Unbiased Differential Visibility Using Fixed-Step Walk-on-Spherical-Caps And Closest Silhouettes

LIFAN WU, NVIDIA, USA
NATHAN MORRICAL, NVIDIA, USA
SAI PRAVEEN BANGARU, NVIDIA, USA
ROHAN SAWHNEY, NVIDIA, USA
SHUANG ZHAO, NVIDIA, USA
CHRIS WYMAN, NVIDIA, USA
RAVI RAMAMOORTHI, NVIDIA, USA
AARON LEFOHN, NVIDIA, USA

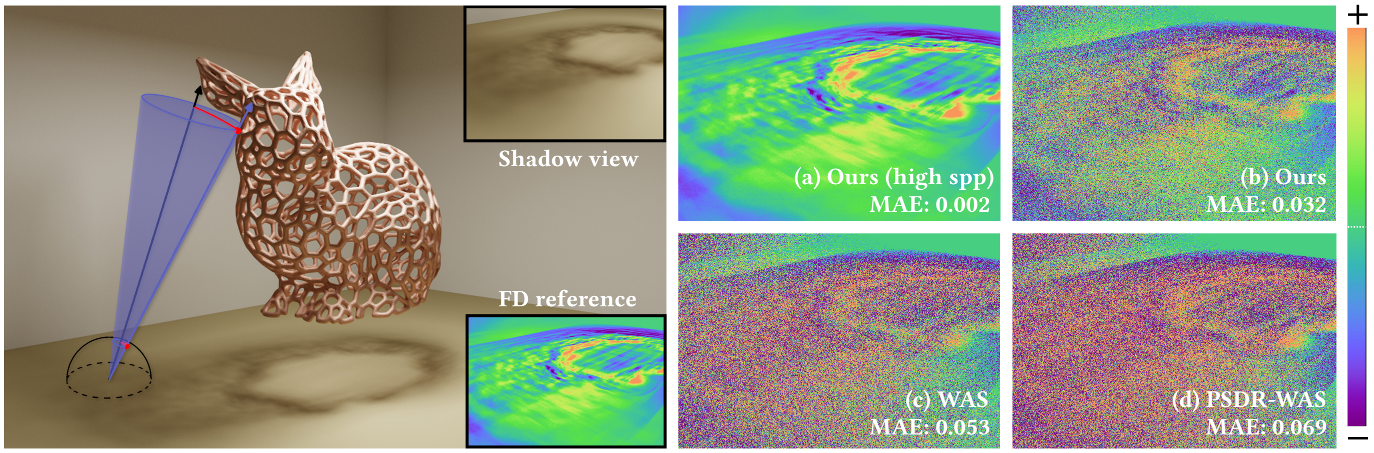


Fig. 1. We introduce a robust method to compute warped-area reparameterization for differential visibility. The key ingredient is a novel velocity construction using fixed-step walk-on-spherical-caps (WoSC) accelerated by cone queries (illustrated in the left image) that find the geodesic closest distance to the boundaries on a unit sphere. In this example, we show derivatives of the shadows, cast by a Voronoi-bunny model [Mehta et al. 2022] with 168k triangles under an area light source, with respect to the y-translation of the bunny. In (a), we present the gradient image computed by our method with a high sample count. In (b)–(d), we show equal-sample comparisons with the baseline methods (WAS [Bangaru et al. 2020] and PSDR-WAS [Xu et al. 2023]).

Computing derivatives of path integrals under evolving scene geometry is a fundamental problem in physics-based differentiable rendering, which requires differentiating discontinuities in the visibility function. Warped-area reparameterization is a powerful technique to compute differential visibility, and key is construction of a velocity field that is continuous in the domain interior and agrees with defined velocities on boundaries. Robustly and efficiently constructing such fields remains challenging.

We present a novel velocity field construction for differential visibility. Inspired by recent Monte Carlo solvers for partial differential equations (PDEs), we formulate the velocity field via Laplace's equation and solve it with a walk-on-spheres (WoS) algorithm. To improve efficiency, we introduce a *fixed-step* WoS that terminates random walks after a fixed step count, resulting in a continuous but non-harmonic velocity field still valid for warped-area reparameterization. Furthermore, to practically apply our method to complex 3D scenes, we propose an efficient cone query to find the closest silhouettes on a boundary. Our cone query finds the closest point under the geodesic distance on a unit sphere, and is analogous to the closest point query by WoS to compute Euclidean distance. As a result, our method generalizes WoS to perform random walks on spherical caps over the unit sphere. We demonstrate that this enables a more robust and efficient unbiased estimator for differential visibility.

Authors' Contact Information: Lifan Wu, lifanw@nvidia.com, NVIDIA, USA; Nathan Morrical, nmorrical@nvidia.com, NVIDIA, USA; Sai Praveen Bangaru, sbangaru@nvidia.com, NVIDIA, USA; Rohan Sawhney, rsawhney@nvidia.com, NVIDIA, USA; Shuang Zhao, shz@ics.uci.edu, NVIDIA, USA; Chris Wyman, cwyman@nvidia.com, NVIDIA, USA; Ravi Ramamoorthi, rramamoorthi@nvidia.com, NVIDIA, USA; Aaron Lefohn, alefohn@nvidia.com, NVIDIA, USA.

ACM 1557-7368/2025/8-ART
https://doi.org/10.1145/3731174

CCS Concepts: • **Computing methodologies → Rendering**.

Additional Key Words and Phrases: Differentiable rendering, Walk on spheres

ACM Reference Format:
Lifan Wu, Nathan Morrical, Sai Praveen Bangaru, Rohan Sawhney, Shuang Zhao, Chris Wyman, Ravi Ramamoorthi, and Aaron Lefohn. 2025. Unbiased Differential Visibility Using Fixed-Step Walk-on-Spherical-Caps And Closest Silhouettes. *ACM Trans. Graph.* 44, 4 (August 2025), 16 pages. https://doi.org/10.1145/3731174

1 Introduction

Physics-based differentiable rendering computes derivatives of radiometric measurements with respect to scene parameters such as object shapes. Recently, differentiable rendering techniques have drawn significant attention due to their applications in a wide range of fields, including but not limited to scene reconstruction, computational fabrication, and 3D generative models.

One of the fundamental problems in physics-based differentiable rendering is computing derivatives of path integrals under evolving scene geometry, which consists of jump discontinuities usually caused by occlusions. This is known as the *differential visibility* problem. One must be careful when differentiating these discontinuities, as directly moving the differentiation operator into the rendering integral does not result in correct derivatives.

To compute differential visibility correctly, two categories of methods have been proposed. The first category involves explicit sampling of visibility discontinuities [Li et al. 2018; Zhang et al. 2019, 2020]. Efficient discontinuity-aware sampling techniques usually rely on guiding data structures and precomputation [Yan et al. 2022; Zhang et al. 2023], since detecting discontinuities in 3D scenes is challenging in general.

Methods in the second category use a geometry-aware reparameterization [Loubet et al. 2019] to avoid sampling discontinuities explicitly, so they do not have to maintain extra data structures. Warped-area reparameterization [Bangaru et al. 2020] is the state-of-the-art method in this category, which applies the divergence theorem to convert the effects of evolving discontinuity boundaries to an area integral defined over the interior domain. The boundary motions are then captured by an interior velocity field that satisfies continuity and boundary conditions. At its core, warped-area reparameterization involves constructing a valid velocity field with these constraints (§3.2). Unfortunately, existing methods for velocity construction suffer from robustness and efficiency issues.

In this paper, we propose a novel velocity field construction for warped-area reparameterization. Our first attempt is formulating the velocity field by Laplace's equation with Dirichlet boundary conditions (§4.1), and our final algorithm is inspired by the walk-on-spheres (WoS) method for solving PDEs "on-demand" at localized regions of interest [Sawhney and Crane 2020]. For the first time, we apply and generalize WoS to warped-area reparameterization for the differential visibility problem, connecting both fields of differentiable rendering and Monte Carlo PDE solvers together.

Our contributions are:

- Introducing a robust and efficient algorithm for warped-area reparameterization, which constructs a valid velocity field using the following components.
- Generalizing the original WoS method in two ways: from random-step to fixed-step (§5.1), and from flat surfaces (walk-on-spheres) to spherical surfaces (walk-on-spherical-caps) (§6.2).
- Deriving directional derivatives of the constructed velocity to ensure unbiased estimation of differential visibility (§5.2).
- Introducing an efficient cone query to find the closest point under the geodesic distance on spherical surfaces (§6.1), enabling a scalable algorithm for complex 3D scenes.

We validate our theory by comparing results of our derivative estimators with references computed by finite differences (Figures 8 and 9), and we demonstrate the robustness and efficiency of our method on differentiable rendering (Figures 1 and 9) and inverse rendering (Fig. 12) examples.

2 Related Work

Our proposed approach applies a Monte Carlo PDE solver that uses a novel closest-silhouette query, to efficiently handle discontinuities in a differentiable renderer. We cover the relevant prior work involving these topics.

2.1 Handling Discontinuities in Differentiable Rendering

Since renderers often exhibit various discontinuities (such as occlusion, discontinuous shading, etc.), a key aspect of differentiating them is computing the non-trivial contribution from these discontinuity boundaries. Our work mainly focuses on visibility discontinuities caused by occlusions. Presently, the set of approaches can be roughly classified into *approximations*, *explicit sampling*, and *reparameterization*.

Approximation-based methods are frequently employed in cases where exact derivatives are unnecessary or derivative accuracy can be improved by increasing the resolution of data structures. Popular works in this category include *SoftRasterizer* and *Implicit Differentiable Renderer* [Liu et al. 2019; Yariv et al. 2021] that use smoothing, while *NVDiffRast* and *Aδ* [Laine et al. 2020; Yang et al. 2022] use pre-filtering. For *physics-based* renderers, which is our focus, approximation-based methods either do not apply or introduce unacceptable bias [Luan et al. 2021; Zhang et al. 2023].

Explicit sampling approaches take an unbiased approach to differentiating physics-based renderers. Li et al. [2018] introduced the edge-sampling method to differentiate a path tracer by decomposing it into "interior" and "boundary" components, and provided an effective yet expensive discontinuity sampler for the latter. Zhang et al. [2020] instead formulated the differentiation in the path space (PSDR). Several techniques have been proposed to improve the sampling efficiency of the boundary integral [Yan et al. 2022; Zhang et al. 2023], usually involving pre-processing and data-structure construction to organize the discontinuities. Since ensuring samplers to be unbiased tends to be the hardest part, some approaches [Zhang et al. 2022; Wang et al. 2024] avoid additional overhead by proposing approximate or relaxed estimators.

Reparameterization-based approaches avoid sampling discontinuities directly, by instead applying a geometry-aware, infinitesimal transformation to the ray or path sampled in standard path tracing. Loubet et al. [2019] were the first to propose such a light-weight reparameterization, though their formulation is biased. Bangaru et al. [2020] proposed warped-area sampling that applies the divergence theorem to the boundary integral to obtain a consistent estimator for the reparameterization. Xu et al. [2023] extended this to the path space by deriving the reparameterized variant of PSDR's boundary integral [Zhang et al. 2020]. Warped-area reparameterization has also been applied to differentiate implicit surfaces such as SDFs [Vicini et al. 2022; Bangaru et al. 2022], where distance information makes computing a valid warp function easy. Our work improves the robustness and efficiency of warped-area reparameterization. Although it might be possible to extend our method to a wider range of geometric representations such as SDFs, we focus on differentiating triangular meshes in this work.

Efficient Differentiable Rendering. With applications in high fidelity 3D reconstruction and generative models, the performance of differentiable rendering is becoming increasingly important. Several works have advanced this area through (i) GPU-based frameworks that leverage hardware acceleration [Jakob et al. 2022a; Bangaru et al. 2023], (ii) systematic optimizations such as adjoint ray tracing [Nimier-David et al. 2020; Vicini et al. 2021], and (iii) variance reduction techniques [Zhang et al. 2021a; Zeltner et al. 2021; Nicolet et al. 2023; Chang et al. 2023; Wang et al. 2023; Belhe et al. 2024], including better data structures for importance sampling explicit discontinuities [Zhang et al. 2021b; Yan et al. 2022; Zhang et al. 2023; Tong et al. 2023].

2.2 Monte Carlo PDE Solvers

Grid-free Monte Carlo methods for solving PDEs have recently emerged as powerful alternatives to grid-based finite-element methods. Unlike the latter, Monte Carlo methods can provide unbiased estimations of a PDE solution at any point of interest, without solving over an entire domain. The most popular approach is the walk-on-spheres (WoS) algorithm, originally by Mueller et al. [1956], and revived into its modern form by Sawhney et al. [2020].

Several extensions have since been proposed to extend the applicable class of PDEs and boundary conditions [Nabizadeh et al. 2021; Sawhney et al. 2022, 2023; Sugimoto et al. 2023; Miller et al. 2024b], while others target variance reduction via caching [Miller et al. 2023; Li et al. 2023], and bidirectional walks [Qi et al. 2022], taking inspiration the Monte Carlo rendering literature. Although prior work in this area is largely limited to random walks in the entire Euclidean space, Sugimoto et al. [2024] introduced a projected walk-on-spheres (PWoS) method that generalizes WoS for surface PDEs. Similarly, our *walk-on-spherical-caps* algorithm also generalizes WoS to spherical surfaces, but for the purpose of reparameterizing differential path integrals rather than solving PDEs. Exploring the applicability of PWoS to our setting is left to future work.

Leveraging recent developments in physics-based differentiable rendering, several recent works develop differential-variants of WoS to estimate parameter derivatives for PDE-constrained inverse problems [Miller et al. 2024a; Yilmazer et al. 2024; Yu et al. 2024]. Though our work focuses on computing spatial derivatives on spherical surfaces, it might be possible to gain additional insights for efficient derivative computation from this line of work.

Finally, Roger et al. [2005] proposed a Monte Carlo estimator to compute derivatives of integrals with deformable domains for sensitivity analysis, which also leads to a similar boundary value interpolation problem. Although they formulated this problem as an n-D Laplace's equation with Dirichlet boundary conditions, their work preceded the recent emerging interest in WoS-based techniques.

2.3 Accelerated Spatial Queries in Graphics

Accelerated queries such as closest-point, k-nearest-neighbors and ray-intersection account for a very large body of work with wide ranging impact. We point the reader to the work by Hjaltson et al. [1999] and Hanan et al. [2005] for general-purpose queries. Our focus is on SIMD-based spatial queries since we target differentiable renderers, which now use hardware-accelerated ray tracing [Jakob et al. 2022a; Bangaru et al. 2023].

