3D Reconstruction with Fast Dipole Sums

Hanyu Chen

CMU-CS-24-102

April 2024

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Ioannis Gkioulekas (Chair)
Matthew O'Toole

Submitted in partial fulfillment of the requirements
for the degree of Master of Science.

Keywords: Computer graphics, computer vision, rendering, 3D reconstruction

Abstract

Reconstructing 3D scenes from multi-view images has always been a challenging problem in computer vision and computer graphics. Traditional methods like structure from motion and multi-view stereo have been widely used for pose estimation and dense point cloud reconstruction. However, these methods have limited ability to reconstruct complex scenes with fine details. Recently, since the introduction of neural radiance fields (NeRF), volumetric neural rendering has shown great promise in reconstructing complex scenes with high fidelity. To accurately reconstruct scene geometry, other works have also proposed ways to directly model the signed-distance function or occupancy of a scene. However, these methods are often slow to train and cannot effectively leverage known scene information.

In this thesis, we propose a novel point-based representation that combines the efficiency of point clouds with the expressiveness of neural rendering. Point clouds are particularly appealing as a scene representation for rendering tasks, as they are the natural output of many 3D sensing modalities, including structure from motion, multi-view stereo, and lidar. They also come with a rich library of geometric queries. In our work, we utilize point clouds to efficiently reconstruct 3D scenes by using the generalized winding number as a proxy for the scene occupancy and by interpolating per-point neural features with appropriate kernels. We leverage the Barnes-Hut approximation and fast dipole sums to perform fast winding number queries and feature interpolation, as well as logarithmic complexity backpropagation for efficient differentiable rendering. We empirically show that our method consistently outperforms existing methods in both reconstruction quality and efficiency on a wide range of real-world scenes.

Acknowledgments

First and foremost, I would like to thank my advisor, Prof. Ioannis Gkioulekas. His guidance and support throughout my time working with him have been invaluable to my research and academic growth. I would also like to thank his Ph.D. student Bailey Miller for his help and insights on the project. Without them, the project would not have been possible.

Furthermore, I am also grateful for Prof. Matthew O'Toole for being on the thesis committee and his valuable feedback on the project.

Contents

List of Figures

List of Tables

Chapter 1

Related work

In this chapter, we review the existing literature on 3D reconstruction, neural rendering, and point-based representations, which are the key components of our work.

1.1 3D reconstruction

Traditionally, structure-from-motion (SfM) [29] and multi-view stereo (MVS) [30] have been two widely used methods for 3D reconstruction from images. SfM estimates camera poses and sparse 3D points from a set of images, while MVS estimates dense 3D points by triangulating points from multiple known views. However, they are limited in their ability to reconstruct fine details and handle complex scenes.

More recently, neural rendering has shown great promise in reconstructing complex scenes with high fidelity. In particular, neural radiance fields (NeRF) [25] uses neural networks to map spatial locations and viewing directions to the emitted radiance and the attenuation coefficient. Instant-NGP [27] uses a multi-resolution hash grid to store neural features that can be efficiently trained to model scene appearance. However, although these methods are able to render complex scenes with high fidelity, they are not well-suited for surface extraction as they do not explicitly model the geometry of the scene.

Other works following NeRF have proposed ways to directly model the geometry of the scene and connect the geometry of the scene to the attenuation coefficient for volume rendering. For example, NeuS [33] proposes directly modeling the occupancy of the scene, while VolSDF [38] proposes modeling the signed-distance function of the scene. These methods allows for more accurate surface extraction.

1.2 Scene representation

Building on top of these neural rendering works, other methods have proposed different ways to represent the scene to either improve the efficiency or quality of 3D reconstruction.

For example, for novel view synthesis tasks, methods like ReLU fields [15] and Plenoxels [28] model the scene using density values and spherical harmonics coefficients stored on a voxel grid, while Point-NeRF [36] optimizes neural features stored on a point cloud. 3D Gaussian

splatting [18] also proposes a point-based representation that directly interpolates density and spherical harmonics coefficients using Gaussian kernels.

For surface extraction, Neuralangelo [21] and NeuS2 [34] propose using multi-resolution hash grids to model the scene geometry, which allows for representing varying levels of detail. Geo-NeuS [9] and Neuralwarp [6] propose enforcing photo-consistency constraints on the scene geometry based on a sparse point cloud obtained from structure-from-motion. Voxurf [35] proposes modeling the scene with a sparse voxel grid to improve efficiency.

1.3 Point clouds

As the natural output of many 3D sensing modalities, point clouds are very commonly used in traditional 3D reconstruction pipelines. For example, Poisson surface reconstruction [17] is widely used for surface reconstruction from point clouds. The generalized winding number [12] can also be used for robust inside-outside queries on point clouds.

In recent neural rendering based methods, point clouds are more often used to impose constraint on the scene geometry or as part of a regularization term. Examples of such work include DS-NeRF [7], which uses depth information from point clouds as supervision for training neural radiance fields, as well as Geo-NeuS and Neuralwarp, as mentioned above, which use sparse point clouds to enforce photo-consistency constraints.

On the other hand, although works like Point-NeRF directly optimizes neural features stored on a point cloud, the features are aggregated using multiple MLPs that limits both efficiency and interpretability. Methods following 3D Gaussian splatting provide an intriguing alternative of directly interpolating features using Gaussian kernels. However, such methods are rasterization-based and are not well-suited for surface extraction and relighting tasks.

Chapter 2

Volumetric neural rendering

2.1 Volumetric light transport background

We first introduce the basic concepts of volumetric light transport using principles from classical volume rendering [14], which will serve as the foundation for our rendering framework.

2.1.1 Radiative transfer equation

The *radiative transfer equation* (RTE) describes the propagation of light in a medium. It is a partial differential equation that models the change in radiance along a ray as it travels through the medium. Assuming an emissive medium with no scattering, the RTE can be written as:

$$dL(\mathbf{x}, \vec{\omega}) = -\sigma(\mathbf{x}, \vec{\omega})L(\mathbf{x}, \vec{\omega})dz + \sigma(\mathbf{x}, \vec{\omega})L_e(\mathbf{x}, \vec{\omega})dz, \tag{2.1}$$

where $L(\mathbf{x}, \vec{\omega})$ is the radiance at point $\mathbf{x}$ in direction $\vec{\omega}$, $L_e(\mathbf{x}, \vec{\omega})$ is the emitted radiance of the medium, and $\sigma(\mathbf{x}, \vec{\omega})$ is the (direction-dependent) attenuation coefficient.

2.1.2 Volume rendering equation

The solution to the RTE is given by the volume rendering equation:

$$L(\mathbf{x}, \vec{\omega}) = \mathrm{T}(\mathbf{x}, \mathbf{x}_z)L(\mathbf{x}_z, \vec{\omega}) + \int_0^z \mathrm{T}(\mathbf{x}, \mathbf{x}_t)\sigma(\mathbf{x}_t, \vec{\omega})L_e(\mathbf{x}_t, \vec{\omega})dt, \tag{2.2}$$

where $\mathrm{T}(\mathbf{x}, \mathbf{x}_t)$ is the transmittance from $\mathbf{x}$ to $\mathbf{x}_t$, given by:

$$\mathrm{T}(\mathbf{x}, \mathbf{x}_t) = \exp\left(-\int_0^t \sigma(\mathbf{x}_s, \vec{\omega})ds\right). \tag{2.3}$$

Notably, we can also consider the probability distribution of the ray termination distance, or the *free-flight distribution*, whose probability density function is given by the product of the transmittance and the attenuation coefficient at the termination point:

$$p_{\mathbf{x},\vec{\omega}}(z) = \sigma(\mathbf{x}_z, \vec{\omega})\mathrm{T}(\mathbf{x}, \mathbf{x}_z). \tag{2.4}$$

Assuming no background light source and near and far bounds t_n and t_f, we can then write the expected color of a camera ray $\mathbf{r}(t) = \mathbf{o} + t\vec{\omega}$ as

$$C(\mathbf{r}) = \int_{t_n}^{t_f} \sigma(\mathbf{r}(t), \vec{\omega}) L_e(\mathbf{r}(t), \vec{\omega}) \exp\left(-\int_{t_n}^{t} \sigma(\mathbf{r}(s), \vec{\omega}) ds\right) dt \tag{2.5}$$

$$= \int_{t_n}^{t_f} p_{\mathbf{o},\vec{\omega}}(t) L_e(\mathbf{r}(t), \vec{\omega}) dt \tag{2.6}$$

2.1.3 Discretization

In practice, as proposed by Max [23], when rendering a single ray, the volume rendering equation is often discretized by sampling the ray at regular intervals and approximating the integral with quadrature methods.

