3D Reconstruction with Fast Dipole Sums

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

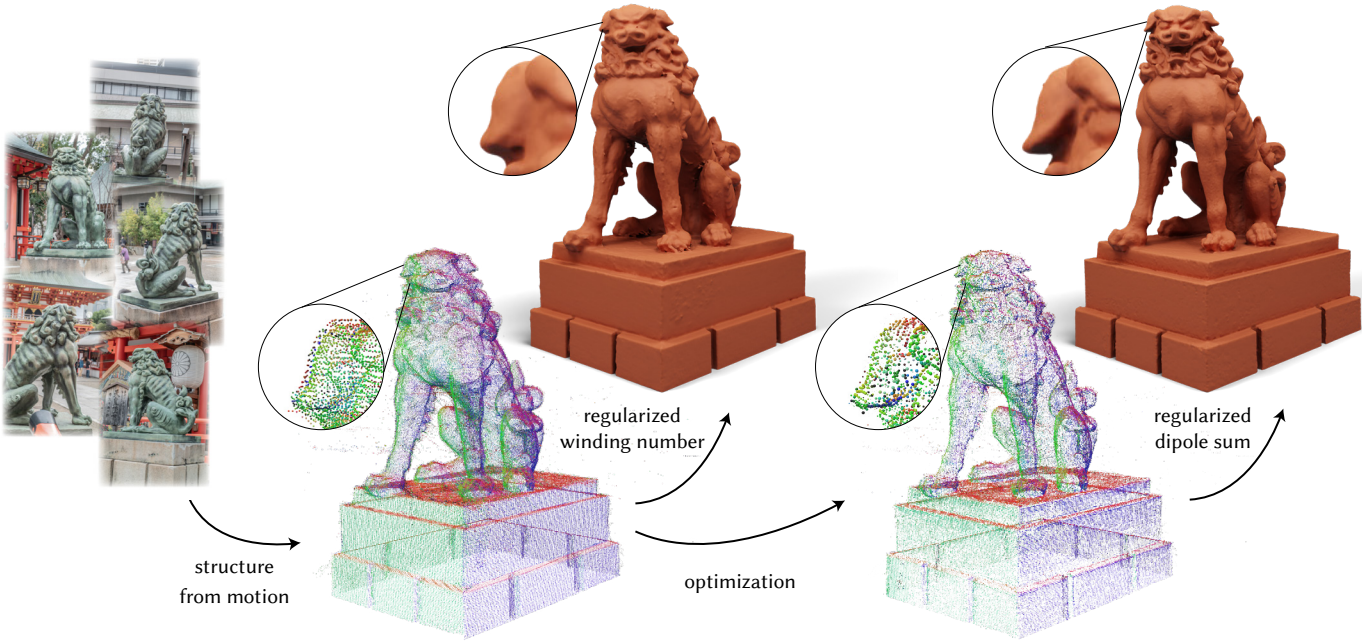

Figure 1. We introduce the regularized dipole sum, a point-based representation for multi-view 3D reconstruction. This representation can model both implicit geometry and radiance fields using per-point attributes, and supports efficient ray tracing and differentiable rendering, thus facilitating optimization using multi-view images. We initialize our regularized dipole sum representation using the dense point cloud output of a structure from motion procedure (COLMAP). 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 method for high-quality 3D reconstruction from multi-view images. Our method uses a new point-based representation, the regularized dipole sum, which generalizes the winding number to allow for interpolation of per-point attributes in point clouds with noisy or outlier points. Using regularized dipole sums, we represent implicit geometry and radiance fields as per-point attributes of a dense point cloud, which we initialize from structure from motion. We additionally derive Barnes-Hut fast summation schemes for accelerated forward and adjoint 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 our method in inverse rendering applications against state-of-the-art alternatives, based on ray tracing of neural representations or rasterization of Gaussian point-based representations. Our method significantly improves 3D reconstruction quality and robustness at equal runtimes, while also supporting more general rendering methods such as shadow rays for direct illumination.

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

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

ACM Reference Format:
Hanyu Chen, Bailey Miller, and Ioannis Gkioulekas. 2024. 3D Reconstruction with Fast Dipole Sums. *ACM Trans. Graph.* 43, 6, Article 192 (December 2024), 18 pages. https://doi.org/10.1145/3687914

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.

ACM 0730-0301/2024/12-ART192
https://doi.org/10.1145/3687914

1 INTRODUCTION

The emergence of neural rendering methods [Tewari et al. 2022] has led to the widespread adoption of a two-stage pipeline for 3D reconstruction from multi-view images: The first stage uses traditional multi-view geometry methods 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 reproduces the multi-view images—an inverse rendering process. The performance of this

pipeline depends critically on the choice of scene representation, motivating 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]) that offer different tradeoffs between expressive power and computational efficiency.

This paper introduces a new scene representation for multi-view 3D reconstruction, the *regularized dipole sum*. This representation uses tailored kernel-based interpolation of point cloud attributes, to model both the scene geometry (an implicit surface) and scene lightfield (a radiance field). Our representation 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; Huang et al. 2024b]: The use of Gaussian kernels allows these methods to perform differentiable rendering using image-space rasterization instead of ray tracing, resulting in impressive computational acceleration. At the same time, the use of rasterization precludes combinations of these representations with advanced rendering features such as direct illumination methods (e.g., shadow rays), which rasterization is incompatible with.

By contrast, we design the regularized 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, equal to the sum of Poisson kernels centered at all point cloud locations. The winding number has useful geometric regularization properties [Lin et al. 2022; Lu et al. 2018; Xu et al. 2023; Metzer et al. 2021], as a jump-harmonic function that approximates the output of robust surface reconstruction algorithms [Kazhdan et al. 2006]. It is also amenable to efficient computation using fast summation methods [Beatson et al. 1997]. Lastly, it can be directly initialized with an optional output of the first-stage structure from motion—a *dense* 3D point cloud of quality approaching that of reconstructions from state-of-the-art neural rendering methods.

The regularized dipole sum generalizes the winding number in several ways that preserve its desirable properties, while also turning it into a point-based representation suitable for inverse rendering applications. As we explain in Section 4, we use regularized kernels and general per-point attributes, to make this representation compatible 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 regularized dipole sums to represent not only the geometry, but also the radiance field of a scene. Lastly, in Section 6, we use fast summation methods to enable efficient computation *and* backpropagation, as needed for inverse rendering.

With the resulting fast dipole sums, we can use ray tracing to optimize a dense 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 dense point cloud that we visualize as a continuous surface using (our regularized generalization of) the winding number. We then use inverse rendering with fast dipole sums to optimize attributes of this point cloud (visualized in the insets), resulting in an improved reconstructed surface. In Section 7, we evaluate our approach against state-of-the-art neural rendering methods 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, greatly improving reconstruction quality and robustness at equal runtimes, while additionally supporting rendering with methods such as shadow rays. We provide interactive visualizations and an open-source implementation on the project website.[1]

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 methods attacked this problem primarily by enforcing inter-image geometric consistency, and triangulating correspondences across different images. Mature structure from motion methods [Schönberger and Frahm 2016] can robustly produce sparse point cloud reconstructions from thousands of images [Snavely et al. 2006]. Additionally, such methods provide optional shading-based refinement capabilities [Schönberger et al. 2016] to turn sparse into *dense* point clouds capturing high geometric detail. Lastly, these methods can cover 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* reconstructions, by directly utilizing and optimizing dense point clouds from structure from motion implementations (Figure 1).

Shading-based refinement and neural rendering. Geometric-only structure-from-motion methods produce point clouds that can have holes in textureless areas where there are no correspondences. They also typically cannot reproduce fine surface details, because they do not exploit shading cues that provide normal information. Shading-aware refinement methods 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.

Recent neural rendering methods 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 methods have adapted this methodological approach for surface reconstruction tasks [Yariv et al. 2021; Oechsle et al. 2021; Wang et al. 2021b]. Unfortunately,

[1] https://imaging.cs.cmu.edu/fast_dipole_sums

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 dense point clouds 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 methods 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 methods 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 point clouds available 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 method originally focused on novel-view synthesis, subsequent works [Guédon and Lepetit 2023; Dai et al. 2024; Huang et al. 2024b] have provided extensions for high-fidelity surface reconstruction. Being point-based, these methods 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 methods such as shadow rays [Ling et al. 2023] (also known as next-event estimation Pharr et al. [2023]) for rendering direct illumination from know light sources. We contribute a point-based representation that uses *ray tracing*—thus maintaining compatibility with such rendering methods—yet is as efficient as Gaussian splatting methods. Compared to these methods, and to concurrent work [Yu et al. 2024] on ray tracing 3D Gaussian representations, our method leverages and extends the advantages afforded by winding number representations [Barill et al. 2018] to produce 3D reconstructions of even higher quality (Section 7).

Point cloud surface reconstruction. Point-based geometry representations have a long history in computer graphics as methods for reconstructing continuous surfaces (either implicit or, after isosurface extraction [Lorensen and Cline 1987], explicit) from point clouds. These methods often find use as post-processing of point clouds from structure from motion methods, and thus are robust to imperfections such as noisy points, outlier points, or holes. Berger et al. [2014] and Huang et al. [2024a] provide detailed reviews. The methods by Fuhrmann and Goesele [2014] and 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 methods [Beatson et al. 1997]. Among this extensive family of methods, we build on the point-based winding number representation Jacobson et al. [2013]; Barill et al. [2018]; Spainhour et al. [2024], because of its attractive properties of geometric regularization, robustness, and efficiency—we provide a review in Section 3. We generalize winding numbers in Sections 4–6 into our regularized 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* $\sigma : \mathbb{R}^3 \times \mathcal{S}^2 \to \mathbb{R}_{\geq 0}$ representing the scene's geometry; and 2. a *radiance field* $\mathrm{L} : \mathbb{R}^3 \times \mathcal{S}^2 \to \mathbb{R}^3_{\geq 0}$ 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 $\mathrm{L}(x, \omega)$ is the incident (RGB) global illumination at x along ω.

This representation allows expressing the RGB intensity (*color*) c captured by a camera ray $r_{o,v}(\tau) \equiv o + \tau v$, $\tau \in \mathbb{R}_{\geq 0}$ with origin o and direction v using the (exponential) *volume rendering equation*:

$$c(o, v) = \int_{\tau_\mathrm{n}}^{\tau_\mathrm{f}} \exp\left(- \int_{\tau_\mathrm{n}}^{\tau} \sigma\big(r_{o,v}(t), v\big)\, \mathrm{d}t \right) \cdot \sigma\big(r_{o,v}(\tau), v\big)\, \mathrm{L}\big(r_{o,v}(\tau), -v\big)\, \mathrm{d}t, \tag{1}$$

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 $\tau_n = \tau_0 < \cdots < \tau_J = \tau_f$:

$$c(o,v) \approx \sum_{j=1}^{J} \exp\left(-\sum_{i=1}^{j} \sigma_i \Delta_i\right)\left(1 - \exp(\sigma_j \Delta_j)\right) \mathrm{L}_j \tag{2}$$

where at each sample location τ_j, $\Delta_j \equiv \tau_j - \tau_{j-1}$, $\sigma_j \equiv \sigma\big(r_{o,v}(\tau_j), v\big)$, and $\mathrm{L}_j \equiv \mathrm{L}\big(r_{o,v}(\tau_j), -v\big)$. Both approaches are differentiable, allowing propagation of gradients from rendered colors c to the attenuation coefficient σ and radiance field L. Rasterization is typically faster but also less general than ray tracing, which allows, e.g., using shadow rays to incorporate direct illumination from known light sources [Bi et al. 2020a,b; Hasselgren et al. 2022; Verbin et al. 2024].

With this representation at hand, NeRF methods reconstruct a 3D scene from multi-view images by using gradient descent methods [Kingma and Ba 2015] to optimize σ and L, so as to minimize an objective comparing real and rendered images—an *inverse rendering* methodology [Loper and Black 2014; Marschner 1998].

Surface reconstruction. To improve the performance of this 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 σ as an analytic function of a scalar field $\mathrm{F} : \mathbb{R}^3 \to \mathbb{R}$—which we term the *geometry field*—controlling an implicit surface representation of the scene geometry $\Gamma_\mathrm{F} \subset \mathbb{R}^3$, i.e., $\Gamma_\mathrm{F} \equiv \left\{x \in \mathbb{R}^3 : \mathrm{F}(x) = 0\right\}$ (with the convention that points where $\mathrm{F}(x) < 0$ are interior points).

