

3D Reconstruction with Fast Dipole Sums

HANYU CHEN, BAILEY MILLER, and IOANNIS GKIOULEKAS, Carnegie Mellon University, USA

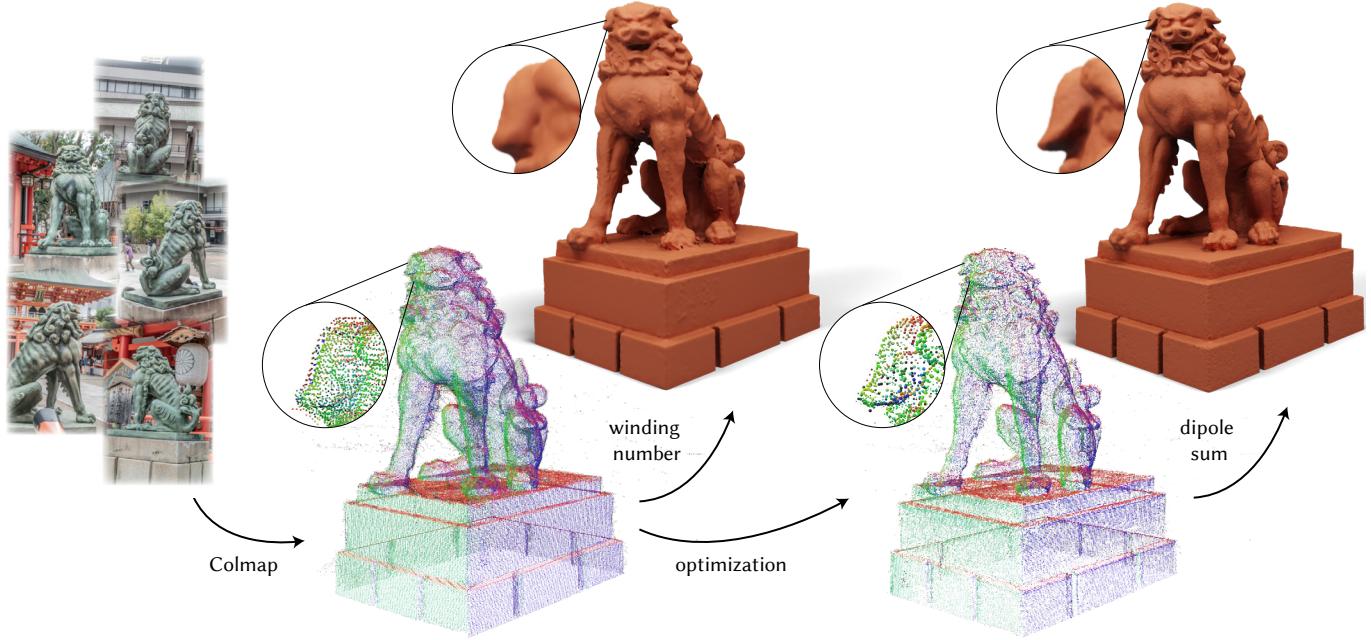


Figure 1. We introduce the dipole sum, a point-based representation for multi-view 3D reconstruction. This representation can model both implicit geometry and radiance fields using per-point attributes in a point cloud. It also supports efficient ray tracing and differentiable rendering, thus facilitating optimization using multi-view images. We initialize our dipole sum representation using the point cloud output of a structure from motion procedure (Colmap), which is required also to obtain camera poses. Bootstrapping from this initialization, we use inverse rendering to optimize per-point attributes (visualized in insets as varying point radii), resulting in a higher-quality surface reconstruction. Images are from the “Komainu / Kobe / Ikuta-jinja” dataset by Open Heritage 3D.

We introduce a technique for the reconstruction of high-fidelity surfaces from multi-view images. Our technique uses a new point-based representation, the *dipole sum*, which generalizes the winding number to allow for interpolation of arbitrary per-point attributes in point clouds with noisy or outlier points. Using dipole sums allows us to represent implicit geometry and radiance fields as per-point attributes of a point cloud, which we initialize directly from structure from motion. We additionally derive Barnes-Hut fast summation schemes for accelerated forward and reverse-mode dipole sum queries. These queries facilitate the use of ray tracing to efficiently and differentiably render images with our point-based representations, and thus update their point attributes to optimize scene geometry and appearance. We evaluate this inverse rendering framework against state-of-the-art alternatives, based on ray tracing of neural representations or rasterization of Gaussian point-based representations. Our technique significantly improves reconstruction quality at equal runtimes, while also supporting more general rendering techniques such as shadow rays for direct illumination. In the supplement, we provide interactive visualizations of our results.

CCS Concepts: • Computing methodologies → Point-based models; Ray tracing.

Authors' address: Hanyu Chen, hanyuche@andrew.cmu.edu; Bailey Miller, bmmiller@andrew.cmu.edu; Ioannis Gkioulekas, igkioule@cs.cmu.edu, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA, 15213, USA.

© 2024 Copyright held by the owner/author(s).

Additional Key Words and Phrases: Winding number, point-based modeling, inverse rendering

Reference Format:

Hanyu Chen, Bailey Miller, and Ioannis Gkioulekas. 2024. 3D Reconstruction with Fast Dipole Sums. *Technical Report 1*, 1 (May 2024), 16 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

1 INTRODUCTION

The emergence of neural rendering techniques [Tewari et al. 2022] has led to the widespread adoption of a two-stage pipeline for multi-view 3D reconstruction: The first stage employs traditional geometric techniques such as structure from motion [Schönberger and Frahm 2016] to estimate unknown parameters required for the second stage—namely, camera poses. The second stage uses gradient-based optimization and differentiable rendering to optimize a scene representation so that it can reproduce available multi-view images—an inverse rendering process. The performance of this pipeline depends critically on the scene representation, and this fact has facilitated the development of various choices (e.g., neural [Mildenhall et al. 2021; Wang et al. 2021b], grid-based [Fridovich-Keil et al. 2022; Karnewar et al. 2022; Wu et al. 2023], hash-encoded [Müller et al. 2022; Wang et al. 2023; Li et al. 2023]), each offering different tradeoffs between expressive power and computational efficiency.

This paper introduces another representation for multi-view 3D reconstruction, the *dipole sum*. This is a point-based representation, where both the scene geometry (an implicit surface) and scene lightfield (a radiance field) are modeled through per-point attributes on a point cloud; in turn, these attributes are interpolated to arbitrary query locations using tailored kernels. Our introduction of the dipole sum continues a recent shift towards point-based representations for neural rendering [Xu et al. 2022]. In particular, point-based representations using 3D Gaussian kernels have recently gained widespread popularity for both novel-view synthesis tasks [Kerbl et al. 2023] and 3D reconstruction [Dai et al. 2024]: The use of Gaussian kernels allows these techniques to perform differentiable rendering using image-space rasterization instead of previous ray-tracing algorithms, resulting in impressive computational acceleration. At the same time, the use of rasterization precludes the combination of these representation with advanced rendering features such as direct illumination estimation (e.g., with shadow rays), which rasterization is incompatible with.

By contrast, we design the dipole sum representation to support efficient differentiable rendering with *ray tracing*. Our representation is fundamentally based on the *winding number* for point clouds Barill et al. [2018]—an approximation to the indicator function of the solid object represented by the point cloud, which can be computed as a sum of Poisson kernels centered at every point cloud location. The winding number has well-known geometric regularity properties [Xu et al. 2023], being a harmonic function that approximates the implicit surface produced by robust surface reconstruction algorithms [Kazhdan et al. 2006]. It is additionally amenable to efficient computation, through combination with fast summation techniques [Beatson et al. 1997]. Lastly, as a point-based representation, it can be directly initialized with the other output of the first-stage structure from motion—a 3D point cloud.

The dipole sum generalizes the winding number in several ways that preserve these desirable properties, while also resulting in a point-based representation that is suitable for inverse rendering applications. In particular, as we explain in Section 4, we use regularized kernels and general per-point attributes, to enable use of this representation with point clouds that are noisy or contain outliers—as point clouds from structure from motion typically do. Then in Section 5, we show how to use dipole sums to represent not only the geometry, but also the radiance field of a scene. Lastly, in Section 6, we use fast summation techniques to enable efficient computation and backpropagation, required for the inverse rendering process.

With the resulting fast dipole sums, we can use ray tracing to optimize a point-based representation initialized directly from structure of motion, by simply updating point-based attributes. Figure 1 shows an example use of our approach: Structure from motion [Schönberger and Frahm 2016] produces a point cloud that we can visualize using the winding number. We can then use inverse rendering with dipole sums to optimize attributes of this point cloud (visualized in the insets), resulting in greatly improved surface reconstruction quality. In Section 7, we systematically evaluate this approach against state-of-the-art neural rendering techniques for surface reconstruction, using neural [Li et al. 2023; Wang et al. 2023] and 3D Gaussian [Dai et al. 2024] representations. Our experiments show that our approach is both efficient and effective, improving

reconstruction quality at equal runtimes, while additionally supporting rendering with techniques such as shadow rays. In the supplement, we provide interactive visualizations of all results.

2 RELATED WORK

Structure from motion. 3D reconstruction from uncalibrated multi-view images, also known as *structure from motion*, is a classical problem in computer vision [Tomasi and Kanade 1990; Ullman 1979]. It has been the subject of extensive theoretical study [Hartley and Zisserman 2003] and engineering efforts [Snavely et al. 2008, Bundler]—we refer to Özyeşil et al. [2017] for a detailed review. Traditional techniques attacked this problem primarily by enforcing inter-image geometric consistency, and triangulating correspondences across different images. Mature structure from motion techniques of this kind can robustly produce 3D reconstructions—typically in the form of point clouds—from thousands of images [Snavely et al. 2006], covering scenes ranging from individual objects [Schönberger and Frahm 2016] to entire cities [Agarwal et al. 2011]. We focus on the first setting, and aim to produce high-fidelity object-level surface reconstructions, by directly utilizing and refining point clouds produced from structure from motion implementations (Figure 1).

Shading-based refinement and neural rendering. Whereas geometric structure-from-motion techniques can produce dense point clouds with good coverage, they can cause holes in textureless areas where there are no correspondences. They also have difficulty producing high surface detail, because they do not exploit shading cues that provide normal information. Shading-aware refinement techniques can refine initial structure from motion reconstructions using either simple shading models [Dai et al. 2017; Langguth et al. 2016; Zollhöfer et al. 2015; Wu et al. 2011] or complex differentiable rendering procedures [Luan et al. 2021]. However, accounting for shading requires also optimizing for ancillary scene information, such as reflectance and global illumination, resulting in a challenging and ill-posed inverse rendering problem.

Recently, neural rendering techniques have made tremendous progress towards overcoming these challenges. We refer to Tewari et al. [2022] for a detailed review, and discuss only the most relevant works. Mildenhall et al. [2021] tackled multi-view reconstruction problems through the combined use of differentiable volume rendering (implemented through ray tracing), neural field representations for both geometry (implicit surfaces) and global illumination (radiance fields), and structure from motion for pose estimation (COLMAP [Schönberger and Frahm 2016]). Though they initially focused on novel-view synthesis, subsequent techniques have adapted this methodological approach for surface reconstruction tasks [Yariv et al. 2021; Oechsle et al. 2021; Wang et al. 2021b]. Unfortunately, the expressive power neural field representations provide comes with two critical caveats: 1. It introduces a severe computational overhead, resulting in very costly inverse rendering optimization. 2. It makes it difficult to leverage the 3D reconstruction output of structure from motion in ways more direct and effective than as just regularization during optimization [Deng et al. 2022; Fu et al. 2022]. We overcome these challenges by developing point-based field representations that are amenable to efficient ray tracing, and can directly optimize the 3D point cloud from structure from motion.

Geometry and radiance field representations. To alleviate the computational complexity issues due to neural field representations, recent work has made rapid progress towards alternative representations for implicit geometry and radiance fields. Grid-based techniques replace neural fields with either dense [Karnewar et al. 2022] or adaptive [Fridovich-Keil et al. 2022; Wu et al. 2023] grids that are efficient to ray trace [Museth et al. 2013] and interpolate, though potentially memory intensive (for dense grids) or difficult to optimize in an end-to-end manner (for adaptive grids). Hash-based techniques replace neural fields with multi-resolution hash encodings [Müller et al. 2022; Wang et al. 2023; Li et al. 2023], which combine expressive power and efficiency. All these approaches can optionally be combined with shallow (thus more efficient) neural networks that post-process interpolated or encoded features. These approaches overcome computational efficiency issues associated with neural fields, though they still do not provide a way to directly use 3D information from structure from motion.