Euclidean Closest Point/Silhouette Queries. There are several proposed methods for accelerated closest-point queries. The linear BVH approach by Jakob et al. [Jakob and Guthe 2021] is considered one of the most performant, though there are several alternatives that operate on grids [Purcell et al. 2003; Leite et al. 2009; Schauer et al. 2016]. Such queries have been used by walk-on-spheres [Sawhney and Crane 2020] (closest point query) and walk-on-stars [Sawhney et al. 2023] (in particular, a closest *silhouette* point query that leverages the spatialized normal cone hierarchy by Johnson et al. [2001]). However, these queries operate on, and compute distances between, points in Euclidean space. Our *closest silhouette ray* query computes geodesic distance for geometry *projected* onto an arbitrarily-positioned unit sphere.

3 Preliminaries

In this section, we summarize the necessary and most relevant background for our work. Interested readers are encouraged to refer to the original papers [Zhang et al. 2020; Bangaru et al. 2020; Xu et al. 2023] for more details.

3.1 Warped-Area Reparameterization

Path integrals for primal rendering. The path integral formulation [Veach 1997] is the foundation of physics-based rendering. It expresses a radiometric measurement as

$$I = \int_{\Omega} f(\bar{\boldsymbol{p}})\, \mathrm{d}\mu(\bar{\boldsymbol{p}})\,, \tag{1}$$

where $\bar{\boldsymbol{p}} = (\boldsymbol{p}_0, \ldots, \boldsymbol{p}_N)$ is a light path of length N with $\boldsymbol{p}_0$ on a light source and $\boldsymbol{p}_N$ on the camera, Ω is the space of all finite-length light paths, f is the measurement contribution function, and μ is the area-product measure.

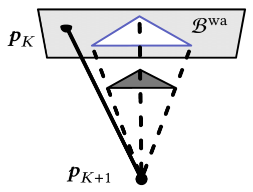
Differential path integrals. Since the integrand f of Eq. (1) has visibility discontinuities caused by geometric occlusions, we use the Reynolds transport theorem to differentiate the path integral with respect to any scene parameter θ, leading to a differential path integral [Zhang et al. 2020]:

$$\partial_\theta I = \int_{\Omega} \partial_\theta f(\bar{\boldsymbol{p}})\, \mathrm{d}\mu(\bar{\boldsymbol{p}}) + \underbrace{\int_{\partial\Omega} f(\bar{\boldsymbol{p}}) V(\bar{\boldsymbol{p}})\, \mathrm{d}\dot{\mu}(\bar{\boldsymbol{p}})}_{=: I_{\mathrm{bdr}}}\,. \tag{2}$$

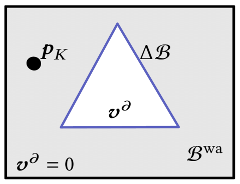
The first term on the right-hand side is an integral over the same path space Ω as the path integral I. The second term I_{bdr} is a *boundary path integral* defined over the boundary path space $\partial\Omega$, which consists of light paths which graze the discontinuous contours of the scene objects. The scalar normal velocity $V(\bar{\boldsymbol{p}})$ captures the motion of the evolving visibility discontinuities.

ALGORITHM 1: Estimating the boundary path integral I_{bdr}.

```
EstIBoundary()
begin
    $\bar{p}$, pdf ← SamplePath();
    $I_{\mathrm{bdr}} \leftarrow 0$;
    for $K = 0$ to $N - 1$ do
        Compute $f_K$ and $\nabla f_K$ via automatic differentiation;
        /* Use fixed-step WoS (Algorithm 3) or
           fixed-step WoSC (Algorithm 7)                  */
        $\boldsymbol{v}_K, \nabla \cdot \boldsymbol{v}_K \leftarrow$ ComputeV($\boldsymbol{p}_K, \boldsymbol{p}_{K+1}$);
        $I_{\mathrm{bdr}} \leftarrow I_{\mathrm{bdr}} + (\nabla f_K \cdot \boldsymbol{v}_K) + f_K (\nabla \cdot \boldsymbol{v}_K)$;
    end
    return $I_{\mathrm{bdr}}$/pdf;
end
```



(a) 3D View



$\boldsymbol{p}_K$
$\Delta\mathcal{B}$
$\boldsymbol{v}^\partial$
$\boldsymbol{v}^\partial = 0$
$\mathcal{B}^{\mathrm{wa}}$

(b) Projection View

Fig. 2. We illustrate the geometric configuration for warped-area reparameterization (WAR) given a single path segment $\overline{\boldsymbol{p}_K \boldsymbol{p}_{K+1}}$ in two views. WAR aims to construct a continuous velocity field $\boldsymbol{v}$ over the visible region $\mathcal{B}^{\mathrm{wa}}$ (the gray region), while ensuring $\boldsymbol{v}$ agrees with the boundary velocities $\boldsymbol{v}^\partial$ defined on the set of boundary curves $\partial\mathcal{B}^{\mathrm{wa}}$. The boundary curves include 1) visibility boundaries $\Delta\mathcal{B}$ caused by occlusion (the blue triangle) and 2) topological boundaries of surfaces (the black rectangle).

Warped-area reparameterization. The key insight of warped-area reparameterization [Bangaru et al. 2020; Xu et al. 2023] is using the divergence theorem to convert *boundary* integrals to *area* integrals. With that, the boundary path integral in Eq. (2) can be rewritten as

$$I_{\mathrm{bdr}} = \int_\Omega \sum_{K=0}^{N-1} [\nabla \cdot (f_K \, \boldsymbol{v}_K)] \, (\boldsymbol{p}_K) \, \mathrm{d}\mu(\bar{\boldsymbol{p}}) \,, \tag{3}$$

where f_K is the measurement contribution function that treats $\boldsymbol{p}_K$ as the only changing variable (all the other vertices are considered as fixed), and $\boldsymbol{v}_K$ is a continuous velocity field. Constructing $\boldsymbol{v}_K$ is the main focus of this paper.

We outline an estimator for I_{bdr} in Algorithm 1. Given a sampled full light path $\bar{\boldsymbol{p}} = (\boldsymbol{p}_0, \ldots, \boldsymbol{p}_N)$, the warped-area reparameterization is computed at every vertex $\boldsymbol{p}_K$ for $0 \le K \le N-1$ (Line 5), meaning that we need to construct various continuous velocity fields $\boldsymbol{v}_K$ for each $\boldsymbol{p}_K$. The measurement contribution function f_K and its gradient ∇f_K can be directly computed via automatic differentiation (AD). On the other hand, constructing a valid velocity field $\boldsymbol{v}_K$ and computing its divergence $\nabla \cdot \boldsymbol{v}_K$ (Line 7) is the main challenge of warped-area reparameterization, which we detail next.

3.2 Problem: Construction of Velocity Fields

We focus on reparameterization over a single path segment $\overline{\boldsymbol{p}_K \boldsymbol{p}_{K+1}}$ and illustrate this in Fig. 2. Assuming $\boldsymbol{p}_{K+1}$ to be the shading point, we aim to construct a velocity field $\boldsymbol{v}$ [1] around the other endpoint $\boldsymbol{p}_K$ for warped-area reparameterization. Specifically, let $\mathcal{B}$ denote object surfaces (e.g., the surface where $\boldsymbol{p}_K$ resides). The velocity field

$$\boldsymbol{v} : \mathcal{B}^{\mathrm{wa}} \to \boldsymbol{T}_{\boldsymbol{p}_K}(\mathcal{B}) \tag{4}$$

should be defined over the visible surface regions with respect to the shading point (the gray region in Fig. 2), i.e.,

$$\mathcal{B}^{\mathrm{wa}} = \{\boldsymbol{x} \in \mathcal{B} \mid \mathbb{V}(\boldsymbol{x} \leftrightarrow \boldsymbol{p}_{K+1}) = 1\} \subset \mathcal{B} \,, \tag{5}$$

and the velocity vectors live on $\boldsymbol{T}_{\boldsymbol{p}_K}(\mathcal{B})$, representing the tangent plane at $\boldsymbol{p}_K$.

The boundary $\partial\mathcal{B}^{\mathrm{wa}}$ of this domain $\mathcal{B}^{\mathrm{wa}}$ consists of two types of boundary curves:

- Visibility boundaries $\Delta\mathcal{B}$ caused by occlusion (the blue triangle in Fig. 2; they are the perspective projections of the occluder's edges). The visibility boundaries may depend on the evolving scene parameter θ, so their velocities $\boldsymbol{v}^\partial = \partial_\theta \boldsymbol{x}$ for all $\boldsymbol{x} \in \Delta\mathcal{B}$ are uniquely determined by the *material-form reparameterization* [Zhang et al. 2020; Xu et al. 2023]. [2]
- Topological boundaries of object surfaces (e.g., edges of a plane shown as the black rectangle in Fig. 2). Because of the material-form reparameterization, they are considered to be static and thus have zero velocity, i.e., $\boldsymbol{v}^\partial = \boldsymbol{0}$.

Our goal is to construct $\boldsymbol{v}$ such that

$$\begin{aligned} \boldsymbol{v} &\in C^0 && \text{on } \mathcal{B}^{\mathrm{wa}} \,, \\ \boldsymbol{v} &= \boldsymbol{v}^\partial && \text{on } \partial\mathcal{B}^{\mathrm{wa}} \,. \end{aligned} \tag{6}$$

While the boundary velocities are uniquely defined, there exist infinitely many valid interior velocity fields $\boldsymbol{v}$ that satisfy the continuity [3] and boundary consistency requirements in Eq. (6). All valid velocity fields lead to the same result for the boundary integral in Eq. (3), but they may have different implications regarding sampling efficiency and robustness, leaving a large design space for constructing $\boldsymbol{v}$.

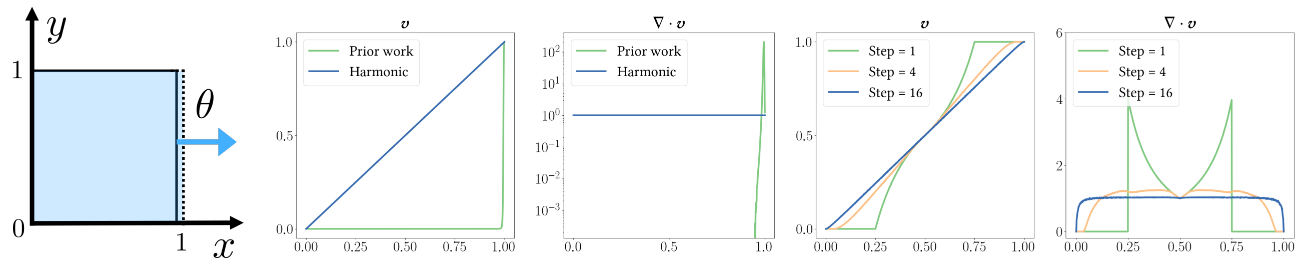
Existing solution. In the following, we present one approach to constructing a valid velocity field $\boldsymbol{v}$, which was used in prior work [Bangaru et al. 2020; Xu et al. 2023]. It takes two steps. First, a discontinuous velocity field $\boldsymbol{v}^{\mathrm{dis}}$ is defined on the entire surface $\mathcal{B}$ such that $\boldsymbol{v}^{\mathrm{dis}}(\boldsymbol{p}) = \boldsymbol{v}^\partial(\boldsymbol{p})$ for all $\boldsymbol{p}$ on the visibility boundaries $\Delta\mathcal{B}$. Then, for any interior point $\boldsymbol{p} \in \mathcal{B}^{\mathrm{wa}}$, a few auxiliary points $\boldsymbol{q}$ are sampled around it and a continuous velocity field $\boldsymbol{v}$ is computed by smoothing $\boldsymbol{v}^{\mathrm{dis}}$ using a spatially varying weighting function w:

$$\boldsymbol{v}(\boldsymbol{p}) = \frac{\int_{\mathcal{B}} w(\boldsymbol{q}; \boldsymbol{p}) \boldsymbol{v}^{\mathrm{dis}}(\boldsymbol{q}) \, \mathrm{d}A(\boldsymbol{q})}{\int_{\mathcal{B}} w(\boldsymbol{q}; \boldsymbol{p}) \, \mathrm{d}A(\boldsymbol{q})} \,, \tag{7}$$

where the weighting function is defined as

$$w(\boldsymbol{q}; \boldsymbol{p}) = \frac{1}{D(\boldsymbol{q}; \boldsymbol{p}) + B(\boldsymbol{q})} \,. \tag{8}$$

[1]We drop the subscript K of the velocity field for notational simplicity.
[2]For the exact formula of $\boldsymbol{v}^\partial$, please refer to Section 3.1.2 of Xu et al.'s [2023] paper.
[3]In fact, $\boldsymbol{v}$ also needs to be differentiable almost everywhere except on a set of measure zero. For example, $\boldsymbol{v}$ cannot be the Weierstrass function.



(a) Problem (b) Prior work [Bangaru et al. 2020] (c) Our fixed-step WoS

Fig. 3. A 1D toy example. (a) We want to use warped-area reparameterization to compute the derivative of the integral I defined in Eq. (10) (the area of the blue region) with respect to the changing discontinuity θ. (b) Plots of the velocity functions and their divergences constructed by the prior method (Eq. (7)) and solving Laplace's equation. (c) Plots of the velocity functions and their divergences constructed by our M-step WoS method with $M = 1$, 4, and 16.

Here, $D(\boldsymbol{q};\boldsymbol{p})$ is the distance function that goes to zero when $\boldsymbol{q}$ approaches $\boldsymbol{p}$, and $B(\boldsymbol{q})$ is the boundary test function that is designed to approach zero when $\boldsymbol{q}$ approaches the visibility boundary. By design, when $\boldsymbol{p}$ is on the visibility boundary, the weighting function $w(\cdot;\boldsymbol{p})$ should behave like a Dirac delta function so that the velocity fields are consistent, i.e., $\boldsymbol{v}(\boldsymbol{p}) = \boldsymbol{v}^{\text{dis}}(\boldsymbol{p})$.

Limitations of prior work. The weighted-average-based velocity interpolation strategy described above has several issues.