We adopt the representation by Miller et al. [2024],[2] which first defines a *vacancy function* in terms of F:

$$\mathrm{v}(x) \equiv \Psi(s\,\mathrm{F}(x)), \tag{3}$$

where $s > 0$ is a user-defined scale factor, and $\Psi : \mathbb{R} \to [0,1]$ is a *sigmoid function* [Han and Moraga 1995; Glorot et al. 2011] equal to the cumulative distribution function of a standard normal distribution. Thus v equals $1/2$ when $\mathrm{F} = 0$ (points on the surface Γ_F), 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 :

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

Then, reconstruction uses the above inverse rendering methodology to optimize (through the differentiable Equations (3) and (4)) the geometry field F instead of the attenuation coefficient σ.

Structure-from-motion initialization. Inverse rendering requires knowledge of camera locations o and poses v for each image in a multi-view dataset, to render images with Equation (1). Inverse rendering methods typically include an initialization stage that uses structure from motion [Özyeşil et al. 2017] to estimate this camera information. The initialization stage additionally outputs a *sparse* point cloud reconstruction of the 3D scene, and Deng et al. [2022] show that the final reconstruction can improve by leveraging this output during subsequent inverse rendering optimization.

Modern structure-from-motion methods such as COLMAP [Schönberger and Frahm 2016] provide an optional refinement process [Schönberger et al. 2016] that outputs a *dense* point cloud reconstruction. Inverse rendering methods skip this refinement process during initialization, despite the fact that: 1. it produces reconstructions of quality competitive with or often even better than what inverse rendering achieves [Wang et al. 2021b] (we revisit this point in Section 7); and 2. its runtime is a lot shorter than the runtime of inverse rendering. Thus, using this refinement process during initialization and leveraging leveraging its dense point cloud output for inverse rendering could help greatly improve performance in terms of both reconstruction quality and computational efficiency.

[2]This representation is similar to NeuS [Wang et al. 2021b], except enforces reciprocity and uses a sigmoid corresponding to a Gaussian process. In our experiments we found that these changes result in significantly improved performance.

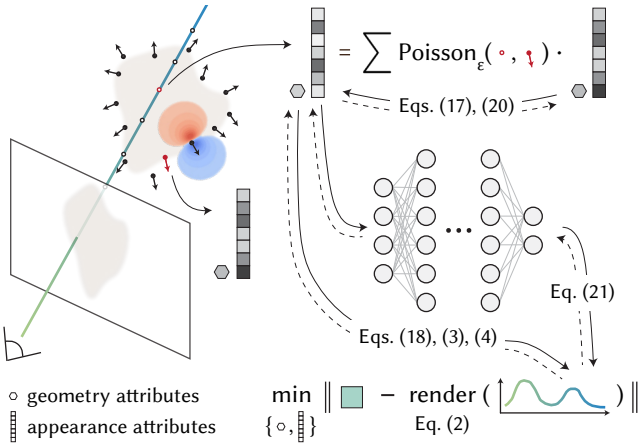

Figure 2. Overview of our method. During forward rendering (indicated by solid arrows), at each sample location along a ray, we interpolate geometry and appearance attributes from a point cloud through a fast primal dipole sum query. We pass appearance attributes through a shallow MLP to predict colors, and use geometry attributes 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.

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

1. facilitates *fast* ray tracing, enabling efficient inverse rendering (like rasterization) without sacrificing generality (shadow rays);
2. leverages *dense* point cloud outputs from structure-for-motion initialization, 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 overviews our overall method.

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 $\Gamma \subset \mathbb{R}^3$. Among its many equivalent definitions [Feng et al. 2023], we use that as a Laplacian *double layer potential with unit moment*, which facilitates the generalizations we consider in Section 4. Then, the *winding number* $\overline{\mathrm{w}} : \mathbb{R}^3 \to \mathbb{R}$ equals:

$$\overline{\mathrm{w}}(x) \equiv \int_\Gamma \mathrm{P}(x,y)1\,\mathrm{d}A(y), \quad \mathrm{P}(x,y) \equiv \frac{1}{4\pi}\frac{n(y)\cdot(y-x)}{\|y-x\|^3}. \tag{5}$$

Here, $n(y)$ is the outward normal vector at point $y \in \Gamma$, and $\mathrm{P} : \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}$ is the *free-space Poisson kernel* for the Laplace equation. We make explicit the factor 1 in Equation (5), for reasons we will explain in Section 4. The scalar field $\overline{\mathrm{w}}$ is jump-harmonic and, when the surface Γ is watertight, equals its binary *indicator function* (a fact known as Gauss' lemma [Folland 1995, Proposition 3.19]):

$$\overline{\mathrm{w}}(x) = \begin{cases} 1, & x \text{ inside } \Gamma, \\ 0, & x \text{ outside } \Gamma, \\ 1/2, & x \text{ on } \Gamma. \end{cases} \tag{6}$$

Point clouds. We next consider the winding number for an *oriented* point cloud $\mathcal{P} \equiv \{(\mathrm{p}_m, \mathrm{n}_m, \mathrm{A}_m)\}_{m=1}^M$, 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 outward 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 $\mathcal{P}$. We use the dense point cloud from structure-from-motion initialization, which provides points p_m and normals n_m, and we estimate area weights A_m as in Barill et al. [2018]. Then, Barill et al. [2018] generalize the winding number to point clouds using a discretization of the double layer potential (5).[3]

Winding number for an oriented point cloud

For an oriented point $\mathcal{P} \equiv \{(\mathrm{p}_m, \mathrm{n}_m, \mathrm{A}_m)\}_{m=1}^M$, its *winding number* $\widetilde{\mathrm{w}} : \mathbb{R}^3 \to \mathbb{R}$ is the scalar field:

$$\widetilde{\mathrm{w}}(x) \equiv \sum_{m=1}^{M} \mathrm{A}_m \, \mathrm{P}(x, \mathrm{p}_m) 1 = \sum_{m=1}^{M} \frac{\mathrm{A}_m}{4\pi} \frac{\mathrm{n}_m \cdot (\mathrm{p}_m - x)}{\|\mathrm{p}_m - x\|^3} 1. \tag{7}$$

Winding number as a geometry field. Though $\widetilde{\mathrm{w}}$ is not a binary scalar field (unlike its continuous counterpart $\overline{\mathrm{w}}$), 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 we can use it to represent a geometry field for inverse rendering (Section 3.1) as:

$$\mathrm{F}_{\mathrm{w}}(x) \equiv \frac{1}{2} - \widetilde{\mathrm{w}}(x), \tag{8}$$

corresponding to an implicit surface $\Gamma_{\mathrm{w}} \equiv \left\{x \in \mathbb{R}^3 : \mathrm{F}_{\mathrm{w}}(x) = 0\right\}$. This representation provides several critical advantages:

- ✓ *Accurate approximation.* Γ_{w} provides an approximation to Γ that becomes exact as point density becomes infinite, and degrades gracefully as the number of points M decreases.
- ✓ *Geometric regularization.* F_{w} is imbued with regularity properties that provide *geometric regularization*. It is *jump-harmonic*, and thus of a smooth nature that has proven useful for geometric optimization tasks [Peng et al. 2021; Lipman 2021]. It is also related to robust 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. We elaborate on these relationships in Sections 4.1 and 5.
- ✓ *Direct initialization.* F_{w} can be directly computed using the point cloud from structure-from-motion initialization. Point queries for $\widetilde{\mathrm{w}}$, and thus F_{w}, use only the point cloud attributes, and do not require meshing or a proxy data structure (e.g., grid or neural).
- ✓ *Fast queries.* Such point queries, *and backpropagating through them*, can be made efficient with logarithmic complexity $O(\log M)$ relative to point cloud size M, as we explain in Section 6. Thus, F_{w} lends itself to efficient ray tracing (which requires multiple point queries along each viewing ray (Equation (1))), even when working with dense point clouds from structure from motion.

Barill et al. [2018] further discuss the benefits of the winding number $\widetilde{\mathrm{w}}$ versus other point-based surface representations. At the same time, $\widetilde{\mathrm{w}}$, and thus F_{w}, have critical shortcomings that make them unsuitable for direct use for inverse rendering:

- ✗ *Numerical instability.* The Poisson kernel $\mathrm{P}(x, y)$ is singular as $x \to y$. The singularity makes the surface Γ_{w} numerical algorithms interface with—e.g., during ray tracing [Gillespie et al. 2024, Section 4.3] or isosurface extraction [Barill et al. 2018, Section 3 & Figure 9]—inaccurate and numerically unstable near $\mathcal{P}$. These numerical issues hinder inverse rendering performance (e.g., due to rays passing near or through points in $\mathcal{P}$, Section 7).
- ✗ *Exact interpolation.* The singularity makes the implicit surface Γ_{w} an *exact interpolant* of the point cloud $\mathcal{P}$. Exact interpolation is undesirable when working with imperfect point clouds with *noisy* point locations [Barill et al. 2018, Section 9], such as those from structure-from-motion initialization.
- ✗ *Outlier sensitivity.* Such point clouds typically also suffer from outlier points (e.g., due to incorrect correspondences) and inaccurate or incorrectly oriented normals. As $\widetilde{\mathrm{w}}$ weighs all points equally, it is can be very sensitive to such defects.

We explain how to overcome these shortcomings in the next section.

4 REGULARIZED DIPOLE SUMS

We introduce a generalization of $\widetilde{\mathrm{w}}$ in Equation (7) that facilitates point-based representations in inverse rendering for both the geometry field F and, as we explain in Section 5.1, the radiance field L. Our generalization changes Equation (7) by replacing: 1. singular with non-singular kernels (Section 4.1); and 2. unit with variable per–point weights (Section 4.2). We also present two technical results (Propositions 1 and 2) that lend theoretical support to our generalization, by relating it to Poisson surface reconstruction [Kazhdan et al. 2006] and stochastic point clouds (resp.).

4.1 Regularization

To overcome shortcomings due to the singular Poisson kernel P, we turn to *regularization* schemes common in methods for the simulation of linear partial differential equations (e.g., method of fundamental solutions, boundary element method [Chen et al. 2024, Section 2.2]). These methods use regularization to address numerical issues arising from singular potential kernels analogous to the issues we encounter in inverse rendering.

A common regularization scheme [Beale et al. 2016; Cortez 2001; Cortez et al. 2005][4] starts from the definition of the Poisson kernel

[3] Throughout we use bars and tildes to indicate correspondences between quantities involving continuous surface integrals and their point-cloud approximations (resp.).

[4] An alternative to regularization is to "desingularize" the Poisson kernel by introducing a small cutoff in the denominator [Lu et al. 2018; Lin et al. 2022]. We found that this

through the Laplacian *Green's function* $\mathrm{G} : \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}$:

$$\mathrm{P}(x, y) \equiv n(y) \cdot \nabla_y \mathrm{G}(x, y), \tag{9}$$

$$\text{where G satisfies: } \Delta_x \mathrm{G}(x, y) = \delta(x - y), \tag{10}$$

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

$$\mathrm{P}_\varepsilon(x, y) \equiv n(y) \cdot \nabla_y \mathrm{G}_\varepsilon(x, y), \tag{11}$$

$$\text{where } \mathrm{G}_\varepsilon \text{ satisfies: } \Delta_x \mathrm{G}_\varepsilon(x, y) = \phi_\varepsilon(x - y). \tag{12}$$

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

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

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

$$\mathrm{P}_\varepsilon(x, y) \equiv \mathrm{S}\left(\frac{\|y - x\|}{\varepsilon}\right) \mathrm{P}(x, y), \tag{14}$$

where $\mathrm{S}(t) \equiv \mathrm{erf}(t) - 2t/\sqrt{\pi} \exp(-t^2)$. Unlike P, P_ε is not singular, as $\mathrm{P}_\varepsilon(y, y) = 3^{-1}\varepsilon^{-3}\pi^{-3/2}$ is finite for $\varepsilon > 0$. The parameter ε controls the trade-off between regularization (restricting how fast $\mathrm{P}_\varepsilon(x, y)$ increases as $\|y - x\| \to 0$) and bias (bounding the difference $\mathrm{P}_\varepsilon - \mathrm{P}$). We can use P_ε to generalize Equation (7) as follows.