Point-based field representations use a point cloud and kernel-based interpolation to compute field quantities needed to express implicit geometry and radiance fields. Xu et al. [2022] proposed this approach for novel-view synthesis, though their use of complex neural network post-processing of point features still introduces significant computational overhead. Kerbl et al. [2023] introduced a point-based representation that uses collections of 3D Gaussians to represent both geometry (volumetric density) and radiance. Critically, they also combine this representation with rasterization—through image-space Gaussian splatting—to eliminate the need for costly ray tracing during volume rendering, thus achieving real-time optimization and rendering performance. Though this technique originally focused on novel-view synthesis, subsequent works Guédon and Lepetit [2023]; Dai et al. [2024] have provided extensions for high-fidelity surface reconstruction. Being point-based, these techniques can directly leverage 3D information from structure from motion. However, in transitioning from ray tracing to rasterization, they sacrifice generality: for example, rasterization rules out rendering techniques such as shadow rays [Ling et al. 2023] (also known as next-event estimation Pharr et al. [2023]) for rendering direct illumination from known light sources. We contribute a point-based representation that achieves high-quality surface reconstruction as efficiently as Gaussian splatting techniques, using *ray tracing*, thus maintaining compatibility with such rendering techniques.

Point cloud surface reconstruction. Point-based geometry representations have a long history in computer graphics as techniques for reconstructing continuous surfaces (either implicit or, after iso-surface extraction Lorensen and Cline [1987], explicit) from point clouds. These techniques often find use as post-processing of point clouds from geometric structure from motion techniques, and thus are robust to imperfections such as noisy points, outlier points, or holes. We refer to Berger et al. [2014] for a review. The techniques by Fuhrmann and Goesele [2014]; Zagorchev and Goshtasby [2011] use anisotropic Gaussian functions and their derivatives to interpolate scalar fields from point locations, and thus bear a strong similarity to the 3D Gaussian splatting representations we discussed above. Carr et al. [2001] use instead more general radial-basis functions for interpolation, combined with fast summation techniques [Beatson

et al. 1997]. Among this extensive family of techniques, we build on the point-based winding number representation Jacobson et al. [2013]; Barill et al. [2018], because of its attractive properties of geometric regularity, robustness, and efficiency—we provide a review in Section 3. We generalize winding numbers in Sections 4–6 into our dipole sum representation, which we use for both implicit geometry and radiance fields. Doing so allows us to achieve efficient inverse rendering of point clouds for high-quality surface reconstruction.

3 BACKGROUND

We discuss background on volume rendering with radiance fields for surface reconstruction, and the winding number for point clouds.

3.1 Inverse volume rendering with radiance fields

We follow the methodology introduced by NeRF [Mildenhall et al. 2021] and represent a 3D scene as a volume comprising two components: 1. an *attenuation coefficient* $\sigma : \mathbb{R}^3 \times \mathcal{S}^2 \rightarrow \mathbb{R}_{\geq 0}$ representing the scene’s geometry; and 2. a *radiance field* $L : \mathbb{R}^3 \times \mathcal{S}^2 \rightarrow \mathbb{R}_{\geq 0}^3$ representing the scene’s (RGB) lightfield. As Miller et al. [2024] explain, at every scene point $x \in \mathbb{R}^3$ and direction $\omega \in \mathcal{S}^2$, the attenuation coefficient $\sigma(x, \omega)$ is the probability density that a ray passing through x along ω will terminate instantly due to intersection with the scene’s geometry. Then, the radiance field $L(x, \omega)$ is the incident (RGB) global illumination at x along ω .

This representation allows expressing the RGB intensity (*color*) c captured by a camera ray $r_{o,v}(\tau) \equiv o + \tau \cdot v$ with origin o and direction v using the (exponential) *volume rendering equation*:

$$c(o, v) = \int_{\tau_n}^{\tau_f} \exp\left(-\int_{\tau_n}^{\tau} \sigma(r_{o,v}(t), v) dt\right) \cdot \sigma(r_{o,v}(\tau), v) L(r_{o,v}(\tau), -v) dt, \quad (1)$$

where τ_n and τ_f are near and far (resp.) integration limits due to the scene’s bounding box. Rasterization approaches [Kerbl et al. 2023; Zwicker et al. 2002; Dai et al. 2024] approximate Equation (1) by projecting (a point-based representation of) σ and L on the image plane, where integration becomes an efficient splatting operation. By contrast, ray tracing approaches approximate $c(o, v)$ with numerical quadrature [Max 1995] using ray samples $\tau_n = \tau_0 < \dots < \tau_J = \tau_f$:

$$c(o, v) \approx \sum_{j=1}^J \exp\left(-\sum_{i=1}^j \sigma_i \Delta_i\right) \cdot (1 - \exp(\sigma_j \Delta_j)) L_j \quad (2)$$

where at each sample location τ_j , $\Delta_j \equiv \tau_j - \tau_{j-1}$, $\sigma_j \equiv \sigma(r_{o,v}(\tau_j), v)$, and $L_j \equiv L(r_{o,v}(\tau_j), -v)$, $j = 1, \dots, J$. Both approaches are differentiable, thus allowing propagation of gradients from rendered colors c to the attenuation coefficient σ and radiance field L . Rasterization approaches are typically faster, but also less general than ray tracing ones, e.g., ray tracing allows using shadow rays to incorporate direct illumination from known light sources [Bi et al. 2020a,b].

With this representation at hand, NeRF techniques reconstruct a 3D scene from a multi-view image datasets in two stages: 1. First, they use structure from motion [Schönberger and Frahm 2016] to estimate camera locations o and poses v for each image, alongside a point cloud reconstruction of the 3D scene. 2. Second, they use gradient descent methods [Kingma and Ba 2015] to optimize σ and L , so as to minimize an objective comparing real images with images

rendered as in Equation (2)—an *inverse rendering* process [Loper and Black 2014; Marschner 1998]. The second stage uses the camera poses output by the first stage, as this information is needed to render images with Equation (1). Deng et al. [2022] show that the final reconstruction can improve by additionally leveraging the point cloud output during the second-stage optimization.

Surface reconstruction. To improve the performance of the above methodology in surface reconstruction tasks, prior work [Wang et al. 2021b; Yariv et al. 2021; Oechsle et al. 2021; Miller et al. 2024] has represented the attenuation coefficient σ as an analytic function of a scalar field $F : \mathbb{R}^3 \rightarrow \mathbb{R}$ —which we term the *geometry field*—controlling an implicit surface representation of the scene geometry $\Gamma \subset \mathbb{R}^3$, i.e., $\Gamma \equiv \{x \in \mathbb{R}^3 : F(x) = 0\}$ (with the convention that points where $F(x) < 0$ are interior points).

We adopt the representation by Miller et al. [2024], which is a reciprocal modification of NeuS [Wang et al. 2021b]. This representation first defines an *occupancy function* in terms of F :

$$v(x) \equiv \Psi(s \cdot F(x)), \quad (3)$$

where $s > 0$ is a user-defined scale factor, and $\Psi : \mathbb{R} \rightarrow [0, 1]$ is the (*logistic*) *sigmoid function* [Han and Moraga 1995; Glorot et al. 2011]. Thus v equals $1/2$ when $F = 0$ (points on the surface Γ), approaches 1 as F increases (exterior points), and 0 as F decreases (interior points). Miller et al. [2024, Equation (12)] relate σ to v as :

$$\sigma(x, \omega) \equiv \frac{|\omega \cdot \nabla v(x)|}{v(x)}. \quad (4)$$

Then, reconstruction uses the two-stage approach we described above, except that the inverse rendering stage optimizes (through the differentiable Equations (3) and (4)) F instead of σ .

Our contribution. Within this context, we develop a point-based representation for the geometry field F and radiance field L that:

1. lends itself to *fast* ray tracing, thus enabling efficient inverse rendering (as with rasterization) without sacrificing generality (shadow rays for direct illumination);
2. allows directly leveraging the point cloud output of structure for motion, optimizing only per-point attributes and a shallow multi-layer perceptron (MLP) during inverse rendering;
3. reconstructs high quality surfaces by implicitly enforcing geometric regularization (e.g., harmonicity).

Figure 2 provides an overview of our overall technique.

3.2 Winding number

We derive our point-based representation as a generalization of the *winding number*, which we discuss next.

Continuous surfaces. We first consider the winding number for a continuous surface $\Gamma \subset \mathbb{R}^3$. There are many equivalent definitions of the winding number [Feng et al. 2023]; we follow Barill et al. [2018] and use its definition as a *jump harmonic* scalar field, as this definition facilitates the generalizations we consider in Section 4. Then, the *winding number* $w : \mathbb{R}^3 \rightarrow \mathbb{R}$ is the scalar field solution to the Laplace boundary value problem (BVP) with jump Dirichlet

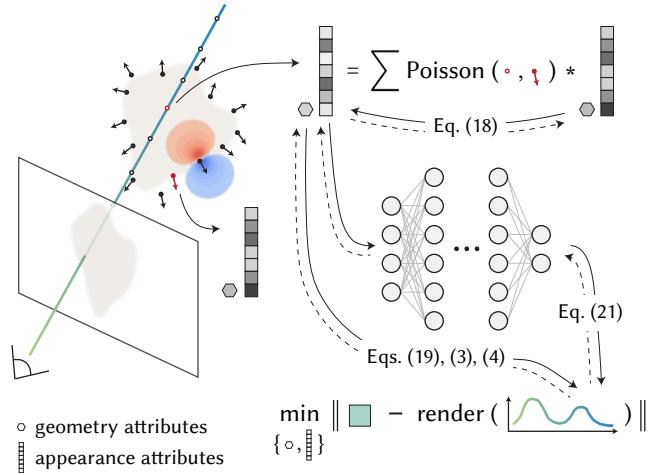


Figure 2. Overview of our method. During forward rendering (indicated by solid arrows), at each sample location, we interpolate geometry and appearance attributes from a point cloud through a fast primal dipole sum query. Appearance attributes are passed through a shallow MLP to predict colors, and geometry attributes are used to compute attenuation coefficients. We integrate along the ray to compute the rendered color and minimize the L^1 -loss between the rendered and ground truth colors. During backpropagation (indicated by dashed arrows), we optimize geometry and appearance attributes of the point cloud through a fast adjoint dipole sum query.

and Neumann boundary conditions:

$$\begin{aligned} \Delta w(x) &= 0 && \text{in } \mathbb{R}^3 \setminus \Gamma, \\ w^+(x) - w^-(x) &= 1 && \text{on } \Gamma, \\ \frac{\partial w^+}{\partial n}(x) - \frac{\partial w^-}{\partial n}(x) &= 0 && \text{on } \Gamma. \end{aligned} \quad (5)$$

Here, $n(x)$ is the outward normal vector at point $x \in \Gamma$, and $w^\pm(x) \equiv \lim_{\varepsilon \rightarrow 0} w(x \pm \varepsilon \cdot n(x))$ are the limit winding number values on either side of the surface Γ along the normal direction. Krutitskii [2001] provide a detailed treatment of such BVPs, and in particular prove the following *boundary integral* expression for their solution:

$$w(x) = \int_{\Gamma} P(x, y) \cdot 1 \, dA(y), \quad P(x, y) \equiv \frac{1}{4\pi} \frac{n(y) \cdot xy}{\|x - y\|^2}, \quad (6)$$

where $xy \equiv y - x / \|yx\|$ is the direction from x to y , and $P : \mathbb{R}^3 \times \mathbb{R}^3 \rightarrow \mathbb{R}$ is the *free-space Poisson kernel* for the Laplacian partial differential equation (PDE). We make explicit the factor 1 in the integral of Equation (6), corresponding to the jump Dirichlet boundary condition in Equation (5), for reasons we will explain in Section 4.

From Equation (6), the winding number $w(x)$ equals the *signed solid angle* the surface Γ subtends at point x . When the surface Γ is the watertight boundary of one or more three-dimensional objects, then $w(x)$ equals their binary *indicator function*— $w(x) = 1$ for points x at the objects’ interior, $w(x) = 0$ otherwise.

Point clouds. We now consider the winding number for an *oriented* point cloud $\mathcal{P} \equiv \{(p_m, n_m, A_m)\}_{m=1}^M$, where for each m we assume that: 1. the point p_m is a sample from an underlying surface Γ ; 2. the vector n_m is the normal of Γ at p_m ; and 3. the scalar A_m is the geodesic Voronoi area on Γ of p_m , i.e., the area of the subset of

Γ where points are closer (in the geodesic distance sense) to p_m than any other point in \mathcal{P} . In practice, we use the point cloud from structure from motion, which has only points p_m and normals n_m available. We estimate area weights A_m following Barill et al. [2018].