- **Biased estimator.** Prior work [Bangaru et al. 2020; Xu et al. 2023] used a biased-but-consistent estimator to compute $\boldsymbol{v}$ in Eq. (7). Specifically, integrals at the numerator and the denominator in Eq. (7) are estimated individually first and followed by the division; it is biased since $\mathbb{E}[1/f] \neq 1/\mathbb{E}[f]$. Although debiasing the estimator using Russian roulette is possible [McLeish 2011; Bangaru et al. 2020], it is expensive and numerically unstable because the weighting function w is unbounded (w approaches the Dirac delta function when points are close to the boundary).
- **Robustness.** In Eq. (8), the distance function and the boundary test function are designed empirically, involving a few hyperparameters that are counterintuitive to tune in practice. Additionally, the unbounded weighting function w often has large values near the boundary and leads to numerical instability.
- **Sampling efficiency.** The resulting velocity field $\boldsymbol{v}$ is zero in most of the interior domain and changes drastically near the visibility boundary. Therefore, its divergence $\nabla\cdot\boldsymbol{v}$ has very narrow support of non-zero values, concentrating near discontinuous boundaries, which makes it difficult to sample.

3.3 Toy Example in 1D

To develop intuition for differentiating integrals with discontinuities, we first show a 1D toy example in Fig. 3. Let y be a step function whose jumping discontinuity is controlled by θ:

$$y(x;\theta) = \begin{cases} 1, & \text{if } x < \theta, \\ 0, & \text{if } x \geq \theta. \end{cases} \tag{9}$$

We want to compute the derivative of an integral

$$I(\theta) = \int_0^\infty y(x;\theta)\,\mathrm{d}x \tag{10}$$

with respect to θ at $\theta_0 = 1$. Note that this derivative can be computed analytically as $\partial_\theta I\,|_{\theta=\theta_0} = 1$.

Alternatively, one can use warped-area reparameterization to compute this derivative as

$$\partial_\theta I\,|_{\theta=1} = \int_0^1 (\nabla\cdot v)\,(x)\,\mathrm{d}x\,, \tag{11}$$

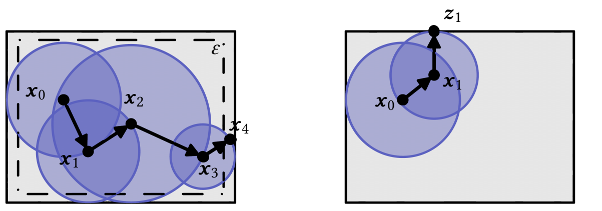
where v should be continuous within the interval $[0,1]$ and equal to the boundary velocities as $v(0) = 0$ (the left end is static) and $v(1) = 1$ (the right end is changing). As shown in Fig. 3 (b), the velocity $\boldsymbol{v}$ constructed using Eq. (7) is mostly zero and changes dramatically near the discontinuity. As a result, its divergence $\nabla\cdot\boldsymbol{v}$ (it is actually $\partial_x v$ in 1D) has a narrow support of non-zero values.

A better way to construct v, which we will discuss in §4.1, is solving a Laplace's equation with Dirichlet boundary conditions $v(0) = 0$ and $v(1) = 1$, leading to a harmonic velocity function $v(x) = x$ (see the blue curve in Fig. 3 (b)). Since its divergence is constant, a Monte Carlo estimator for computing $\partial_\theta I$ in Eq. (11) will have zero variance in this 1D example.

In general, a harmonic velocity field does not have analytic solutions, so we have to compute it numerically. The WoS method [Muller 1956; Sawhney and Crane 2020] is a powerful tool to construct such harmonic velocity fields, but it introduces noise due to random walks. We propose a *fixed-step* WoS method (§5) to improve the efficiency of velocity construction. Fig. 3 (c) shows the velocity functions constructed using different numbers of steps and their corresponding divergences. They are all valid constructions for warped-area reparameterization, leading to the same result of $\partial_\theta I = 1$.

4 Overview

We aim to construct a valid velocity field $\boldsymbol{v}$ over the interior domain given the boundary velocities $\boldsymbol{v}^\partial$. Because the requirements of $\boldsymbol{v}$ formulated in Eq. (6) look like a boundary value problem, one could construct $\boldsymbol{v}$ as the solution to a Laplace's equation with Dirichlet boundary conditions. Our first attempt to tackle the velocity construction problem is using the walk-on-spheres (WoS) algorithm to solve this Laplace's equation (§4.1). We show that it already addresses the issues of existing methods (discussed earlier in §3.2) to some extent: the WoS estimators are unbiased and robust, and the



(a) Random-step WoS (b) Fixed-step WoS with $M = 1$.

Fig. 4. (a) The original random-step WoS algorithm starts at x_0 and stops at x_4, which is inside the ε-shell near the boundary. (b) Our fixed-step walk stops after one step at x_1 and grabs the value from its closest point z_1 on the boundary.

resulting harmonic velocity field is very smooth and easy to sample (see Fig. 3 (b)).

There are several ways to solve Laplace's equations such as finite element methods (FEM). We instead choose to use WoS because it can be evaluated "on demand" at any point without discretizing and computing solutions over the entire domain. For the first time, we show that the warped-area reparameterization for differential visibility can be computed using WoS, connecting the realms of differentiable rendering and Monte Carlo PDE solvers.

Moreover, to push this idea further, we discuss two challenges that emerge from the WoS-based velocity construction and propose our solutions to address them (§4.2). First, we introduce a *fixed-step* WoS algorithm that terminates after M steps, and grabs values from the closest points on the boundaries, making the velocity estimation more efficient and robust (§5). Second, to make our method scalable to complex 3D scenes with a large number of triangles, we present a novel *walk-on-spherical-caps* algorithm equipped with an efficient *closest silhouette ray* query, which extends the original WoS method from flat surfaces (2D Euclidean space) to spherical surfaces (§6).

4.1 First Attempt: Constructing v Using Walk-on-Spheres

According to our problem specification in §3.1, given boundary velocities v^∂, our goal is to construct a continuous velocity field v over the interior domain $\mathcal{B}^{\text{wa}}$ which is a subset of 2D surfaces (e.g., the triangle where p resides) defined in Eq. (5). This is a typical boundary value problem that can be modeled as a Laplace's equation with Dirichlet boundary conditions:

$$\begin{aligned} \Delta v(p) &= 0 && \text{on } \mathcal{B}^{\text{wa}}\,, \\ v(p) &= v^\partial(p) && \text{on } \partial\mathcal{B}^{\text{wa}}\,. \end{aligned} \tag{12}$$

The velocity construction problem boils down to solving this equation on 2D surfaces. The resulting velocity field v, which is the solution to this PDE, satisfies the continuity (v is harmonic) and boundary consistency (constrained by the Dirichlet boundary conditions) requirements defined in Eq. (6). We use WoS to estimate the harmonic velocity field v, which we briefly recap below:

Walk-on-Spheres. The WoS algorithm estimates the solution $v(p)$ at any interior point $p \in \mathcal{B}^{\text{wa}}$ by sampling random walks jumping on n-dimensional spheres; in our case, it is technically 2D "walk-on-circles" because the Laplace's equation is defined on 2D surfaces. The WoS algorithm is based on the mean value property of harmonic functions:

$$v(p) = \frac{1}{|B(p,r)|}\int_{B(p,r)} v(y)\,\mathrm{d}y = \frac{1}{|\partial B(p,r)|}\int_{\partial B(p,r)} v(z)\,\mathrm{d}z\,, \tag{13}$$

where $B(p,r)$ is a 2D inner disk that is centered at p with radius r and entirely inside the interior domain $\mathcal{B}^{\text{wa}}$, and its boundary $\partial B(p,r)$ is a circle.

We illustrate a 2D WoS random walk in Fig. 4 (a). The walk starts at $x_0 = p$. At each step k, it randomly jumps to a next point x_{k+1} uniformly sampled on an inner circle. To achieve faster convergence, we often use the maximum inner circle ∂B_{x_k} that is centered at x_k with radius

$$D(x_k) = \|x_k - \text{cp}(x_k)\|\,, \tag{14}$$

where $\text{cp}(x_k)$ indicates the closest point on the boundaries with respect to x_k:

$$\text{cp}(x_k) = \operatorname*{argmin}_{z\in\partial\mathcal{B}^{\text{wa}}} \|x_k - z\|\,. \tag{15}$$

Note that $D(\cdot)$ is actually a (unsigned) distance field to the boundaries. The random walk continues until it reaches a point near the boundaries, i.e., the distance function $D(x_k)$ is smaller than a stopping threshold ε. In summary, a WoS estimator can be formulated recursively as

$$\langle v(x_k)\rangle = \begin{cases} v^\partial(\text{cp}(x_k)), & \text{if } D(x_k) < \varepsilon, \\ \langle v(x_{k+1})\rangle, & x_{k+1} \sim \mathcal{U}(\partial B_{x_k}), \end{cases} \tag{16}$$

where $\mathcal{U}(\cdot)$ denotes the uniform distribution. This WoS estimator has negligible bias that comes from the ε-shell near the boundaries, and it terminates in $O(\log 1/\varepsilon)$ steps [Binder and Braverman 2012]

We also want to compute the gradient of v [Sawhney and Crane 2020], which can be estimated recursively with Eq. (16) :

$$\langle \nabla v(x_0)\rangle = \frac{|\partial B_x|}{|B_x|}\underbrace{\langle u(x_1)\rangle}_{\text{Eq. (16)}}\frac{x_1 - x_0}{D(x_0)}\,, \quad x_1 \sim \mathcal{U}(\partial B_{x_0})\,. \tag{17}$$

4.2 Challenges and Our Solution

Inefficiency due to long walks. Random walks in the original WoS algorithm terminate when they reach the ε-shell near the boundaries, which takes a random number of steps (so we refer to it as *random-step* WoS). This randomness may lead to inefficiency because it might take many steps for a random walk to terminate when the shapes of the boundaries are complex.

To improve the efficiency of the WoS algorithm, we introduce a *fixed-step* variant that terminates random walks after a fixed number of M steps and grabs the value from the closest point on the boundaries (see Fig. 4 (b)). Since a random walk is guaranteed to terminate after M steps, we can avoid having potentially long walks, especially when the stopping threshold ε is small (to keep the bias low) or the shapes of boundaries are complex. It also creates a more balanced workload distribution for parallel computing; otherwise, multiple random walks in the random-step WoS may stop in a different number of steps. Lastly, the hyperparameter of fixed-step WoS is the maximum number of steps M rather than the stopping threshold ε. It removes the bias caused by ε and enables the flexibility of balancing the computational cost and the smoothness of the velocity field.

We discuss the details of our fixed-step WoS algorithm in §5. A critical question is whether the velocity field constructed by the fixed-step WoS is still valid for warped-area reparameterization. In §5.1, we demonstrate that the constructed velocity field still satisfies the continuity and boundary consistency requirements in Eq. (6), even though it is not harmonic anymore. Once we validate the velocity construction, we show how to estimate its directional derivatives and compute the reparameterization using our fixed-step WoS estimators in §5.2.

Scalability with geometric complexity. In the WoS process, computing the closest point function $\mathrm{cp}(\cdot)$ defined in Eq. (15) can be computationally expensive. It requires explicitly computing the visibility boundaries with respect to arbitrary shading points, which effectively treats every shading point as a "camera" and "rasterizes" all the triangles in the scene (as potential occluders). Each "rasterization" takes linear time proportional to the number of triangles, and it is impractical to precompute the visibility boundaries at all shading points.

In §6, we show how to make our method scalable to complex 3D scenes with a large number of triangles. We observe that, although $\mathrm{cp}(\cdot)$ is expensive to compute in general, its geodesic variant $\mathrm{cp}_{\mathbb{S}^2}(\cdot)$ defined spherical surfaces as Eq. (30) can be efficiently computed by traversing a spatial hierarchy (§6.1). With this efficient geodesic closest point query, we introduce the walk on spherical caps (WoSC) algorithm, which generalizes the original WoS method from flat surface (2D Euclidean space) to the spherical domain $\mathbb{S}^2$. We develop fixed-step WoSC estimators to construct a valid velocity field on the spherical surface and estimate its directional derivatives, which are used in our final warped-area reparameterization algorithm (§6.2).

5 Reparameterization Using Fixed-Step Walk-on-Spheres

To alleviate the inefficiency caused by potential long random walks in the original WoS algorithm, we propose a new *fixed-step* walk-on-spheres method that is guaranteed to terminate after a fixed number of M steps. We first prove that the constructed velocity field $\boldsymbol{v}^{(M)}$ satisfies the continuity and boundary consistency requirements described in Eq. (6) (§5.1). Then, we derive its directional derivatives $\partial_{\boldsymbol{d}}\boldsymbol{v}^{(M)}$ with respect to any vector $\boldsymbol{d}$ on a tangent plane, and we present the warped-area reparameterization algorithm using our fixed-step WoS estimators for $\boldsymbol{v}^{(M)}$ and $\partial_{\boldsymbol{d}}\boldsymbol{v}^{(M)}$ (§5.2).

5.1 Walk-on-Spheres: From Random Steps to Fixed Steps

Our fixed-step WoS algorithm terminates in M steps rather than a random number of steps (stop until reaching the ε-shell) as in the original WoS algorithm. Besides the aforementioned benefits in efficiency and robustness, using the fixed-step WoS for warped-area reparameterization actually makes more sense because the reparameterization only requires a continuous velocity field; a harmonic one with C^{∞}-continuity is not needed for this purpose.

We illustrate our fixed-step WoS algorithm in Fig. 4 (b) and present the pseudocode in Algorithm 2. It is very similar to the original WoS except that 1) random walks will terminate after M steps (Line 5) and 2) the next point is sampled inside the disk rather than on the

ALGORITHM 2: Estimating the velocity $\boldsymbol{v}^{(M)}$

```
1  EstV($\boldsymbol{p}_K, \boldsymbol{p}_{K+1}, M$)
2  begin
3      Compute the visibility boundaries $\Delta\mathcal{B}$ with respect to $\boldsymbol{p}_{K+1}$;
       /* Find the closest point to boundaries */
4      $\boldsymbol{z} \leftarrow \mathrm{cp}(\boldsymbol{p}_K)$;
       /* Terminate if reaching the maximum step */
5      if $M = 0$ then
6          $\boldsymbol{v} \leftarrow \boldsymbol{v}^{\partial}(\boldsymbol{z})$;
7      end
8      else
9          $D \leftarrow \|\boldsymbol{p}_K - \boldsymbol{z}\|$;
10         $\boldsymbol{y}, \mathrm{pdf} \leftarrow \mathrm{SampleInsideDisk}(\boldsymbol{p}_K, D)$;
11         $\mathrm{area} \leftarrow \pi D^2$;
12         $\boldsymbol{v} \leftarrow$ EstV($\boldsymbol{y}, \boldsymbol{p}_{K+1}, M-1$)$/(\mathrm{pdf} \cdot \mathrm{area})$;
13     end
14     return $\boldsymbol{v}$;
15 end
```

circle (Line 10; since we use the volumetric version [4] of the mean value theorem). In the following, we will verify that our M-step WoS algorithm (for $M > 0$) constructs a valid velocity field that satisfies the continuity and boundary consistency requirements.