Regularized winding number for an oriented point cloud

For an oriented point $\mathcal{P} \equiv \{(\mathrm{p}_m, \mathrm{n}_m, \mathrm{A}_m)\}_{m=1}^M$, and a regularization parameter $\varepsilon \geq 0$, its *regularized winding number* $\widetilde{\mathrm{w}}_\varepsilon : \mathbb{R}^3 \to \mathbb{R}$ is the scalar field:

$$\widetilde{\mathrm{w}}_\varepsilon(x) \equiv \sum_{m=1}^{M} \mathrm{A}_m \, \mathrm{P}_\varepsilon(x, \mathrm{p}_m) 1, \tag{15}$$

where P_ε is the regularized Poisson kernel in Equation (14).

Unlike $\widetilde{\mathrm{w}}$, $\widetilde{\mathrm{w}}_\varepsilon$ can be robustly evaluated arbitrarily close to points in $\mathcal{P}$, and its 1/2-level set does *not* exactly interpolate those points. Thus, it escapes the first two shortcomings we identified at the end of Section 3.2. During inverse rendering (Section 5.2), we can use additional image-based losses to penalize large deviations between the level set and $\mathcal{P}$, while also allowing for inexact interpolation to account for noise. Figure 3 uses the dense point cloud output from dense structure-from-motion initialization for a scene from the BlendedMVS dataset [Yao et al. 2020] to compare: 1. 2D slices of the original and regularized winding number fields, and 2. meshed isosurfaces extracted from them using marching cubes [Lorensen and Cline 1987]. Simply using the regularized winding number on this initial point cloud, without any training, already produces high-quality meshes comparable to those from state-of-the-art inverse rendering methods, as we quantify in Section 7.

approach results in worse performance in inverse rendering experiments (Section 7), corroborating the arguments of Cortez [2001] in favor of regularization.

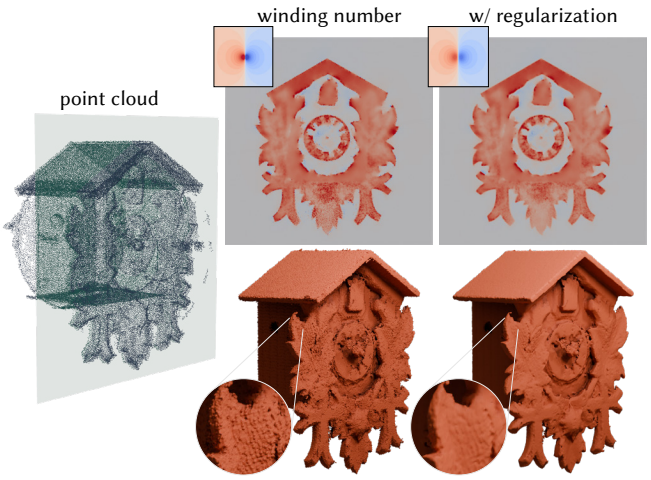

Figure 3. Using the original and regularized winding number fields on the unoptimized point cloud (left) for the BlendedMVS CLOCK scene. The top row shows planar slices of the two fields: The original winding number is very noisy near point cloud locations due to the singular Poisson kernel, whereas the regularized winding number is much smoother. The insets visualize the singular and regularized kernels. The bottom row shows meshes extracted from the two fields using marching cubes: The original winding number results in strong artifacts, which the regularized winding number fixes.

Relationship to Poisson surface reconstruction. We remark on a relationship between the regularized winding number $\widetilde{\mathrm{w}}_\varepsilon$ in Equation (15), and *Poisson surface reconstruction (PSR)* [Kazhdan et al. 2006; Kazhdan and Hoppe 2013]; this relationship highlights the useful geometric regularization properties of $\widetilde{\mathrm{w}}_\varepsilon$, and its generalization in Equation (17). Like Equation (15), PSR uses an oriented point cloud to compute a scalar field that approximates the continuous winding number (5) for the underlying surface Γ. This scalar field has proven to enable robust surface reconstruction, thanks to the regularity properties of the Poisson equation used to compute it. As a result, PSR has become a workhorse for point-based surface reconstruction [Berger et al. 2014; Huang et al. 2024a]. Unfortunately, using PSR in inverse rendering is prohibitively expensive, as querying (and differentiating) its scalar field output requires constructing a grid and performing a global Poisson solve operation [Peng et al. 2021]. However, we prove in Appendix A.1 the following result.

Proposition 1: Poisson surface reconstruction

The regularized winding number $\widetilde{\mathrm{w}}_\varepsilon$ is the solution to the Poisson equation of Kazhdan et al. [2006].

Proposition 1 shows that we can use the regularized winding number $\widetilde{\mathrm{w}}_\varepsilon$ in Equation (15) (and its generalization in Equation (17)) as a geometry representation for inverse rendering that has the same regularity and robustness properties as PSR, while remaining efficient to render and differentiate (Section 6). Feng et al. [2023, Section 1.1.2] and Barill et al. [2018, Section 2.1] have previously discussed the relationship between the *non-regularized* winding number $\widetilde{\mathrm{w}}$ in Equation (5) and PSR. However, the two are equivalent only *asymptotically*, at the limit of zero-variance Gaussian blurring of normals

in PSR [Kazhdan et al. 2006, Equation (2)]. By contrast, the equivalence Proposition 1 is *exact* for all blur variances—intuitively, the Gaussian blurring of normals in PSR is equivalent to the Gaussian regularization of the Green's function and Poisson kernel in Equations (11) and (12). The need to incorporate Gaussian regularization helps resolve, both theoretically and in practice (Figure 3) performance discrepancies between the winding number and PSR in, e.g., isosurface extraction [Barill et al. 2018, Section 3 & Figure 9].

4.2 Variable moment

To overcome shortcomings due to outlier points and inaccurate normals, we modify Equation (15) to use variable per-point weights—thus allowing to deemphasize outliers. This modification yields a point-based representation suitable for both the geometry field and radiance field (Section 5.1) in inverse rendering.

To this end, we can replace the unit moment in Equation (5) with a *variable moment* $\mathrm{f} : \Gamma \to \mathbb{R}$. Its corresponding *double layer potential* $\overline{\mathrm{f}} : \mathbb{R}^3 \to \mathbb{R}$ is the scalar field [Folland 1995, Section 3.C]:

$$\overline{\mathrm{f}}(x) \equiv \int_\Gamma \mathrm{P}(x, y)\, \mathrm{f}(y)\, \mathrm{d}A(y). \tag{16}$$

For any sufficiently smooth moment f, $\overline{\mathrm{f}}$ is *jump-harmonic* [Krutitskii 2001]: it satisfies Laplace's equation at $x \in \mathbb{R}^3 \setminus \Gamma$, and has a jump discontinuity equal to f at $x \in \Gamma$, analogously to Equation (6) for $\overline{\mathrm{w}}$.

Using the moment values on the point cloud, $\mathrm{f}_m \equiv \mathrm{f}(\mathrm{p}_m), \mathrm{p}_m \in \mathcal{P}$, and the regularized kernel P_ε to circumvent singularity issues, we arrive at a regularized point-cloud approximation of Equation (16).[5]

Regularized dipole sum for an oriented point cloud

For an oriented point $\mathcal{P} \equiv \{(\mathrm{p}_m, \mathrm{n}_m, \mathrm{A}_m)\}_{m=1}^M$, a regularization parameter $\varepsilon \geq 0$, and a moment function with point samples $\mathrm{f}_m, m = 1, \ldots, M$, the corresponding *regularized dipole sum* $\widetilde{\mathrm{w}}_\varepsilon : \mathbb{R}^3 \to \mathbb{R}$ is the scalar field:

$$\widetilde{\mathrm{f}}_\varepsilon(x) \equiv \sum_{m=1}^M \mathrm{A}_m\, \mathrm{P}_\varepsilon(x, \mathrm{p}_m)\, \mathrm{f}_m\,. \tag{17}$$

where P_ε is the regularized Poisson kernel in Equation (14).

The point-cloud winding number and its regularized form in Equations (7) and (15) are special cases, i.e., $\widetilde{\mathrm{w}} = \widetilde{1}_0$ and $\widetilde{\mathrm{w}}_\varepsilon = \widetilde{1}_\varepsilon$. The regularized dipole sum maintains the advantages of the point-cloud winding number we listed in Section 3.2, while addressing its shortcomings. We can thus treat $\mathrm{f}_m,\ m = 1, \ldots, M$ as a *learnable* per-point *geometry attribute*, and use its corresponding dipole sum to define a point-based representation for the geometry field:

$$\mathrm{F}(x) \equiv \frac{1}{2} - \widetilde{\mathrm{f}}_\varepsilon(x), \tag{18}$$

We then convert F to a vacancy v and attenuation coefficient σ using Equations (3) and (4). We initialize $\widetilde{\mathrm{f}}_\varepsilon$ to equal the regularized winding number $\widetilde{\mathrm{w}}_\varepsilon$ in Equation (15) by using initial geometry attribute values $\mathrm{f}_m = 1$, which we then optimize during inverse rendering to update the scene geometry (Section 5.2). Figure 4 visualizes these initial and optimized fields on the teaser scene. Compared to the winding number, allowing non-unit values for f_m during the inverse rendering process serves two goals: 1. It allows the process to diminish the influence of point cloud outliers, by decreasing their geometry attribute f_m. 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 correct noisy point locations or holes in textureless regions) *without* changing the point locations p_m in $\mathcal{P}$; as we explain in Section 6.3, fixing point locations facilitates faster inverse rendering.

Stochastic point cloud interpretation. Our generalization of the winding number $\widetilde{\mathrm{w}}$ in Equation (5) into the regularized dipole sum $\widetilde{\mathrm{f}}_\varepsilon$ in Equation (17) was motivated by the need for improved robustness when working with imperfect point clouds that include noisy and outlier points. We can model such a point cloud as *stochastic*, treating both point locations and normals as random variables. The winding number of such a stochastic point cloud is itself a random variable. We then prove the following relationship between this random variable and the regularized dipole sum.

Proposition 2: Stochastic point cloud

We assume that $\mathcal{P}$ is a stochastic point cloud such that, for each point: 1. its location is a 3D Gaussian random variable $\mathrm{P}_m \sim \mathcal{N}(\mathrm{p}_m, \varepsilon I)$, where I is the 3×3 identity matrix; 2. its normal N_m is a spherical random variable with conditional mean direction n_m and mean resultant length f_m, i.e., $\mathbb{E}[\mathrm{N}_m \mid \mathrm{P}_m] = \mathrm{f}_m\, \mathrm{n}_m$. We also assume that all other point cloud attributes are deterministic. Then, the expected value of $\widetilde{\mathrm{w}}$ in Equation (7) equals:

$$\mathbb{E}_{\{\mathrm{P}_m, \mathrm{N}_m\}_{m=1}^M}[\widetilde{\mathrm{w}}(x)] = \sum_{m=1}^M \mathrm{A}_m\, \mathrm{P}_\varepsilon(x, \mathrm{p}_m)\, \mathrm{f}_m = \widetilde{\mathrm{f}}_\varepsilon. \tag{19}$$

We refer to Mardia and Jupp [2009, Chapter 9] for background on spherical random variables, and prove Proposition 2 in Appendix A.2. Proposition 2 lends theoretical support to our use of a regularized dipole sum to represent the geometry of point clouds $\mathcal{P}$ with noisy ($\varepsilon > 0$) and outlier ($\mathrm{f}_m \approx 0$)[6] points. It also suggests the option to use different values ε_m, or even covariance matrices Σ_m, to model varying and anisotropic per-point uncertainty [Fuhrmann and Goesele 2014]. Empirically, we did not find doing so beneficial, and thus use a global ε value that we select as we explain in Section 5.2.

5 INVERSE RENDERING WITH POINT-BASED FIELDS

We derived a point-based representation for the geometry field in inverse rendering. To complete our inverse rendering pipeline, we introduce a point-based representation for the radiance field (Section 5.1), and explain how to optimize both fields (Section 5.2).