The boundary integral (6) suggests the following generalization of the winding number for point clouds [Barill et al. 2018]:

$$w_{pc}(x) \equiv \sum_{m=1}^M A_m P(x, p_m) \cdot 1 = \sum_{m=1}^M \frac{A_m}{4\pi} \frac{n_m \cdot x p_m}{\|x - p_m\|^2} \cdot 1. \quad (7)$$

Winding number as a geometry field. Though w_{pc} is not a binary scalar field (unlike its continuous counterpart w), its behavior is still suggestive of the continuous surface Γ underlying \mathcal{P} : As Barill et al. [2018] show, it approaches $1/2$ at points near the continuous surface Γ underlying \mathcal{P} , increases towards its interior, and decreases towards its exterior. Thus at first glance, it appears that we can use it to represent a geometry field for inverse rendering (Section 3.1) as:

$$F_{pc}(x) \equiv \frac{1}{2} - w_{pc}(x). \quad (8)$$

This representation provides several critical advantages:

1. The corresponding implicit surface $\Gamma_{pc} \equiv \{x \in \mathbb{R}^3 : F_{pc}(x) = 0\}$ provides an approximation to Γ that becomes exact as point density converges to infinity, and degrades gracefully as the number of points M decreases.
2. F_{pc} is imbued with regularity properties that provide useful *geometric regularization*. It is *harmonic*, and thus of a smooth nature that has proven useful for geometric optimization tasks [Liu et al. 2022]. It is also closely related to geometric representations [Kazhdan et al. 2006; Belyaev et al. 2013] and interpolation schemes [Floater et al. 2005; Ju et al. 2005] that have found great success in reconstruction applications thanks to their robustness. We elaborate on these connections below and in Section 5.
3. F_{pc} can be directly computed using the point cloud from structure-from-motion initialization. Point queries for F_{pc} , and thus w_{pc} , use only the point cloud and per-point attributes, and do not require meshing or a proxy data structure (e.g., grid or neural).
4. Such point queries, *and backpropagating through them*, can be achieved efficiently with logarithmic complexity $O(\log M)$ relative to point cloud size M , as we explain in Section 6. Thus, F_{pc} lends itself to efficient ray tracing, which requires multiple point queries along each viewing ray (Equation (1)).

Barill et al. [2018] further discuss the benefits of the winding number w_{pc} versus other point-based surface representations. Unfortunately, w_{pc} , and thus F_{pc} , have a few critical shortcomings that make them unsuitable for direct use for inverse rendering. We explain these shortcomings in Section 4, where we overcome them by developing a generalization of w_{pc} that facilitates point-based representation of both the geometry field F and the radiance field L .

Relationship to Poisson surface reconstruction. To emphasize the useful geometric regularization properties of the point-cloud winding number w_{pc} , we remark on its relationship with *Poisson surface reconstruction (PSR)* [Kazhdan et al. 2006; Kazhdan and Hoppe 2013]. Both techniques take as input an oriented point cloud and output a scalar field that approximates the harmonic solution to the BVP (5) for the underlying continuous surface Γ . As Feng et al. [2023];

Barill et al. [2018] explain, the limit behavior of both techniques is the same, recovering the same harmonic function. Even away from that limit, PSR outputs robust surface reconstructions thanks to the regularity properties of the Laplace and Poisson BVPs it uses [Peng et al. 2021], and thus has become a workhorse for point-based surface reconstruction [Berger et al. 2014]. However, querying the scalar field output of PSR requires performing a global Poisson solve operation, which is prohibitively expensive for inverse rendering. By contrast, using w_{pc} , and its generalization in Section 4, allows us to efficiently query and render an approximation to the PSR output with similar regularity and robustness properties.

4 REGULARIZED DIPOLE SUMS

We introduce a generalization of Equation (7) that will serve as the basis for our point-based representation for the geometry and radiance fields in inverse rendering. Our generalization is twofold, involving the use of regularized Poisson kernels and general Dirichlet boundary conditions. We first define our generalization, then explain how we use it to represent the geometry and radiance fields.

Regularized Poisson kernel. The Poisson kernel $P(x, y)$ is singular as $x \rightarrow y$. In theory, the singularity makes the implicit surface Γ_{pc} an *exact interpolant* of the point cloud \mathcal{P} . In practice, the singularity makes the *effective* surface Γ_{pc} numerical algorithms interface with—e.g., during ray tracing [Gillespie et al. 2024, Section 4.3] or iso-surface extraction [Barill et al. 2018, Section 3, Figure 9]—inaccurate and numerically unstable near \mathcal{P} . Additionally, exact interpolation is undesirable when working with *noisy* point clouds [Barill et al. 2018, Section 9], such as those from structure-from-motion initialization.

To overcome these issues, we turn to the method of *regularized fundamental solutions*, developed in PDE simulation [Beale et al. 2016; Cortez 2001; Cortez et al. 2005] to address similar numerical issues from singular potential kernels. Its starting point is the definition of the Poisson kernel through the *Green’s function* (or *fundamental solution*) $G : \mathbb{R}^3 \times \mathbb{R}^3 \rightarrow \mathbb{R}$ of the Laplace PDE:

$$P(x, y) \equiv n(y) \cdot \nabla_x G(x, y), \quad (9)$$

$$\text{where } G \text{ satisfies: } \Delta G(x, y) = \delta(x - y), \quad (10)$$

and δ is the Dirac delta distribution in \mathbb{R}^3 . The method of regularized fundamental solutions replaces δ with a *nascent delta function*, that is, a function $\phi_\varepsilon(x - y)$ satisfying $\lim_{\varepsilon \rightarrow 0} \phi_\varepsilon(x - y) = \delta(x - y)$. Then, we can define the regularized Green’s function G_ε and Poisson kernel P_ε exactly analogously to Equations (9) and (10):

$$P_\varepsilon(x, y) \equiv n(y) \cdot \nabla_x G_\varepsilon(x, y), \quad (11)$$

$$\text{where } G_\varepsilon \text{ satisfies: } \Delta G_\varepsilon(x, y) = \phi_\varepsilon(x - y). \quad (12)$$

It follows that $\lim_{\varepsilon \rightarrow 0} G_\varepsilon = G$ and $\lim_{\varepsilon \rightarrow 0} P_\varepsilon = P$. A common choice of nascent delta function is the Gaussian function:

$$\phi_\varepsilon(x - y) \equiv \frac{1}{\varepsilon \sqrt{2\pi}} \cdot \exp\left(-\frac{\|x - y\|^2}{2\varepsilon^2}\right), \quad (13)$$

with corresponding regularized Poisson kernel [Beale et al. 2016]:

$$P_\varepsilon(x, y) \equiv P(x, y) \cdot S\left(\frac{\|x - y\|}{\varepsilon}\right), \quad (14)$$

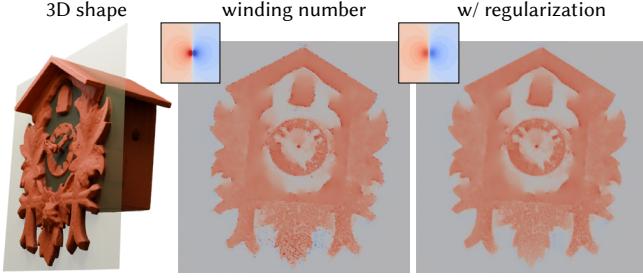


Figure 3. Visualization of the original and regularized winding number fields on the BlendedMVS clock scene. The original winding number field is extremely noisy near individual points of the point cloud due to the singular Poisson kernel, while the regularized winding number field is relatively smooth. The insets show winding number fields induced by a single point.

where $S(t) \equiv \text{erf}(t) - 2/\sqrt{\pi} \cdot t \cdot \exp(-t^2)$. Unlike P , P_ε is not singular, as $P_\varepsilon(x, x) = 3^{-1} \varepsilon^{-3} \pi^{-3/2}$ is finite for $\varepsilon > 0$. The parameter ε controls the trade-off between regularization (restricting how fast $P_\varepsilon(x - y)$ increases as $\|x, y\| \rightarrow 0$) and bias (controlling the difference $P_\varepsilon - P$).

We use P_ε in Equation (7) to get the *regularized winding number*:

$$w_{pc,\varepsilon}(x) \equiv \sum_{m=1}^M A_m P_\varepsilon(x, p_m) \cdot 1. \quad (15)$$

Unlike w_{pc} , $w_{pc,\varepsilon}$ will *not* exactly interpolate point in \mathcal{P} . We can use regularization during inverse rendering (Section 5.3) to penalize large deviations, while also allowing for inexact interpolation to account for noise in point cloud locations. Figure 3 compares 2D slices of the singular versus regularized Poisson kernel, as well as 2D slices of the original and regularized winding number fields for a 3D model from the BlendedMVS dataset [Yao et al. 2020], using the point cloud output from structure from motion initialization.

Regularized dipole sum. In addition to noisy points, point clouds from structure from motion can suffer from outlier points (e.g., due to incorrect correspondences) and holes (e.g., in surface areas without texture). We generalize Equation (15) to a point-based representation with increased descriptive power that: 1. we can use as the geometry field in inverse rendering to alleviate both noise and outlier issues; and 2. we can additionally use as a representation for the radiance field. We elaborate on both uses in Section 5.

To this end, we modify the BVP (5) defining the winding number to use an arbitrary *Dirichlet data* function $b : \Gamma \rightarrow \mathbb{R}$:

$$\begin{aligned} \Delta u^b(x) &= 0 && \text{in } \mathbb{R}^3 \setminus \Gamma, \\ u^{b+}(x) - u^{b-}(x) &= b(x) && \text{on } \Gamma, \\ \frac{\partial u^{b+}}{\partial n}(x) - \frac{\partial u^{b-}}{\partial n}(x) &= 0 && \text{on } \Gamma. \end{aligned} \quad (16)$$

Its solution u^b equals the *double-layer potential* [Krutitskii 2001]:

$$u^b(x) \equiv \int_{\Gamma} P(x, y) \cdot b(y) dA(y). \quad (17)$$

Using the Dirichlet data values on the point cloud, $b_m \equiv b(p_m)$, $p_m \in \mathcal{P}$, and the regularized kernel P_ε to circumvent singularity issues, we

arrive at a regularized point-cloud approximation of Equation (17):

$$u_{pc,\varepsilon}^b(x) \equiv \sum_{m=1}^M A_m P_\varepsilon(x, p_m) \cdot b_m. \quad (18)$$

For any b , we term $u_{pc,\varepsilon}^b(x)$ the corresponding (*regularized*) *dipole sum*, adopting from Barill et al. [2018] the term *dipole* for the Poisson kernel of an oriented point. The point-cloud winding number in Equation (7) and its regularized form in Equation (15) are special cases of dipole sums, i.e., $w_{pc} = u_{pc,0}^1$ and $w_{pc,\varepsilon} = u_{pc,\varepsilon}^1$.

5 POINT-BASED FIELD REPRESENTATIONS

We now can introduce our point-based representations for the geometry and radiance fields we use for inverse rendering. We augment the point cloud from structure from motion with additional per-point attributes, representing samples of different Dirichlet data functions in Equation (16): $\mathcal{P} := \left\{ (p_m, n_m, A_m, f_m, \ell_m^1, \dots, \ell_m^K) \right\}_{m=1}^M$. We then use dipole sums as in Equation (18) to *interpolate* these per-point attributes to field quantities.

5.1 Geometry field representation

We generalize the geometry field of Equation (8) as a dipole sum for the *geometry attribute* f :

$$F(x) \equiv \frac{1}{2} - u_{pc,\varepsilon}^f(x), \quad (19)$$

We then convert $F(x)$ to an occupancy and attenuation coefficient $\sigma(x, \omega)$ using Equations (3) and (4). We initialize F to equal the regularized winding number $w_{pc,\varepsilon}$ in Equation (15) by using initial values of 1 for all geometry attributes f , which we then optimize during inverse rendering to update the scene geometry. Figure 4 visualizes these initial and optimized field quantities on the teaser scene. Compared to the winding number, allowing non-unit values for f during inverse rendering serves two goals: 1. It allows the process to automatically diminish the influence of any outlier points in the point cloud, by decreasing their geometry attribute. The point cloud insets in Figure 1 visualize this effect, by scaling point radii by their optimized geometry attribute. 2. It allows the process to modify the scene geometry (e.g., to account for noise in point locations or better fill holes in textureless regions) *without* changing the point locations p in \mathcal{P} ; as we explain in Section 6.3, keeping point locations constant facilitates faster inverse rendering.

Probabilistic interpretation. In Appendix A, we prove an alternative derivation of $u_{pc,\varepsilon}^f(x)$ as the *expected value* of $w_{pc}(x)$ for a point cloud \mathcal{P} where: 1. point locations are isotropic Gaussian random variables with mean p_m and variance ε ; 2. at each point cloud location, the normal is a spherical random variable with mean direction n_m and circular variance $1 - f_m$, conditional on the point location. This derivation lends additional support to our use of a dipole sum to represent the geometry of a noisy point clouds \mathcal{P} with potential outliers (i.e., $f \approx 0$). It also suggests the possibility of using different values ε_m , $m = 1, \dots, M$ to model varying per-point uncertainty [Fuhrmann and Goesele 2014]. Empirically, we did not find such a practice beneficial, and thus opt to use a global value ε that we select as we explain in Section 5.3.