Given a ray $\mathbf{r}(t) = \mathbf{o} + t\vec{\omega}$ and discrete samples $t_0, t_1, \ldots, t_N$ along the ray, the color of the ray is approximated as

$$C(\mathbf{r}) \approx \sum_{i=0}^{N-1} p_i L_e(\mathbf{r}(t_i), \vec{\omega}), \tag{2.7}$$

where the free-flight distribution is also approximated as

$$p_i = \alpha_i \sum_{j=0}^{i-1} (1 - \alpha_j), \quad \alpha_i = 1 - \exp\left(-\sigma(\mathbf{r}(t_i), \vec{\omega}) \Delta t_i\right). \tag{2.8}$$

Notably, this reduces volume rendering to alpha compositing, where the color of the ray is accumulated by blending the color of each sample with the accumulated color.

Rendering a scene with the volume rendering equation requires knowing the attenuation coefficient σ and emitted radiance L_e at every point in the scene. Traditionally, these quantities are estimated using physical measurements and known material properties. However, more recently, neural networks have been used as a tool to directly estimate these quantities from images of the scene, as we will discuss in the next section.

2.2 Neural rendering

Recent works following the introduction of neural fields (NeRF) [25] have shown that neural networks are capable of representing complex scenes and rendering them with high fidelity. We briefly review the key concepts of neural rendering, which we will build upon in our work.

2.2.1 Neural fields

Neural fields are neural networks, or functions $f_\Theta : (\mathbf{x}, \vec{\omega}) \rightarrow (c, \sigma)$, that map a 3D spatial location $\mathbf{x} = (x, y, z)$ and a 2D viewing direction $\vec{\omega} = (\theta, \phi)$ to the emitted radiance and the attenuation coefficient at the given spatial location in the given viewing direction [25].

Neural fields are often trained to minimize the error between rendered pixels and ground truth pixels, which are obtained by sampling from a dataset of captured images of the scene. The

trained neural field can then be used to render novel views of the scene by sampling camera rays and evaluating the neural field at the desired spatial location and viewing directions.

Despite its success in rendering complex scenes, neural fields have limitations in extracting surfaces from the scene, as they do not explicitly model the geometry of the scene. Naively, one can also use neural fields to directly extract surfaces from the scene by thresholding the attenuation coefficient σ at a certain value. However, this approach is not ideal for extracting surfaces, and often results in incorrect or noisy surfaces.

To address this limitation, other works have proposed ways to directly model the geometry of the scene and connect the geometry of the scene to the attenuation coefficient for volume rendering, as we will discuss in the next section.

2.2.2 Neural surface representations

To address the limitations of using the attenuation coefficient to extract surfaces, works including [26, 33, 38] have proposed ways to directly model the *signed-distance function* (SDF) or *occupancy* of a scene and convert them into attenuation coefficients for volume rendering. These works have shown that neural surface representations can be used to both render high-quality images of scenes and extract surfaces from the scene, by using marching cubes [22] to find the 0.5-crossing of the occupancy, for example.

Specifically, given the occupancy of a scene, we adopt the method proposed by Miller et al. [26] and compute the attenuation coefficients by the formula

$$\sigma(\mathbf{x}, \vec{\omega}) = \frac{|\vec{\omega} \cdot \nabla o(\mathbf{x})|}{1 - o(\mathbf{x})}, \tag{2.9}$$

where $o : \mathbb{R}^3 \to [0, 1]$ is the occupancy function that represents the probability of a point in space being inside of an object. We further discuss how these concepts relate to our representation in section 3.1.

The original discrete formulation makes the assumption that the attenuation coefficient is constant within each discrete segment of the ray. As proposed by both Wang et al. [33] and Miller et al. [26], we can relax this assumption to allow for non-constant but monotone attenuation coefficients within each segment. Then, we can directly estimate α_i from the occupancy as

$$\alpha_i = \frac{|o(\mathbf{r}(t_i)) - o(\mathbf{r}(t_{i+1}))|}{1 - \min\{o(\mathbf{r}(t_i)), o(\mathbf{r}(t_{i+1}))\}}. \tag{2.10}$$

This alternative formulation also avoids the need to explicitly compute the attenuation coefficient and thus the gradient of the occupancy function, which requires additional computation and are often noisy.

Chapter 3

Winding number and dipole sums

3.1 The winding number

To lay the foundation for our point-based representation, we begin with an introduction on the winding number for surfaces and point clouds, which we will generalize in 3.2 for our implicit surface representation.

3.1.1 Winding number for surfaces

We first consider a continuous surface $\Gamma \subset \mathbb{R}^3$. There are many equivalent definitions of the winding number [8]; we follow Barill et al. [1] and use its definition as a *jump harmonic* scalar field. Then, the *winding number* $\mathrm{w} : \mathbb{R}^3 \to \mathbb{R}$ is the scalar field solution to the Laplace boundary value problem (BVP) with jump Dirichlet and Neumann boundary conditions:

$$\triangle \, \mathrm{w}(x) = 0 \text{ in } \mathbb{R}^3 \setminus \Gamma, \tag{3.1}$$

$$\mathrm{w}^+(x) - \mathrm{w}^-(x) = 1 \text{ on } \Gamma, \tag{3.2}$$

$$\partial \mathrm{w}^+ / \partial n(x) - \partial \mathrm{w}^- / \partial n(x) = 0 \text{ on } \Gamma. \tag{3.3}$$

Here, $n(x)$ is the normal at point $x \in \Gamma$, and $\mathrm{w}^{\pm}(x) \equiv \lim_{\varepsilon \to 0} \mathrm{w}(x \pm \varepsilon \cdot n(x))$ are the winding number values on either side of the surface Γ along the normal direction. Krutitskii [20] provide a detailed treatment of such BVPs, and in particular prove the following *boundary integral* expression for their solution:

$$\mathrm{w}(x) = \int_\Gamma \mathrm{P}(x, y) \cdot 1 \, \mathrm{d}\sigma(y), \qquad \mathrm{P}(x, y) \equiv \frac{1}{4\pi} \frac{\langle n(y), \widehat{xy} \rangle}{\|x - y\|^2}, \tag{3.4}$$

where $\widehat{xy} \equiv y - x / \|yx\|$ is the direction from x to y, and $\mathrm{P} : \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}$ is the *free-space Poisson kernel* for the Laplacian partial differential equation (PDE). We make explicit the factor 1 in the integral of 3.4, corresponding to the jump Dirichlet boundary condition 3.2, for reasons we will explain in 3.2.

When the surface Γ is the watertight boundary of one or more three-dimensional objects, then the winding number equals their binary *indicator function*—$\mathrm{w}(x) = 1$ for points x at the objects' interior, $\mathrm{w}(x) = 0$ otherwise.

3.1.2 Winding number for point clouds

We now consider an *oriented* point cloud $\mathcal{P} \equiv \{(p_m, n_m, A_m)\}_{m=1}^{M}$, where for each m we assume that:

1. the point p_m is a sample from an underlying surface Γ;
2. the vector n_m is the normal of Γ at p_m; and
3. the scalar A_m is the geodesic Voronoi area on Γ of p_m, i.e., the area of the subset of Γ where points are closer (in the geodesic distance sense) to p_m than any other point in the point cloud.