5.1 Radiance field representation

Our derivation of the regularized dipole sum in Section 4 focused on point-based representation of scene geometry. However, Equation (17) provides a way to interpolate any *learnable* per-point

[5] Following Barill et al. [2018, Section 3.1] and Gotsman and Hormann [2024, Section 2], we use the term *dipole* because the Poisson kernel $\mathrm{P}(x, y)$ equals the electric potential of a dipole centered at y and polarized in the direction of $n(y)$.

[6] In this stochastic interpretation, $\mathrm{f}_m = 0$ means that the random normal N_m (conditional on P_m) is *uniformly* distributed on the sphere. Then, the direction of the corresponding dipole is completely uncertain, and on expectation the dipole vanishes.

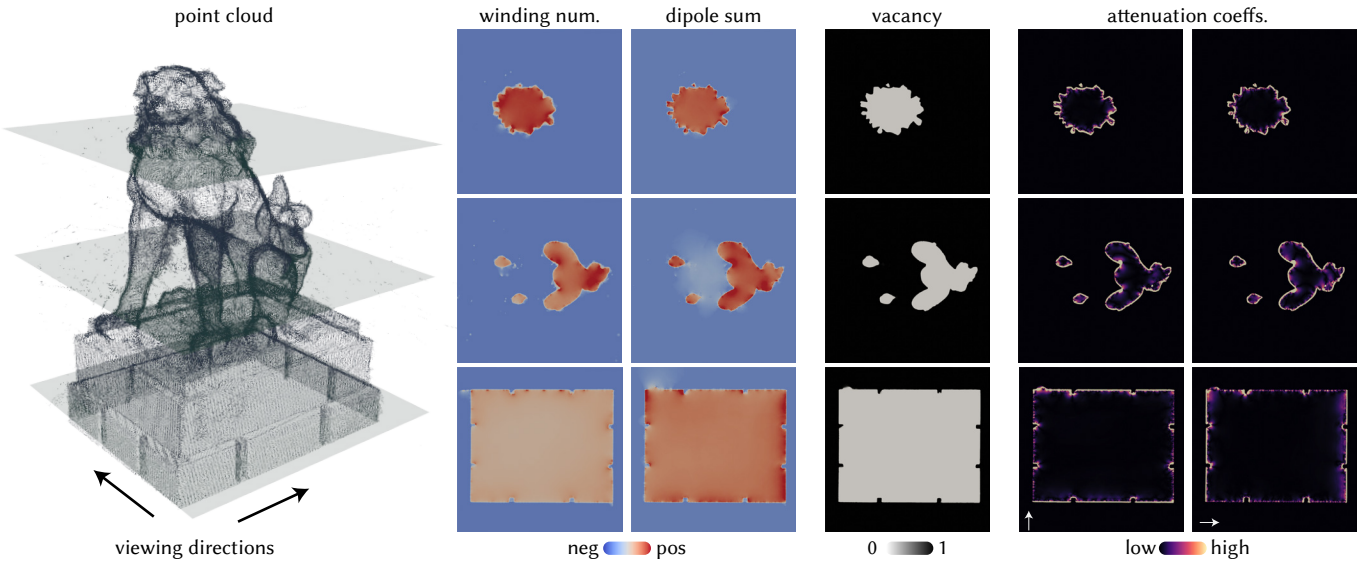

Figure 4. We visualize on the teaser scene geometry-related field quantities that we use for inverse rendering. From left to right: the initial geometry field with unit geometry attributes (equal to the regularized winding number in Equation (15)), the optimized geometry field with learned geometry attributes (Equation (18)), the optimized vacancy field (Equation (3)), and attenuation coefficients computed (Equation (4)) along two different viewing directions.

attributes—corresponding to point samples of the continuous moment of a double-layer potential, or equivalently the jump-Dirichlet boundary condition of a Laplace equation—to scalar fields for use in inverse rendering. Thus, the regularized dipole sum lends itself as a point-based representation also for the radiance field.

To this end, we first interpolate a set of per-point *appearance attributes* ℓ^k using regularized dipole sums $\widetilde{\ell}^k_\varepsilon(x)$ as in Equation (17),

$$\widetilde{\ell}^k_\varepsilon(x) \equiv \sum_{m=1}^{M} \mathrm{A}_m \, \mathrm{P}_\varepsilon(x, \mathrm{p}_m) \, \ell^k_m, \;\; k = 1, \ldots, K. \tag{20}$$

We then represent the radiance field $\mathrm{L}(x, \omega)$ as the output of a shallow multi-layer perceptron (MLP) that takes as input the values $\widetilde{\ell}^k_\varepsilon(x)$, position x and (encoded) direction ω, and the *implicit surface normal* from the geometry field $\mathrm{n}_{\mathrm{imp}}(x) \equiv \nabla \mathrm{F}(x)/\|\nabla \mathrm{F}(x)\|$:[7]

$$\mathrm{L}(x, \omega) \equiv \mathrm{MLP}\Big(x, \omega, \mathrm{n}_{\mathrm{imp}}(x), \widetilde{\ell}^1_\varepsilon(x), \ldots, \widetilde{\ell}^K_\varepsilon(x)\Big). \tag{21}$$

The radiance field L and the geometry field F (Equation (18)) are intertwined, as the regularized dipole sums for appearance (Equation (20)) and geometry (Equation (17)) share the same weights—determined by point cloud locations p_m, area weights A_m, and normals n_m.

Relationship to mean value interpolation. Our use of regularized dipole sums in Equation (20) to interpolate per-point appearance attributes ℓ is closely related to 3D interpolation with *mean value coordinates* [Floater et al. 2005; Ju et al. 2005]. Using Equations (5) and (16), we can write the mean value interpolant (and its point-cloud approximation) at $x \in \mathbb{R}^3$ of a function $\ell : \Gamma \to \mathbb{R}$ as:

$$\mathrm{mv}^\ell(x) \equiv \frac{\overline{\ell}(x)}{\overline{\mathrm{w}}(x)} \approx \frac{\widetilde{\ell}(x)}{\widetilde{\mathrm{w}}(x)}. \tag{22}$$

Mean value interpolation has found widespread use in computer graphics and other areas [Hormann and Sukumar 2017; Chen et al. 2024; de Goes and Desbrun 2024], a success in large part thanks to the geometric regularization properties of the mean value interpolant [Ju et al. 2005, Section 2]. These properties and empirical success lend support to our choice of (regularized) dipole sums as a point-based representation for the radiance field. 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.

5.2 Inverse rendering optimization

Our overall scene representation comprises a point cloud $\mathcal{P} \equiv \left\{\left(\mathrm{p}_m, \mathrm{n}_m, \mathrm{A}_m, \mathrm{f}_m, \ell^1_m, \ldots, \ell^K_m\right)\right\}_{m=1}^{M}$ with per-point locations, normals, area weights, geometry attribute, and appearance attributes; as well as the parameters of the MLP in Equation (21). We use this representation to compute the geometry field F (Equations (17) and (18)) and radiance field L (Equations (20) and (21)). During the inverse rendering stage, we synthesize images using volume rendering and ray tracing (Equation (2)) combined with F and L. We then optimize the point cloud $\mathcal{P}$ and MLP by minimizing the loss:

$$\mathcal{L}_{\mathrm{rendering}} + \mathcal{L}_{\mathrm{entropy}} + \mathcal{L}_{\mathrm{winding}} + \mathcal{L}_{\mathrm{normal}}, \tag{23}$$

where each summand includes an appropriate weight, and:

1. $\mathcal{L}_{\mathrm{rendering}}$ is the L^1-loss between input and rendered images;
2. $\mathcal{L}_{\mathrm{entropy}}$ is a per-ray entropy loss inspired from Kim et al. [2022] (we provide details in Appendix B);

[7] We experimented with a representation where the interpolated appearance attributes $\widetilde{\ell}^k_\varepsilon(x)$ are *spherical harmonic coefficients* that are convertible to radiance $\mathrm{L}(x, \omega)$ through a rotation operation, as advocated by Karnewar et al. [2022] and 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.

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

struct TreeNode
  $\widehat{\mathrm{p}}, \widehat{\mathrm{A}}, \widehat{\mathrm{r}}, \widehat{\mathrm{b}} \leftarrow$ TreeUpdate ▷*Immutable node attributes initialized using Equations (24) and (25)*
  $\widehat{\mathrm{db}} \leftarrow 0$ ▷*Mutable node gradient attribute*
  function GetContribution(x, ε)
    return $\widehat{\mathrm{A}}\, \mathrm{S}(\|\widehat{\mathrm{p}}-x\|/\varepsilon)(\widehat{\mathrm{p}}-x)/\|\widehat{\mathrm{p}}-x\|^3 \cdot \widehat{\mathrm{b}}$ ▷*Compute node contribution to dipole sum using Equation (26)*
  end function
  function IncrementGradient($\widetilde{\mathrm{db}}_\varepsilon, x, \varepsilon$)
    $\widehat{\mathrm{db}}$ += $\widehat{\mathrm{A}}\, \mathrm{S}(\|\widehat{\mathrm{p}}-x\|/\varepsilon)(\widehat{\mathrm{p}}-x)/\|\widehat{\mathrm{p}}-x\|^3 \cdot \widetilde{\mathrm{db}}_\varepsilon$ ▷*Increment node gradient attribute using Equation (53)*
  end function
  function GetChildren
    return `listOfChildrenNodes` ▷*Return a list of children nodes, or empty list if node is a leaf*
  end function
Input: *A query point x, the root node of a tree structure* node*, a control parameter β.*
Output: *Dipole sum $\widetilde{\mathrm{b}}_\varepsilon(x)$.*
function PrimalQuery(x, node, ε, β)
  if $\|x - \mathrm{node}.\widehat{\mathrm{p}}\| > \beta \cdot \mathrm{node}.\widehat{\mathrm{r}}$ **then return** node.GetContribution(x, ε) ▷*If the query point is far from the cluster, terminate*
  `listOfChildrenNodes` ← node.GetChildren ▷*Get list of children nodes*
  if IsEmpty(`listOfChildrenNodes`) **then return** node.GetContribution(x, ε) ▷*If the node is a leaf, terminate*
  $\widetilde{\mathrm{b}}_\varepsilon \leftarrow 0$ ▷*Initialize dipole sum value*
  for `child` **in** `listOfChildrenNodes` **do**
    $\widetilde{\mathrm{b}}_\varepsilon$ += PrimalQuery(x, `child`, ε, β) ▷*Iterate over all children nodes*
  return $\widetilde{\mathrm{b}}_\varepsilon$
end function
Input: *A gradient $\widetilde{\mathrm{db}}_\varepsilon$, a query point x, the root node of a tree structure* node*, a control parameter β.*
function AdjointQuery($\widetilde{\mathrm{db}}_\varepsilon, x$, node, ε, β)
  if $\|x - \mathrm{node}.\widehat{\mathrm{p}}\| > \beta \cdot \mathrm{node}.\widehat{\mathrm{r}}$ **then** node.IncrementGradient($\widetilde{\mathrm{db}}_\varepsilon, x, \varepsilon$) **return** ▷*If the query point is far from the cluster, terminate*
  `listOfChildrenNodes` ← node.GetChildren ▷*Get list of children nodes*
  if IsEmpty(`listOfChildrenNodes`) **then** node.IncrementGradient($\widetilde{\mathrm{db}}_\varepsilon, x, \varepsilon$) **return** ▷*If the node is a leaf, terminate*
  for `child` **in** `listOfChildrenNodes` **do**
    AdjointQuery($\widetilde{\mathrm{db}}_\varepsilon, x$, `child`, ε, β) ▷*Iterate over all children nodes*
end function

3. $\mathcal{L}_{\text{winding}}$ aggregates losses $\|\mathrm{f}_m - 1\|^2$ on the point cloud;
4. $\mathcal{L}_{\text{normal}}$ aggregates losses $\left\|\mathrm{n}_m - \mathrm{n}_{m,\text{init}}\right\|^2$ on the point cloud.

The loss $\mathcal{L}_{\text{winding}}$ regularizes the geometry field F by penalizing large deviations between the regularized dipole sum $\widetilde{\mathrm{f}}_\varepsilon$ and the regularized winding number $\widetilde{\mathrm{w}}_\varepsilon$. The loss $\mathcal{L}_{\text{normal}}$ penalizes large changes to point normals compared to their initial values.