Base cases. We start with $M = 0$, just grabbing values from the closest point on the boundaries without any walk. Assuming that $\boldsymbol{v}^{\partial}$ is continuous on the boundaries, this results in a piecewise continuous function

$$\boldsymbol{v}^{(0)}(\boldsymbol{p}) = \boldsymbol{v}^{\partial}(\mathrm{cp}(\boldsymbol{p}))\,. \tag{18}$$

Clearly, this function $\boldsymbol{v}^{(0)}$ satisfies boundary consistency because $\mathrm{cp}(\boldsymbol{p}) = \boldsymbol{p}$ for any $\boldsymbol{p} \in \partial\mathcal{B}^{\mathrm{wa}}$. However, it does not meet the continuity requirement because it has discontinuities when $\boldsymbol{p}$ is on the medial axis of the boundaries, i.e., $\boldsymbol{p}$ has more than one closest point.

When $M = 1$, we use the volumetric version of the mean value property and construct $\boldsymbol{v}^{(1)}$ by taking average of $\boldsymbol{v}^{(0)}$ over the maximum inner disk:

$$\boldsymbol{v}^{(1)}(\boldsymbol{p}) = \frac{1}{|B_{\boldsymbol{p}}|}\int_{B_{\boldsymbol{p}}} \boldsymbol{v}^{(0)}(\boldsymbol{y})\,\mathrm{d}\boldsymbol{y}, \tag{19}$$

where $B_{\boldsymbol{p}}$ is the maximum inner disk that is centered at $\boldsymbol{p}$ and has radius $D(\boldsymbol{p})$, and $|B_{\boldsymbol{p}}| = \pi(D(\boldsymbol{p}))^2$ denotes its area. Note that $\boldsymbol{v}^{(1)}$ also satisfies the boundary consistency. We prove its continuity, i.e. $\boldsymbol{v}^{(1)} \in \boldsymbol{C}^0$, in Appendix A.

Generalization. For any $M > 1$, the fixed-step WoS process runs recursively and we have

$$\boldsymbol{v}^{(M)}(\boldsymbol{p}) = \frac{1}{|B_{\boldsymbol{p}}|}\int_{B_{\boldsymbol{p}}} \boldsymbol{v}^{(M-1)}(\boldsymbol{y})\,\mathrm{d}\boldsymbol{y}. \tag{20}$$

Starting from $\boldsymbol{v}^{(1)} \in \boldsymbol{C}^0$, each step of integration raises the smoothness of the function by one degree. So $\boldsymbol{v}^{(M)} \in \boldsymbol{C}^{M-1} \subseteq \boldsymbol{C}^0$ for any $M \geq 1$ is a valid velocity field for warped-area reparameterization.

[4] Sampling on the circle's boundary as was originally done in the work of Sawhney and Crane [2020] will break the continuity of the constructed velocity $\boldsymbol{v}$. This is easy to verify using the same 1D example in §3.3.

ALGORITHM 3: Computing the velocity $\boldsymbol{v}$ and its divergence $\nabla \cdot \boldsymbol{v}$.

```
1  ComputeV($\boldsymbol{p}_K, \boldsymbol{p}_{K+1}$)
2  begin
       /* Estimate the velocity $\boldsymbol{v}$ */
3      $\boldsymbol{v} \leftarrow$ EstV($\boldsymbol{p}_K, \boldsymbol{p}_{K+1}, M$) ;        // Algorithm 2
       /* Estimate the divergence $\nabla \cdot \boldsymbol{v}$ */
4      $\nabla \cdot \boldsymbol{v} \leftarrow 0$;
5      Specify a pair of orthogonal unit vectors $\boldsymbol{s}$ and $\boldsymbol{t}$ on the tangent
         plane at $\boldsymbol{p}_K$;
6      for $\boldsymbol{d} \in \{\boldsymbol{s}, \boldsymbol{t}\}$ do
7          $\partial_{\boldsymbol{d}}\boldsymbol{v} \leftarrow$ EstDerV($\boldsymbol{p}_K, \boldsymbol{p}_{K+1}, \boldsymbol{d}, M$) ;        // Algorithm 4
8          $\nabla \cdot \boldsymbol{v} \leftarrow \nabla \cdot \boldsymbol{v} + (\partial_{\boldsymbol{d}}\boldsymbol{v} \cdot \boldsymbol{d})$;
9      end
10     return $\boldsymbol{v}, \nabla \cdot \boldsymbol{v}$;
11 end
```

As M goes to infinity, our M-step WoSC behaves like the random-step variant, resulting in a harmonic function with C^∞-continuity.

5.2 Reparameterization with $\boldsymbol{v}^{(M)}$ and $\partial_{\boldsymbol{d}}\boldsymbol{v}^{(M)}$

In addition to $\boldsymbol{v}^{(M)}$, warped-area reparameterization also requires computing the divergence of the constructed velocity field $\boldsymbol{v}^{(M)}$, which is outlined in Algorithm 3. Since the field $\boldsymbol{v}^{(M)}$ lives on the tangent plane at $\boldsymbol{p}_K$ (see Eq. (4)), we can find a pair of orthogonal unit vectors $(\boldsymbol{s}, \boldsymbol{t})$ on this tangent plane (Line 5) and formulate the divergence as

$$\nabla \cdot \boldsymbol{v}^{(M)} = \begin{pmatrix} \partial_s \\ \partial_t \end{pmatrix} \cdot \begin{pmatrix} \boldsymbol{v}^{(M)} \cdot \boldsymbol{s} \\ \boldsymbol{v}^{(M)} \cdot \boldsymbol{t} \end{pmatrix} = (\partial_s \boldsymbol{v}^{(M)} \cdot \boldsymbol{s}) + (\partial_t \boldsymbol{v}^{(M)} \cdot \boldsymbol{t})\,. \tag{21}$$

As a result, computing $\nabla \cdot \boldsymbol{v}^{(M)}$ boils down to computing the directional derivative $\partial_{\boldsymbol{d}}\boldsymbol{v}^{(M)}$ with respect to any unit vector $\boldsymbol{d}$ on the tangent plane at $\boldsymbol{p}_K$ (Lines 6–9).

Estimating $\partial_{\boldsymbol{d}}\boldsymbol{v}^{(M)}$. Since $\boldsymbol{v}^{(M)}$ is not harmonic, the gradient estimator for the original WoS described in Eq. (17) does not work anymore. In the following, we derive the directional derivative $\partial_{\boldsymbol{d}}\boldsymbol{v}^{(M)}$ and develop an unbiased fixed-step WoS estimator for it (presented in Algorithm 4).

We first take the derivative on both sides of Eq. (20) and apply the product rule, yielding

$$\begin{aligned} \partial_{\boldsymbol{d}}\boldsymbol{v}^{(M)}(\boldsymbol{p}) = {} & \partial_{\boldsymbol{d}} \left(\frac{1}{|B_{\boldsymbol{p}}|} \right) \underbrace{\int_{B_{\boldsymbol{p}}} \boldsymbol{v}^{(M-1)}(\boldsymbol{y})\,\mathrm{d}\boldsymbol{y}}_{I^{(M-1)}(\boldsymbol{p})} \\ & + \frac{1}{|B_{\boldsymbol{p}}|} \underbrace{\partial_{\boldsymbol{d}} \left(\int_{B_{\boldsymbol{p}}} \boldsymbol{v}^{(M-1)}(\boldsymbol{y})\,\mathrm{d}\boldsymbol{y} \right)}_{\partial_{\boldsymbol{d}} I^{(M-1)}(\boldsymbol{p})}. \end{aligned} \tag{22}$$

The first term on the RHS of Eq. (22) is easier to handle. The derivative of the normalization factor $\partial_{\boldsymbol{d}}(1/|B_{\boldsymbol{p}}|)$ can be computed as

$$\partial_{\boldsymbol{d}} \left(\frac{1}{|B_{\boldsymbol{p}}|} \right) = \partial_{\boldsymbol{d}} \left(\frac{1}{\pi D^2} \right) = -\frac{2}{\pi D^3} \partial_{\boldsymbol{d}} D(\boldsymbol{p}), \tag{23}$$

ALGORITHM 4: Estimating the directional derivative $\partial_{\boldsymbol{d}}\boldsymbol{v}^{(M)}$

```
1  EstDerV($\boldsymbol{p}_K, \boldsymbol{p}_{K+1}, \boldsymbol{d}, M$)
2  begin
3      $D \leftarrow \|\boldsymbol{p}_K - \mathrm{cp}(\boldsymbol{p}_K)\|$;
4      dInv$A \leftarrow \frac{-2}{\pi D^3}\partial_{\boldsymbol{d}} D$ ;        // Eq. (23)
5      $\boldsymbol{v} \leftarrow$ EstV($\boldsymbol{p}_K, \boldsymbol{p}_{K+1}, M$) ;        // Algorithm 2
6      $I \leftarrow |B_{\boldsymbol{p}}|\boldsymbol{v}$;
7      inv$A \leftarrow 1/(\pi D^2)$;
8      d$I \leftarrow$ EstDerI($\boldsymbol{p}_K, \boldsymbol{p}_{K+1}, \boldsymbol{d}, M$) ;        // Line 11
9      return dInv$A \cdot I$ + inv$A \cdot$ d$I$ ;        // Eq. (22)
10 end
11 EstDerI ($\boldsymbol{p}_K, \boldsymbol{p}_{K+1}, \boldsymbol{d}$, M)
12 begin
13     $\boldsymbol{x} \leftarrow \mathrm{cp}(\boldsymbol{p}_K)$;
14     $D \leftarrow \|\boldsymbol{p}_K - \boldsymbol{x}\|$;
15     $\boldsymbol{z}$, pdf $\leftarrow$ SampleOnCircle($\boldsymbol{p}_K, D$);
16     $\boldsymbol{v} \leftarrow$ EstV($\boldsymbol{z}, \boldsymbol{p}_{K+1}, M-1$);
17     $\boldsymbol{n}^\partial \leftarrow (\boldsymbol{z} - \boldsymbol{p}_K)/\|\boldsymbol{z} - \boldsymbol{p}_K\|$;
       /* Compute boundary motion vector */
18     $\boldsymbol{g}_D \leftarrow (\boldsymbol{p}_K - \boldsymbol{x})/D$ ;        // Eq. (25)
19     $\boldsymbol{V}^\partial \leftarrow \boldsymbol{d} + (\boldsymbol{d} \cdot \boldsymbol{g}_D)\boldsymbol{n}^\partial$ ;        // Eq. (26)
20     return $\boldsymbol{v}(\boldsymbol{V}^\partial \cdot \boldsymbol{n}^\partial)$/pdf;
21 end
```

where $\partial_{\boldsymbol{d}} D(\boldsymbol{p})$ is the directional derivative of the distance field D and can be computed via automatic differentiation (Lines 3–4). The integral $I^{(M-1)}$ equals $|B_{\boldsymbol{p}}|\,\boldsymbol{v}^{(M)}$ by the definition in Eq. (20) (Lines 5–6).

The second term involves the integral's directional derivative $\partial_{\boldsymbol{d}} I^{(M-1)}$. Since the integration domain $B_{\boldsymbol{p}}$ changes along with the direction $\boldsymbol{d}$, we can use the Reynolds transport theorem to rewrite this derivative as a boundary integral:

$$\partial_{\boldsymbol{d}} I^{(M-1)}(\boldsymbol{p}) = \int_{\partial B_{\boldsymbol{p}}} \boldsymbol{v}^{(M-1)}(\boldsymbol{z})(\boldsymbol{V}^\partial_{\boldsymbol{d}}(\boldsymbol{z}) \cdot \boldsymbol{n}^\partial(\boldsymbol{z}))\,\mathrm{d}\boldsymbol{z}\,. \tag{24}$$

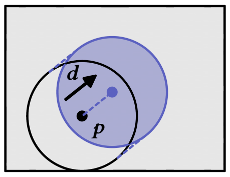
In this equation, the integration domain becomes a circle $\partial B_{\boldsymbol{p}}$ (boundary of the 2D disk $B_{\boldsymbol{p}}$). To estimate this integral, we first sample a point $\boldsymbol{z}$ on the circle (Line 15), compute $\boldsymbol{v}^{(M-1)}$ at $\boldsymbol{z}$ recursively (Line 16), and compute the dot product of the (outward) boundary normal $\boldsymbol{n}(\boldsymbol{z}) = (\boldsymbol{z} - \boldsymbol{p})/\|\boldsymbol{z} - \boldsymbol{p}\|$ (Line 17) and the boundary motion vector $\boldsymbol{V}^\partial_{\boldsymbol{d}}(\boldsymbol{z})$ (Lines 18–19).

As illustrated in Fig. 5, the boundary motion vector $\boldsymbol{V}^\partial_{\boldsymbol{d}}(\boldsymbol{z})$ captures the motion of the integration domain, which is a circle in this case, as we perturb its center $\boldsymbol{p}$ along the direction $\boldsymbol{d}$. Assuming the circle's center $\boldsymbol{p}$ moves along $\boldsymbol{d}$ by a unit magnitude, the motion at a point $\boldsymbol{z}$ on the circle is the composition of the following two components:

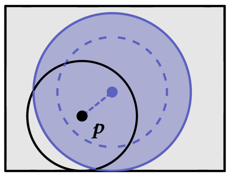
(1) translation in the direction $\boldsymbol{d}$ by the same magnitude as the circle's center (Fig. 5 (a)), and

(2) translation in the direction $\boldsymbol{n}^\partial(\boldsymbol{z})$ due to the change of the circle's radius (Fig. 5 (b)), which is the distance field function $D(\boldsymbol{p})$ defined in Eq. (14).