In practice, if only the points p_m are available, we can estimate the normals n_m and area weights A_m using standard techniques (e.g., by fitting and Voronoi-tessellating a plane to each point's k-nearest neighbors [11, 31]).

As Barill et al. [1] explain, the boundary integral representation 3.4 directly suggests the following generalization of the winding number for point clouds:

$$\mathrm{w}_{\mathrm{pc}}(x) \equiv \sum_{m=1}^{M} A_m \, \mathrm{P}(x, p_m) \cdot 1 = \sum_{m=1}^{M} \frac{A_m}{4\pi} \frac{\langle n_m, \widehat{x p_m} \rangle}{\|x - p_m\|^2} \cdot 1. \tag{3.5}$$

Barill et al. [1] show that w_{pc} is a non-binary scalar field that approaches $1/2$ at points near the boundary of the continuous surface Γ underlying the point cloud $\mathcal{P}$, increases towards its interior, and decreases towards its exterior. Thus, the $1/2$-level set of w_{pc} is an implicit surface that approximates Γ; this approximation becomes exact as point density converges to infinity, and degrades gracefully as the number M of points decreases.

3.1.3 Barnes-Hut approximation

Evaluating the winding number $\mathrm{w}_{\mathrm{pc}}(x)$ at a query point x using 3.5 has linear complexity $O(M)$ with respect to the point cloud size M; for large point clouds, doing so can be exceedingly expensive, especially if we need to query w_{pc} at multiple points (as we will later in this section). Barill et al. [1] show how to compute $\mathrm{w}_{\mathrm{pc}}(x)$ with logarithmic complexity using the *Barnes-Hut fast summation method* [2]. This method first creates a tree data structure (e.g., octree [24]) whose nodes hierarchically subdivide points in the point cloud into clusters, with leaf nodes corresponding to individual points. Each tree node t has a centroidal location, area-weighted normal, and radius

$$\tilde{p}_t \equiv \frac{\sum_{m \in \mathcal{L}(t)} A_m p_m}{\sum_{m \in \mathcal{L}(t)} A_m}, \qquad \tilde{n}_t \equiv \sum_{m \in \mathcal{L}(t)} A_m n_m \cdot 1, \qquad \tilde{r}_t \equiv \max_{m \in \mathcal{L}(t)} \|p_m - \tilde{p}_t\|. \tag{3.6}$$

where $\mathcal{L}(t)$ is the set of leaf (i.e., single-point) nodes that are successors of t in the tree hierarchy. We purposefully include an additional factor of 1 in the are-weighted normal computation, for reasons we will explain in 3.2.3.

Then, for each query point x, the Barnes-Hut methods performs a depth-first tree traversal; at each node t, if x is sufficiently far from its centroid (i.e., $\|x - \tilde{p}_t\| > \beta \tilde{r}_t$), the node's successors

are not visited and the sum of contributions to $\mathrm{w}_{\mathrm{pc}}(x)$ from all leaves in $\mathcal{L}(t)$ is approximated as:

$$\sum_{m \in \mathcal{L}(t)} A_m \, \mathrm{P}(x, p_m) \cdot 1 \approx \widetilde{\mathrm{P}}(x, \tilde{p}_t) \equiv \frac{1}{4\pi} \frac{\langle \tilde{n}_t, \widehat{x\tilde{p}_t} \rangle}{\|x - \tilde{p}_t\|^2}. \tag{3.7}$$

This approximation expresses the fact that, due to the squared-distance falloff of the Poisson kernel in 3.4, the *far-field* influence of a cluster of points can be represented by a single point at the cluster's centroid. We also note that the area weight is not included in the far-field approximation, as it is already accounted for in the normal computation in 3.6.

3.1.4 Rendering point clouds with winding numbers

Barill et al. [1] show that efficient winding number queries facilitate several point cloud operations, e.g., meshing, inside-outside tests, and Boolean composition. Our goal in this paper is to show that, with appropriate modifications (3.2), these queries facilitate also forward and inverse rendering of geometry represented as point clouds.

Specifically, we can use the winding number w_{pc} to define an implicit surface Γ_{pc}

$$\Gamma_{\mathrm{pc}} \equiv \left\{ x \in \mathbb{R}^3 : \mathrm{w}_{\mathrm{pc}}(x) = 1/2 \right\}, \tag{3.8}$$

and, by naturally viewing the winding number as an occupancy function and following Miller et al. [26, Equation (12)], we can compute a volumetric attenuation coefficient σ_{pc} as:

$$\sigma_{\mathrm{pc}}(x, \omega) \equiv \frac{|\omega \cdot \nabla \, \mathrm{w}_{\mathrm{pc}}(x)|}{1 - \mathrm{w}_{\mathrm{pc}}(x)} \tag{3.9}$$

Then, for surface rendering, we can perform ray casting and visibility queries on the point cloud, by intersecting the isosurface Γ_{pc} using ray marching [10]. Likewise, for volumetric rendering, we can compute free-flight distribution and transmittance queries through the point cloud, by accumulating the coefficient σ_{pc} along a ray. All these ray operations use only the point cloud attributes, and do not require meshing or using a proxy (e.g., grid or neural) for the point cloud. Additionally, though each ray operation requires winding number queries at multiple ray points, they remain efficient thanks to the Barnes-Hut method. Lastly, backpropagating through the expressions for σ_{pc}, and w_{pc} to update point cloud parameters is straightforward and efficient, as we discuss in 3.3.

Unfortunately, despite these attractive properties, the winding number w_{pc}—and associated isosurface Γ_{pc} and attenuation coefficient σ_{pc}—have a few critical shortcomings that make them unsuitable for direct use in rendering applications. We explain these shortcomings, and how to overcome them, in the next section.

3.1.5 Relationship to Poisson surface reconstruction

Before we continue, we remark on a relationship between the point cloud winding number w_{pc} and Poisson surface reconstruction [16, 17]. As Barill et al. [1] explain, both techniques compute, from an oriented point cloud, a scalar field that approximates the true winding number,

corresponding to the solution of BVP (3.1, 3.2, 3.3) for the continuous surface underlying the point cloud. The *limit* behaviors of the two scalar fields are equivalent. However, whereas the approximation of 3.5 can be efficiently computed directly from the point cloud, the approximation by Poisson surface reconstruction requires solving an expensive Poisson integration problem, making it impractical for forward and (especially) inverse rendering applications. Using w_{pc} allows us to efficiently render an approximation to the implicit surface output by Poisson surface reconstruction, without the need for a Poisson solver.

3.2 Fast dipole sums

We introduce a generalization of 3.5 that will serve as our point-based representation for both geometry and radiance in inverse rendering applications. We first derive our generalization, then explain its advantages.

3.2.1 General Dirichlet conditions

We generalize the BVP (3.1, 3.2, 3.3) to use an arbitrary *Dirichlet data* function $\mathrm{f} : \Gamma \to \mathbb{R}$ for the jump Dirichlet boundary condition:

$$\triangle\, \mathrm{u}(x) = 0 \text{ in } \mathbb{R}^3 \setminus \Gamma, \tag{3.10}$$

$$\mathrm{u}^+(x) - \mathrm{u}^-(x) = \mathrm{f}(x) \text{ on } \Gamma, \tag{3.11}$$

$$\partial \mathrm{u}^+/\partial n(x) - \partial \mathrm{u}^-/\partial n(x) = 0 \text{ on } \Gamma. \tag{3.12}$$

We also augment the point cloud $\mathcal{P} \coloneqq \{(p_m, n_m, A_m, \mathrm{f}_m)\}_{m=1}^M$ to include the Dirichlet data as a per-point attribute, $\mathrm{f}_m \equiv \mathrm{f}(p_m)$. We will use u^{f} to denote the solution to this BVP for specific Dirichlet data f. Then, we can modify (3.4, 3.5) to express u^{f} and its point-based approximation as [20]:

$$\mathrm{u}^{\mathrm{f}}(x) \equiv \int_\Gamma \mathrm{P}(x,y) \cdot \mathrm{f}(y)\, \mathrm{d}\sigma(y), \qquad \mathrm{u}^{\mathrm{f}}_{\mathrm{pc}}(x) \equiv \sum_{m=1}^{M} A_m\, \mathrm{P}(x, p_m) \cdot \mathrm{f}_m\,. \tag{3.13}$$

3.2.2 Regularized Poisson kernel

The Poisson kernel $\mathrm{P}(x,y)$ is singular as $x \to y$; this makes the value of $\mathrm{u}^{\mathrm{f}}_{\mathrm{pc}}$ at locations x near a point p_m in the point cloud numerically unstable, and undefined at p_m.