We initialize $\mathcal{P}$ with locations p_m and normals n_m using the *dense* point cloud output of COLMAP [Schönberger and Frahm 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 regularized winding number), and appearance attributes ℓ_m^k using Gaussian random variates. Inverse rendering optimizes: the normals, geometry attributes, 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 fast 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 (noisy points, outliers, holes) in the dense structure-from-motion point cloud.

6 BARNES-HUT FAST SUMMATION

Rendering with our point-based representations requires *evaluating* dipole sums (Equations (17) and (20)) at multiple locations along each viewing ray, to compute the geometry (Equation (18)) and radiance (Equation (21)) fields in Equation (1). Inverse rendering with these representations requires additionally *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 (i.e., as summations by iterating over all points), primal and adjoint queries have linear complexity $O(M)$ relative to point cloud size M. Consequently, during inverse rendering, these queries become the main computational burden when working with even moderately large point clouds; and become

prohibitively expensive when working with the dense point clouds output by structure-from-motion initialization.

Fortunately, it is possible to dramatically accelerate both types of queries, enabling inverse rendering at speeds competitive with rasterization methods [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 regularized 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 *moment attributes* stored in $\mathcal{P}$—namely, the geometry attribute f and the appearance attributes $\ell^k,\ k = 1, \dots, K$.

6.1 Acceleration of primal queries

The Barnes-Hut method 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 set $\mathcal{L}(t)$ of all leaf nodes that are successors of t in the tree hierarchy. We follow Barill et al. [2018] and assign the node area, location, area, and radius (resp.) attributes:

$$\widehat{\mathrm{A}}_t \equiv \sum_{m \in \mathcal{L}(t)} \mathrm{A}_m,\ \widehat{\mathrm{p}}_t \equiv \frac{1}{\widehat{\mathrm{A}}_t} \sum_{m \in \mathcal{L}(t)} \mathrm{A}_m\, \mathrm{p}_m,\ \widehat{\mathrm{r}}_t \equiv \max_{m \in \mathcal{L}(t)} \|\mathrm{p}_m - \widehat{\mathrm{p}}_t\|, \tag{24}$$

as well as *vector-valued* moment attributes:

$$\widehat{\mathrm{b}}_t \equiv \frac{1}{\widehat{\mathrm{A}}_t} \sum_{m \in \mathcal{L}(t)} \mathrm{A}_m\, \mathrm{n}_m\, \mathrm{b}_m, \tag{25}$$

which absorb the leaf nodes' moment *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., $\|x - \widehat{\mathrm{p}}_t\| > \beta \widehat{\mathrm{r}}_t$, where β is a user-defined parameter), the node's successors are not visited. Instead, the sum of contributions from all leaf nodes in $\mathcal{L}(t)$ to the dipole sum is approximated using the node's attributes:

$$\sum_{m \in \mathcal{L}(t)} \mathrm{A}_m\, \mathrm{P}_\varepsilon(x, \mathrm{p}_m)\, \mathrm{b}_m \approx \widehat{\mathrm{A}}_t\, \mathrm{S}\left(\frac{\|\widehat{\mathrm{p}}_t - x\|}{\varepsilon}\right) \frac{\widehat{\mathrm{b}}_t \cdot (\widehat{\mathrm{p}}_t - x)}{\|\widehat{\mathrm{p}}_t - x\|^3}. \tag{26}$$

This approximation expresses the fact that, due to the squared-distance falloff of P_ε, 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.

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 a set of additional *mutable* gradient attributes $\mathrm{d}\widehat{\mathrm{b}}_t$ (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 $O(QM)$. 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 dense point cloud output. The node hierarchy in the tree depends on only the point cloud locations p_m. Thus, as we choose not to update these locations during inverse rendering (Section 5.2), we create the tree structure only *once* rather than after each gradient operation. Choosing otherwise would introduce significant computational overhead.

At each iteration, we must twice update the moment attributes $\widehat{\mathrm{b}}_t$ 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 moment attributes. Primal and inverse rendering with Equation (1) requires performing, at every sampled ray location x, dipole sum queries for all moment attributes b stored in the point cloud—namely, the geometry attribute f and the appearance attributes $\ell^k,\ k = 1, \dots, K$. 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 query points x is an exciting future direction.

7 EXPERIMENTAL EVALUATION

We evaluate our method against state-of-the-art methods for multi-view surface reconstruction: Gaussian surfels [Dai et al. 2024], which combines a point-based representation with rasterization; and Neus2 [Wang et al. 2023] and Neuralangelo [Li et al. 2023], which both

combine a hybrid hashgrid-neural representation with ray tracing. All three methods aim for high-quality surface outputs, but place different emphasis on computational efficiency (Gaussian surfels, NeuS2) versus reconstruction fidelity (Neuralangelo). In summary, our results suggest that our method provides, at equal runtimes, improved reconstruction quality and robustness compared to these alternatives. Additionally, though we do not include direct comparisons, our results (Table 1) additionally suggest that our method improves reconstruction quality compared to other concurrent 3D Gaussian methods, including 2D Gaussian splatting [Huang et al. 2024b, Table 1] and Gaussian opacity fields [Yu et al. 2024, Table 2]. We provide additional results and code on the project website.

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]. Our code is available on the project website.

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 encode viewing directions with real spherical harmonics up to degree 3, and use $K = 32$ appearance attributes ℓ^k, for which we found it beneficial to skip the foreshortening term when computing dipole sums (Equation (20)). We apply weight normalization [Salimans and Kingma 2016] for stable training. We limit the radiance field inside a bounding sphere, and use a background network based on NeRF++ [Zhang et al. 2020] to model the exterior of the sphere.

Ray sampling. We sample ray locations in Equation (2) as in Miller et al. [2024]: For each camera ray, we identify the first zero-crossing of the geometry field F by densely placing 1024 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, 48 dense samples around the first crossing, and 8 sparse samples between the first crossing and the far limit. Otherwise, we place 80 samples uniformly between the near and far limits.

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 method 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 ray marching procedure we described above; thus we use the latter for simplicity.

Training. We use Adam [Kingma and Ba 2015] with a batch size 4096 rays for optimization. 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 1000, 3000, and 20000 iterations takes 3 min, 8 min, and 1 hour (resp.) on a single NVIDIA RTX 4090 GPU.

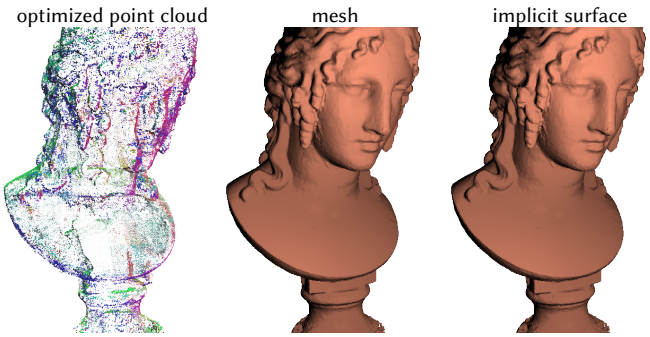

Figure 5. Our regularized dipole sum representation allows us to directly ray trace the optimized point cloud (where we use color to visualize normals, and size to visualize geometry attributes), achieving the same results as ray tracing a mesh without the need to extract one.

Point growing. As we mentioned in Section 4.2, the use of non-unit geometry attributes for the geometry field F 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 10% additional points relative to the original dense point cloud from structure from motion, as we show in Figure 10.

Mesh extraction. We produce meshes by extracting the zero-level set of F using marching cubes [Lorensen and Cline 1987], at 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 kernel. By contrast, thanks to the regularized Poisson kernel, we can extract artifact-free meshes using marching cubes, as we show in Figure 3. 2. We need to extract meshes *only* for quantitative comparisons with other methods. We can directly and efficiently ray trace our geometry field using ray marching with fast primal queries (and optionally Harnack tracing [Gillespie et al. 2024]), as we show in Figure 5.

7.2 Comparison to prior work

We evaluate our method against NeuS2 [Wang et al. 2023], Gaussian 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 train our method, NeuS2, and Gaussian surfels without mask supervision and evaluate their extracted meshes using the DTU evaluation script. For each method, we present results for *runtimes* of 5 minutes, 10 minutes, and 1 hour, which we measure as follows to ensure fair comparisons: For NeuS2 and Gaussian surfels, runtime equals training time; for our method, runtime includes the time of

Figure 6. Qualitative comparisons on the BlendedMVS (left) and DTU (right) datasets. The dashed circles indicate areas of interest. NeuS2 captures fine details, but produces noisy meshes with structural artifacts. Gaussian surfels produces floater artifacts that require manual filtering. By contrast, our method produces clean meshes with correct and artifact-free geometry. We provide interactive visualizations of results on the entire datasets on the project website.

the refinement process in COLMAP (which the other two methods do not require), thus decreasing training time (e.g., about 2 minutes COLMAP refinement and 3 minutes training for a total runtime of 5 minutes). For Neuralangelo, we report DTU evaluation scores from their paper, and do not report scores on BlendedMVS as on multiple scenes it failed to produce meaningful reconstructions (e.g., second row of Figure 7). Lastly, we also compare against meshes extracted using our regularized winding number (Equation (15)) on the dense point cloud output by COLMAP without training—effectively, the initialization of our method. We provide quantitative results in Tables 1 and 2, qualitative results in Figures 6–8, and interactive visualizations on the project website.

Quantitatively, we observe that our method overall outperforms Gaussian surfels and NeuS2 at all runtimes on both DTU and BlendedMVS. Moreover, our method consistently improves reconstruction quality with additional training time. By contrast, NeuS2 and Gaussian surfels either stagnate or even degrade performance with additional training time. Our method at 1 hour of runtime also outperforms Neuralangelo at 18 hours of training on the DTU dataset.

In all cases, the quantitative improvements also translate to visual qualitative improvements on the extracted meshes. Figures 6 and 7 show some examples, but we encourage using the interactive visualization on the project website to better assess qualitative differences. Our method occasionally takes longer to recover finer details, because it keeps point cloud positions fixed to regularize geometry and prevent introduction of geometric defects. Even with this regularization, as training time increases, our method only improves reconstructed geometry; and with sufficient training time, it reliably recovers finer geometric details. Figure 8 shows some examples visualizing the training progression of our method.

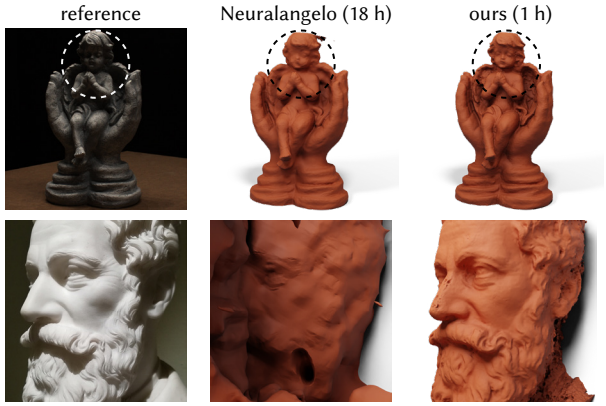

Figure 7. Our method produces higher-quality reconstructions than Neuralangelo on DTU scenes at 1/18 of the runtime (top row). Neuralangelo fails on BlendedMVS scenes when few views are available (bottom row).