The first component is trivial to compute. For the second component, the motion direction aligns with $\boldsymbol{n}^\partial(\boldsymbol{z})$, and the translation



(a) Step 1: Translation along d



(b) Step 2: Scaling the circle

Fig. 5. The computation of the boundary motion vector V_d^∂ consists of two steps. Suppose the circle's center $\boldsymbol{p}$ translates along the direction $\boldsymbol{d}$ by a small magnitude. (a) We first translate the entire circle along with its center by the same direction and magnitude. (b) We then scale the circle so that it is still tangential to the boundary.

magnitude $(\boldsymbol{d} \cdot \boldsymbol{g}_D)$ is the change rate of the distance field D with respect to $\boldsymbol{d}$, where

$$\boldsymbol{g}_D = \frac{\boldsymbol{p} - \mathrm{cp}(\boldsymbol{p})}{\|\boldsymbol{p} - \mathrm{cp}(\boldsymbol{p})\|} \tag{25}$$

denotes the gradient vector of the distance field at $\boldsymbol{p}$. To sum up, the boundary motion vector is computed as

$$V_d^\partial(z) = \boldsymbol{d} + (\boldsymbol{d} \cdot \boldsymbol{g}_D)\boldsymbol{n}^\partial(z)\,. \tag{26}$$

Putting these equations together, the final result of the directional derivative is

$$\begin{aligned}\partial_d v^{(M)}(\boldsymbol{p}) &= -\frac{2}{D}\partial_d D(\boldsymbol{p}) \cdot v^{(M)}(\boldsymbol{p}) \\ &+ \frac{1}{\pi D^2}\int_{\partial B_p} v^{(M-1)}(z)\left(\boldsymbol{d} \cdot (\boldsymbol{g}_D + \boldsymbol{n}^\partial(z))\right) \mathrm{d}z\,.\end{aligned} \tag{27}$$

In the top row of Fig. 8, we verify the correctness of our fixed-step WoS derivative estimator by comparing its results with references generated by finite differences.

6 Practical Differential Visibility in 3D Using Walk-on-Spherical-Caps And Closest Silhouettes

We have constructed a valid velocity field $v^{(M)}$ using fixed-step WoS. Unfortunately, this method is not immediately practical for computing differential visibility, because the closest point queries $\mathrm{cp}(\cdot)$ in the WoS process is expensive to compute.

In this section, we present a practical algorithm to address this challenge. We first introduce an algorithm to find the *closest silhouettes* given a path segment, which takes sub-linear time and effectively finds the closest points under the geodesic (angular) metric on a unit sphere (§6.1). With this efficient closest silhouette query, we can construct a valid velocity field $u^{(M)}$ on the spherical surface $\mathbb{S}^2$ using fixed-step *walk-on-spherical-caps* estimators (§6.2); in this case, random walks live on the spherical surface and jump between spherical caps, which are the intersections between cones and the unit sphere, instead of 2D circles (see Fig. 6). Lastly, our final warped-area reparameterization algorithm uses fixed-step WoSC estimators to compute $u^{(M)}$ and its directional derivatives $\partial_d u^{(M)}$ on the spherical surface.

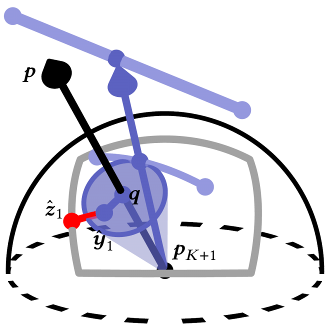


Fig. 6. Illustration of fixed-step walk-on-spherical-caps (WoSC) with $M = 1$. A random walk starts at bq, jumps to $\hat{\boldsymbol{y}}_1$ inside a spherical cap (the blue region), which is the intersection between a cone and the unit sphere, and ends by grabbing the boundary value at $\hat{z}_1$.

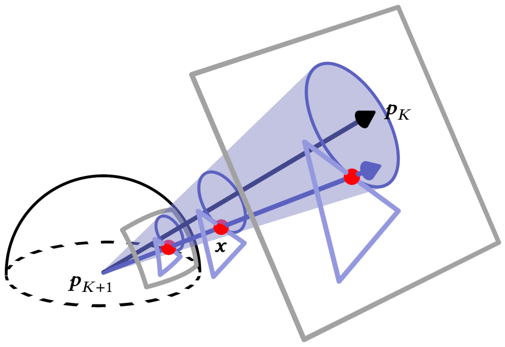


Fig. 7. We define a spherical projection from the tangent plane at $\boldsymbol{p}_K$ to the unit sphere centered at the shading point $\boldsymbol{p}_{K+1}$ (here only shows a hemisphere). Our new geodesic closest point function $\mathrm{cp}_{\mathbb{S}^2}$ find the closest silhouette point $\boldsymbol{x}$ on the triangle occluder, which effectively minimizes the geodesic distance between its spherical projection and the boundary arcs on the spherical surface.

6.1 Finding Closest Silhouette Points

Our fixed-step WoS estimators will become practical as long as we can efficiently find the closest points on the boundaries. In the following, we present an efficient algorithm to find the closest silhouette points on edges, which effectively finds the closest points minimizing the geodesic (angular) distance on a unit sphere. Having the ability of finding closest points on the spherical surface, we propose a practical algorithm called walk-on-spherical-caps in the next subsection, which is a variant of WoS running on the spherical surface rather than flat surfaces.

Spherical projection. As illustrated in Fig. 7, given a path segment $\overline{\boldsymbol{p}_K\boldsymbol{p}_{K+1}}$, we project the endpoint $\boldsymbol{p}_K$ and the boundary curves in $\partial\mathcal{B}^{\mathrm{wa}}$ (e.g., the visibility boundaries with red color in Fig. 7) to a unit sphere centered at the shading point $\boldsymbol{o} := \boldsymbol{p}_{K+1}$. After this spherical projection, a point $\boldsymbol{p}$ on the surface becomes a point $\boldsymbol{q} = (\boldsymbol{p}-\boldsymbol{o})/\|\boldsymbol{p}-\boldsymbol{o}\|$ on the unit sphere, and the boundary curves become a set of spherical arcs $\partial\mathcal{B}^{\mathrm{wa}}_{\mathbb{S}^2}(\boldsymbol{o})$ on $\mathbb{S}^2$ (the blue spherical triangle and the gray spherical rectangle in Fig. 7). The boundary velocities on the projected spherical arcs are inherited from $\boldsymbol{v}^\partial$ defined on surfaces:

$$\boldsymbol{u}^\partial((\boldsymbol{p}-\boldsymbol{o})/\|\boldsymbol{p}-\boldsymbol{o}\|) := \boldsymbol{v}^\partial(\boldsymbol{p}), \text{ for all } \boldsymbol{p} \in \partial\mathcal{B}^{\text{wa}}. \tag{28}$$

In this case, finding the closest point between $\boldsymbol{q}$ and the projected spherical arcs $\partial\mathcal{B}^{\text{wa}}_{\mathbb{S}^2}(\boldsymbol{o})$ is actually minimizing the geodesic (angular) distance on the unit sphere, instead of the Euclidean distance on a flat surface. Specifically, the closest point function on the spherical surface is defined as

$$\text{cp}_{\mathbb{S}^2}(\boldsymbol{q};\boldsymbol{o}) = \underset{\boldsymbol{q}'\in\partial\mathcal{B}^{\text{wa}}_{\mathbb{S}^2}(\boldsymbol{o})}{\text{argmax}} \;(\boldsymbol{q}\cdot\boldsymbol{q}'). \tag{29}$$

Note that minimizing the geodesic distance is equivalent to maximizing the dot product between two unit vectors on the sphere.

Because all the spherical arcs are projected from the edges of object surfaces in the scene (e.g., the blue triangle occluder in Fig. 7), the geodesic closest point function can be rewritten as

$$\text{cp}_{\mathbb{S}^2}(\boldsymbol{q};\boldsymbol{o}) = \underset{\boldsymbol{x}\in E}{\text{argmax}}\,(\boldsymbol{q}\cdot\frac{\boldsymbol{x}-\boldsymbol{o}}{\|\boldsymbol{x}-\boldsymbol{o}\|}), \tag{30}$$

where E is the set of surface edges that are visible to $\boldsymbol{o}$.

Closest silhouette point queries. The geodesic closest point function defined in Eq. (30) has a geometric interpretation. Given a cone that originates at $\boldsymbol{o}$ and has a central direction $\boldsymbol{q}$, the function $\text{cp}_{\mathbb{S}^2}(\boldsymbol{q};\boldsymbol{o})$ finds its minimum half-angle such that the cone intersects (in fact it is tangential to) scene objects. Inspired by this, we propose an efficient algorithm accelerated by a spatial hierarchy to compute $\text{cp}_{\mathbb{S}^2}(\boldsymbol{q};\boldsymbol{o})$, which is described in Algorithm 5.

We initialize a cone with the specified origin, direction, and half-angle as input, and start traversing the hierarchy from the root node. At each node, we check whether the cone intersects the bounding box of its child nodes (Line 13) and continue the traversal if so (Line 14). Upon reaching a leaf node, we compute the geodesic closest point function from all the triangle edges in the leaf node based on Eq. (30) (Line 5), and then update the result accordingly (Lines 6–8). Thanks to the accelerating spatial hierarchy, this algorithm takes sub-linear time in terms of the number of triangles, making it practical for complex 3D scenes.

Accelerating spatial hierarchy. In theory, our method does not require any additional accelerating spatial hierarchy since we can reuse the existing BVH for ray tracing. In practice, because this BVH cannot be accessed beyond hardware accelerated (RTX) ray intersection, we cannot easily repurpose it for our closest silhouette cone queries. Instead, we use a binary BVH constructed by the `fcpw` library [Sawhney 2021], which is a software implementation in Slang, still running on the GPU, but without any hardware acceleration. Our BVH uses a CPU-parallel construction that is relatively unoptimized compared to the most recent GPU-based BVH construction algorithms. For example, BVH construction for the scene in Fig. 1 takes about 100 milliseconds, and computing one million cone queries via BVH traversal on the GPU takes about 20 milliseconds. Developing an optimized BVH construction and traversal algorithm for our cone queries is an interesting topic for future work.

ALGORITHM 5: Computing the geodesic closest point on $\mathbb{S}^2$.

```
1  CPOnSphere($q$, $o$, node, minHalfAngle)
2  begin
3      cp ← ∅ ;                                        // Initialization
4      if node is a leaf node then
5          Compute the geodesic closest point $x$ and its corresponding
            half-angle $a$ from all edges inside the node ;  // Eq. (30)
           /* Update cp with a smaller half-angle                      */
6          if $a$ < minHalfAngle then
7              cp ← $x$; minHalfAngle ← $a$;
8          end
9      end
10     else
11         for child ∈ {node.left, node.right} do
12             bbox ← bounding box of child;
13             if bbox intersects the cone with minHalfAngle then
                   /* Traverse the subtree                             */
14                 $x$, $a$ ← CPOnSphere($q$, $o$, child, minHalfAngle);
                   /* Update cp with a smaller half-angle              */
15                 if $a$ < minHalfAngle then
16                     cp ← $x$; minHalfAngle ← $a$;
17                 end
18             end
19         end
20     end
21     return cp, minHalfAngle;
22 end
```

To speed up the BVH traversal, we use the following heuristics. First, we set the initial half-angle of the cone to be the angle subtended by the hit triangle that $\boldsymbol{p}_K$ lies on. Tuning the size of the initial half-angle offers a trade-off between the speed of the traversal and the smoothness of the constructed velocity field, and finding the optimal value is left as future work. Second, we use a back-face culling heuristic that terminates the traversal if the bounding box of the current node is completely behind the tangent plane at $\boldsymbol{p}_K$.

6.2 Fixed-Step Walk-on-Spherical-Caps: From Flat to Spherical Surfaces

Equipped with the efficient closest silhouette queries, we can now construct a valid velocity field $\boldsymbol{u}^{(M)}$ on the spherical surface, enabling a practical warped-area reparameterization algorithm for 3D differential visibility. In a high level, our final warped-area reparameterization algorithm consists of two steps.

(1) We construct $\boldsymbol{u}^{(M)}$ on the spherical surface with the fixed-step walk-on-spherical-caps algorithm. Our fixed-step WoSC estimator is very similar to the WoS variant described in Algorithm 2, except that we use the geodesic closest point function $\text{cp}_{\mathbb{S}^2}$ rather than the Euclidean closest point function cp.

(2) A valid velocity field $\boldsymbol{u}^{(M)}$ on the spherical surface induces a valid $\boldsymbol{v}$ on the flat surface (i.e., $\mathcal{B}^{\text{wa}}$ that $\boldsymbol{p}_K$ lies on). We can get $\boldsymbol{v}$ and $\partial_{\boldsymbol{d}}\boldsymbol{v}$ for warped-area reparameterization by computing $\boldsymbol{u}^{(M)}$ and $\partial_{\boldsymbol{d}}\boldsymbol{u}^{(M)}$ on the spherical surface.

We discuss each of them as follows.

ALGORITHM 6: Estimating the velocity $\boldsymbol{u}^{(M)}$ on $\mathbb{S}^2$

```
1  EstU($q, o, M$)
2  begin
       /* Find the geodesic closest point on spherical
          boundaries using Algorithm 5                    */
3      $z$, minHalfAngle ← CPOnSphere($q, o$, root, initialHalfAngle);
       /* Terminate if reaching the maximum step          */
4      if $M = 0$ then
5          $u \leftarrow u^\partial(z)$;
6      end
7      else
8          $y$, pdf ← SampleInsideSphericalCap($q, o$, minHalfAngle);
9          area ← $2\pi(1 - \cos(\text{minHalfAngle}))$ ;        // Eq. (33)
10         $u$ ← EstU($y, o, M-1$)/(pdf · area);
11     end
12     return $u$;
13 end
```

Constructing $\boldsymbol{u}^{(M)}$ using fixed-step WoSC. The formulation of $\boldsymbol{u}^{(M)}$ on the spherical surface is very similar to its variant $\boldsymbol{v}^{(M)}$ on the flat surface (Eqs. (18, 20)), except that we use $\text{cp}_{\mathbb{S}^2}$ rather than cp and the integration domain is a spherical cap $C_{\boldsymbol{q}}$ rather than a disk $B_{\boldsymbol{p}}$.