To overcome this issue, we use the method of *regularized fundamental solutions* developed in PDE simulation [3, 4, 5] to address similar numerical issues from these singularities. Its starting point is the definition of the Poisson kernel through the *Green's function* (or *fundamental solution*) $\mathrm{G} : \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}$ of the Laplace PDE:

$$\mathrm{P}(x,y) \equiv n(y) \cdot \nabla_x\, \mathrm{G}(x,y) \text{ where G satisfies } \triangle\, \mathrm{G}(x,y) = \delta(x-y), \tag{3.14}$$

where δ is the Dirac delta distribution in $\mathbb{R}^3$. The method of regularized fundamental solutions replaces δ with a *nascent delta function*, that is, a function $\phi_\varepsilon(x-y)$ satisfying $\lim_{\varepsilon\to 0} \phi_\varepsilon(x-y) =$

$\delta(x-y)$. Then, we can define the regularized Green's function G_ε and Poisson kernel P_ε exactly analogously to 3.14:

$$\mathrm{P}_\varepsilon(x,y) \equiv n(y)\cdot\nabla_x \mathrm{G}_\varepsilon(x,y) \text{ where } \mathrm{G}_\varepsilon \text{ satisfies } \triangle \mathrm{G}_\varepsilon(x,y) = \phi_\varepsilon(x-y), \tag{3.15}$$

where it follows that $\lim_{\varepsilon\to 0}\mathrm{G}_\varepsilon = \mathrm{G}$ and $\lim_{\varepsilon\to 0}\mathrm{P}_\varepsilon = \mathrm{P}$. A common nascent delta function is the Gaussian function $\phi_\varepsilon(x-y) \equiv {}^1\!/_{\varepsilon\sqrt{2\pi}}\cdot\exp\left(-{}^{\|x-y\|^2}\!/_{2\varepsilon^2}\right)$. The corresponding regularized Poisson kernel equals [3]:

$$\mathrm{P}_\varepsilon(x,y) \equiv \mathrm{P}(x,y)\cdot \mathrm{S}\left(\frac{\|x-y\|}{\varepsilon}\right), \text{ where } \mathrm{S}(t) \equiv \mathrm{erf}(t) - \frac{2}{\sqrt{\pi}}\cdot t\cdot\exp\left(-t^2\right). \tag{3.16}$$

The parameter ε provides control between regularization (restricting how fast P_ε increases as it approaches singularity) and bias (controlling the difference between P_ε and P).

3.2.3 Regularized dipole sum

As shown in 3.5 and 3.13, the point-cloud winding number is the solution to the BVP for constant unit data, i.e., $\mathrm{w}_{\mathrm{pc}}(x) = \mathrm{u}^1_{\mathrm{pc}}(x)$. However, as explained above, naively using the Poisson kernel for the dipole sum in 3.17 leads to numerical instability during both optimization and mesh extraction, as the kernel is singular near the surface. The regularized dipole sum in 3.17 overcomes this issue by using the regularized Poisson kernel, which is always finite and provides a stable approximation to the true dipole sum. Moreover, in practice, using constant unit data for the dipole sum is also not always ideal, as it is not robust to noisy point clouds and gaps and holes in the point cloud.

Therefore, we instead use the regularized dipole sum with non-unit Dirichlet data,

$$\mathrm{u}^{\mathrm{f}}_{\mathrm{pc},\varepsilon}(x) \equiv \sum_{m=1}^{M} A_m\,\mathrm{P}_\varepsilon(x,p_m)\cdot \mathrm{f}_m, \tag{3.17}$$

where ε is the regularization strength and f_m is the per-point Dirichlet data, both of which are learnable parameters. This allows us to accurately represent the underlying geometry of the point cloud, while also providing a stable and robust representation for rendering applications.

We note that although this dipole sum is written as a sum over all points in the point cloud, introducing the regularization term and non-unit Dirichlet data does not prevent us from using the Barnes-Hut method to efficiently compute the dipole sum with logarithmic complexity. Indeed, the regularized dipole sum can be computed using the same tree data structure but with a modified area-weighted normal that also takes into account the non-unit Dirichlet data:

$$\tilde{n}_t \equiv \sum_{m\in\mathcal{L}(t)} A_m n_m\cdot \mathrm{f}_m \tag{3.18}$$

The contributions from the leaf nodes in the far-field approximation are approximated as:

$$\sum_{m\in\mathcal{L}(t)} A_m\,\mathrm{P}_\varepsilon(x,p_m)\cdot \mathrm{f}_m \approx \widetilde{\mathrm{P}}_\varepsilon(x,\tilde{p}_t) \equiv \frac{1}{4\pi}\frac{\langle \tilde{n}_t, \widehat{x\tilde{p}_t}\rangle}{\|x-\tilde{p}_t\|^2}\,\mathrm{S}\left(\frac{\|x-\tilde{p}_t\|}{\varepsilon}\right). \tag{3.19}$$

Again, both the area weights and the Dirichlet data are already accounted for when computing the area-weighted normal in 3.18, and are not included in the far-field approximation in 3.19. Crucially, this is *not* the same as separately aggregating area weights and Dirichlet data for the node; instead, this correctly computes the summed contribution from the leaf nodes under the approximation that they are concentrated at the centroid of the node.

3.2.4 Rendering with regularized dipole sums

Finally, we introduce two modifications that makes the regularized dipole sum suitable for rendering applications.

First, although for watertight surfaces, the winding number is a binary indicator function, in practice, when discretized on a potentially noisy point cloud, the winding number can easily take on values greater than 1 or less than 0. To ensure that the occupancy function derived from the winding number is well-defined, we apply a *sigmoid* function to the regularized dipole sum:

$$\mathrm{o}(x) \equiv \Phi_s\left(\mathrm{u}^{\mathrm{f}}_{\mathrm{pc},\varepsilon}(x) - \frac{1}{2}\right), \qquad \Phi_s(x) \equiv \frac{1}{1+\exp(-sx)}, \tag{3.20}$$

where Φ_s is the logistic sigmoid with an additional scale parameter s, which maps the regularized dipole sum to the range $[0, 1]$.

The point cloud winding number does not guarantee a sharp transition between the inside and outside of the surface, which is crucial for representing sharp geometric features. The scale parameter of the sigmoid function allows us to control the sharpness of the transition without affecting the geometry of the underlying implicit surface, as the $1/2$-level set remains unchanged.

Second, in addition to per-point Dirichlet data used to represent geometry, we can similarly store per-point neural features to also model the appearance of the scene. However, unlike for geometry, we do not want sharp discontinuity in the appearance of the scene near the surface. To address this, we use a modified version of the Poisson kernel to *interpolate* d-dimensional per-point neural features $h_m \in \mathbb{R}^d$ at query points x:

$$h(x) \equiv \sum_{m=1}^{M} A_m \, \mathrm{P}^{\mathrm{mod}}(x, p_m) \cdot h_m, \qquad \mathrm{P}^{\mathrm{mod}}(x, y) \equiv \frac{1}{4\pi \|x-y\|^2}, \tag{3.21}$$

We omit the regularized version of the modified kernel and the corresponding Barnes-Hut approximation for brevity, as they are analogous to the original Poisson kernel. The modified kernel ensures that the neural features are smoothly interpolated across the scene without sharp discontinuities.