Table 1. Chamfer distances on DTU for different runtimes. (N.2: Neus2, G.S.: Gaussian surfels, N.A.: Neuralangelo, init.: regularized winding number on the dense COLMAP point cloud without training.)

	init.	5 m			10 m			18 h	1 h
	ours	N.2	G.S.	**ours**	N.2	G.S.	**ours**	N.A.	**ours**
24	1.82	0.78	0.68	0.80	0.75	0.62	0.64	0.37	0.45
37	1.34	0.64	0.77	0.77	0.65	0.76	0.69	0.72	0.67
40	0.54	1.04	0.56	0.38	1.06	0.49	0.35	0.35	0.32
55	0.60	0.30	0.47	0.39	0.28	0.48	0.36	0.35	0.31
63	0.76	1.01	0.86	0.91	1.00	0.84	0.90	0.87	0.93
65	1.37	0.62	1.06	0.94	0.59	1.08	0.79	0.54	0.67
69	1.45	0.68	0.86	0.78	0.67	0.88	0.76	0.53	0.53
83	0.95	1.17	1.09	0.69	1.18	1.09	0.72	1.29	0.79
97	1.78	1.00	1.31	1.00	1.04	1.31	0.97	0.97	0.91
105	0.88	0.71	0.74	0.61	0.74	0.75	0.59	0.73	0.63
106	0.80	0.55	0.83	0.73	0.54	1.05	0.60	0.47	0.48
110	1.43	0.89	1.76	0.93	0.84	1.76	0.83	0.74	0.57
114	0.60	0.36	0.52	0.47	0.37	0.52	0.39	0.32	0.32
118	0.94	0.47	0.64	0.55	0.43	0.67	0.49	0.41	0.40
122	0.71	0.45	0.59	0.49	0.43	0.61	0.42	0.43	0.39
avg.	**1.06**	0.71	0.85	**0.70**	0.70	0.86	**0.63**	0.61	**0.56**

Table 2. Chamfer distances on BlendedMVS for different runtimes. (N.2: Neus2, G.S.: Gaussian surfels, init.: regularized winding number on the dense COLMAP point cloud without training; ✗ indicates failure to converge.)

	init.	5 m			10 m			1 h		
	ours	N.2	G.S.	**ours**	N.2	G.S.	**ours**	N.2	G.S.	**ours**
bas.	0.52	0.76	0.55	0.62	0.72	0.54	0.61	0.75	0.47	0.45
bea.	0.51	0.88	0.62	0.44	0.89	0.65	0.42	0.94	0.71	0.39
bre.	1.06	0.72	0.44	0.45	0.77	0.48	0.27	0.58	0.68	0.22
cam.	0.60	0.86	0.92	0.53	0.84	0.83	0.56	0.82	0.89	0.57
clo.	0.82	1.40	1.71	0.68	1.33	1.14	0.66	1.41	1.44	0.65
cow	0.52	0.66	1.95	0.56	0.64	2.01	0.54	0.64	2.69	0.56
dog	0.98	1.22	1.53	0.77	1.23	1.54	0.69	1.21	1.71	0.61
dol.	0.84	0.74	0.85	0.70	0.71	0.87	0.70	0.70	0.84	0.75
dra.	1.86	0.96	2.46	0.83	0.91	1.75	0.66	0.97	1.58	0.50
dur.	1.22	✗	1.43	1.04	✗	1.38	1.00	✗	1.47	0.98
fou.	1.22	1.23	1.54	0.97	1.30	1.68	0.96	1.22	1.71	0.88
gun.	0.61	0.34	0.84	0.37	0.35	0.45	0.36	0.41	0.55	0.32
hou.	0.71	0.96	0.82	0.72	1.01	0.83	0.70	1.07	0.89	0.51
jad.	2.15	1.64	1.59	1.83	1.54	2.10	1.80	1.63	1.67	1.66
man	1.10	0.55	0.97	1.09	0.54	1.08	0.82	0.55	1.55	0.55
mon.	0.67	0.39	0.73	0.42	0.35	0.87	0.41	0.35	1.43	0.36
scu.	0.67	0.62	1.23	0.66	0.59	1.86	0.62	0.58	2.01	0.56
sto.	0.79	0.92	0.64	0.64	0.78	0.68	0.63	0.79	0.70	0.53
avg.	**0.94**	0.87	1.16	**0.74**	0.85	1.15	**0.69**	0.86	1.28	**0.61**

By contrast, the alternative methods occasionally recover finer details earlier in training, but are prone to introducing geometric defects that cannot be resolved with additional training time (e.g., NeuS2 and Gaussian surfels reconstructions of the dog head and camera screen, in second and third row (resp.) of Figure 6). These defects can even result in complete failure to extract a meaningful mesh (e.g., Neuralangelo in second row of Figure 7). Additionally, the alternative methods often add higher frequency details not present in the input images (e.g., NeuS2 reconstruction of the bear first row of Figure 6) giving the false impression of increased detail.

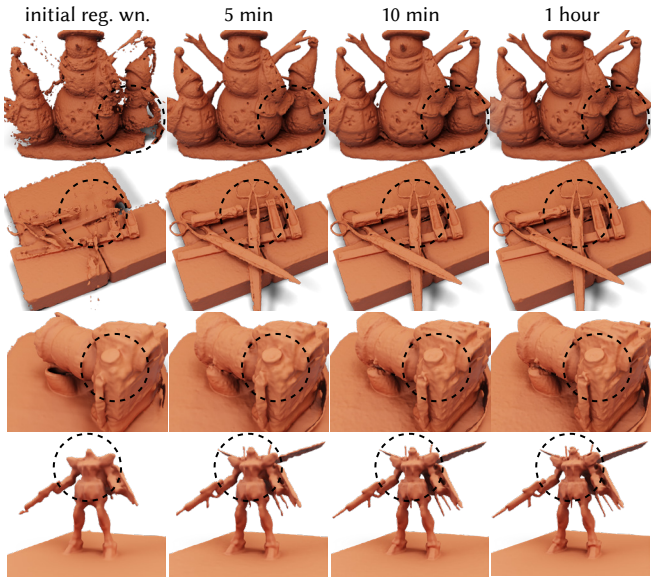

Figure 8. Progression of our method on DTU (top two rows) and BlendedMVS (bottom two rows) scenes. The leftmost column shows the mesh extracted from the initial regularized winding number field, and the remaining columns show meshes at runtimes of 5 minutes, 10 minutes, and 1 hour. Our method significantly improves the initial mesh within 5 minutes (3 minutes training), and continues to refine details with additional training.

Lastly, our method reconstructs more accurate meshes than its initialization. Notably, this initialization already provides a high quality 3D reconstruction, and in several cases better than what NeuS2 and Gaussian surfels after an hour of training (Table 2)! This behavior highlights the importance of leveraging *dense* point cloud initialization from structure from motion in subsequent inverse rendering. To summarize, our method overall ensures robust performance by providing reliable geometry improvement and fine feature recovery, and outperforms alternative methods at equal (NeuS2, Gaussian surfels) or order-of-magnitude shorter (Neuralangelo) runtimes.

7.3 Ablation study

To evaluate the impact of different components of our method in overall performance, we perform an ablation study using the BlendedMVS dataset and the same experimental protocol as in Section 7.2. We evaluate the following variants of our method: 1. removing each of the entropy, winding, and normal losses in Equation (23) during inverse rendering optimization; 2. removing kernel regularization; 3. removing point growing; and 4. removing normal training and keeping normals fixed to their initial values. We provide quantitative results in Table 3. We observe that performance deteriorates in all cases, suggesting that each of the components we consider in this ablation study contributes positively to overall performance.

The component that has the largest impact in performance is removing kernel regularization. We were not able to *completely*

Table 3. Chamfer distances on BlendedMVS for ablation study. Labels indicate components we *remove* from the full method we evaluate in Table 2.

✗	entropy loss	winding loss	normal loss	kernel reg.	point grow.	normal train.
avg.	0.66	0.65	0.69	0.73	0.64	0.68

remove regularization (i.e., use $\varepsilon = 0$ in Equation (14)), as doing so resulted in training failures in all scenes because of numerical errors (undefined values). Instead, we resorted to using a small cutoff in the denominator of the Poisson kernel in Equation (9)—an approach termed "desingularization" by Cortez [2001]. In addition to significantly worsening quantitative scores in Table 3, using desingularization results in extracted meshes with strong artifacts, similar to those we show in Figure 3 for the unoptimized mesh.

7.4 Rendering with shadow rays

Compared to other fast point-based methods such as Gaussian 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 improvements in both mesh reconstruction and novel view synthesis.

Experiment setup. We re-render the Lego scene from the NeRF Realistic Synthetic dataset [Mildenhall et al. 2021] with Lambertian materials and illumination from two point light sources. We render 200 images from random viewpoints and point-light positions that vary from image to image. We process these images with COLMAP to extract an initial dense 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:

$$\mathrm{L}_{\mathrm{sh.rays}}(x,\omega) \equiv \sum\nolimits_{i=1,2} \mathrm{L}_{\mathrm{d}}^{i}(x,\omega) + \mathrm{MLP}\Big(x, \omega, \mathrm{n}_{\mathrm{imp}}(x), \widetilde{\ell}_{\varepsilon}^{1}(x), \ldots, \widetilde{\ell}_{\varepsilon}^{K}(x)\Big), \tag{27}$$

where for each light source:

$$\mathrm{L}_{\mathrm{d}}^{i}(x,\omega) \equiv \alpha(x)\,\mathrm{T}(x,l_i)\,\frac{\mathrm{n}_{\mathrm{imp}}(x)\cdot(l_i-x)}{\|l_i-x\|^3}. \tag{28}$$

Here, l_i is the position of the i-th light source, α is the albedo at x, and $\mathrm{T}(x,l_i)$ 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 9 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.

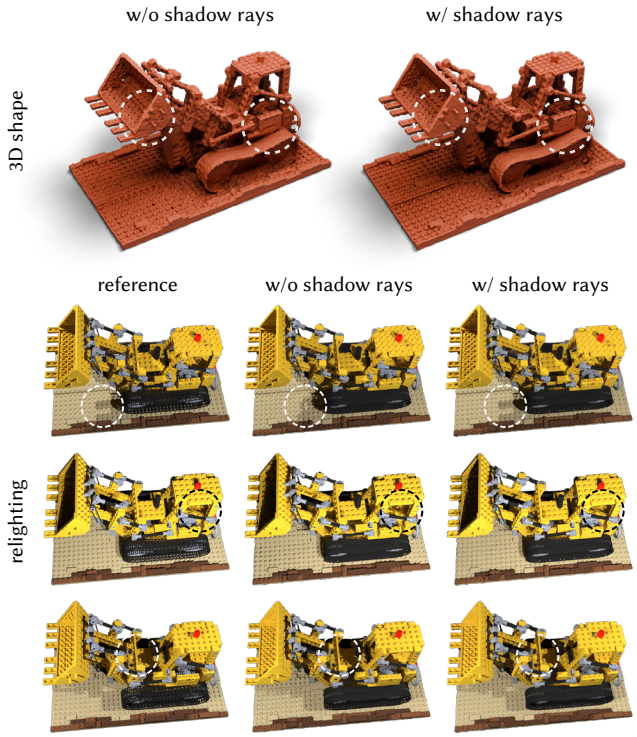

Figure 9. Comparison of extracted meshes and rendered images from training with and without shadow rays on the NeRF Realistic Synthetic Lego scene. Optimizing with shadow rays results in extracted meshes that have fewer artifacts and finer details. Additionally, it results in images rendered under novel lighting that have more accurate shadows.

8 LIMITATIONS AND DISCUSSION

We introduced the regularized 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 and reconstruction quality better than rasterization methods, while maintaining the generality afforded by ray tracing. Starting from dense structure-from-motion initialization, dipole sums additionally produce surface reconstructions of 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., Gaussian surfels [Dai et al. 2024] for point-based, and Voxurf [Wu et al. 2023] for

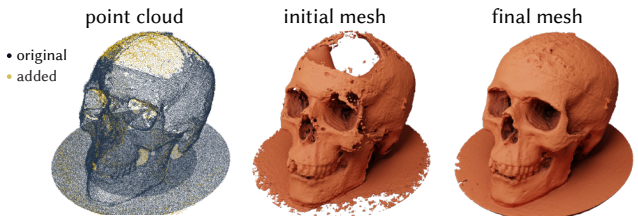

Figure 10. Visualization of original and added points on the point cloud for the DTU SKULL scene (left), and extracted meshes from the original (middle) and final (right) point clouds. Our point-growing method fills in regions with large gaps in the point cloud (e.g., top of the skull) and, together with optimized geometry attributes, fixes these gaps in the final extracted mesh.

grid-based) report difficulties producing accurate surface reconstructions in areas of strong specular appearance. Our method and Voxurf alleviate the issue using shallow MLPs to predict appearance from interpolated features, which inevitably introduces a computational overhead. Gaussian 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 method also uses ray tracing, it could adapt this approach by incorporating such features as point attributes rather than neural network outputs.