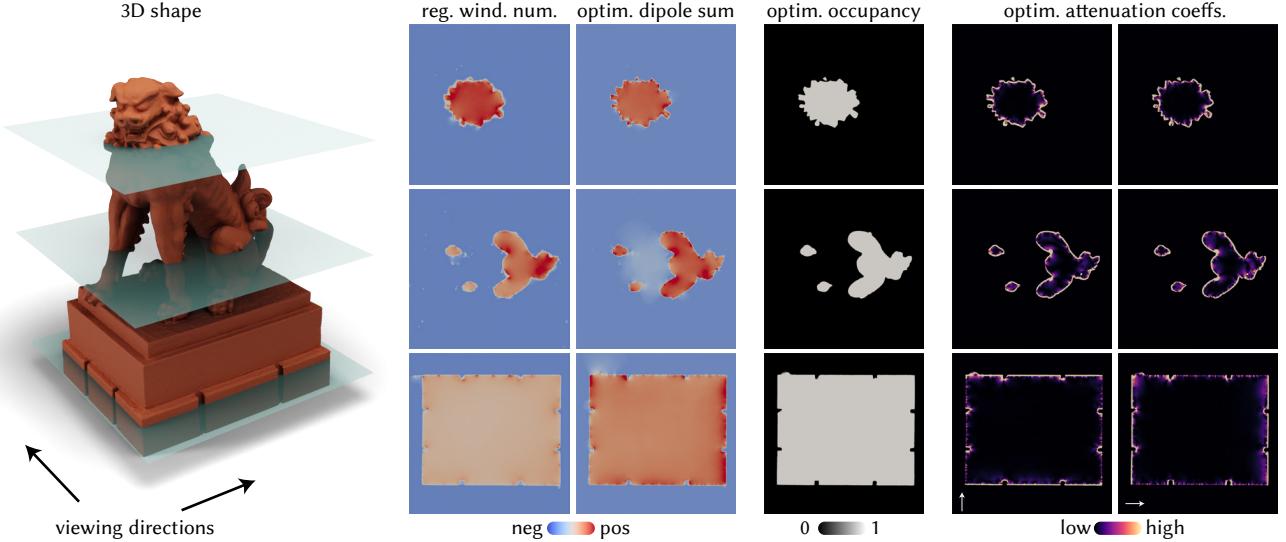


Figure 4. We visualize on the teaser scene geometry-related field quantities that we use for inverse rendering. From left to right: the initial geometry field with unit geometry attributes (equal to the regularized winding number in Equation (15)), the optimized geometry field with learned geometry attributes (Equation (19)), the optimized occupancy field (Equation (3)), and attenuation coefficients computed (Equation (4)) along two different viewing directions.

5.2 Radiance field representation

We first interpolate the *appearance attributes* ℓ^k using dipole sums:

$$\ell^k(x) \equiv u_{pc,\ell}^{\ell^k}(x), k = 1, \dots, K. \quad (20)$$

We then represent the radiance field $L(x, \omega)$ as the output of a shallow multi-layer perceptron (MLP) that takes as input the values $\ell^k(x)$, (encoded) position x and direction ω , and the *implicit surface normal* from the geometry field $n_{imp}(x) \equiv \nabla F(x)/\|\nabla F(x)\|$:¹

$$L(x, \omega) \equiv \text{MLP}\left(x, \omega, n_{imp}(x), \ell^1(x), \dots, \ell^K(x)\right). \quad (21)$$

The radiance field L and geometry field F are intertwined, as the dipole sums for both geometry and appearance attributes share the same weights P_ℓ in Equation (18) (determined by point cloud locations p_m , area weights A_m , and normals n_m).

Relationship to mean value interpolation. The dipole sum in Equation (18), and its use in Equations (19) and (20) to interpolate per-point attributes to scalar fields, is closely related to 3D interpolation using *mean value coordinates* [Floater et al. 2005; Ju et al. 2005]. Using Equations (6) and (17), we can express the mean value interpolant (and its point-cloud approximation) at point $x \in \mathbb{R}^3$ of a function b defined on a surface Γ as:

$$mv^b(x) \equiv \frac{u^b(x)}{w(x)} \approx \frac{u_{pc}^b(x)}{w_{pc}(x)}. \quad (22)$$

¹We experimented with a representation where the interpolated appearance attributes $\ell^k(x)$ are *spherical harmonic coefficients* that are convertible to radiance $L(x, \omega)$ through a rotation operation, as advocated by Karnewar et al. [2022]; Fridovich-Keil et al. [2022]. Unfortunately, this approach, though sufficient for rendering high-quality novel views, resulted in surface artifacts around regions of strong specular appearance. Dai et al. [2024, Section 5] report similar issues, which they alleviate by using monocular normal priors. We instead followed Wu et al. [2023] and used an MLP to post-process the interpolated appearance attributes, to elide supervised data-driven priors.

Mean value interpolation has found widespread use in computer graphics and other areas [Hormann and Sukumar 2017], a success in large part thanks to the strong regularity properties (e.g., smoothness) of the mean value interpolant [Ju et al. 2005, Section 2]. These properties and empirical success lend further support to our choice of (regularized) dipole sums as a point-based representation for both geometry and radiance fields. In our representation, we omit normalization (denominator in Equation (22)), as we found empirically that the *linear precision* property it enforces inhibits the ability of the radiance field to reproduce specular highlights.

5.3 Inverse rendering with point-based fields

During the inverse rendering stage, we synthesize images using volume rendering and ray tracing (Equation (2)) combined with our point-based geometry F and radiance L fields. We then optimize the point cloud \mathcal{P} controlling F and L by minimizing the loss:

$$\mathcal{L}_{\text{rendering}} + \mathcal{L}_{\text{entropy}} + \mathcal{L}_{\text{normal}} + \mathcal{L}_{\text{interpolation}}, \quad (23)$$

where each summand includes an appropriate weight, and:

1. $\mathcal{L}_{\text{rendering}}$ is the L^1 -loss between input and rendered images;
2. $\mathcal{L}_{\text{entropy}}$ is a per-ray entropy loss inspired from Kim et al. [2022] (we provide details in Appendix B);
3. $\mathcal{L}_{\text{normal}}$ and $\mathcal{L}_{\text{interpolation}}$ aggregate losses $\|n_{imp}(p_m) - n_m\|^2$ and $\|F(p_m)\|^2$ (resp.) on the point cloud.

The losses $\mathcal{L}_{\text{interpolation}}$ and $\mathcal{L}_{\text{normal}}$ regularize the geometry field F to be consistent with the point cloud \mathcal{P} , by encouraging interpolation ($\mathcal{L}_{\text{interpolation}}$) and normal consistency ($\mathcal{L}_{\text{normal}}$) at all point cloud locations. These losses can be computed without singularities, thanks to our use of the regularized Poisson kernel in Equation (14).

We initialize \mathcal{P} with locations p_m and normals n_m from structure from motion initialization (COLMAP [Schönberger and Frahm

Algorithm 1 Barnes-Hut accelerated primal and adjoint queries for fast dipole sums.

```

1: struct TREENODE
2:    $\bar{p}, \bar{A}, \bar{r}, \bar{b} \leftarrow \text{TREEUPDATE}$                                 ▷Immutable node attributes initialized using Equations (24) and (25)
3:    $\bar{db} \leftarrow 0$                                                        ▷Mutable node gradient attribute
4:   function GETCONTRIBUTION( $x, \epsilon$ )
5:     return  $\bar{A} \cdot x\bar{p}/\|x-\bar{p}\|^2 \cdot S(\|x-\bar{p}\|/\epsilon) \cdot \bar{b}$           ▷Compute node contribution to dipole sum using Equation (26)
6:   end function
7:   function INCREMENTGRADIENT( $du^b, x, \epsilon$ )
8:      $\bar{db} += \bar{A} \cdot x\bar{p}/\|x-\bar{p}\|^2 \cdot S(\|x-\bar{p}\|/\epsilon) \cdot du^b$       ▷Increment node gradient attribute using Equation (43)
9:   end function
10:  function GETCHILDREN
11:    return listOfChildrenNodes                                         ▷Return a list of children nodes, or empty list if node is a leaf
12:  end function

Input: A query point  $x$ , the root node of a tree structure node, a control parameter  $\beta$ .
Output: Dipole sum  $u^b(x)$ .
13:  function PRIMALQUERY( $x, \text{node}, \epsilon, \beta$ )
14:    if  $\|x - \text{node}.\bar{p}\| > \beta \cdot \text{node}.\bar{r}$  then return node.GETCONTRIBUTION( $x, \epsilon$ )          ▷If the query point is far from the cluster, terminate
15:    listOfChildrenNodes  $\leftarrow$  node.GETCHILDREN                                         ▷Get list of children nodes
16:    if IsEMPTY(listOfChildrenNodes) then return node.GETCONTRIBUTION( $x, \epsilon$ )          ▷If the node is a leaf, terminate
17:     $u^b \leftarrow 0$                                                        ▷Initialize dipole sum value
18:    for child in listOfChildrenNodes do                                         ▷Iterate over all children nodes
19:       $u^b += \text{PRIMALQUERY}(x, \text{child}, \epsilon, \beta)$ 
20:    return  $u^b$ 
21:  end function

Input: A gradient  $du^b$ , a query point  $x$ , the root node of a tree structure node, a control parameter  $\beta$ .
22:  function ADJOINTQUERY( $du^b, x, \text{node}, \epsilon, \beta$ )
23:    if  $\|x - \text{node}.\bar{p}\| > \beta \cdot \text{node}.\bar{r}$  then node.INCREMENTGRADIENT( $du^b, x, \epsilon$ ) return          ▷If the query point is far from the cluster, terminate
24:    listOfChildrenNodes  $\leftarrow$  node.GETCHILDREN                                         ▷Get list of children nodes
25:    if IsEMPTY(listOfChildrenNodes) then node.INCREMENTGRADIENT( $du^b, x, \epsilon$ ) return          ▷If the node is a leaf, terminate
26:    for child in listOfChildrenNodes do                                         ▷Iterate over all children nodes
27:      ADJOINTQUERY( $du^b, x, \text{child}, \epsilon, \beta$ )
28:    end function

```

2016]), and area weights A_m computed as in Barill et al. [2018]. We initialize the geometry attributes f_m to 1 (equal to the winding number), and appearance attributes t_m^k , $k = 1, \dots, K$ using Gaussian random variates. Inverse rendering optimizes the geometry and appearance attributes of \mathcal{P} , the global scale s and regularization ϵ parameters in Equations (3) and (14) (resp.), and the MLP parameters in Equation (21). Importantly, we do not optimize the area weights and locations in \mathcal{P} , to facilitate faster forward and inverse rendering—we elaborate in Section 6. Instead, the geometry and appearance attributes provide us with enough degrees of freedom to represent high-quality geometry and appearance, and correct defects (outliers, holes) in the structure-from-motion point cloud.

6 BARNES-HUT FAST SUMMATION

Rendering with the point-based field representations in Section 5 requires *evaluating* dipole sums (Equation (18)) at multiple locations along each viewing ray, to compute the geometry (Equation (19)) and radiance fields (Equations (20) and (21)) in Equation (1). Inverse rendering with these representations additionally requires *backpropagating* through each dipole sum, to compute derivatives of

per-point attributes. We term such evaluation and backpropagation operations *primal* and *adjoint* (resp.) *dipole sum queries*, using terminology from differentiable rendering [Nimier-David et al. 2020; Vicini et al. 2021; Stam 2020]. Implemented naively, both primal and adjoint queries have linear complexity $O(M)$ relative to point cloud size M , as they require iterating over all points. For large point clouds—such as those by structure from motion—these queries become the main computational burden during inverse rendering, and can even become prohibitively expensive.