With interpolated neural features, we follow Wang et al. [33] and use a neural network to predict the radiance at a query point x with viewing direction ω and neural features $h(x)$:

$$L(x, \omega) \equiv N(x, \omega, h(x)), \tag{3.22}$$

In particular, we also follow Verbin et al. [32] and use spherical harmonics to encode the viewing direction instead of the positional encoding original proposed by Mildenhall et al. [25].

3.3 Logarithmic complexity backpropagation

In this section, we show how we can utilize the Barnes-Hut approximation to efficiently backpropagate gradients through the octree data structure, and how this allows us to efficiently optimize the point cloud parameters. For simplicity, we focus on deriving the gradient computation for the per-point Dirichlet data, but the same principles apply to the per-point neural features.

3.3.1 Gradient accumulation

Denoting the rendered pixels as $\widehat{C}_i$ and ground truth pixels as C_i, we can define the loss function as the mean absolute error between the rendered and ground truth pixels:

$$\text{loss} = \frac{1}{N}\sum_{i=1}^{N}\left\|\widehat{C}_i - C_i\right\|_1. \tag{3.23}$$

Since we use the Barnes-Hut approximation during foward rendering, the loss can be viewed as a function of both the per-point parameters of the point cloud and per-node parameters of the octree structure.

Assuming we have a point cloud of n points and make m winding number queries during rendering, naively using autograd to compute the gradients of the loss with respect to all of the parameters would require $O(n \cdot m)$ time at best: for each single query, backpropagation requires computing the gradient at all octree nodes that are visited during forward rendering. This is identical to what happens during forward rendering and only takes $O(\log n)$ time. However, since the per-node parameters are essentially functions of the per-point parameters of its leaves, this means that for each node t we visit, we would also need to visit all of its leaves $\mathcal{L}(t)$ and accumulate gradients at its leaves. This results in asympotically slower backpropagation compared to forward rendering.

To address this, we introduce a two-stage backpropagation scheme:

1. In the first stage, we *detach* the per-node parameters from the per-point parameters and compute the gradients of the loss with respect to only the per-node parameters. For m winding number queries during rendering, this only requires $O(m \log n)$ time, identical to forward rendering.
2. In the second stage, given the gradients of the per-node parameters, we can use the chain rule to compute the gradient of the per-point parameters. Since each point can only be the leaf of $O(\log n)$ many nodes, in the worst case, this step still only requires $O(n \log n)$ time. Importantly, this is only a one-time cost for each iteration of training, and does not need to be done for each individual query.

In summary, this two stage backpropagation scheme allows us to run backpropagation in $O((n + m) \log n)$ time, compared to the naive $O(n \cdot m)$ time. In practice, n and m can be on the order of thousands and millions, respectively, making this optimization crucial for efficient training.

3.3.2 Gradient computation for nodes

Next, we elaborate on how the gradient of the loss with respect to the per-node parameters can be computed efficiently, which is essential to the first stage of our backpropagation scheme. For each point m, we denote its Dirichlet data as f_m and the gradient of the per-point parameters as $\nabla \mathrm{f}_m$.

For each node t, we note that the per-node Dirichlet data is not explicitly stored but as part of the area-weighted normal $\tilde{n}_t$ as shown in 3.18. In other words, instead of computing the gradient of the per-node Dirichlet data, we need to instead compute the gradient of area-weighted normal, which we denote as $\nabla \tilde{n}_t$.

We consider a single node t and all queries $Q(t)$ that visit this node during forward rendering. For each query, we denote the corresponding regularized dipole sum as $\mathrm{u}^{\mathrm{f}}_{\mathrm{pc},\varepsilon}(x)$ and the gradient of the regularized dipole sum as $\nabla \mathrm{u}^{\mathrm{f}}_{\mathrm{pc},\varepsilon}(x)$. We can then compute the gradient of the area-weighted normal as:

$$\nabla \tilde{n}_t = \sum_{x \in Q(t)} \nabla \mathrm{u}^{\mathrm{f}}_{\mathrm{pc},\varepsilon}(x) \cdot \frac{1}{4\pi} \frac{\widehat{x\tilde{p}_t}}{\|x - \tilde{p}_t\|^2} \mathrm{S}\left(\frac{\|x - \tilde{p}_t\|}{\varepsilon}\right). \tag{3.24}$$

In practice, we loop over all queries and traverse the octree to accumulate the gradients of the area-weighted normal at each node. This allows us to compute the gradient of the loss with respect to the per-node parameters in $O(m \log n)$ time.

3.3.3 Propogating gradient to points

Then, the second stage of backpropagation requires propogating gradients stored on the nodes of the octree down to its leaves, i.e., the individual points of the point cloud. During the construction of the octree structure, we computed the area-weighted normal for each node as shown in 3.18. Given the gradient of the area-weighted normal $\nabla \tilde{n}_t$, we can compute the gradient of the per-point Dirichlet data as:

$$\nabla \mathrm{f}_m = \sum_{t \in \mathcal{T}(m)} \langle \nabla \tilde{n}_t \cdot A_m n_m \rangle, \tag{3.25}$$

where $\mathcal{T}(m)$ is the set of nodes that are ancestors of point m in the octree. In practice, we loop over all nodes of the octree and propogate gradients down to the leaves of each node.

The second stage can be done independently of the first stage, and only needs to be done once for each iteration of training. This allows us to compute the gradient of the loss with respect to the per-point parameters in an additional $O(n \log n)$ time.

3.3.4 Optimization

Finally, we can use the gradients computed in the two stages to optimize the per-point parameters of the point cloud. We can use any optimization algorithm, such as Adam [19], to update the per-point parameters.

After updating the per-point parameters, we can then recompute the new area-weighted normal for each node of the octree efficiently in $O(n \log n)$ time, according to the same equation as in 3.18. This summarizes how we can efficiently optimize the learnable parameters of the point cloud using the Barnes-Hut approximation in a differentiable rendering setting.

Chapter 4

Experimental evaluation

In this chapter, we discuss our implementation details and present the results of our method on a variety of tasks, including 3D reconstruction of real-world scenes and multi-bounce rendering in scenes with known lighting. We compare our method to existing methods and show that our method consistently outperforms existing methods in both reconstruction quality and efficiency.

4.1 Implementation details

We first elaborate on our implementation details for both neural rendering and point cloud queries, and how they are integrated during forward rendering and backpropagation.

4.1.1 Neural rendering

We implement our neural rendering pipeline in a simplified version of the NeuS [33] codebase. In each iteration of training, the forward rendering pipeline consists of several stages:

1. **Camera ray generation:** We randomly choose a single image from the training dataset and generate camera rays for randomly selected pixels in the image. A batch size of 4096 rays is used during training.
2. **Intersection:** We compute near and far planes by intersecting camera rays with the bounding sphere of the point cloud. Then, we intersect camera rays with the scene by densely querying the winding number at 1024 uniformly sampled points along each ray between the near and far planes by finding the first 0.5-crossing.
3. **Importance sampling:** We place 16 sparse samples along the ray between the near plane and the first crossing point, 32 dense samples near the first crossing point, and 16 sparse samples between the first crossing point and the far plane.
4. **Rendering:** We query the winding number and neural features at each sample point from the point cloud. The winding number is converted into the occupancy (3.20), while the neural features are passed into a color network to compute the radiance (3.22). The occupancy along each ray is converted into opacity values (2.10) that are then used to accumulate the radiance along the ray (2.7). A background network is used to model the scene background

outside of the bounding sphere, following Zhang et al. [39].

5. **Backpropagation:** We compute the loss between the rendered colors and the ground truth colors using the mean absolute error loss. PyTorch autograd is used to directly compute the gradients for the neural network parameters. Backpropogation for the point cloud parameters is more complex and is discussed in the next section.
6. **Optimization:** We use the Adam optimizer with a learning rate of $3 \cdot 10^{-3}$ for the neural networks and $1 \cdot 10^{-2}$ for the neural features and Dirichlet data of the point cloud. We train the model for 50,000 iterations on a single NVIDIA RTX 4090 GPU.