Suppose $\boldsymbol{o} := \boldsymbol{p}_{K+1}$ is the fixed shading point and the center of the unit sphere. Starting from $M = 0$, we have

$$\boldsymbol{u}^{(0)}(\boldsymbol{q}) = \boldsymbol{u}^\partial(\text{cp}_{\mathbb{S}^2}(\boldsymbol{q};\boldsymbol{o}))\,. \tag{31}$$

For $M > 0$, $\boldsymbol{u}^{(M)}$ is defined recursively by averaging $\boldsymbol{u}^{(M-1)}$ over the maximum inner spherical cap $C_{\boldsymbol{q}}$ (the blue region in Fig. 6):

$$\boldsymbol{u}^{(M)}(\boldsymbol{q}) = \frac{1}{|C_{\boldsymbol{q}}|}\int_{C_{\boldsymbol{q}}} \boldsymbol{u}^{(M-1)}(\boldsymbol{y})\,\mathrm{d}\boldsymbol{y}\,, \tag{32}$$

where $C_{\boldsymbol{q}}$ is the maximum inner spherical cap with the central direction $\boldsymbol{q}$ and the half-angle $A(\boldsymbol{q})$ (i.e., the geodesic closest distance between $\boldsymbol{q}$ and the spherical boundaries), and $|C_{\boldsymbol{q}}|$ denotes its area:

$$|C_{\boldsymbol{q}}| = 2\pi(1 - \cos A(\boldsymbol{q}))\,. \tag{33}$$

Because $\boldsymbol{v}^{(M)}$ and $\boldsymbol{u}^{(M)}$ have almost identical formulations, $\boldsymbol{u}^{(M)}$ inherits the properties of $\boldsymbol{v}^{(M)}$: it satisfies the continuity and boundary consistency requirements on the spherical surface. We illustrate the fixed-step WoSC algorithm in Fig. 6 and outline it in Algorithm 6. The major differences compared with its WoS variant in Algorithm 2 are highlighted in orange color (Lines 3 and 8).

Computing directional derivatives $\partial_{\boldsymbol{d}}\boldsymbol{u}^{(M)}$. We present the final result of the directional derivative $\partial_{\boldsymbol{d}}\boldsymbol{u}^{(M)}(\boldsymbol{q})$ with respect to a direction $\boldsymbol{d}$ on the tangent plane at $\boldsymbol{q}$:

$$\begin{aligned}\partial_{\boldsymbol{d}}\boldsymbol{u}^{(M)}(\boldsymbol{q}) = &-\frac{\sin A}{1-\cos A}\partial_{\boldsymbol{d}}A(\boldsymbol{q})\cdot\boldsymbol{u}^{(M)}(\boldsymbol{q})\\ &+\frac{1}{|C_{\boldsymbol{q}}|}\int_{\partial C_{\boldsymbol{q}}} \boldsymbol{u}^{(M-1)}(\boldsymbol{z})\left(\hat{\boldsymbol{V}}^\partial_{\boldsymbol{d}}(\boldsymbol{z})\cdot\hat{\boldsymbol{n}}^\partial(\boldsymbol{z})\right)\mathrm{d}\boldsymbol{z}\,,\end{aligned} \tag{34}$$

where $A(\cdot)$ is the geodesic (angular) distance field on the unit sphere and $\hat{\boldsymbol{n}}^\partial(\boldsymbol{z})$ is the projected (outward) normal vector of the spherical cap boundary onto the tangent plane at $\boldsymbol{z}$. The projected boundary motion vector $\hat{\boldsymbol{V}}^\partial_{\boldsymbol{d}}(\boldsymbol{z})$ onto the tangent plane at $\boldsymbol{z}$ is written as

$$\begin{aligned}\hat{\boldsymbol{V}}^\partial_{\boldsymbol{d}}(\boldsymbol{z}) &= \boldsymbol{V}^\partial_{\boldsymbol{d}}(\boldsymbol{z}) - \left(\boldsymbol{V}^\partial_{\boldsymbol{d}}(\boldsymbol{z})\cdot\boldsymbol{z}\right)\boldsymbol{z}\,,\text{ and}\\ \boldsymbol{V}^\partial_{\boldsymbol{d}}(\boldsymbol{z}) &= \left(\frac{\boldsymbol{q}\times\boldsymbol{d}}{\|\boldsymbol{q}\times\boldsymbol{d}\|}\times\boldsymbol{z}\right) + \left(\boldsymbol{d}\cdot\boldsymbol{g}_A\right)\hat{\boldsymbol{n}}^\partial(\boldsymbol{z})\,,\end{aligned} \tag{35}$$

where $\times$ represents cross product and $\boldsymbol{g}_A$ is the gradient of the geodesic distance field $A(\cdot)$ on the unit sphere.

Similar to the derivation in §4, the boundary motion vector is decomposed into two components. The first component is slightly different; the translation on flat surfaces becomes a rotation on the spherical surface, and the cross product term denotes the angular velocity of this rotation. The second component is similar, capturing the change of a spherical cap's size.

In the bottom row of Fig. 8, we validate the spherical version of our directional derivative estimator in Eq. (34).

Reparameterization with $\boldsymbol{u}^{(M)}$ and $\partial_{\boldsymbol{d}}\boldsymbol{u}^{(M)}$. We now present our final warped-area reparameterization algorithm using fixed-step WoSC estimators. It is outlined in Algorithm 7, and we highlight the major differences compared with its WoS variant (Algorithm 3) in orange color (Lines 4, 8, and 9).

Our constructed field $\boldsymbol{u}^{(M)}$ on the spherical surface induces a valid velocity field $\boldsymbol{v}$ on the flat surface:

$$\boldsymbol{v}(\boldsymbol{p}) := \boldsymbol{u}^{(M)}(\boldsymbol{q})\,, \tag{36}$$

where $\boldsymbol{q} = (\boldsymbol{p} - \boldsymbol{p}_{K+1})/\|\boldsymbol{p} - \boldsymbol{p}_{K+1}\|$ is the spherical projection of $\boldsymbol{p}$. So the velocity $\boldsymbol{v}$ can be estimated using our fixed-step WoSC estimator (Line 4).

Furthermore, the directional derivative $\partial_{\boldsymbol{d}}\boldsymbol{v}(\boldsymbol{p})$ with respect to a direction $\boldsymbol{d}$ on the surface equals the directional derivative $\partial_{\hat{\boldsymbol{d}}}\boldsymbol{u}^{(M)}(\boldsymbol{q})$ with respect to the projection of $\boldsymbol{d}$ on the tangent plane at $\boldsymbol{q}$:

$$\hat{\boldsymbol{d}} = \frac{\boldsymbol{d}}{\|\boldsymbol{p} - \boldsymbol{p}_{K+1}\|} - \left(\frac{\boldsymbol{d}}{\|\boldsymbol{p} - \boldsymbol{p}_{K+1}\|}\cdot\boldsymbol{q}\right)\boldsymbol{q}\,. \tag{37}$$

We show the proof of

$$\partial_{\boldsymbol{d}}\boldsymbol{v}(\boldsymbol{p}) = \partial_{\hat{\boldsymbol{d}}}\boldsymbol{u}^{(M)}(\boldsymbol{q}) \tag{38}$$

in Appendix B. As a result, we can estimate the divergence $\nabla\cdot\boldsymbol{v}$ using our fixed-step WoSC estimator (Lines 8 and 9). This concludes our final warped-area reparameterization algorithm.

In Fig. 9, we validate our final warped-area reparameterization algorithm by comparing the differentiable rendering results with reference images computed by finite differences.

7 Results

We implement our method on the GPU using the Falcor rendering system [Kallweit et al. 2022] and the Slang shading language [Bangaru et al. 2023]. All the experiments are run on a single NVIDIA RTX 4090 GPU.

7.1 Validation

Directional derivative estimators. We first validate our directional derivative estimators in Eqs. (27) and (34) as a unit test. In the top row of Fig. 8, we define a 2D boundary value problem on a flat plane and provide the boundary shape (the blue polygon) and boundary values as input. We then use 1-step WoS to compute $\boldsymbol{v}^{(1)}$ and its directional derivative $\partial_{\boldsymbol{d}}\boldsymbol{v}^{(1)}$ with respect to a pre-determined vector

ALGORITHM 7: Computing the velocity $\boldsymbol{v}$ and its divergence $\nabla \cdot \boldsymbol{v}$.

```
ComputeV($\boldsymbol{p}_K, \boldsymbol{p}_{K+1}$)
begin
    $\boldsymbol{q} \leftarrow (\boldsymbol{p}_K - \boldsymbol{p}_{K+1})/\|\boldsymbol{p}_K - \boldsymbol{p}_{K+1}\|$;
    /* Estimate the velocity $\boldsymbol{v}$ */
    $\boldsymbol{v} \leftarrow$ EstU($\boldsymbol{q}, \boldsymbol{p}_{K+1}, M$) ;          // Algorithm 6
    /* Estimate the divergence $\nabla \cdot \boldsymbol{v}$ */
    $\nabla \cdot \boldsymbol{v} \leftarrow 0$;
    Specify a pair of orthogonal unit vectors $\boldsymbol{s}$ and $\boldsymbol{t}$ on the surface
      where $\boldsymbol{p}_K$ resides;
    for $\boldsymbol{d} \in \{\boldsymbol{s}, \boldsymbol{t}\}$ do
        $\hat{\boldsymbol{d}} \leftarrow$ Project($\boldsymbol{p}_K, \boldsymbol{p}_{K+1}, \boldsymbol{d}$) ;          // Line 14
        $\partial_{\hat{\boldsymbol{d}}}\boldsymbol{v} \leftarrow$ EstDerU($\boldsymbol{q}, \boldsymbol{p}_{K+1}, \hat{\boldsymbol{d}}, M$) ;          // Eq. (34)
        $\nabla \cdot \boldsymbol{v} \leftarrow \nabla \cdot \boldsymbol{v} + (\partial_{\hat{\boldsymbol{d}}}\boldsymbol{v} \cdot \boldsymbol{d})$;
    end
    return $\boldsymbol{v}, \nabla \cdot \boldsymbol{v}$;
end
Project($\boldsymbol{p}, \boldsymbol{o}, \boldsymbol{d}$)
begin
    $\ell \leftarrow \|\boldsymbol{p} - \boldsymbol{o}\|$;
    $\boldsymbol{q} \leftarrow (\boldsymbol{p} - \boldsymbol{o})/\ell$;
    $\hat{\boldsymbol{d}} \leftarrow \boldsymbol{d}/\ell - ((\boldsymbol{d}/\ell) \cdot \boldsymbol{q})\boldsymbol{q}$ ;          // Eq. (37)
    return $\hat{\boldsymbol{d}}$;
end
```

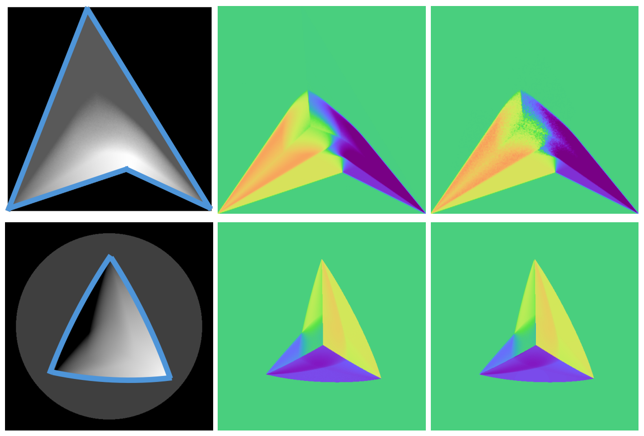

(a) $\boldsymbol{v}^{(1)}$
(b) $\partial_{\boldsymbol{d}}\boldsymbol{v}^{(1)}$
(c) FD ref.

Fig. 8. Validation of our directional derivative estimators on flat surfaces (top row) and spherical surfaces (bottom row; visualized by projecting the hemisphere onto a disk). (a) Given values at the boundaries (shown in blue segments) as input, we compute the interior values using our fixed-step estimators with $M = 1$. The estimated derivatives with respect to a direction $\boldsymbol{d}$ in (b) using Eqs. (27) and (34) match the references in (c) computed by finite differences.

$\boldsymbol{d}$ inside the polygon (shown in Fig. 8 (a) and (b) respectively). We compare the derivative estimates to the reference values computed by finite differences (FD) in Fig. 8 (c). In the bottom row, we repeat a similar validation on a unit hemisphere using 1-step walk-on-spherical-caps, where the boundary shape is a spherical triangle now.

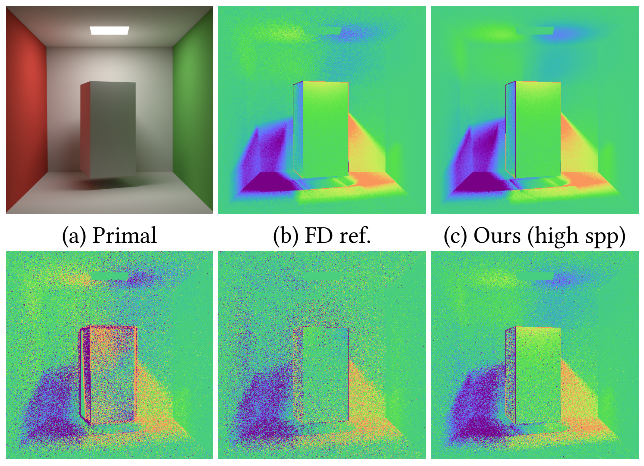


Fig. 9. In this example, we compute derivatives with respect to the x-translation of the center box. We first validate our warped-area reparameterization algorithm by comparing our result in (c) with the reference gradient image computed by finite differences in (b). In (d)–(f), we show equal-sample comparisons with the baseline methods, WAS [Bangaru et al. 2020] and PSDR-WAS [Xu et al. 2023].

Gradient images. With the directional derivatives validated, we now validate our final warped-area reparameterization algorithm by comparing the gradient images computed using our method to the reference images computed using finite differences.

The example in Fig. 9 (a)–(c) computes image derivatives with respect to the x-translation of the center box. The gradient image computed by our method (c) closely matches the FD reference (b).

7.2 Evaluation

Ablation: number of WoSC steps. The number of WoSC steps M is the most important hyperparameter in our method. Although using more steps lead to smoother velocity fields, it comes with a higher computational cost, mainly due to a higher required number of cone queries. The number of cone queries, and thus the running time of our algorithm, scales linearly in the step count M. In practice, we do not observe significant benefits from using more than one step.