Fortunately, it is possible to dramatically accelerate both types of queries, enabling inverse rendering at speeds competitive with rasterization techniques Dai et al. [2024]. In particular, Barill et al. [2018] show how to perform primal queries for the winding number with *logarithmic complexity* $O(\log M)$, using the classical *Barnes-Hut fast summation method* [Barnes and Hut 1986]. We adopt their approach, which we adapt below to generalized dipole sums. Then, we show how to use Barnes-Hut fast summation to perform also adjoint queries with logarithmic complexity. To simplify discussion, throughout this section we use b as a stand-in for any of the *Dirichlet*

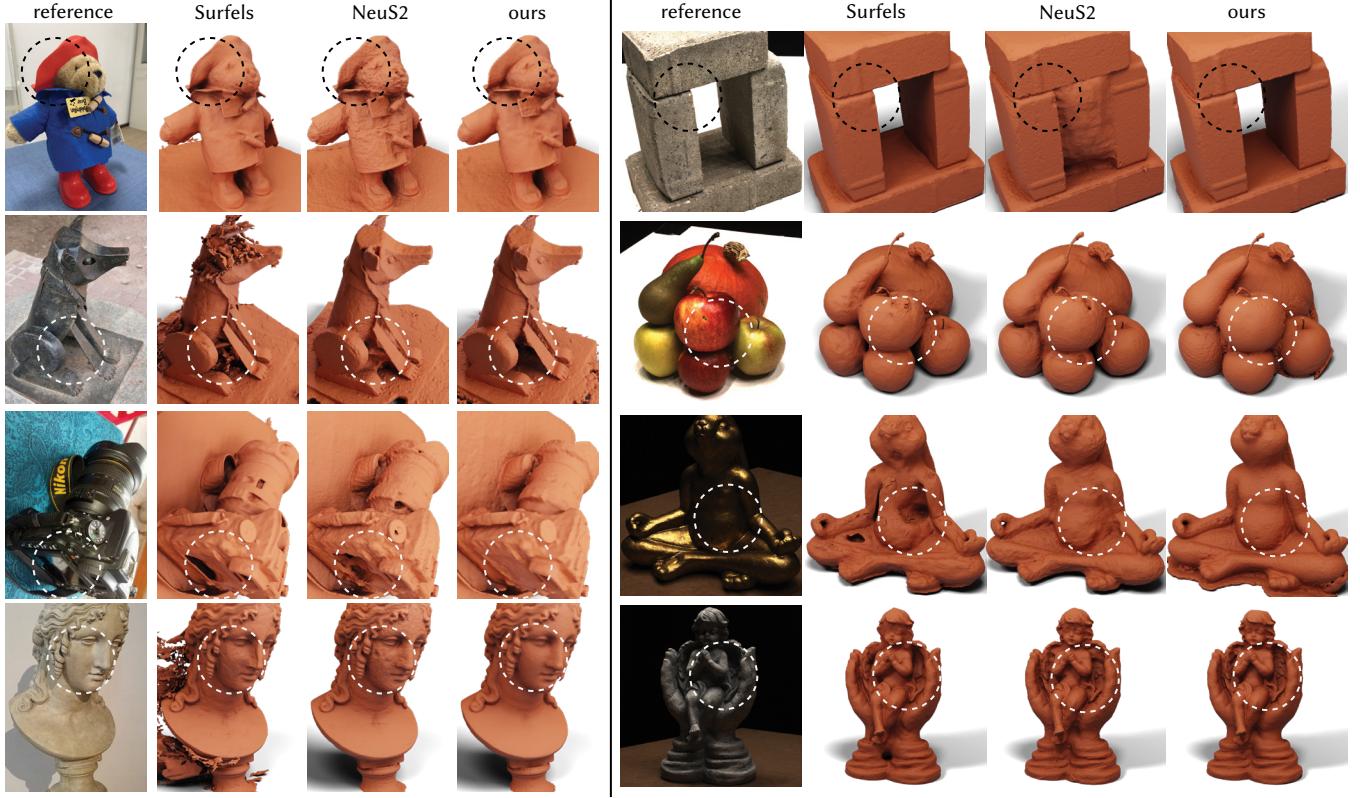


Figure 5. Qualitative comparisons on the BlendedMVS (left) and DTU (right) datasets. The dashed circles indicate areas of interest. We observe that while NeuS2 is capable of capturing fine details, it produces noisy meshes with artifacts that are likely caused by the multi-resolution hash grid representation. Gaussian Surfels produces floater artifacts that require manual filtering due to the lack of object mask supervision. In contrast, our method produces clean meshes with correct geometry and free of artifacts. We provide interactive visualizations of results on the entire datasets in the supplement.

data attributes stored in \mathcal{P} —namely, the geometry attribute f and the appearance attributes ℓ^k , $k = 1, \dots, K$.

6.1 Acceleration of primal queries

The Barnes-Hut fast first creates a tree data structure (e.g., octree [Meagher 1982]) whose nodes hierarchically subdivide the point cloud \mathcal{P} into clusters, with leaf nodes corresponding to individual points. Each tree node t is assigned a centroidal radius and attributes representative of the cluster comprising all leaf (i.e., single-point) nodes that are successors of t in the tree hierarchy. Denoting by $\mathcal{L}(t)$ the set of successor leaf nodes, we follow Barill et al. [2018] and assign the node area, location, area, and radius (resp.) attributes:

$$\bar{A}_t \equiv \sum_{m \in \mathcal{L}(t)} A_m, \quad \bar{p}_t \equiv \frac{1}{\bar{A}_t} \sum_{m \in \mathcal{L}(t)} A_m p_m, \quad \bar{r}_t \equiv \max_{m \in \mathcal{L}(t)} \|p_m - \bar{p}_t\|, \quad (24)$$

as well as *vector-valued* Dirichlet data attributes:

$$\bar{b}_t \equiv \frac{1}{\bar{A}_t} \sum_{m \in \mathcal{L}(t)} A_m n_m b_m, \quad (25)$$

which absorb the leaves’ Dirichlet data *and* normal attributes.

Then, for a primal query at point x , the Barnes-Hut method performs a depth-first tree traversal: at each node t , if x is sufficiently

far from the node’s centroid (i.e., $\|x - \bar{p}_t\| > \beta \cdot \bar{r}_t$, where β is a user-defined parameter), the node’s successors are not visited. Instead, the sum of contributions from all leaves in $\mathcal{L}(t)$ to the dipole sum is approximated using the node’s attributes:

$$\sum_{m \in \mathcal{L}(t)} A_m P_\varepsilon(x, p_m) \cdot b_m \approx \bar{A}_t \frac{\bar{b}_t \cdot x \bar{p}_t}{\|x - \bar{p}_t\|^2} \cdot S\left(\frac{\|x - \bar{p}_t\|}{\varepsilon}\right), \quad (26)$$

This approximation expresses the fact that, due to the squared-distance falloff of the Poisson kernel in Equation (6), the *far-field* influence of a cluster of points can be represented by a single point at the cluster’s centroid. Algorithm 1 (lines 13–21) summarizes the accelerated primal queries.

6.2 Acceleration of adjoint queries

A naive implementation of adjoint queries by using automatic differentiation (e.g., autograd [Paszke et al. 2017]) would result in linear complexity $O(M)$, even though the differentiated primal query has logarithmic complexity $O(\log M)$. The reason is that, even if the primal query stopped tree traversal at a node t , the node’s attributes would be functions of those of all successor leaf nodes. Thus, the adjoint query would still end up visiting all leaf nodes.

Table 1. Chamfer distances on the DTU dataset. (Surf. = Surfels, N.A. = Neuralangelo, mean* ignores scans 63, 83, 105.)

	5 m			10 m			18 h	1 h
	NeuS2	Surf.	ours	NeuS2	Surf.	ours	N.A.	ours
24	0.78	0.68	0.76	0.75	0.62	0.62	0.37	0.46
37	0.64	0.77	0.72	0.65	0.76	0.68	0.72	0.66
40	1.04	0.56	0.37	1.06	0.49	0.35	0.35	0.33
55	0.30	0.47	0.41	0.28	0.48	0.36	0.35	0.31
63	1.01	0.86	0.91	1.00	0.84	0.90	0.87	0.90
65	0.62	1.06	1.07	0.59	1.08	0.80	0.54	0.75
69	0.68	0.86	0.81	0.67	0.88	0.63	0.53	0.55
83	1.17	1.09	0.72	1.18	1.09	0.73	1.29	0.75
97	1.00	1.31	0.89	1.04	1.31	0.91	0.97	0.84
105	0.71	0.74	0.58	0.74	0.75	0.61	0.73	0.62
106	0.55	0.83	0.65	0.54	1.05	0.54	0.47	0.47
110	0.89	1.76	1.03	0.84	1.76	0.84	0.74	0.64
114	0.36	0.52	0.45	0.37	0.52	0.35	0.32	0.32
118	0.47	0.64	0.57	0.43	0.67	0.45	0.41	0.39
122	0.45	0.59	0.43	0.43	0.61	0.41	0.43	0.38
mean	0.71	0.85	0.69	0.70	0.86	0.61	0.61	0.56
mean*	0.65	0.84	0.65	0.64	0.85	0.58	0.52	0.51

To maintain logarithmic complexity during inverse rendering, we use at each gradient iteration a two-stage backpropagation scheme:

1. At the start of the iteration, after the tree updates, we *detach* the node attributes from the corresponding leaf node attributes. During inverse rendering, each adjoint query backpropagates gradients to *only* the nodes visited by the corresponding primal query. Each node locally accumulates rendering gradients.
2. After inverse rendering concludes, we perform a *single* full tree traversal to propagate accumulated gradients from all nodes to the leaf nodes. The resulting gradients are used to update point cloud attributes at the end of the iteration.

This two-stage process requires storing at each tree node additional set *mutable* gradient attributes \bar{db}_t (one for each of the geometry and appearance attributes), to accumulate backpropagated gradients. Overall, if we perform a total of Q queries during each gradient iteration, naive backpropagation would result in complexity $O(Q \cdot M)$. Our two-stage backpropagation has instead complexity $O(Q \log M + M \log M)$: $O(Q \log M)$ for the adjoint queries in the first stage; and $O(M \log M)$ for the full traversal at the second stage (updating M leaf nodes, each with $O(\log M)$ ancestors).

Algorithm 1 (lines 22–28) summarizes the accelerated adjoint queries in the first stage, which we implement exactly analogously to primal queries. The second stage is likewise easy to implement automatic differentiation. We provide details in Appendix C.

6.3 Acceleration details

We conclude this section by highlighting some salient details regarding our Barnes-Hut acceleration scheme.

Tree construction and update. We construct the octree data structure after structure from motion, using its output point cloud. The node hierarchy in the tree depends on only the point cloud locations

Table 2. Chamfer distances on the BlendedMVS dataset (Surf. = Surfels).

	5 m			10 m			1 h		
	NeuS2	Surf.	ours	NeuS2	Surf.	ours	NeuS2	Surf.	ours
bask.	1.92	1.49	2.48	1.71	1.51	1.43	1.81	1.35	1.22
bear	2.23	1.76	1.38	2.47	1.82	1.33	2.19	2.07	1.19
brea.	1.83	1.37	2.64	1.90	1.64	1.69	1.66	1.33	1.65
cam.	1.83	1.74	1.62	1.77	1.92	1.61	1.88	2.11	1.58
cloc.	2.64	3.48	1.99	2.51	3.27	1.86	2.57	3.42	1.88
cow	1.53	4.01	2.12	1.46	4.61	1.62	1.49	4.78	1.32
dog	2.41	3.10	2.78	2.34	2.78	2.41	2.31	2.86	2.03
doll	1.63	1.91	3.79	1.55	1.93	3.15	1.55	1.99	1.86
drag.	2.00	3.31	1.98	1.90	3.00	1.69	1.92	2.83	1.43
dur.	3.39	3.24	2.11	3.39	3.30	2.13	3.39	3.49	2.08
foun.	2.26	2.60	2.20	2.18	2.88	2.09	2.19	3.85	1.94
gun.	0.94	1.36	1.11	1.02	1.47	1.05	1.18	2.26	0.99
hous.	2.28	1.91	2.51	2.25	2.06	2.38	2.26	1.98	1.82
jade	3.38	3.70	3.95	3.32	3.67	3.46	3.31	4.11	3.30
man	1.53	2.48	2.57	1.49	2.61	1.63	1.48	3.19	1.44
mon.	1.30	1.32	1.31	1.18	1.52	1.24	1.17	2.00	1.21
scul.	1.44	2.98	1.42	1.38	3.18	1.32	1.32	3.65	1.22
ston.	1.74	1.86	2.16	1.78	1.87	1.82	1.64	2.01	1.50
mean	2.02	2.42	2.23	1.98	2.50	1.88	1.96	2.74	1.65

p_m . Thus, as we choose not to update these locations during inverse rendering (Section 5.3), we only need to create the tree structure *once* rather than after each gradient operation. Choosing otherwise would introduce significant computational overhead.

At each iteration, we must twice update only the Dirichlet data attributes \bar{b}_t of the tree nodes: once at the start of the iteration, to account for updated point cloud attributes after a gradient step; and once at its end, during the second-stage of backpropagation. Both updates are efficient, introducing an overhead analogous to about a couple additional ray casting queries during rendering.

Queries for multiple features. Primal and inverse rendering with Equation (1) requires at every sampled ray location x dipole sum queries for all Dirichlet data attributes b stored in the point cloud—namely, the geometry attribute f and the appearance attributes ℓ^k , $k = 1, \dots, K$. As the tree traversal pattern depends on only x , we can return all these attributes with a single primal query and tree traversal—and likewise for adjoint queries. Further accelerating performance by using packet queries [Wald et al. 2014] for multiple points x is an exciting future direction.