4.1.2 Point cloud queries

A naive PyTorch implementation of the Barnes-Hut approximation and fast dipole sums for point cloud queries is inhibitive for training due to the large number of point cloud queries required for each ray. To address this, we implement the Barnes-Hut approximation and fast dipole sums in C++ and CUDA and use PyTorch's extension API to interface with the neural rendering pipeline in Python.

We implement the Barnes-Hut approximation by building an octree structure in CUDA. The centroids, weighted normals, radii, and leaves of each node are stored as tensors in GPU memory. During forward rendering, we use custom CUDA kernels to traverse the octree and compute fast dipole sums for large numbers of point cloud queries in parallel. The resulting dipole sums are then passed back to the neural rendering pipeline in Python for further processing. In particular, these dipole sums are treated as leaf nodes in the PyTorch computation graph and are effectively detached from the per-point parameters of the point cloud. The gradients of the loss with respect to the dipole sums are then directly computed using PyTorch autograd. These gradients are then passed into a custom CUDA kernel to propogate the gradients to the per-point parameters of the point cloud.

4.1.3 Training Setup

We evaluate our method against Neuralangelo [21] and Colmap [29, 30] on the DTU dataset [13] and Blended MVS datasets [37]. We train our models on a single NVIDIA RTX4090 GPU. Each scene takes around 3-4 hours to train to convergence.

For the DTU dataset, we filter the extracted mesh using object masks provided in the DTU dataset, and evaluate the mesh quality using the Chamfer distance, computed using the official evaluation script. For the Blended MVS dataset, since ground truth meshes are not provided, we only present qualitative results.

4.2 3D reconstruction

In this section, we present quantitative and qualitative results of our method in comparison to Neuralangelo [21] and Colmap [29, 30].

4.2.1 DTU

For the DTU dataset, the quantitative reults are summarized in Table 4.1. Due to inaccurate object masks and ground truth point clouds for scans 63, 83, and 105, we exclude these scans from the mean Chamfer distance calculation.

Scan	Ours	Neuralangelo	Colmap
24	0.46	**0.37**	1.00
37	**0.65**	0.72	1.37
40	**0.33**	0.35	0.93
55	**0.33**	0.35	0.43
63	0.95	**0.87**	1.10
65	0.78	**0.54**	0.65
69	**0.53**	**0.53**	0.57
83	**1.23**	1.29	1.48
97	**0.84**	0.97	1.09
105	**0.70**	0.73	0.83
106	**0.46**	0.47	0.52
110	**0.55**	0.74	1.20
114	0.33	**0.32**	0.35
118	**0.37**	0.41	0.49
122	**0.36**	0.43	0.54
Mean[†]	**0.50**	0.52	0.76

[†]Mean calculation excludes scans 63, 83, and 105.

Table 4.1: Chamfer distance evaluations on the DTU dataset.

Our method consistently outperforms Neuralangelo and Colmap in terms of the mean Chamfer distance on the DTU dataset, aside from scans 24 and 65, which are challenging for our method due to the presence of fine details and textureless regions. We will further discuss limitations of our method in section 5.2.

Furthermore, the DTU dataset is challenging for our method due to having no views behind the objects, leading to large gaps in the point cloud behind the objects and thus making it difficult to represent a closed surface with the dipole sum. Having learnable Dirichlet data partially addresses this issue, but to make our method more robust, we also implement a point growing procedure that iteratively adds points to the point cloud by sampling rays in the scene and adding points at the first 0.5-crossing. This allows us to fill in the gaps in the point cloud and produce more accurate reconstructions.

4.2.2 Blended MVS

For the Blended MVS dataset, since there are no ground truth meshes or point clouds available, we only present qualitative evaluation in Figures 4.1 and 4.2.

Figure 4.1: Qualitative results on the Blended MVS dataset. The first column is the input image, the second column is the Colmap reconstruction, the third column is the Neuralangelo reconstruction, and fourth column is our reconstruction.

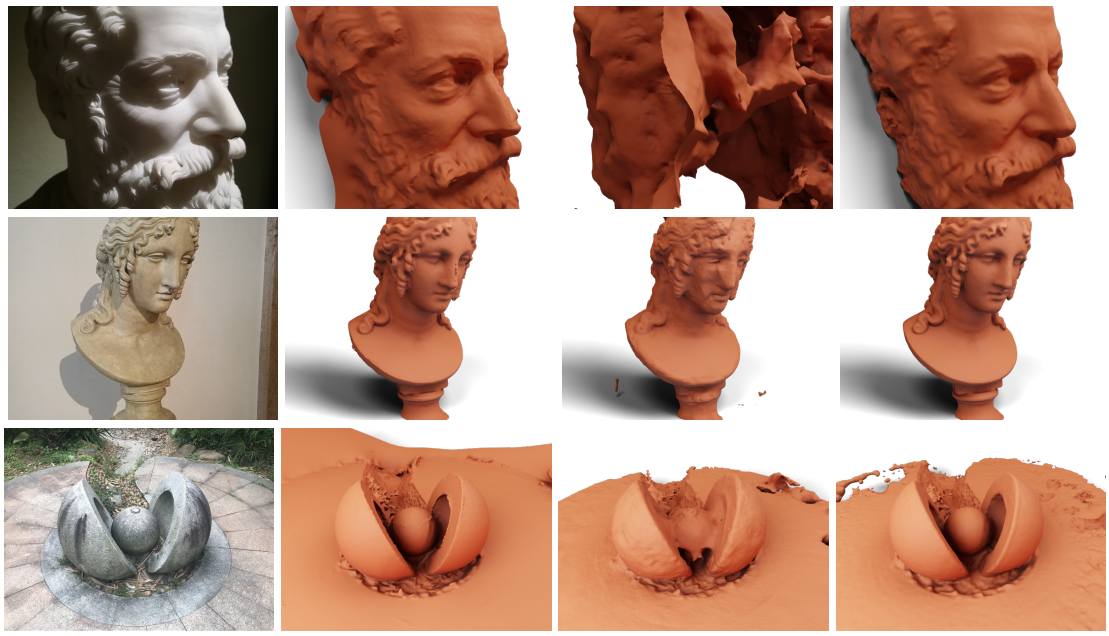

Figure 4.2: Qualitative results on the Blended MVS dataset (continued).

In general, our method produces more detailed and accurate reconstructions compared to Neuralangelo and Colmap with less noise. Notably, our method is able to perform well in cases where the number of input images is limited, such as in the MAN and DOG scenes due to its ability to leverage existing geometry information of the scenes via the point cloud. In contrast, Neuralangelo completely fails to reconstruct those scenes.

4.3 Multi-bounce rendering

In this section, we present results of our method on multi-bounce rendering in scenes with known lighting. We demonstrate that our method is able to accurately render complex scenes and leverage known lighting information through tracing shadow rays. This is an important advantage of our method over rasterization-based methods, such as 3D Gaussian splatting [18].

4.3.1 Experimental setup

We evaluate our method on the LEGO scene from the synthetic NeRF dataset [25]. We set up the scene in Blender with two point light sources that randomly vary across views. We use a fully diffuse Lambertian material for the scene objects and render the scene from 200 views with known camera poses.

During training, at each sample along the primary ray, we make direct connections to both light sources and sample shadow rays to to the light sources. The normals and albedos at each point are predicted from neural features interpolated from the point cloud. Assuming a fully diffuse Lambertian material, we compute the direct illumination using the Lambertian reflectance model. Then, we use a separate shallow neural network to model indirect illumination with the

light source direction as additional input. These two components are then combined to compute the final radiance at each sample point.

For comparison, we also implement a baseline method that models both direct and indirect illumination using a single neural network with the light source direction as additional input without explicitly sampling shadow rays.