In Fig. 10, we show an equal-time comparison using $M = 1$, 2, and 4, which uses 1000, 480, and 310 primary samples per pixel, respectively. Each primary sample simulates one auxiliary WoSC random walk. This example (also shown in Fig. 1) computes derivatives of the shadows cast by the Voronoi-bunny model [Mehta et al. 2022] with 168k triangles under an area light source, with respect to the y-translation of the bunny. Using different numbers of steps, the mean L1 errors of the gradient images compared with the FD reference image are 0.017, 0.028, and 0.041, respectively. When the cone queries become more efficient in the future, using more steps may be potentially beneficial.

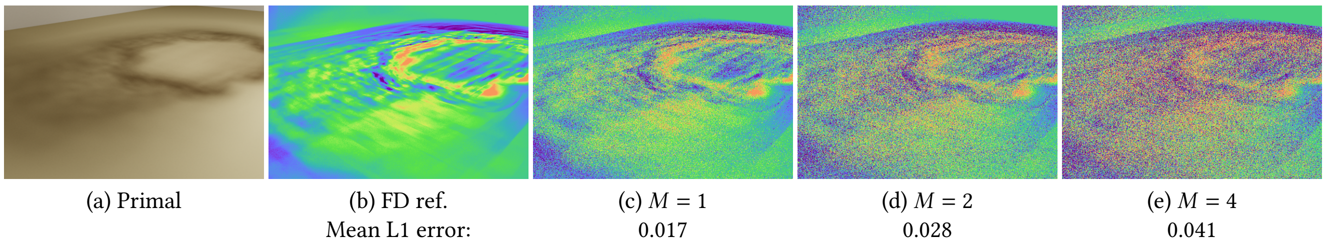


Fig. 10. An equal-time ablation study on the number of WoSC steps M. In practice, we do not observe significant benefits from using more than one step, due to the linearly increasing computational cost.

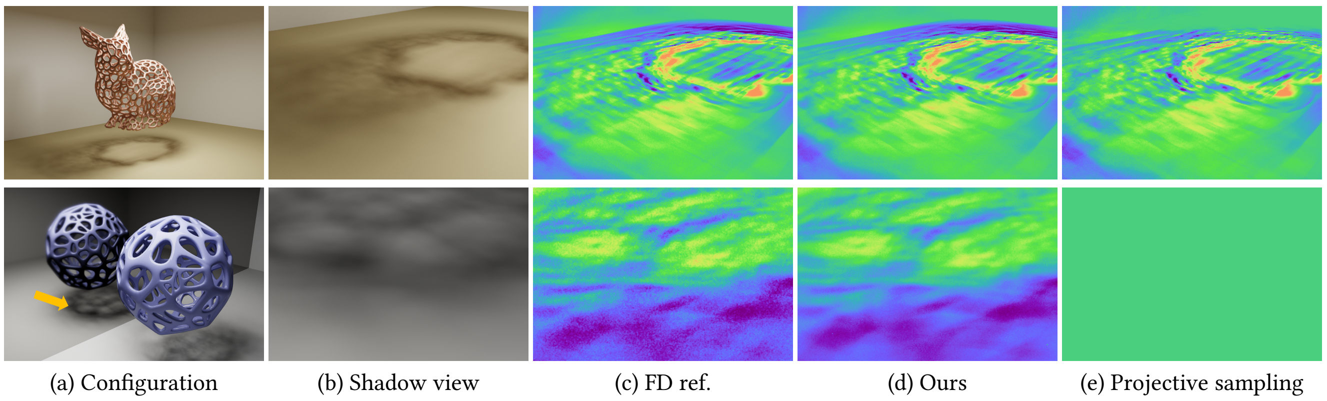


Fig. 11. Comparison with projective sampling [Zhang et al. 2023]. The example on the top row computes derivatives of the shadows cast by the Voronoi-bunny model, and the example on the bottom row computes derivatives of the shadows seen through a mirror. Our results closely match the FD reference images, showing the robustness of our method under complicated light transport configurations. On the other hand, the results of projective sampling deviate from the reference images.

Baselines. We compare our method to two state-of-the-art baselines in warped-area reparameterization: the warped-area sampling method by Bangaru et al. [2020] (denoted as WAS) and the path-space warped-area sampling method by Xu et al. [2023] (denoted as PSDR-WAS). For WAS, we use its public implementation in the Falcor renderer [Kallweit et al. 2022]. For PSDR-WAS, we use the code released by the authors, which uses the Enzyme automatic differentiation framework [Moses and Churavy 2020] and runs on the CPU.

Equal-sample gradient image comparison. Since our method and the baselines run in different environments, it is hard to do equal-time comparisons, so we perform equal-sample comparisons instead. To shed light on the practicality of our method, we compare the performance of our cone queries (Algorithm 5) using the following experiment. We implement the cone query in the Slang shading language, port it to the open-source `fcpw` library [Sawhney 2021], and compare its performance to the (shadow) ray intersection query in `fcpw`. Using a Stanford dragon model with 100k triangles, it takes roughly 30ms for our method to answer 1M cone queries, which is about 10× slower than performing 1M ray intersection queries.

We compare the gradient image renderings to baseline methods (WAS [Bangaru et al. 2020] and PSDR-WAS [Xu et al. 2023]). Both baselines use 8 auxiliary samples to compute the velocities in Eq. 7 for warped-area parameterization, and all methods use the same number of primary samples per pixel.

As shown in Fig. 1, our method outperforms the baselines in the equal-sample comparison because of the robust velocity construction strategy. From Fig. 1 (b)–(d), the mean L1 errors compared to the FD reference image are 0.032, 0.053, and 0.069, respectively. Using a high sample count (Fig. 1 (a)), our result converges closely to the reference.

In Fig. 9 (d)–(f), we show another equal-sample comparison. Our result has lower noise level compared to both baselines.

Comparison with explicit boundary sampling. In Fig. 11, we demonstrate two examples of our method compared with the state-of-the-art explicit boundary sampling technique, projective sampling [Zhang et al. 2023]. We use the authors' public implementation in Mitsuba 3 [Jakob et al. 2022b], and both methods use a high sample count to produce converged gradient images.

In the first example (top row of Fig. 11) of the Voronoi-bunny scene, the gradient image computed by our method closely matches the FD reference image, while the projective sampling result deviates from the reference and has obvious artifacts on the top-right region. The reason might be that the projection is not always well-behaved when building the guiding structure for the Voronoi-bunny model with many thin geometric structures, which is referred to as the difficult "blade-of-grass" case in their paper.

The second example (bottom row of Fig. 11) computes derivatives of the shadows cast by the Voronoi-sphere model [Mehta et al. 2022] with 72k triangles seen through a mirror, with respect to the x-translation of the Voronoi-sphere. Our result matches the FD reference image closely, showing the robustness of our method under complicated light transport configurations. On the other hand, projective sampling fails to capture the mirrored shadows since its guiding structure cannot be built due to the direct specular connection to the pinhole camera, which might be addressed by the specular manifold sampling techniques [Zeltner et al. 2020].

Inverse rendering comparison. In Fig. 12, we demonstrate an inverse rendering example optimizing the occluder's shape (i.e., mesh vertex positions) by only looking at its shadows. This example involves 50 unknown variables. Starting from a disk, we can converge to the target flower shape. On the other hand, the baseline method fails to converge to the optimal solution due to its inaccurate gradient estimates.

8 Conclusion

In this work, we introduced a fixed-step walk-on-spherical-caps algorithm for differentiable rendering, which constructs a velocity field that respects the constraints needed for unbiased warped-area reparameterization. Our method makes non-trivial modifications to the walk on spheres algorithm, namely, adapting it from Euclidean to spherical domains, providing principled derivative estimation for our fixed-step implementation, and developing a cone query that is scalable to complex 3D scenes. Our resulting velocity fields provide higher quality derivatives compared to prior warped area reparameterization approaches, both in terms of bias and variance, enabling more robust differentiable rendering. Future work involves further optimizing our cone query on the GPU, as well as exploration of recent manifold-based WoS approaches [Sugimoto et al. 2024] to our setting.

Acknowledgments

We thank the anonymous reviewers for their constructive suggestions. We are grateful to Steve Marschner, Wenzel Jakob, Jacob Munkberg, and Jon Hasselgren for their insightful discussions and feedback on the paper. We also thank Peiyu Xu for his help with the 3D scenes and the comparison to PSDR-WAS.

References

Sai Bangaru, Michael Gharbi, Tzu-Mao Li, Fujun Luan, Kalyan Sunkavalli, Milos Hasan, Sai Bi, Zexiang Xu, Gilbert Bernstein, and Fredo Durand. 2022. Differentiable Rendering of Neural SDFs through Reparameterization. In *ACM SIGGRAPH Asia 2022 Conference Proceedings (SIGGRAPH Asia '22).* Article 22, 9 pages.

Sai Bangaru, Lifan Wu, Tzu-Mao Li, Jacob Munkberg, Gilbert Bernstein, Jonathan Ragan-Kelley, Fredo Durand, Aaron Lefohn, and Yong He. 2023. SLANG.D: Fast, Modular and Differentiable Shader Programming. *ACM Trans. Graph.* 42, 6 (2023), 1–28.

Sai Praveen Bangaru, Tzu-Mao Li, and Frédo Durand. 2020. Unbiased Warped-Area Sampling for Differentiable Rendering. *ACM Trans. Graph.* 39, 6 (2020), 245:1–245:18.

Yash Belhe, Bing Xu, Sai Praveen Bangaru, Ravi Ramamoorthi, and Tzu-Mao Li. 2024. Importance Sampling BRDF Derivatives. *ACM Trans. Graph.* 43, 3, Article 25 (2024), 21 pages.

Ilia Binder and Mark Braverman. 2012. The rate of convergence of the walk on spheres algorithm. *Geometric and Functional Analysis* 22, 3 (2012), 558–587.

Wesley Chang, Venkataram Sivaram, Derek Nowrouzezahrai, Toshiya Hachisuka, Ravi Ramamoorthi, and Tzu-Mao Li. 2023. Parameter-space ReSTIR for Differentiable and Inverse Rendering. In *ACM SIGGRAPH 2023 Conference Proceedings (SIGGRAPH '23).* Association for Computing Machinery, New York, NY, USA, 18:1–18:10.

Gísli R. Hjaltason and Hanan Samet. 1999. Distance browsing in spatial databases. *ACM Trans. Database Syst.* 24, 2 (1999), 265–318.

J. Jakob and M. Guthe. 2021. Optimizing LBVH-Construction and Hierarchy-Traversal to accelerate kNN Queries on Point Clouds using the GPU. *Computer Graphics Forum* 40, 1 (2021), 124–137.

Wenzel Jakob, Sébastien Speierer, Nicolas Roussel, Merlin Nimier-David, Delio Vicini, Tizian Zeltner, Baptiste Nicolet, Miguel Crespo, Vincent Leroy, and Ziyi Zhang. 2022b. *Mitsuba 3 renderer.* https://mitsuba-renderer.org.

Wenzel Jakob, SÃ©bastien Speierer, Nicolas Roussel, and Delio Vicini. 2022a. Dr.Jit: A Just-In-Time Compiler for Differentiable Rendering. *ACM Trans. Graph.* 41, 4 (2022), 124:1–124:19.

David E. Johnson and Elaine Cohen. 2001. Spatialized normal come hierarchies. In *Proceedings of the 2001 Symposium on Interactive 3D Graphics (I3D '01).* 129–134.

Simon Kallweit, Petrik Clarberg, Craig Kolb, Tom'aš Davidovič, Kai-Hwa Yao, Theresa Foley, Yong He, Lifan Wu, Lucy Chen, Tomas Akenine-Möller, Chris Wyman, Cyril Crassin, and Nir Benty. 2022. The Falcor Rendering Framework. https://github.com/NVIDIAGameWorks/Falcor

Samuli Laine, Janne Hellsten, Tero Karras, Yeongho Seol, Jaakko Lehtinen, and Timo Aila. 2020. Modular Primitives for High-Performance Differentiable Rendering. *ACM Trans. Graph.* 39, 6 (2020).

Pedro Jose Silva Leite, Joao Marcelo Xavier Natario Teixeira, Thiago Souto Maior Cordeiro de Farias, Veronica Teichrieb, and Judith Kelner. 2009. Massively Parallel Nearest Neighbor Queries for Dynamic Point Clouds on the GPU. In *2009 21st International Symposium on Computer Architecture and High Performance Computing.* 19–25.

Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. 2018. Differentiable Monte Carlo ray tracing through edge sampling. *ACM Trans. Graph.* 37, 6 (2018), 222:1–222:11.

Zilu Li, Guandao Yang, Xi Deng, Christopher De Sa, Bharath Hariharan, and Steve Marschner. 2023. Neural Caches for Monte Carlo Partial Differential Equation Solvers. In *SIGGRAPH Asia 2023 Conference Papers.* 1–10.

Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. 2019. Soft Rasterizer: A Differentiable Renderer for Image-based 3D Reasoning. *The IEEE International Conference on Computer Vision (ICCV)* (Oct 2019).

Guillaume Loubet, Nicolas Holzschuch, and Wenzel Jakob. 2019. Reparameterizing discontinuous integrands for differentiable rendering. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 1–14.

Fujun Luan, Shuang Zhao, Kavita Bala, and Zhao Dong. 2021. Unified Shape and SVBRDF Recovery using Differentiable Monte Carlo Rendering. *Computer Graphics Forum* 40, 4 (2021), 101–113.

Don McLeish. 2011. A general method for debiasing a Monte Carlo estimator. *Monte Carlo methods and applications* 17, 4 (2011), 301–315.

Ishit Mehta, Manmohan Chandraker, and Ravi Ramamoorthi. 2022. A level set theory for neural implicit evolution under explicit flows. In *European Conference on Computer Vision.* 711–729.

Bailey Miller, Rohan Sawhney, Keenan Crane, and Ioannis Gkioulekas. 2023. Boundary Value Caching for Walk on Spheres. *ACM Trans. Graph.* 42, 4 (2023), 1–11.

Bailey Miller, Rohan Sawhney, Keenan Crane, and Ioannis Gkioulekas. 2024a. Differential Walk on Spheres. *ACM Trans. Graph.* 43, 6 (2024), 1–18.

Bailey Miller, Rohan Sawhney, Keenan Crane, and Ioannis Gkioulekas. 2024b. Walkin'robin: Walk on stars with robin boundary conditions. *ACM Trans. Graph.* 43, 4 (2024), 1–18.