7 EXPERIMENTAL EVALUATION

We evaluate our technique against COLMAP initialization, as well as state-of-the-art techniques for multi-view surface reconstruction: Surfels [Dai et al. 2024], which combines a point-based representation with rasterization; NeuS2 [Wang et al. 2023] and Neuralangelo [Li et al. 2023], which both combine a hybrid hashgrid-neural representation with ray tracing. All three techniques aim for high-quality surface outputs, but place different emphasis on computational efficiency (Surfels, NeuS2) versus reconstruction fidelity (Neuralangelo). In this section, we summarize our experiments and findings.



Figure 6. Our method produces higher-quality reconstructions than Neuralangelo on DTU scenes at 1/18 of the running time. Neuralangelo fails catastrophically on BlendedMVS scenes when there are few views available (second row).

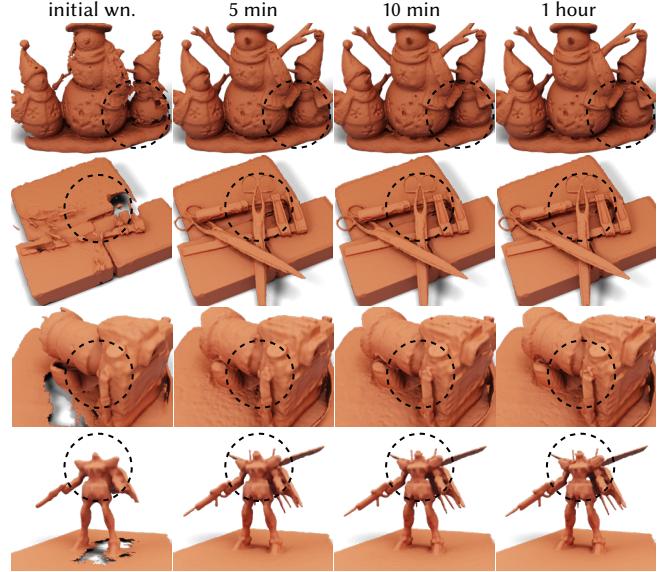


Figure 7. Training progression of our method on scenes from the DTU (top two rows) and BlendedMVS (bottom two rows) datasets. The leftmost column shows the mesh extracted from the initial winding number field, and remaining columns show the mesh after 5 minutes, 10 minutes, and 1 hour of training, respectively. Our method provides significant improvements over the initial mesh within 5 minutes of training and continues to refine details with additional training time.

The extensive supplement includes an HTML document with interactive visualizations of all experiments.

7.1 Implementation details

We built our codebase in PyTorch [Paszke et al. 2019] based on the NeuS codebase [Wang et al. 2021a]. We implemented custom C++ and CUDA extensions for building the octree and performing fast primal and adjoint dipole sum queries, following the original C++ implementation of fast winding numbers [Barill et al. 2018] in libigl [Jacobson et al. 2018]. We provide our codebase in the supplement.

Radiance field details. We design the MLP in Equation (21) similarly to the appearance network of NeuS [Wang et al. 2021b]—4 hidden layers, each with 256 neurons and ReLU activations. We use $K = 32$ appearance attributes t^k , and encode viewing directions with real spherical harmonics up to degree 3. We apply weight normalization [Salimans and Kingma 2016] for stable training.

Ray sampling. We sample the ray locations in Equation (2) using the procedure by Miller et al. [2024]: For each camera ray, we identify the first zero-crossing of the geometry field in Equation (19) by densely placing 1,024 samples along the ray between the near and far limits. If a zero-crossing is found, we place 24 sparse samples between the near limit and the first crossing, 64 dense samples around the first crossing, and 8 sparse samples between the first crossing and the far limit. Otherwise, we place 96 samples uniformly.

We note that an advantage of representing the geometry field as a dipole sum is that we can compute the first zero-crossing along a ray efficiently (with logarithmic complexity in terms of number of dipole sum queries) using *Harnack tracing* [Gillespie et al. 2024, Section 4.3]—a technique analogous to sphere tracing for signed distance functions [Hart 1996], except designed for (near-)harmonic functions. In practice, because our fast primal queries contribute only minor overhead to the overall inverse rendering runtime, we found that Harnack tracing provided negligible acceleration compared to the standard ray marching procedure we described above.

Training. We optimize our point-based representation using Adam [Kingma and Ba 2015] with a batch size 4,096 rays. We use a learning rate of 1×10^{-2} for point cloud attributes, and 3×10^{-3} for the radiance field MLP. We use a linear warmup schedule for the first 200 iterations, and a cosine decay schedule for the remaining iterations. We use different numbers of iterations depending on the experiment—training for 2k, 5k, 20k iterations takes 5 min, 10 min, and 1 hour (resp.) on a single NVIDIA RTX 4090 GPU.

Point growing. As we explain in Section 5, the use of non-unit geometry attributes for the geometry field of Equation (19) helps fill point cloud holes due to textureless regions. In practice, we found it useful to *also* grow a small number of additional points during inverse rendering. We perform point growing every 500 iterations, by sampling random rays and computing their first intersection with the geometry field. At each intersection, we add a point if the distance to the closest point in the point cloud is greater than a threshold. For each new point, we initialize its attributes by averaging those of its neighbors, compute a normal using PCA [Hoppe et al. 1992], then recompute the area weights of the entire point cloud. We found that we need to grow only about 5% additional points relative to the original point cloud from structure from motion.

Mesh extraction. For quantitative evaluation, we produce meshes by extracting the zero-level set of the geometry field using marching cubes [Lorensen and Cline 1987]. We use a grid resolution of 512^3 for DTU and 1024^3 for BlendedMVS. We make two observations: 1. Barill et al. [2018] suggest using bisection root-finding to extract

meshes from the winding number field, to avoid artifacts due to the singular Poisson kernels. By contrast, our use of regularized Poisson kernels allow us to directly extract meshes using marching cubes. 2. We need to extract meshes *only* for quantitative evaluation with other techniques. Our geometry field can be directly and efficiently ray traced using ray marching with fast primal queries.

7.2 Comparison to prior work

We evaluate our method against NeuS2 [Wang et al. 2023], Surfels [Dai et al. 2024], and Neuralangelo [Li et al. 2023], on the DTU [Aanæs et al. 2016] and BlendedMVS [Yao et al. 2020] datasets. We summarize our results at 5 minutes, 10 minutes, and 1 hour of training in Tables 1 and 2, and provide qualitative results in Figure 5 as well as in interactive visualizations in the supplement. For the DTU dataset, we report chamfer distance averages over all scans, as well as excluding scans 63, 83, and 105: We found that the ground truth point clouds for these scans are highly inaccurate, and thus often favor qualitatively worse reconstructions, an issue also reported by Li et al. [2023, Appendix D and personal communication].

We train our method, NeuS2, and Surfels without mask supervision and evaluate their extracted meshes using the official DTU evaluation script. For Neuralangelo, we directly report DTU evaluation results from their paper. We do not report results for Neuralangelo on BlendedMVS as on multiple scenes it failed to produce meaningful reconstructions (e.g., second row of Figure 6).

We observe our method consistently outperforms Surfels at all runtimes in both datasets. Compared to NeuS2, our method performs worse at 5 minutes on the BlendedMVS dataset, but better at all other runtimes on both DTU and BlendedMVS. In all cases, the quantitative improvements also translate to visual qualitative improvements on the extracted meshes (Figure 5 and supplement).

We noticed that on both datasets, our method consistently improves reconstruction quality with additional training time. By contrast, NeuS2 and Surfels either stagnate or even degrade performance with additional training time. Moreover, our method at 1 hour of training also outperforms Neuralangelo at 18 hours of training on the DTU dataset. Lastly, our method in all cases improves upon the mesh extracted from the original structure-from-motion point cloud using the regularized winding number field. We visualize the training progression of our method in Figure 7.

7.3 Rendering with shadow rays

Compared to other fast point-based methods such as Surfels [Dai et al. 2024], our method uses ray tracing instead of rasterization. Ray tracing provides greater flexibility than rasterization in terms of rendering algorithms and light transport effects it can be used for. A salient example in the context of 3D reconstruction is rendering direct illumination via shadow rays [Ling et al. 2023] when reconstructing scenes with known illumination—doing so is not possible with rasterization methods. We use a synthetic example to demonstrate that this additional flexibility translates to both higher-quality extracted meshes and more accurate novel view synthesis.

Experiment setup. We re-render the LEGO scene from the NeRF Realistic Synthetic dataset [Mildenhall et al. 2021] with fully Lambertian materials and illumination from two point light sources. We

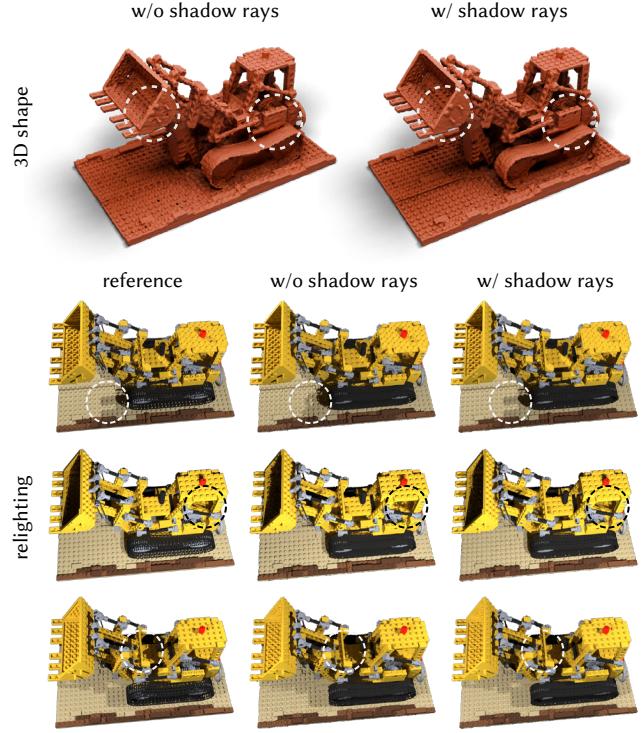


Figure 8. Comparison of extracted meshes and rendered images from training with and without shadow rays on the NeRF Realistic Synthetic LEGO scene. Optimizing with shadow rays results in extracted meshes with fewer artifacts and finer details as well as images rendered under novel lighting with more accurate shadows.

render 200 images from random viewpoints and random point light positions that vary from image to image. We process these images with COLMAP to extract an initial point cloud, normals, and camera poses—we use ground truth point light positions for each view.

Inverse rendering. We optimize this initialization using our method with and without shadow rays. Without shadow rays, our method works exactly as before, using the radiance field representation of Equation (21) to model global (direct and indirect) illumination.

With shadow rays, we augment Equation (21) to include direct illumination terms for the two point light sources:

$$\begin{aligned} L_{\text{sh.rays}}(x, \omega) &\equiv \sum_{i=1,2} L_d^i(x, \omega) \\ &+ \text{MLP}\left(x, \omega, n_{\text{imp}}(x), \ell^1(x), \dots, \ell^K(x)\right), \end{aligned} \quad (27)$$

where for each light source:

$$L_d^i(x, \omega) \equiv \alpha(x) T(x, l_i) \frac{n_{\text{imp}}(x) \cdot x l_i}{\|x - l_i\|^2}. \quad (28)$$

Here, l_i is the position of the i -th light source, α is the albedo at x , and $T(x, l_i)$ is the exponential transmittance between x and l_i . We compute albedo as an additional output of the MLP, and transmittance using quadrature (Equation (2)) and fast queries.

Results. We compare in Figure 8 extracted meshes and images rendered under novel lighting, after optimizing with and without shadow rays. We observe that optimizing with shadow rays results in extracted meshes with fewer artifacts and finer details. Additionally, the corresponding images have accurate shadows, compared to clearly implausible shadows otherwise. These results demonstrate that our method benefits from the generality of ray tracing, while achieving efficiency comparable to rasterization.

8 LIMITATIONS AND DISCUSSION

We introduced the dipole sum, a point-based representation for inverse rendering of 3D geometry. This representations allows modeling, ray tracing, and optimizing both implicit geometry and radiance fields using point cloud attributes. Coupled with Barnes-Hut acceleration, dipole sums enable multi-view 3D reconstruction at speeds comparable to rasterization techniques, while maintaining the generality afforded by ray tracing. Starting from structure-from-motion initialization, dipole sums additionally produce surface reconstructions of comparable or better quality than neural representations, while escaping overfitting issues or computational overheads those encounter. We conclude with a discussion of some limitations of our work, and the future research directions they suggest.

Dealing with specular appearance. Both our work and prior work studying representations other than neural (e.g., Surfels [Dai et al. 2024] for point-based, and Voxsurf [Wu et al. 2023] for grid-based) report difficulties producing accurate surface reconstructions in areas of strong specular appearance. Our technique and Voxsurf alleviate the issue using shallow MLPs to predict appearance from interpolated features, which inevitably introduces a computational overhead. Surfels instead rely on data-driven priors, which in turn introduces reliance on supervised training and generalizability issues. Previous work on neural representations showed improved handling of specular appearance through the use of “roughness” [Verbin et al. 2022] or “anisotropy” [Miller et al. 2024] features that are combined with spherical-harmonic radiance representations during ray tracing. As our technique also uses ray tracing, it could adapt this approach by incorporating such features as point attributes rather than neural network outputs.