4.3.2 Results

We first present qualitative comparison of the extracted meshes on the LEGO scene in Figure 4.3. The ability to explicitly model shadow rays allows our method to produce meshes with higher accuracy in regions with fine details and less noise in flatter regions, compared to the baseline method.

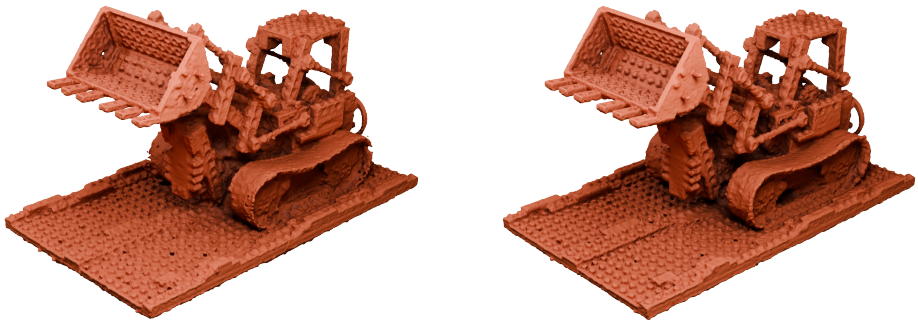

Figure 4.3: Qualitative comparison of the extracted meshes on the LEGO scene. The first image shows the mesh reconstructed without shadow rays and the second image show the mesh reconstructed with shadow rays.

Moreover, although novel view synthesis is not the focus of our work, we also demonstrate that modeling shadow rays allows us to render novel views of the LEGO scene with high fidelity, as shown in Figure 4.4. In particular, we note that the shadows on the ground are much more accurate in the third image, which is rendered with shadow rays, compared to the baseline.

This is an important feature that allows us to easily relight scenes with arbitrary light source locations and render novel views with high fidelity, which is not possible with rasterization-based methods. For example, in Figure 4.5, we show that we can swap the light source locations in two different views of the LEGO scene and render novel views with high fidelity.

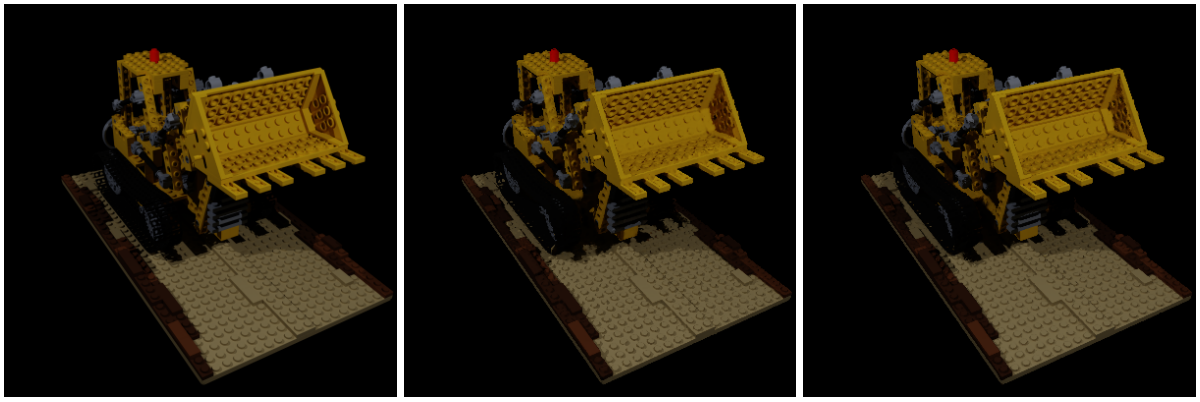

Figure 4.4: Qualitative comparison of novel view synthesis on the LEGO scene. The first image is the ground truth, the second image is rendered without shadow rays, and the third image is rendered with shadow rays.

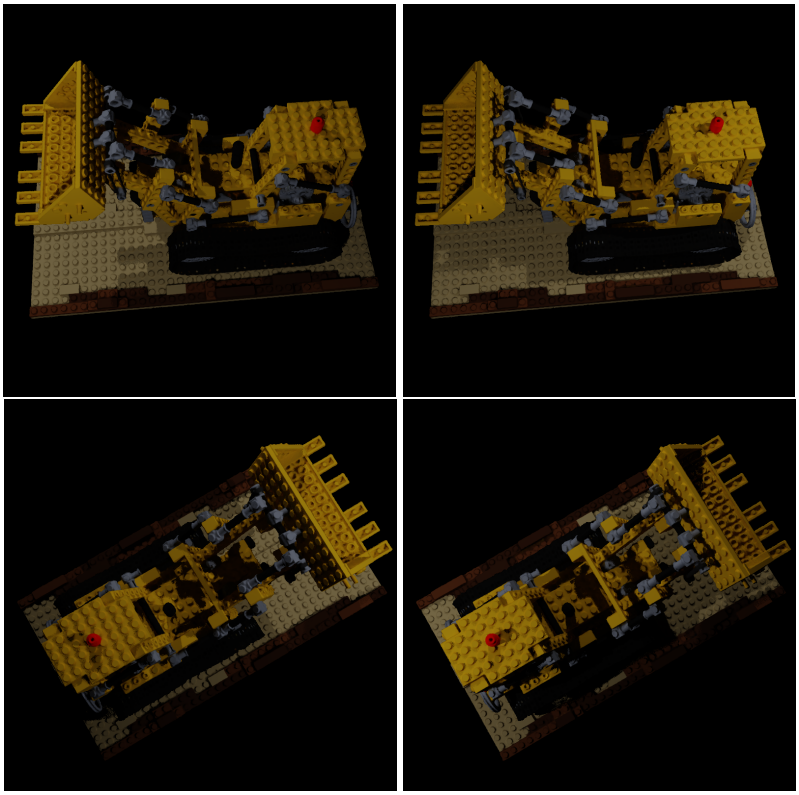

Figure 4.5: Demonstration of relighting on the LEGO scene. The images in each row are rendered with the same camera pose but different light source locations, while the images in each column are rendered with the same light source location but different camera poses.

Chapter 5

Conclusion

5.1 Contributions

In this section, we summarize the contributions of our work in relation to existing literature.

5.1.1 Efficient 3D Reconstruction

Our work introduces a novel point-based representation for efficient 3D reconstruction based on the generalized winding number. We derive, from a principled approach, a further generalization of the winding number that allows for arbitrary learnable boundary conditions, which we call the "dipole sum". We use the Barnes-Hut approximation to efficiently compute the dipole sum in logarithmic time complexity, and we propose a novel two-stage backpropagation algorithm to compute gradients efficiently.

We show that our method outperforms existing state-of-the-art methods, such as Neuralangelo, in both reconstruction quality and efficiency on a wide range of real-world scenes, and present qualitative and quantitative results on the DTU and BlendedMVS datasets. Unlike most existing methods, by leverging known scene information derived from traditional structure from motion and multi-view stereo pipelines, our method is able to produce high-quality reconstructions with minimal input data.

5.1.2 Multi-bounce Rendering

We demonstrate that our point-based representation can be used for efficient differentiable multi-bounce rendering under known lighting conditions, which is not possible for rasterization-based approaches like 3D Gaussian splatting. This allows us to produce more accurate rendering and model complex scene geometry. We show that our method outperforms a baseline method that does not explicitly account for shadow rays in terms of both mesh extraction and novel view synthesis.

5.2 Limitations

Finally, we discuss the limitations of our work and potential directions for future research.

5.2.1 Fine Details

Our method is limited in its ability to reconstruct varying levels of detail in the scene, such as thin structures and fine textures. This is due to the inherent limitations of point-based representations, which are unable to capture fine details without a large number of points, making them computationally expensive to train and evaluate. Future work could potentially explore ways to incorporate multi-resolution hash grids into the point-based representation to capture fine details more effectively, or use the point-based representation as a pre-processing step for more complex representations.