William Moses and Valentin Churavy. 2020. Instead of rewriting foreign code for machine learning, automatically synthesize fast gradients. *Advances in neural information processing systems* 33 (2020), 12472–12485.

Mervin E Muller. 1956. Some continuous Monte Carlo methods for the Dirichlet problem. *The Annals of Mathematical Statistics* (1956), 569–589.

Mohammad Sina Nabizadeh, Ravi Ramamoorthi, and Albert Chern. 2021. Kelvin transformations for simulations on infinite domains. *ACM Trans. Graph.* 40, 4 (2021), 97:1–97:15.

Baptiste Nicolet, Fabrice Rousselle, Jan Novák, Alexander Keller, Wenzel Jakob, and Thomas Müller. 2023. Recursive Control Variates for Inverse Rendering. *ACM Trans. Graph.* 42, 4 (2023).

Merlin Nimier-David, Sébastien Speierer, Benoît Ruiz, and Wenzel Jakob. 2020. Radiative backpropagation: an adjoint method for lightning-fast differentiable rendering. *ACM Trans. Graph.* 39, 4 (2020), 146:1–146:15.

Timothy J. Purcell, Craig Donner, Mike Cammarano, Henrik Wann Jensen, and Pat Hanrahan. 2003. Photon Mapping on Programmable Graphics Hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware.* 41–50.

Yang Qi, Dario Seyb, Benedikt Bitterli, and Wojciech Jarosz. 2022. A bidirectional formulation for Walk on Spheres. *Computer Graphics Forum* 41, 4 (2022), 51–62.

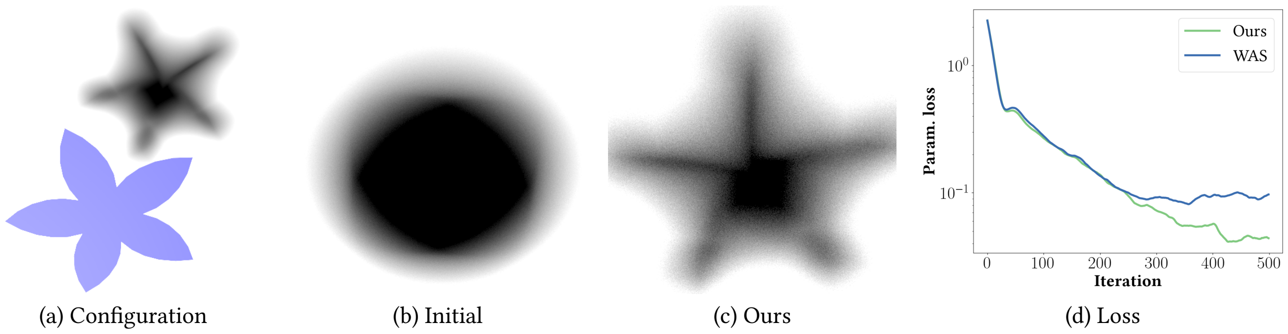


Fig. 12. We present an inverse rendering example that optimizes the occluder's shape by only looking at shadows. The target shape is shown in (a). Starting from an initial shape in (b), our method can converge to the desired shape in (c). We compare the convergence rates between our method and the baseline method [Bangaru et al. 2020] in (d).

Maxime Roger, Stéphane Blanco, Mouna El Hafi, and Richard Fournier. 2005. Monte Carlo estimates of domain-deformation sensitivities. *Physical review letters* 95, 18 (2005), 180601.

Hanan Samet. 2005. *Foundations of Multidimensional and Metric Data Structures (The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling)*. Morgan Kaufmann Publishers Inc.

Rohan Sawhney. 2021. *FCPW: Fastest Closest Points in the West.*

Rohan Sawhney and Keenan Crane. 2020. Monte Carlo geometry processing: a grid-free approach to PDE-based methods on volumetric domains. *ACM Trans. Graph.* 39, 4 (2020), 123:1–123:18.

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

Rohan Sawhney, Dario Seyb, Wojciech Jarosz, and Keenan Crane. 2022. Grid-free Monte Carlo for PDEs with spatially varying coefficients. *ACM Trans. Graph.* 41, 4 (July 2022).

Johannes Schauer, Janusz Bedkowski, Karol Majek, and Andreas Nüchter. 2016. Performance comparison between state-of-the-art point-cloud based collision detection approaches on the CPU and GPU. *IFAC* 49, 30 (2016), 54–59.

Ryusuke Sugimoto, Terry Chen, Yiti Jiang, Christopher Batty, and Toshiya Hachisuka. 2023. A Practical Walk-on-Boundary Method for Boundary Value Problems. *ACM Trans. Graph.* 42, 4, Article 81 (July 2023), 16 pages.

Ryusuke Sugimoto, Nathan King, Toshiya Hachisuka, and Christopher Batty. 2024. Projected Walk on Spheres: A Monte Carlo Closest Point Method for Surface PDEs. In *SIGGRAPH Asia 2024 Conference Papers*. 1–10.

Xiaochun Tong, Hsueh-Ti Derek Liu, Yotam Gingold, and Alec Jacobson. 2023. Differentiable Heightfield Path Tracing with Accelerated Discontinuities. In *SIGGRAPH 2023 Conference Proceedings (SIGGRAPH '23)*.

Eric Veach. 1997. *Robust Monte Carlo methods for light transport simulation.* Vol. 1610. Stanford University PhD thesis.

Delio Vicini, Sébastien Speierer, and Wenzel Jakob. 2021. Path Replay Backpropagation: Differentiating Light Paths Using Constant Memory and Linear Time. *ACM Trans. Graph.* 40, 4, Article 108 (2021), 108:1–108:14 pages.

Delio Vicini, SÃ©bastien Speierer, and Wenzel Jakob. 2022. Differentiable Signed Distance Function Rendering. *ACM Trans. Graph.* 41, 4 (2022), 125:1–125:18.

Yu-Chen Wang, Chris Wyman, Lifan Wu, and Shuang Zhao. 2023. Amortizing Samples in Physics-Based Inverse Rendering Using ReSTIR. *ACM Trans. Graph.* 42, 6 (2023), 214:1–214:17.

Zichen Wang, Xi Deng, Ziyi Zhang, Wenzel Jakob, and Steve Marschner. 2024. A Simple Approach to Differentiable Rendering of SDFs. In *ACM SIGGRAPH Asia 2024 Conference Proceedings.*

Peiyu Xu, Sai Bangaru, Tzu-Mao Li, and Shuang Zhao. 2023. Warped-Area Reparameterization of Differential Path Integrals. *ACM Trans. Graph.* 42, 6 (2023), 213:1–213:18.

Kai Yan, Christoph Lassner, Brian Budge, Zhao Dong, and Shuang Zhao. 2022. Efficient estimation of boundary integrals for path-space differentiable rendering. *ACM Trans. Graph.* 41, 4 (2022), 123:1–123:13.

Yuting Yang, Connelly Barnes, Andrew Adams, and Adam Finkelstein. 2022. A δ: autodiff for discontinuous programs-applied to shaders. *ACM Trans. Graph.* 41, 4 (2022), 1–24.

Lior Yariv, Jiatao Gu, Yoni Kasten, and Yaron Lipman. 2021. Volume rendering of neural implicit surfaces. In *Thirty-Fifth Conference on Neural Information Processing Systems.*

Ekrem Fatih Yilmazer, Delio Vicini, and Wenzel Jakob. 2024. Solving Inverse PDE Problems using Monte Carlo Estimators. *ACM Trans. Graph.* 43 (2024).

Z. Yu, L. Wu, Z. Zhou, and S. Zhao. 2024. A Differential Monte Carlo Solver For the Poisson Equation. In *ACM SIGGRAPH 2024 Conference Proceedings.*

Tizian Zeltner, Iliyan Georgiev, and Wenzel Jakob. 2020. Specular Manifold Sampling for Rendering High-Frequency Caustics and Glints. *ACM Trans. Graph.* 39, 4 (2020).

Tizian Zeltner, Sébastien Speierer, Iliyan Georgiev, and Wenzel Jakob. 2021. Monte Carlo estimators for differential light transport. *ACM Trans. Graph.* 40, 4 (2021), 78:1–78:16.

Cheng Zhang, Zhao Dong, Michael Doggett, and Shuang Zhao. 2021a. Antithetic sampling for Monte Carlo differentiable rendering. *ACM Trans. Graph.* 40, 4 (2021), 77:1–77:12.

Cheng Zhang, Bailey Miller, Kai Yan, Ioannis Gkioulekas, and Shuang Zhao. 2020. Path-space differentiable rendering. *ACM Trans. Graph.* 39, 4 (2020), 143:1–143:19.

Cheng Zhang, Lifan Wu, Changxi Zheng, Ioannis Gkioulekas, Ravi Ramamoorthi, and Shuang Zhao. 2019. A differential theory of radiative transfer. *ACM Trans. Graph.* 38, 6 (2019), 227:1–227:16.

Cheng Zhang, Zihan Yu, and Shuang Zhao. 2021b. Path-space differentiable rendering of participating media. *ACM Trans. Graph.* 40, 4 (2021), 76:1–76:15.

Kai Zhang, Fujun Luan, Zhengqi Li, and Noah Snavely. 2022. IRON: Inverse Rendering by Optimizing Neural SDFs and Materials from Photometric Images. In *IEEE Conf. Comput. Vis. Pattern Recog.*

Ziyi Zhang, Nicolas Roussel, and Wenzel Jakob. 2023. Projective Sampling for Differentiable Rendering of Geometry. *ACM Trans. Graph.* 42, 6 (2023), 212:1–212:14.

A Proof of $\boldsymbol{v}^{(1)} \in \boldsymbol{C}^0$

We apply a change of variable

$$\boldsymbol{y}(\boldsymbol{x};\boldsymbol{p}) = \boldsymbol{p} + D(\boldsymbol{p})\boldsymbol{x} \tag{39}$$

to map from $\boldsymbol{x}$ in a unit disk $\hat{B}$ to $\boldsymbol{y} \in B_{\boldsymbol{p}}$ bijectively and in an area-preserving manner. The Jacobian determinant of this transformation

$$J_{\boldsymbol{y}}(\boldsymbol{p}) = [D(\boldsymbol{p})]^2 \tag{40}$$

is the ratio between the areas of $B_{\boldsymbol{p}}$ and $\hat{B}$. With that, we can rewrite Eq. (19) as an integral with a fixed integration domain:

$$\boldsymbol{v}^{(1)}(\boldsymbol{p}) = \frac{1}{|B_{\boldsymbol{p}}|}\int_{\hat{B}} \boldsymbol{v}^{(0)}(\boldsymbol{y}(\boldsymbol{x};\boldsymbol{p}))J_{\boldsymbol{y}}(\boldsymbol{p})\,\mathrm{d}\boldsymbol{x}. \tag{41}$$

To demonstrate the continuity, we need to check whether $\boldsymbol{v}^{(1)}(\boldsymbol{p}_0)$ equals $\lim_{\boldsymbol{p}\to\boldsymbol{p}_0} \boldsymbol{v}^{(1)}(\boldsymbol{p})$. We observe that all the terms ($|B_{\boldsymbol{p}}|$, $\boldsymbol{y}$, and $J_{\boldsymbol{y}}$) are continuous except for $\boldsymbol{v}^{(0)}$. Since $\boldsymbol{v}^{(0)}$ is piecewise continuous, $\boldsymbol{v}^{(0)}(\boldsymbol{y}_0)$ equals $\lim_{\boldsymbol{y}\to\boldsymbol{y}_0} \boldsymbol{v}^{(0)}(\boldsymbol{y})$ almost everywhere, except for points at the intersection of $B_{\boldsymbol{p}}$ and the medial axis of boundaries (denoted as $\mathrm{med}(\partial\mathcal{B}^{\mathrm{wa}})$), i.e., $\boldsymbol{y}_0 \in \mathcal{S} := \mathrm{med}(\partial\mathcal{B}^{\mathrm{wa}}) \cap B_{\boldsymbol{p}}$. Fortunately, $\mathcal{S}$ has measure zero and thus does not affect the result of the integral. Therefore, $\boldsymbol{v}^{(1)}$ is a continuous function of class $\boldsymbol{C}^0$, and it is differentiable almost everywhere except on the medial axis (we present its directional derivatives in Sec. 5.2).

B Proof of directional derivatives

For a small $\varepsilon > 0$, the directional derivative $\partial_{\boldsymbol{d}} \boldsymbol{v}$ is written as

$$\partial_{\boldsymbol{d}} \boldsymbol{v}(\boldsymbol{p}) = \lim_{\varepsilon \to 0} \frac{\boldsymbol{v}(\boldsymbol{p} + \varepsilon \boldsymbol{d}) - \boldsymbol{v}(\boldsymbol{p})}{\varepsilon}. \tag{42}$$

Since $\boldsymbol{q}$ is the projection of $\boldsymbol{p}$, we know that $\boldsymbol{v}(\boldsymbol{p}) = \boldsymbol{u}^{(M)}(\boldsymbol{q})$. By the relationship in projective geometry, because $\hat{\boldsymbol{d}}$ is the projection of $\boldsymbol{d}$ onto the tangent plane at $\boldsymbol{q}$, the point $\boldsymbol{q} + \varepsilon \hat{\boldsymbol{d}}$ is also the projection of $\boldsymbol{p} + \varepsilon \boldsymbol{d}$ onto the tangent plane at $\boldsymbol{q}$. When ε approaches zero, we have

$$\lim_{\varepsilon \to 0} \boldsymbol{v}(\boldsymbol{p} + \varepsilon \boldsymbol{d}) = \lim_{\varepsilon \to 0} \boldsymbol{v}(\boldsymbol{q} + \varepsilon \hat{\boldsymbol{d}}) = \lim_{\varepsilon \to 0} \boldsymbol{u}^{(M)}(\boldsymbol{q} + \varepsilon \hat{\boldsymbol{d}}) \,. \tag{43}$$

Combining Eqs. (42) and (43), we have the final result

$$\partial_{\boldsymbol{d}} \boldsymbol{v}(\boldsymbol{p}) = \lim_{\varepsilon \to 0} \frac{\boldsymbol{u}^{(M)}(\boldsymbol{q} + \varepsilon \hat{\boldsymbol{d}}) - \boldsymbol{u}^{(M)}(\boldsymbol{q})}{\varepsilon} = \partial_{\hat{\boldsymbol{d}}} \boldsymbol{u}^{(M)}(\boldsymbol{q}) \,. \tag{44}$$