Global illumination and surface rendering. Our dipole sum representation is designed for efficient ray tracing. Thus, it is compatible, in principle, with more general (primal and differentiable) rendering algorithms. We have demonstrated this compatibility only in a restricted fashion, through a combination of dipole sums with shadow rays for direct illumination estimation (Section 7). In the future, it would be interesting to investigate combinations of dipole sums with other direct illumination algorithms [Bitterli et al. 2020], or global illumination algorithms such as path tracing and bidirectional path tracing [Pharr et al. 2023]. Additionally, we focused on volume rendering, but our dipole sum representation is also compatible with surface rendering formulations, which lead to improved surface reconstruction [Cai et al. 2022; Luan et al. 2021] at the cost of needing to account for visibility discontinuities in the represented implicit surface [Vicini et al. 2022; Bangaru et al. 2022].

Applications beyond 3D reconstruction. We evaluated our point-based representation only in the narrow context of inverse rendering for 3D reconstruction. However, representations such as our dipole sum—comprising a tailored *combination* of point cloud attributes, an interpolation kernel, and fast summation queries—can be useful more broadly for a variety of graphics and vision tasks, analogously to multiresolution hashgrids [Müller et al. 2022]. Broader adoption could be facilitated by investigation of alternative fast summation techniques [Beatson et al. 1997], and data-driven optimization of interpolation kernels [Chen et al. 2023; Ryan et al. 2022].

REFERENCES

- Henrik Aanæs, Rasmus Ramsøb Jensen, George Vogiatzis, Engin Tola, and Anders Bjørholm Dahl. 2016. Large-Scale Data for Multiple-View Stereopsis. *International Journal of Computer Vision* (2016), 1–16.
- Sameer Agarwal, Yasutaka Furukawa, Noah Snavely, Ian Simon, Brian Curless, Steven M Seitz, and Richard Szeliski. 2011. Building rome in a day. *Commun. ACM* 54, 10 (2011), 105–112.
- Sai Praveen Bangaru, Michael Gharbi, Fujun Luan, Tzu-Mao Li, Kalyan Sunkavalli, Milos Hasan, Sai Bi, Zexiang Xu, Gilbert Bernstein, and Fredo Durand. 2022. Differentiable rendering of neural sdf’s through reparameterization. In *SIGGRAPH Asia 2022 Conference Papers*. 1–9.
- Gavin Barill, Neil G Dickson, Ryan Schmidt, David IW Levin, and Alec Jacobson. 2018. Fast winding numbers for soups and clouds. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–12.
- Josh Barnes and Piet Hut. 1986. A hierarchical O(N log N) force-calculation algorithm. *nature* 324, 6096 (1986), 446–449.
- J Thomas Beale, Wenzun Ying, and Jason R Wilson. 2016. A simple method for computing singular or nearly singular integrals on closed surfaces. *Communications in Computational Physics* 20, 3 (2016), 733–753.
- Rick Beatson, Leslie Greengard, et al. 1997. A short course on fast multipole methods. *Wavelets, multilevel methods and elliptic PDEs* 1 (1997), 1–37.
- Alexander Belyaev, Pierre-Alain Fayolle, and Alexander Pasko. 2013. Signed L^p-distance fields. *Computer-Aided Design* 45, 2 (2013), 523–528.
- Matthew Berger, Andrea Tagliasacchi, Lee M Seversky, Pierre Alliez, Joshua A Levine, Andrei Sharf, and Claudio T Silva. 2014. State of the art in surface reconstruction from point clouds. In *35th Annual Conference of the European Association for Computer Graphics, Eurographics 2014-State of the Art Reports*. The Eurographics Association.
- Sai Bi, Zexiang Xu, Pratul Srinivasan, Ben Mildenhall, Kalyan Sunkavalli, Miloš Hašan, Yannick Hold-Geoffroy, David Kriegman, and Ravi Ramamoorthi. 2020a. Neural reflectance fields for appearance acquisition. *arXiv preprint arXiv:2008.03824* (2020).
- Sai Bi, Zexiang Xu, Kalyan Sunkavalli, Miloš Hašan, Yannick Hold-Geoffroy, David Kriegman, and Ravi Ramamoorthi. 2020b. Deep reflectance volumes: Reliable reconstructions from multi-view photometric images. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III*. Springer, 294–311.
- Benedikt Bitterli, Chris Wyman, Matt Pharr, Peter Shirley, Aaron Lefohn, and Wojciech Jarosz. 2020. Spatiotemporal reservoir resampling for real-time ray tracing with dynamic direct lighting. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 148–1.
- Guangyan Cai, Kai Yan, Zhao Dong, Ioannis Gkioulekas, and Shuang Zhao. 2022. Physics-based inverse rendering using combined implicit and explicit geometries. In *Computer Graphics Forum*, Vol. 41. Wiley Online Library, 129–138.
- Jonathan C Carr, Richard K Beatson, Jon B Cherrie, Tim J Mitchell, W Richard Fright, Bruce C McCallum, and Tim R Evans. 2001. Reconstruction and representation of 3D objects with radial basis functions. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. 67–76.
- Zhang Chen, Zhong Li, Liangchen Song, Lele Chen, Jingyi Yu, Junsong Yuan, and Yi Xu. 2023. Neurbf: A neural fields representation with adaptive radial basis functions. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 4182–4194.
- Ricardo Cortez. 2001. The method of regularized Stokeslets. *SIAM Journal on Scientific Computing* 23, 4 (2001), 1204–1225.
- Ricardo Cortez, Lisa Fauci, and Alexei Medovikov. 2005. The method of regularized Stokeslets in three dimensions: analysis, validation, and application to helical swimming. *Physics of Fluids* 17, 3 (2005).
- Angela Dai, Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Christian Theobalt. 2017. Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface reintegration. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 1.
- Pinxuan Dai, Jiamin Xu, Wenxiang Xie, Xinguo Liu, Huamin Wang, and Weiwei Xu. 2024. High-quality Surface Reconstruction using Gaussian Surfels. In *SIGGRAPH 2024 Conference Papers*. Association for Computing Machinery. <https://doi.org/10.1145/3641519.3657441>

- Kangle Deng, Andrew Liu, Jun-Yan Zhu, and Deva Ramanan. 2022. Depth-supervised nerf: Fewer views and faster training for free. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 12882–12891.
- Nicole Feng, Mark Gillespie, and Keenan Crane. 2023. Winding Numbers on Discrete Surfaces. *ACM Transactions on Graphics (TOG)* (2023).
- Michael S Floater, Géza Kós, and Martin Reimers. 2005. Mean value coordinates in 3D. *Computer Aided Geometric Design* 22, 7 (2005), 623–631.
- Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinzhong Chen, Benjamin Recht, and Angjoo Kanazawa. 2022. Plenoxels: Radiance fields without neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 5501–5510.
- Qiancheng Fu, Qingshan Xu, Yew-Soon Ong, and Wenbing Tao. 2022. Geo-Neus: Geometry-Consistent Neural Implicit Surfaces Learning for Multi-view Reconstruction. In *Advances in Neural Information Processing Systems*, Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (Eds.). <https://openreview.net/forum?id=JvIFpZOjLF4>
- Simon Fuhrmann and Michael Goesele. 2014. Floating scale surface reconstruction. *ACM Transactions on Graphics (ToG)* 33, 4 (2014), 1–11.
- Mark Gillespie, Denise Yang, Mario Botsic, and Keenan Crane. 2024. Ray Tracing Harmonic Functions. *ACM Trans. Graph.* 43, 4 (2024).
- Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 315–323.
- Antoine Guédon and Vincent Lepetit. 2023. SuGaR: Surface-Aligned Gaussian Splatting for Efficient 3D Mesh Reconstruction and High-Quality Mesh Rendering. *arXiv preprint arXiv:2311.12775* (2023).
- Jun Han and Claudio Moraga. 1995. The influence of the sigmoid function parameters on the speed of backpropagation learning. In *International workshop on artificial neural networks*. Springer, 195–201.
- John C Hart. 1996. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer* 12, 10 (1996), 527–545.
- Richard Hartley and Andrew Zisserman. 2003. *Multiple view geometry in computer vision*. Cambridge university press.
- Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. 1992. Surface reconstruction from unorganized points. In *Proceedings of the 19th annual conference on computer graphics and interactive techniques*. 71–78.
- Kai Hormann and N Sukumar. 2017. *Generalized barycentric coordinates in computer graphics and computational mechanics*. CRC press.
- Alec Jacobson, Ladislav Kavan, and Olga Sorkine-Hornung. 2013. Robust inside-outside segmentation using generalized winding numbers. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 1–12.
- Alec Jacobson, Daniele Panozzo, et al. 2018. libigl: A simple C++ geometry processing library. <https://libigl.github.io/>.
- Tao Ju, Scott Schaefer, and Joe Warren. 2005. Mean value coordinates for closed triangular meshes. *ACM Trans. Graph.* 24, 3 (jul 2005), 561–566. <https://doi.org/10.1145/1073204.1073229>
- Animesh Karnewar, Tobias Ritschel, Oliver Wang, and Niloy Mitra. 2022. Relu fields: The little non-linearity that could. In *ACM SIGGRAPH 2022 Conference Proceedings*. 1–9.
- Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. 2006. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, Vol. 7. 0.
- Michael Kazhdan and Hugues Hoppe. 2013. Screened poisson surface reconstruction. *ACM Transactions on Graphics (ToG)* 32, 3 (2013), 1–13.
- Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 2023. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics* 42, 4 (2023), 1–14.
- Mijeong Kim, Seounguk Seo, and Bohyung Han. 2022. Infonerf: Ray entropy minimization for few-shot neural volume rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 12912–12921.
- Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. <http://arxiv.org/abs/1412.6980>
- Pavel A Krutitskii. 2001. The jump problem for the Laplace equation. *Applied Mathematics Letters* 14, 3 (2001), 353–358.
- Fabian Langguth, Kalyan Sunkavalli, Sunil Hadap, and Michael Goesele. 2016. Shading-aware multi-view stereo. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part III* 14. Springer, 469–485.
- Zhaoshuo Li, Thomas Müller, Alex Evans, Russell H Taylor, Mathias Unterath, Ming-Yu Liu, and Chen-Hsuan Lin. 2023. Neuralangelo: High-Fidelity Neural Surface Reconstruction. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Jingwang Ling, Zhibo Wang, and Feng Xu. 2023. Shadowneus: Neural sdf reconstruction by shadow ray supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 175–185.
- Hsueh-Ti Derek Liu, Francis Williams, Alec Jacobson, Sanja Fidler, and Or Litany. 2022. Learning smooth neural functions via lipschitz regularization. In *ACM SIGGRAPH 2022 Conference Proceedings*. 1–13.
- Matthew M Loper and Michael J Black. 2014. OpenDR: An approximate differentiable renderer. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6–12, 2014, Proceedings, Part VII* 13. Springer, 154–169.
- William E. Lorensen and Harvey E. Cline. 1987. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '87)*. 163–169. <https://doi.org/10.1145/37401.37422>
- Fujun Luan, Shuang Zhao, Kavita Bala, and Zhao Dong. 2021. Unified shape and svbrdf recovery using differentiable monte carlo rendering. In *Computer Graphics Forum*, Vol. 40. Wiley Online Library, 101–113.
- Kanti V Mardia and Peter E Jupp. 2009. *Directional statistics*. John Wiley & Sons.
- Stephen Robert Marschner. 1998. *Inverse rendering for computer graphics*. Cornell University.
- Nelson Max. 1995. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics* 1, 2 (1995), 99–108.
- Donald Meagher. 1982. Geometric modeling using octree encoding. *Computer graphics and image processing* 19, 2 (1982), 129–147.
- Ben Mildenhall, Pratul Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. 2021. NeRF: Representing scenes as neural radiance fields for view synthesis. *Commun. ACM* 65, 1 (2021), 99–106.
- Bailey Miller, Hanyu Chen, Alice Lai, and Ioannis Gkioulekas. 2024. Objects as volumes: A stochastic geometry view of opaque solids. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant neural graphics primitives with a multiresolution hash encoding. *ACM transactions on graphics (TOG)* 41, 4 (2022), 1–15.
- Ken Museth, Jeff Lait, John Johanson, Jeff Budsberg, Ron Henderson, Mihai Alden, Peter Cucka, David Hill, and Andrew Pearce. 2013. OpenVDB: an open-source data structure and toolkit for high-resolution volumes. In *Acm siggraph 2013 courses*. 1–1.
- Merlin Nimier-David, Sébastien Speirer, Benoît Ruiz, and Wenzel Jakob. 2020. Radiative backpropagation: An adjoint method for lightning-fast differentiable rendering. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 146–1.
- Michael Oechslé, Songyou Peng, and Andreas Geiger. 2021. Unisurf: Unifying neural implicit surfaces and radiance fields for multi-view reconstruction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 5589–5599.
- Onur Özyeşil, Vladislav Voroninski, Ronen Basri, and Amit Singer. 2017. A survey of structure from motion*. *Acta Numerica* 26 (2017), 305–364.
- Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch. (2017).
- Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).
- Songyou Peng, Chiyu Jiang, Yiyi Liao, Michael Niemeyer, Marc Pollefeys, and Andreas Geiger. 2021. Shape as points: A differentiable poisson solver. *Advances in Neural Information Processing Systems* 34 (2021), 13032–13044.
- Matt Pharr, Wenzel Jakob, and Greg Humphreys. 2023. *Physically based rendering: From theory to implementation*. MIT Press.
- John P Ryan, Sebastian E Ament, Carla P Gomes, and Anil Damle. 2022. The fast kernel transform. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 11669–11690.
- Tim Salimans and Durk P Kingma. 2016. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *Advances in neural information processing systems* 29 (2016).
- Johannes Lutz Schönberger and Jan-Michael Frahm. 2016. Structure-from-Motion Revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Noah Snavely, Steven M Seitz, and Richard Szeliski. 2006. Photo tourism: exploring photo collections in 3D. In *ACM siggraph 2006 papers*. 835–846.
- Noah Snavely, Steven M Seitz, and Richard Szeliski. 2008. Modeling the world from internet photo collections. *International journal of computer vision* 80 (2008), 189–210.
- Jos Stam. 2020. Computing Light Transport Gradients using the Adjoint Method. *arXiv preprint arXiv:2006.15059* (2020).
- Ayush Tewari, Justus Thies, Ben Mildenhall, Pratul Srinivasan, Edgar Treitschke, Wang Yifan, Christoph Lassner, Vincent Sitzmann, Ricardo Martin-Brualla, Stephen Lombardi, et al. 2022. Advances in neural rendering. In *Computer Graphics Forum*, Vol. 41. Wiley Online Library, 703–735.
- Carlo Tomasi and Takeo Kanade. 1990. Shape and motion without depth. In *Proceedings of the DARPA Image Understanding Workshop*. 258.
- Shimon Ullman. 1979. The interpretation of structure from motion. *Proceedings of the Royal Society of London. Series B. Biological Sciences* 203, 1153 (1979), 405–426.