5.2.2 Noisy Point Clouds

While our method is reasonably robust to noisy point clouds due to having learnable boundary conditions and our point growing procedure, it still inherits the same limitation of dense point cloud reconstruction pipelines like Colmap. For example, in textureless regions or highly specular regions, Colmap may fail to produce any points, leading to incomplete reconstructions. This means that our method may also struggle to accurately reconstruct a smooth surface in these regions. Future work could potentially explore more robust ways to account for noise and holes in the point cloud, such as implementing a more robust point growing procedure or allow point locations to be optimizable during training.

Bibliography

[1] Gavin Barill, Neil G Dickson, Ryan Schmidt, David IW Levin, and Alec Jacobson. Fast winding numbers for soups and clouds. *ACM Transactions on Graphics (TOG)*, 37(4): 1–12, 2018. 3.1.1, 3.1.2, 3.1.2, 3.1.3, 3.1.4, 3.1.5

[2] Josh Barnes and Piet Hut. A hierarchical o (n log n) force-calculation algorithm. *nature*, 324(6096):446–449, 1986. 3.1.3

[3] J Thomas Beale, Wenjun Ying, and Jason R Wilson. A simple method for computing singular or nearly singular integrals on closed surfaces. *Communications in Computational Physics*, 20(3):733–753, 2016. 3.2.2, 3.2.2

[4] Ricardo Cortez. The method of regularized stokeslets. *SIAM Journal on Scientific Computing*, 23(4):1204–1225, 2001. 3.2.2

[5] Ricardo Cortez, Lisa Fauci, and Alexei Medovikov. The method of regularized stokeslets in three dimensions: analysis, validation, and application to helical swimming. *Physics of Fluids*, 17(3), 2005. 3.2.2

[6] François Darmon, Bénédicte Bascle, Jean-Clément Devaux, Pascal Monasse, and Mathieu Aubry. Improving neural implicit surfaces geometry with patch warping. In *CVPR*, 2022. 1.2

[7] Kangle Deng, Andrew Liu, Jun-Yan Zhu, and Deva Ramanan. Depth-supervised NeRF: Fewer views and faster training for free. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022. 1.3

[8] Nicole Feng, Mark Gillespie, and Keenan Crane. Winding numbers on discrete surfaces. *ACM Transactions on Graphics (TOG)*, 2023. 3.1.1

[9] Qiancheng Fu, Qingshan Xu, Yew-Soon Ong, and Wenbing Tao. Geo-neus: Geometry-consistent neural implicit surfaces learning for multi-view reconstruction. *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. 1.2

[10] John C Hart. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12(10):527–545, 1996. 3.1.4

[11] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Surface reconstruction from unorganized points. In *Proceedings of the 19th annual conference on computer graphics and interactive techniques*, pages 71–78, 1992. 3.1.2

[12] Alec Jacobson, Ladislav Kavan, and Olga Sorkine-Hornung. Robust inside-outside segmentation using generalized winding numbers. *ACM Transactions on Graphics (TOG)*, 32

(4):1–12, 2013. 1.3

[13] Rasmus Jensen, Anders Dahl, George Vogiatzis, Engin Tola, and Henrik Aanæs. Large scale multi-view stereopsis evaluation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 406–413, 2014. 4.1.3

[14] James T. Kajiya and Brian P Von Herzen. Ray tracing volume densities. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '84, page 165–174, New York, NY, USA, 1984. Association for Computing Machinery. ISBN 0897911385. doi: 10.1145/800031.808594. URL https://doi.org/10.1145/800031.808594. 2.1

[15] Animesh Karnewar, Tobias Ritschel, Oliver Wang, and Niloy J. Mitra. Relu fields: The little non-linearity that could. *Transactions on Graphics (Proceedings of SIGGRAPH)*, 41 (4):13:1–13:8, 2022. doi: 10.1145/3528233.3530707. 1.2

[16] Michael Kazhdan and Hugues Hoppe. Screened poisson surface reconstruction. *ACM Transactions on Graphics (ToG)*, 32(3):1–13, 2013. 3.1.5

[17] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, volume 7, page 0, 2006. 1.3, 3.1.5

[18] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), July 2023. URL https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/. 1.2, 4.3

[19] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980. 3.3.4

[20] Pavel A Krutitskii. The jump problem for the laplace equation. *Applied Mathematics Letters*, 14(3):353–358, 2001. 3.1.1, 3.2.1

[21] Zhaoshuo Li, Thomas Müller, Alex Evans, Russell H Taylor, Mathias Unberath, Ming-Yu Liu, and Chen-Hsuan Lin. Neuralangelo: High-fidelity neural surface reconstruction. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023. 1.2, 4.1.3, 4.2

[22] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '87, page 163–169, New York, NY, USA, 1987. Association for Computing Machinery. ISBN 0897912276. doi: 10.1145/37401.37422. URL https://doi.org/10.1145/37401.37422. 2.2.2

[23] Nelson Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99–108, 1995. 2.1.3

[24] Donald Meagher. Geometric modeling using octree encoding. *Computer graphics and image processing*, 19(2):129–147, 1982. 3.1.3

[25] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020. 1.1, 2.2, 2.2.1, 3.2.4, 4.3.1

[26] Bailey Miller, Hanyu Chen, Alice Lai, and Ioannis Gkioulekas. A theory of volumetric representations for opaque solids. *arXiv preprint arXiv:2312.15406*, 2023. 2.2.2, 2.2.2, 3.1.4

[27] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, July 2022. doi: 10.1145/3528223.3530127. URL `https://doi.org/10.1145/3528223.3530127`. 1.1

[28] Sara Fridovich-Keil and Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *CVPR*, 2022. 1.2

[29] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 1.1, 4.1.3, 4.2

[30] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise view selection for unstructured multi-view stereo. In *European Conference on Computer Vision (ECCV)*, 2016. 1.1, 4.1.3, 4.2

[31] Jonathan Richard Shewchuk. Triangle: Engineering a 2d quality mesh generator and delaunay triangulator. In *Workshop on applied computational geometry*, pages 203–222. Springer, 1996. 3.1.2

[32] Dor Verbin, Peter Hedman, Ben Mildenhall, Todd Zickler, Jonathan T. Barron, and Pratul P. Srinivasan. Ref-NeRF: Structured view-dependent appearance for neural radiance fields. *CVPR*, 2022. 3.2.4

[33] Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. *arXiv preprint arXiv:2106.10689*, 2021. 1.1, 2.2.2, 2.2.2, 3.2.4, 4.1.1

[34] Yiming Wang, Qin Han, Marc Habermann, Kostas Daniilidis, Christian Theobalt, and Lingjie Liu. Neus2: Fast learning of neural implicit surfaces for multi-view reconstruction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023. 1.2

[35] Tong Wu, Jiaqi Wang, Xingang Pan, Xudong Xu, Christian Theobalt, Ziwei Liu, and Dahua Lin. Voxurf: Voxel-based efficient and accurate neural surface reconstruction. In *International Conference on Learning Representations (ICLR)*, 2023. 1.2

[36] Qiangeng Xu, Zexiang Xu, Julien Philip, Sai Bi, Zhixin Shu, Kalyan Sunkavalli, and Ulrich Neumann. Point-nerf: Point-based neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5438–5448, June 2022. 1.2

[37] Yao Yao, Zixin Luo, Shiwei Li, Jingyang Zhang, Yufan Ren, Lei Zhou, Tian Fang, and Long Quan. Blendedmvs: A large-scale dataset for generalized multi-view stereo networks.

Computer Vision and Pattern Recognition (CVPR), 2020. 4.1.3

[38] Lior Yariv, Jiatao Gu, Yoni Kasten, and Yaron Lipman. Volume rendering of neural implicit surfaces. In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021. 1.1, 2.2.2

[39] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. Nerf++: Analyzing and improving neural radiance fields. *arXiv:2010.07492*, 2020. 4