Dealing with large textureless regions. Our method directly uses the dense point cloud from structure-from-motion initialization, which makes it sensitive to artifacts such as large holes and missing surfaces in that point cloud (e.g., due to textureless regions where structure from motion fails). Our method mitigates these artifacts through the use of learnable per-point geometry attributes (Section 4.2) and point growing (Section 7.1), but the resulting reconstruction of very large textureless regions can still be noisy—Figure 10 shows an example. Adopting more elaborate point growing procedures from prior work [Xu et al. 2022; Kerbl et al. 2023] could enable our technique to more effective mitigate such artifacts.

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). 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]. In the future, it would be interesting to investigate combinations of dipole sums with other direct illumination [Bitterli et al. 2020] and global illumination [Pharr et al. 2023] algorithms, in both volume and surface rendering formulations.

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 methods [Beatson et al. 1997], and data-driven optimization of interpolation kernels [Chen et al. 2023; Ryan et al. 2022].

ACKNOWLEDGMENTS

We thank Keenan Crane, Rohan Sawhney, and Nicole Feng for many helpful discussions, and the authors of Dai et al. [2024]; Wang et al. [2023]; Li et al. [2023] for help running experimental comparisons. This work was supported by NSF award 1900849, NSF Graduate Research Fellowship DGE2140739, an NVIDIA Graduate Fellowship for Miller, and a Sloan Research Fellowship for Gkioulekas.

REFERENCES

Henrik Aanæs, Rasmus Ramsbøl Jensen, George Vogiatzis, Engin Tola, and Anders Bjorholm Dahl. 2016. Large-Scale Data for Multiple-View Stereopsis. *International Journal of Computer Vision* (2016).

Sameer Agarwal, Yasutaka Furukawa, Noah Snavely, Ian Simon, Brian Curless, Steven M Seitz, and Richard Szeliski. 2011. Building rome in a day. *Commun. ACM* (2011).

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 sdfs through reparameterization. In *ACM SIGGRAPH Asia Conference Papers*.

Gavin Barill, Neil G Dickson, Ryan Schmidt, David IW Levin, and Alec Jacobson. 2018. Fast winding numbers for soups and clouds. *ACM Trans. Graph.* 37, 4 (2018), 1–12.

Josh Barnes and Piet Hut. 1986. A hierarchical O (N log N) force-calculation algorithm. *Nature* (1986).

James Thomas Beale, Wenjun Ying, and Jason R Wilson. 2016. A simple method for computing singular or nearly singular integrals on closed surfaces. *Communications in Computational Physics* (2016).

Rick Beatson, Leslie Greengard, et al. 1997. A short course on fast multipole methods. *Wavelets, multilevel methods and elliptic PDEs* (1997).

Alexander Belyaev, Pierre-Alain Fayolle, and Alexander Pasko. 2013. Signed Lp-distance fields. *Computer-Aided Design* (2013).

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 *Eurographics State of the Art Reports*.

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: Relightable reconstructions from multi-view photometric images. In *European Conference on Computer Vision*. Springer.

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 Trans. Graph.* (2020).

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*.

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*.

Jiong Chen, Florian Schaefer, and Mathieu Desbrun. 2024. Lightning-fast Method of Fundamental Solutions. *ACM Trans. Graph.* (2024).

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 *IEEE/CVF International Conference on Computer Vision*.

Ricardo Cortez. 2001. The method of regularized Stokeslets. *SIAM Journal on Scientific Computing* (2001).

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* (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 Trans. Graph.* (2017).
Pinxuan Dai, Jiamin Xu, Wenxiang Xie, Xinguo Liu, Huamin Wang, and Weiwei Xu. 2024. High-quality Surface Reconstruction using Gaussian Surfels. In *ACM SIGGRAPH Conference Papers*.
Fernando de Goes and Mathieu Desbrun. 2024. Stochastic Computation of Barycentric Coordinates. *ACM Trans. Graph.* (2024).
Kangle Deng, Andrew Liu, Jun-Yan Zhu, and Deva Ramanan. 2022. Depth-supervised nerf: Fewer views and faster training for free. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*.
Lawrence C Evans. 2022. *Partial differential equations*. American Mathematical Society.
Nicole Feng, Mark Gillespie, and Keenan Crane. 2023. Winding Numbers on Discrete Surfaces. *ACM Trans. Graph.* (2023).
Michael S Floater, Géza Kós, and Martin Reimers. 2005. Mean value coordinates in 3D. *Computer Aided Geometric Design* (2005).
Gerald B Folland. 1995. *Introduction to partial differential equations*. Princeton University Press.
Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. 2022. Plenoxels: Radiance fields without neural networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*.
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*.
Simon Fuhrmann and Michael Goesele. 2014. Floating scale surface reconstruction. *ACM Trans. Graph.* (2014).
Mark Gillespie, Denise Yang, Mario Botsch, and Keenan Crane. 2024. Ray Tracing Harmonic Functions. *ACM Trans. Graph.* (2024).
Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep sparse rectifier neural networks. In *International Cconference on Artificial Intelligence and Statistics*.
Craig Gotsman and Kai Hormann. 2024. A Linear Method to Consistently Orient Normals of a 3D Point Cloud. In *ACM SIGGRAPH Conference Papers*.
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*.
John C Hart. 1996. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer* (1996).
Richard Hartley and Andrew Zisserman. 2003. *Multiple view geometry in computer vision*. Cambridge university press.
Jon Hasselgren, Nikolai Hofmann, and Jacob Munkberg. 2022. Shape, Light, and Material Decomposition from Images using Monte Carlo Rendering and Denoising. In *Advances in Neural Information Processing Systems*.
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*.
Kai Hormann and N Sukumar. 2017. *Generalized barycentric coordinates in computer graphics and computational mechanics*. CRC press.
Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger, and Shenghua Gao. 2024b. 2D Gaussian Splatting for Geometrically Accurate Radiance Fields. In *ACM SIGGRAPH Conference Papers*.
Zhangjin Huang, Yuxin Wen, Zihao Wang, Jinjuan Ren, and Kui Jia. 2024a. Surface reconstruction from point clouds: A survey and a benchmark. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2024).
Alec Jacobson, Ladislav Kavan, and Olga Sorkine-Hornung. 2013. Robust inside-outside segmentation using generalized winding numbers. *ACM Trans. Graph.* (2013).
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.* (2005).
Animesh Karnewar, Tobias Ritschel, Oliver Wang, and Niloy Mitra. 2022. Relu fields: The little non-linearity that could. In *ACM SIGGRAPH Conference Proceedings*.
Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. 2006. Poisson surface reconstruction. In *Eurographics Symposium on Geometry processing*.
Michael Kazhdan and Hugues Hoppe. 2013. Screened poisson surface reconstruction. *ACM Trans. Graph.* (2013).
Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 2023. 3d Gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.* (2023).
Mijeong Kim, Seonguk Seo, and Bohyung Han. 2022. Infonerf: Ray entropy minimization for few-shot neural volume rendering. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*.
Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*.
Pavel A Krutitskii. 2001. The jump problem for the Laplace equation. *Applied Mathematics Letters* (2001).
Fabian Langguth, Kalyan Sunkavalli, Sunil Hadap, and Michael Goesele. 2016. Shading-aware multi-view stereo. In *European Conference on Computer Vision*. Springer.
Zhaoshuo Li, Thomas Müller, Alex Evans, Russell H Taylor, Mathias Unberath, Ming-Yu Liu, and Chen-Hsuan Lin. 2023. Neuralangelo: High-Fidelity Neural Surface Reconstruction. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*.
Siyou Lin, Dong Xiao, Zuoqiang Shi, and Bin Wang. 2022. Surface Reconstruction from Point Clouds without Normals by Parametrizing the Gauss Formula. *ACM Trans. Graph.* (2022).
Jingwang Ling, Zhibo Wang, and Feng Xu. 2023. Shadowneus: Neural sdf reconstruction by shadow ray supervision. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*.
Yaron Lipman. 2021. Phase Transitions, Distance Functions, and Implicit Neural Representations. In *International Conference on Machine Learning*.
Matthew M Loper and Michael J Black. 2014. OpenDR: An approximate differentiable renderer. In *European Conference on Computer Vision*. Springer.
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*.
Wenjia Lu, Zuoqiang Shi, Jian Sun, and Bin Wang. 2018. Surface Reconstruction Based on the Modified Gauss Formula. *ACM Trans. Graph.* (2018).
Fujun Luan, Shuang Zhao, Kavita Bala, and Zhao Dong. 2021. Unified shape and svbrdf recovery using differentiable monte carlo rendering. In *Computer Graphics Forum*.
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* (1995).
Donald Meagher. 1982. Geometric modeling using octree encoding. *Computer graphics and image processing* (1982).
Gal Metzer, Rana Hanocka, Denis Zorin, Raja Giryes, Daniele Panozzo, and Daniel Cohen-Or. 2021. Orienting point clouds with dipole propagation. *ACM Trans. Graph.* (2021).
Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. 2021. NeRF: Representing scenes as neural radiance fields for view synthesis. *Commun. ACM* (2021).
Bailey Miller, Hanyu Chen, Alice Lai, and Ioannis Gkioulekas. 2024. Objects as volumes: A stochastic geometry view of opaque solids. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*.
Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.* (2022).
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 courses*.
Merlin Nimier-David, Sébastien Speierer, Benoît Ruiz, and Wenzel Jakob. 2020. Radiative backpropagation: An adjoint method for lightning-fast differentiable rendering. *ACM Trans. Graph.* (2020).
Michael Oechsle, Songyou Peng, and Andreas Geiger. 2021. Unisurf: Unifying neural implicit surfaces and radiance fields for multi-view reconstruction. In *IEEE/CVF International Conference on Computer Vision*.
Onur Özyeşil, Vladislav Voroninski, Ronen Basri, and Amit Singer. 2017. A survey of structure from motion. *Acta Numerica* (2017).
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* (2021).
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.
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* (2016).
Johannes Lutz Schönberger and Jan-Michael Frahm. 2016. Structure-from-Motion Revisited. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*.
Johannes L Schönberger, Enliang Zheng, Jan-Michael Frahm, and Marc Pollefeys. 2016. Pixelwise view selection for unstructured multi-view stereo. In *European Conference on Computer Vision*.
Noah Snavely, Steven M Seitz, and Richard Szeliski. 2006. Photo tourism: exploring photo collections in 3D. In *ACM SIGGRAPH papers*.

Noah Snavely, Steven M Seitz, and Richard Szeliski. 2008. Modeling the world from internet photo collections. *International journal of computer vision* (2008).
Jacob Spainhour, David Gunderman, and Kenneth Weiss. 2024. Robust Containment Queries over Collections of Rational Parametric Curves via Generalized Winding Numbers. *ACM Trans. Graph.* (2024).
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 Tretschk, Wang Yifan, Christoph Lassner, Vincent Sitzmann, Ricardo Martin-Brualla, Stephen Lombardi, et al. 2022. Advances in neural rendering. In *Computer Graphics Forum*.
Carlo Tomasi and Takeo Kanade. 1990. Shape and motion without depth. In *Proceedings of the DARPA Image Understanding Workshop*.
Shimon Ullman. 1979. The interpretation of structure from motion. *Proceedings of the Royal Society of London. Series B. Biological Sciences* (1979).
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 *IEEE/CVF Conference on Computer Vision and Pattern Recognition*.
Dor Verbin, Ben Mildenhall, Peter Hedman, Jonathan T Barron, Todd Zickler, and Pratul P Srinivasan. 2024. Eclipse: Disambiguating illumination and materials using unintended shadows. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*.
Delio Vicini, Sébastien Speierer, and Wenzel Jakob. 2021. Path replay backpropagation: Differentiating light paths using constant memory and linear time. *ACM Trans. Graph.* (2021).
Delio Vicini, Sébastien Speierer, and Wenzel Jakob. 2022. Differentiable signed distance function rendering. *ACM Trans. Graph.* (2022).
Ingo Wald, Sven Woop, Carsten Benthin, Gregory S Johnson, and Manfred Ernst. 2014. Embree: a kernel framework for efficient CPU ray tracing. *ACM Trans. Graph.* (2014).
Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. 2021a. NeuS codebase. https://github.com/Totoro97/NeuS.
Peng Wang, Lingjie 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* (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 *IEEE/CVF International Conference on Computer Vision*.
Chenglei Wu, Bennett Wilburn, Yasuyuki Matsushita, and Christian Theobalt. 2011. High-quality shape from multi-view stereo and shading under general illumination. In *IEEE Conference on Computer Vision and Pattern Recognition*.
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*.
Qiangeng Xu, Zexiang Xu, Julien Philip, Sai Bi, Zhixin Shu, Kalyan Sunkavalli, and Ulrich Neumann. 2022. Point-nerf: Point-based neural radiance fields. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*.
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 Trans. Graph.* (2023).
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. *IEEE/CVF Conference on Computer Vision and Pattern Recognition* (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).
Zehao Yu, Torsten Sattler, and Andreas Geiger. 2024. Gaussian opacity fields: Efficient and compact surface reconstruction in unbounded scenes. *arXiv preprint arXiv:2404.10772* (2024).
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* (2011).
Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. 2020. NeRF++: Analyzing and Improving Neural Radiance Fields. *arXiv:2010.07492* (2020).
Michael Zollhöfer, Angela Dai, Matthias Innmann, Chenglei Wu, Marc Stamminger, Christian Theobalt, and Matthias Nießner. 2015. Shading-based refinement on volumetric signed distance functions. *ACM Trans. Graph.* (2015).
Matthias Zwicker, Hanspeter Pfister, Jeroen Van Baar, and Markus Gross. 2002. EWA splatting. *IEEE Transactions on Visualization and Computer Graphics* (2002).