- Dor Verbin, Peter Hedman, Ben Mildenhall, Todd Zickler, Jonathan T Barron, and Pratul P Srinivasan. 2022. Ref-NeRF: Structured view-dependent appearance for neural radiance fields. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 5481–5490.
- Delio Vicini, Sébastien Speierer, and Wenzel Jakob. 2021. Path replay backpropagation: Differentiating light paths using constant memory and linear time. *ACM Transactions on Graphics (TOG)* 40, 4 (2021), 1–14.
- Delio Vicini, Sébastien Speierer, and Wenzel Jakob. 2022. Differentiable signed distance function rendering. *ACM Transactions on Graphics (TOG)* 41, 4 (2022), 1–18.
- Ingo Wald, Sven Woop, Carsten Benthin, Gregory S Johnson, and Manfred Ernst. 2014. Embree: a kernel framework for efficient CPU ray tracing. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 1–8.
- Peng Wang, Lingjiu Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. 2021a. NeuS codebase. <https://github.com/Totoro97/NeuS>.
- Peng Wang, Lingjiu Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. 2021b. NeuS: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. *Advances in Neural Information Processing Systems* 34 (2021).
- Yiming Wang, Qin Han, Marc Habermann, Kostas Daniilidis, Christian Theobalt, and Lingjie Liu. 2023. NeuS2: Fast Learning of Neural Implicit Surfaces for Multi-view Reconstruction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.
- Chenglei Wu, Bennett Wilburn, Yasuyuki Matsushita, and Christian Theobalt. 2011. High-quality shape from multi-view stereo and shading under general illumination. In *CVPR 2011*. IEEE, 969–976.
- Tong Wu, Jiaqi Wang, Xingang Pan, Xudong Xu, Christian Theobalt, Ziwei Liu, and Dahua Lin. 2023. Voxurf: Voxel-based Efficient and Accurate Neural Surface Reconstruction. In *International Conference on Learning Representations (ICLR)*.
- Qiangeng Xu, Zexiang Xu, Julien Philip, Sai Bi, Zhixin Shu, Kalyan Sunkavalli, and Ulrich Neumann. 2022. Point-nerf: Point-based neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 5438–5448.
- Rui Xu, Zhiyang Dou, Ningna Wang, Shiqing Xin, Shuangmin Chen, Mingyan Jiang, Xiaohu Guo, Wenping Wang, and Changhe Tu. 2023. Globally consistent normal orientation for point clouds by regularizing the winding-number field. *ACM Transactions on Graphics (TOG)* 42, 4 (2023), 1–15.
- Yao Yao, Zixin Luo, Shiwei Li, Jingyang Zhang, Yufan Ren, Lei Zhou, Tian Fang, and Long Quan. 2020. BlendedMVS: A Large-scale Dataset for Generalized Multi-view Stereo Networks. *Computer Vision and Pattern Recognition (CVPR)* (2020).
- Lior Yariv, Jiatao Gu, Yoni Kasten, and Yaron Lipman. 2021. Volume rendering of neural implicit surfaces. *Advances in Neural Information Processing Systems* 34 (2021), 4805–4815.
- Lyubomir G Zagorchev and Arthur Ardeshir Goshtasby. 2011. A curvature-adaptive implicit surface reconstruction for irregularly spaced points. *IEEE Transactions on Visualization and Computer Graphics* 18, 9 (2011), 1460–1473.
- Michael Zollhöfer, Angela Dai, Matthias Ihmnnann, Chenglei Wu, Marc Stamminger, Christian Theobalt, and Matthias Nießner. 2015. Shading-based refinement on volumetric signed distance functions. *ACM Transactions on Graphics (ToG)* 34, 4 (2015), 1–14.
- Matthias Zwicker, Hanspeter Pfister, Jeroen Van Baar, and Markus Gross. 2002. EWA splatting. *IEEE Transactions on Visualization and Computer Graphics* 8, 3 (2002), 223–238.

A PROBABILISTIC INTERPRETATION OF DIPOLE SUM

We assume that for each point in \mathcal{P} , its location is a 3D Gaussian random variable $\tilde{p}_m \sim N(p_m, \epsilon I)$, where I is the 3×3 identity matrix. Additionally, we assume that its normal \tilde{n}_m is a spherical random variable such that $\mathbb{E}[\tilde{n}_m | \tilde{p}_m] = f_m n_m$; that is, its conditional *mean direction* equals \tilde{p}_m and its conditional *mean resultant length* equals f_m [Mardia and Jupp 2009, Chapter 9]. Lastly, we assume that all other point cloud attributes are deterministic. Then, the expected value of w_{pc} in Equation (7) becomes:

$$\mathbb{E}_{\{\tilde{p}_m, \tilde{n}_m\}_{m=1}^M} [w_{pc}(x)] = \sum_{m=1}^M A_m \mathbb{E}_{\tilde{p}_m, \tilde{n}_m} [P(x, \tilde{p}_m)] \quad (29)$$

$$= \sum_{m=1}^M A_m \mathbb{E}_{\tilde{p}_m} [\mathbb{E}_{\tilde{n}_m} [P(x, \tilde{p}_m) | \tilde{p}_m]] \quad (30)$$

$$\begin{aligned} &= \sum_{m=1}^M A_m \\ &\quad \cdot \mathbb{E}_{\tilde{p}_m} [\mathbb{E}_{\tilde{n}_m} [\tilde{n}_m \nabla G(x, \tilde{p}_m) | \tilde{p}_m]] \quad (31) \\ &= \sum_{m=1}^M A_m \mathbb{E}_{\tilde{p}_m} [f_m n_m \nabla G(x, \tilde{p}_m)] \quad (32) \\ &= \sum_{m=1}^M A_m f_m n_m \nabla \mathbb{E}_{\tilde{p}_m} [G(x, \tilde{p}_m)] \quad (33) \\ &= \sum_{m=1}^M A_m f_m n_m \nabla G_\epsilon(x, \tilde{p}_m) \quad (34) \\ &= \sum_{m=1}^M A_m f_m P_\epsilon(x, \tilde{p}_m) \quad (35) \\ &= u_{pc, \epsilon}^f. \quad (36) \end{aligned}$$

In this sequence: (29) follows from linearity of expectation; (30) follows from the law of total expectation; (31) follows from the definition in (9); (32) follows from the assumptions on \tilde{n}_m ; (33) follows from the fact that differentiation and expectation commute; (35) follows from the definition in (14); and (36) follows from the definition in Equation (18). The only non-trivial step is (34). From the assumption that \tilde{p}_m is a Gaussian random variable, we have (up to a constant scale that we omit for simplicity):

$$\mathbb{E}_{\tilde{p}_m} [G(x, \tilde{p}_m)] \propto \int_{y \in \mathbb{R}^3} G(x, y) \exp\left(-\frac{\|x - y\|^2}{2\epsilon^2}\right) dy \quad (37)$$

$$\propto \int_{y \in \mathbb{R}^3} G(x, y) \phi_\epsilon(x - y) dy \quad (38)$$

$$= G_\epsilon(x, y), \quad (39)$$

where (38) follows from the definition in (13). The step (39) follows from the fact that (36) is equivalent, by the properties of the Green's function, to the solution of the BVP in the definition of Equation (12).

In this probabilistic interpretation, $f_m = 0$ is equivalent to the random normal \tilde{n}_m (conditioned on \tilde{p}_m) being *uniformly* distributed on the sphere. Then, the direction of the corresponding dipole is completely uncertain, and on expectation the dipole vanishes.

B ENTROPY LOSS

The *free-flight distribution* [Miller et al. 2024] of a ray $r_{o,v}(\tau)$,

$$p_{o,v}^{ff}(\tau) \equiv \exp\left(-\int_0^\tau \sigma(r_{o,v}(t), v) dt\right) \cdot \sigma(r_{o,v}(\tau), v), \quad (40)$$

is the probability density function for a first intersection occurring at τ . For surface-like volumes, the free-flight distribution should approximate a Dirac delta. We can encourage such behavior by penalizing the Shannon entropy of the free-flight distribution along each ray—low entropy favors peaked unimodal distributions. To do so, we use quadrature (Equation (2)) to form a discrete approximation of the free-flight distribution at the ray samples $\tau_n = \tau_0 < \dots < \tau_J = \tau_f$:

$$p_j \equiv \exp\left(-\sum_{i=1}^j \sigma_i \Delta_i\right) \cdot (1 - \exp(\sigma_j \Delta_j)). \quad (41)$$

We then compute the Shannon entropy of the vector $[p_1, \dots, p_J]$,

$$H(o, v) \equiv - \sum_{j=1}^J p_j \log p_j. \quad (42)$$

We accumulate such entropies for all rays in the loss $\mathcal{L}_{\text{entropy}}$.

C BACKPROPAGATION DETAILS

As in Section 6, throughout this section we use b as a stand-in for any of the *Dirichlet data attributes* stored in \mathcal{P} —namely, the geometry attribute f and the appearance attributes ℓ^k , $k = 1, \dots, K$. As we discuss in Section 6.3, in practice we implement the backpropagation operations in Equations (25) and (43) for all these attributes as vector operations updating all attributes in parallel.

Backpropagation to nodes. An adjoint query backpropagates a derivative $d u_{pc,\varepsilon}^b(x)$ —provided by differentiable rendering—to all tree nodes that contributed to this dipole sum during the corresponding primal query. At each such node t , the query increments the

(vector-valued) gradient attribute \bar{db}_t by an amount that follows from differentiating Equation (26):

$$\bar{A}_t \frac{x \bar{p}_t}{\|x - \bar{p}_t\|^2} \cdot S\left(\frac{\|x - \bar{p}_t\|}{\varepsilon}\right) \cdot d u_{pc,\varepsilon}^b(x). \quad (43)$$

Second-stage backpropagation to leaves. This stage backpropagates accumulated gradient attributes \bar{db}_t from all nodes to leaf nodes—i.e., nodes corresponding to individual points p_m , $m = 1, \dots, M$ in the point cloud \mathcal{P} . For each such leaf node, we denote by $\mathcal{A}(m)$ the set of its ancestor nodes in the tree. Then, by differentiating Equation (25), we can express this backpropagation stage as simply:

$$d b_m = \sum_{t \in \mathcal{A}(m)} \frac{A_m}{\bar{A}_t} n_m \cdot \bar{db}_t. \quad (44)$$

Each leaf node has $O(\log M)$ ancestors, thus total complexity of the second stage is $O(M \log M)$. In practice we implement Equation (44) as a matrix-vector multiplication that has negligible cost.