A PROOFS

We prove the two propositions we presented in Section 4.

A.1 Proof of Proposition 1

Poisson surface reconstruction computes a scalar field as the solution to the following Poisson equation [Kazhdan et al. 2006, Section 3]:[8]

$$\Delta\, \mathrm{u}(x) = -\nabla \cdot \mathrm{N}(x), \quad x \in \mathbb{R}^3, \tag{29}$$

where the *normal field* $\mathrm{N} : \mathbb{R}^3 \to \mathbb{R}^3$ in the right-hand side equals:

$$\mathrm{N}(x) \equiv \sum_{m=1}^{M} \phi_\varepsilon(x - \mathrm{p}_m)\, \mathrm{A}_m\, \mathrm{n}_m, \tag{30}$$

and ϕ_ε is the Gaussian function in Equation (13). As the domain of Equation (29) is $\mathbb{R}^3$, which is unbounded: 1. existence of a solution requires that the right-hand side term decays sufficiently fast, which is true for $\nabla \cdot \mathrm{N}$ thanks to the Gaussians in Equation (30); 2. uniqueness of that solution requires imposing a condition at infinity, and as u should approximate the indicator in Equation (6), the appropriate condition is that $|\mathrm{u}| \to 0$ as $\|x\| \to \infty$. Under these conditions, the solution of Equation (29) equals the *Newtonian potential* with moment $\nabla \cdot \mathrm{N}$ [Evans 2022, Section 2.2.1.b, Theorem 1]:

$$\mathrm{u}(x) = -\int_{\mathbb{R}^3} \mathrm{G}(x, y) \nabla_y \cdot \mathrm{N}(y)\, \mathrm{d}y. \tag{31}$$

From Equation (30), this solution becomes:

$$\mathrm{u}(x) = -\sum_{m=1}^{M} \mathrm{A}_m \overbrace{\int_{\mathbb{R}^3} \mathrm{G}(x, y) \nabla_y \cdot \phi_\varepsilon(y - \mathrm{p}_m)\, \mathrm{n}_m \, \mathrm{d}y}^{\equiv \mathrm{g}_m(x)}. \tag{32}$$

We consider each of the M integrals separately. Denoting by $\mathrm{B}(x, R) \subset \mathbb{R}^3$ the ball with center x and radius R, we have:

$$\begin{aligned}
\mathrm{g}_m(x) &= \lim_{R\to\infty} \int_{\mathrm{B}(x,R)} \mathrm{G}(x, y) \nabla_y \cdot \phi_\varepsilon(y - \mathrm{p}_m)\, \mathrm{n}_m \, \mathrm{d}y && (33)\\
&= \lim_{R\to\infty} \left\{ \int_{\partial \mathrm{B}(x,R)} \mathrm{G}(x, y) \phi_\varepsilon(y - \mathrm{p}_m)\, \mathrm{n}_m \cdot y - x/R \, \mathrm{d}A(y) \right.\\
&\qquad \left. - \int_{\mathrm{B}(x,R)} \nabla_y \mathrm{G}(x, y) \cdot \mathrm{n}_m\, \phi_\varepsilon(y - \mathrm{p}_m)\, \mathrm{d}y \right\} && (34)\\
&= 0 + \int_{\mathbb{R}^3} \nabla_x \mathrm{G}(x, y) \cdot \mathrm{n}_m\, \phi_\varepsilon(y - \mathrm{p}_m)\, \mathrm{d}y && (35)\\
&= \nabla_x \mathrm{G}_\varepsilon(x - \mathrm{p}_m) \cdot \mathrm{n}_m && (36)\\
&= -\mathrm{P}_\varepsilon(x, \mathrm{p}_m). && (37)
\end{aligned}$$

In this sequence: (33) reexpresses the unbounded integration domain; (34) uses integration by parts; (35) uses the distributive property of limits and the facts that $\mathrm{G}(x, y)\phi_\varepsilon(y - \mathrm{p}_m) = o(\|y - x\|^{-2})$ and $\nabla_y \mathrm{G}(x, y) = -\nabla_x \mathrm{G}(x, y)$; (36) follows from the definition in (12) and the properties of the Green's function; and (37) follows from the definition in (11). Then, from Equations (15), (32) and (37),

$$\mathrm{u}(x) = \sum_{m=1}^{M} \mathrm{A}_m\, \mathrm{P}_\varepsilon(x, \mathrm{p}_m) = \widetilde{\mathrm{w}}_\varepsilon(x). \tag{38}$$

This concludes our proof. We note two differences with the numerical implementation of PSR by Kazhdan et al. [2006]:

[8] The minus sign at the right-hand side is because we use outward normals, whereas Kazhdan et al. [2006] use inward ones.

1. To make the Poisson equation (29) amenable to a linear-system solver, Kazhdan et al. [2006] impose Dirichlet boundary conditions on a bounding volume of the point cloud. For the true indicator function in Equation (6), these conditions and our condition that $\mathrm{u} \to 0$ at infinity are equivalent. However, for point-cloud approximations, they are not equivalent and the choice between them is arbitrary [Kazhdan and Hoppe 2013, Section 4.4].
2. Kazhdan et al. [2006] suggest variable per-point standard deviations ε_m. Proposition 1 still holds in that case, except using ε_m in Equation (15). We comment on this suggestion in Section 4.2.

A.2 Proof of Proposition 2

Under the assumptions of Proposition 2, we have from Equation (7):

$$\mathbb{E}_{\{\mathrm{P}_m,\mathrm{N}_m\}_{m=1}^M}[\widetilde{\mathrm{w}}(x)] = \sum_{m=1}^M \mathrm{A}_m \, \mathbb{E}_{\mathrm{P}_m,\mathrm{N}_m}[\mathrm{P}(x,\mathrm{P}_m)] \tag{39}$$

$$= \sum_{m=1}^M \mathrm{A}_m \cdot \mathbb{E}_{\mathrm{P}_m}\Big[\mathbb{E}_{\mathrm{N}_m}[\mathrm{P}(x,\mathrm{P}_m) \,|\, \mathrm{P}_m]\Big] \tag{40}$$

$$= \sum_{m=1}^M \mathrm{A}_m \, \mathbb{E}_{\mathrm{P}_m}\Big[\mathbb{E}_{\mathrm{N}_m}[\mathrm{N}_m \nabla \mathrm{G}(x,\mathrm{P}_m) \,|\, \mathrm{P}_m]\Big] \tag{41}$$

$$= \sum_{m=1}^M \mathrm{A}_m \, \mathbb{E}_{\mathrm{P}_m}[\mathrm{f}_m \, \mathrm{n}_m \, \nabla \mathrm{G}(x,\mathrm{P}_m)] \tag{42}$$

$$= \sum_{m=1}^M \mathrm{A}_m \, \mathrm{n}_m \, \nabla \, \mathbb{E}_{\mathrm{P}_m}[\mathrm{G}(x,\mathrm{P}_m)] \, \mathrm{f}_m \tag{43}$$

$$= \sum_{m=1}^M \mathrm{A}_m \, \mathrm{n}_m \, \nabla \mathrm{G}_\varepsilon(x,\mathrm{p}_m) \, \mathrm{f}_m \tag{44}$$

$$= \sum_{m=1}^M \mathrm{A}_m \, \mathrm{P}_\varepsilon(x,\mathrm{p}_m) \, \mathrm{f}_m \tag{45}$$

$$= \widetilde{\mathrm{f}}_\varepsilon. \tag{46}$$

In this sequence: (39) follows from linearity of expectation; (40) follows from the law of total expectation; (41) follows from the definition in (9); (42) follows from the assumptions on N_m; (43) follows from the fact that differentiation and expectation commute; (45) follows from the definition in (11); and (46) follows from the definition in Equation (17). The only non-trivial step is (44). From the assumption that P_m is a Gaussian random variable, we have (up to a constant scale that we omit for simplicity):

$$\mathbb{E}_{\mathrm{P}_m}[\mathrm{G}(x,\mathrm{P}_m)] \propto \int_{y\in\mathbb{R}^3} \mathrm{G}(x,y) \exp\!\left(-\frac{\|x-y\|^2}{2\varepsilon^2}\right) \mathrm{d}y \tag{47}$$

$$\propto \int_{y\in\mathbb{R}^3} \mathrm{G}(x,y)\phi_\varepsilon(x-y) \, \mathrm{d}y \tag{48}$$

$$= \mathrm{G}_\varepsilon(x,y), \tag{49}$$

where (48) follows from the definition in (13). The step (49) follows from the fact that (46) is equivalent, by the properties of the Green's function, to the solution of the partial differential equation in (12).

B ENTROPY LOSS

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

$$p^{\mathrm{ff}}_{o,v}(\tau) \equiv \exp\!\left(-\int_0^\tau \sigma\big(r_{o,v}(t),v\big)\,\mathrm{d}t\right)\sigma\big(r_{o,v}(\tau),v\big), \tag{50}$$

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 $\tau_\mathrm{n} = \tau_0 < \cdots < \tau_J = \tau_\mathrm{f}$:

$$p_j \equiv \exp\!\left(-\sum_{i=1}^j \sigma_i \Delta_i\right)\big(1-\exp\big(\sigma_j\Delta_j\big)\big). \tag{51}$$

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

$$H(o,v) \equiv -\sum_{j=1}^J p_j \log p_j. \tag{52}$$

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

C BACKPROPAGATION DETAILS

As in Section 6, throughout this section we use b as a stand-in for any of the *moment attributes* stored in $\mathcal{P}$—namely, the geometry attribute f and the appearance attributes ℓ^k, $k = 1,\ldots,K$. As we discuss in Section 6.3, in practice we implement the backpropagation operations in Equations (25) and (53) for all these attributes as vector operations updating all attributes in parallel.

Backpropagation to nodes. An adjoint query backpropagates a derivative $\widetilde{\mathrm{db}}_\varepsilon(x)$—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 $\widehat{\mathrm{db}}_t$ by an amount that follows from differentiating Equation (26):

$$\widehat{\mathrm{A}}_t \, \mathrm{S}\!\left(\frac{\|\widehat{\mathrm{p}}_t - x\|}{\varepsilon}\right) \frac{\widehat{\mathrm{p}}_t - x}{\|\widehat{\mathrm{p}}_t - x\|^3} \cdot \widetilde{\mathrm{db}}_\varepsilon(x). \tag{53}$$

Second-stage backpropagation to leaf nodes. This stage backpropagates accumulated gradient attributes $\widehat{\mathrm{db}}_t$ from all nodes to leaf nodes corresponding to individual points $\mathrm{p}_m, m = 1,\ldots,M$ in $\mathcal{P}$. 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:

$$\mathrm{db}_m = \sum_{t\in\mathcal{A}(m)} \frac{\mathrm{A}_m}{\widehat{\mathrm{A}}_t} \, \mathrm{n}_m \, \widehat{\mathrm{db}}_t. \tag{54}$$

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 (54) as a matrix-vector multiplication that has negligible cost.