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's 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 ahve 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

Things to cite here: - Nick's paper, Belkin paper, on point cloud geometry - SIGGRAPH 2023 paper on fixing normal orientations - Point-NeRF - CMU paper on using point clouds - GeoNeuS - Pointersect - Gaussian splatting - Kaleidoscopic, other stereo that produce area weights - Alec's winding number papers - PSR and SPSR - Neural surface papers - Differentiable PSR

Chapter 2

Background

2.1 Volumetric light transport background

We first introduce the basic concepts of volumetric light transport using principles from classical volume rendering [9], 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 [14], 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

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

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) [16] 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 $f_\Theta : (\mathbf{x}, \vec{\omega}) \to (c, \sigma)$, 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 [16].

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 implicit surfaces

To address the limitations of using the attenuation coefficient to extract surfaces, works including [17, 19, 20] 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 implicit surfaces can be used to both render high-quality images of scenes and extract surfaces from the scene, by using marching cubes [13] to find the 0.5-crossing of the occupancy, for example.

Specifically, for converting the occupancy into attenuation coefficients, we adopt the method proposed by Miller et al. [17], which is given 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 2.3.4.

2.3 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 3.1 for our implicit surface representation.

2.3.1 Winding number for surfaces

We first consider a continuous surface $\Gamma \subset \mathbb{R}^3$. There are many equivalent definitions of the winding number [6]; 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:

$$\triangle \mathrm{w}(x) = 0 \text{ in } \mathbb{R}^3 \setminus \Gamma, \tag{2.10}$$

$$\mathrm{w}^+(x) - \mathrm{w}^-(x) = 1 \text{ on } \Gamma, \tag{2.11}$$

$$\partial \mathrm{w}^+/\partial n(x) - \partial \mathrm{w}^-/\partial n(x) = 0 \text{ on } \Gamma. \tag{2.12}$$

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 [12] provide a detailed treatment of such BVPs, and in particular prove the following *boundary integral* expression for their solution:

$$\mathrm{w}(x) = \int_\Gamma \mathrm{P}(x, y) \cdot 1 \, \mathrm{d}\sigma(y), \qquad \mathrm{P}(x, y) \equiv \frac{1}{4\pi} \frac{n(y) \cdot \widehat{xy}}{\|x - y\|^2}, \tag{2.13}$$

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 2.13, corresponding to the jump Dirichlet boundary condition 2.11, for reasons we will explain in 3.1.

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.

2.3.2 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 [8, 18]).

As Barill et al. [1] explain, the boundary integral representation 2.13 directly suggests the following generalization of the winding number for point clouds:

$$\mathrm{w}_{\mathrm{pc}}(x) \equiv \sum_{m=1}^{M} A_m \, \mathrm{P}(x, p_m) \cdot 1 = \sum_{m=1}^{M} \frac{A_m}{4\pi} \frac{n_m \cdot \widehat{xp_m}}{\|x - p_m\|^2} \cdot 1. \tag{2.14}$$

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.

2.3.3 Barnes-Hut approximation

Evaluating the winding number $\mathrm{w}_{\mathrm{pc}}(x)$ at a query point x using 2.14 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 [15]) 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, weighed normal, area, and radius

$$\tilde{p}_t \equiv \frac{\sum_{m\in\mathcal{L}(t)} A_m p_m}{\tilde{A}_t}, \qquad \tilde{n}_t \equiv \frac{\sum_{m\in\mathcal{L}(t)} A_m n_m}{\tilde{A}_t}, \tag{2.15}$$

$$\tilde{A}_t \equiv \sum_{m\in\mathcal{L}(t)} A_m, \qquad \tilde{r}_t \equiv \max_{m\in\mathcal{L}(t)} \|p_m - \tilde{p}_t\|, \tag{2.16}$$

where $\mathcal{L}(t)$ is the set of leaf (i.e., single-point) nodes that are successors of t in the tree hierarchy. 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:

$$\sum_{m\in\mathcal{L}(t)} A_m \, \mathrm{P}(x, p_m) \cdot 1 \approx \tilde{A}_t \, \mathrm{P}(x, \tilde{p}_t) \cdot 1. \tag{2.17}$$

This approximation expresses the fact that, due to the squared-distance falloff of the Poisson kernel in 2.13, the *far-field* influence of a cluster of points can be represented by a single point at the cluster's centroid.

2.3.4 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 (3.1), these queries facilitate also forward and inverse rendering of geometry represented as point clouds.

Specifically, we observe that we can use the winding number w_{pc} and occupancy o_{pc} to define an implicit surface Γ_{pc} and, following Miller et al. [17, Equation (12)], volumetric attenuation coefficient σ_{pc} as:

$$\Gamma_{\mathrm{pc}} \equiv \left\{x \in \mathbb{R}^3 : \mathrm{w}_{\mathrm{pc}}(x) = 1/2\right\}, \qquad \sigma_{\mathrm{pc}}(x, \omega) \equiv \frac{|\omega \cdot \nabla \, \mathrm{o}_{\mathrm{pc}}(x)|}{1 - \mathrm{o}_{\mathrm{pc}}(x)}. \tag{2.18}$$

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 [7]. 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}, o_{pc}, and w_{pc} to update point cloud parameters is straightforward and efficient, as we discuss in 3.2.

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.

2.3.5 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 [10, 11]. 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 (2.10, 2.11, 2.12) for the continuous surface underlying the point cloud. The *limit* behaviors of the two scalar fields are equivalent. However, whereas the approximation of 2.14 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.

Chapter 3

Method

3.1 Fast dipole sums

We introduce a generalization of 2.14 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.

3.1.1 General Dirichlet conditions

We generalize the BVP (2.10, 2.11, 2.12) to use an arbitrary *Dirichlet data* function $\mathrm{f} : \Gamma \to \mathbb{R}$ for the jump Dirichlet boundary condition:

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

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

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

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 (2.13, 2.14) to express u^f and its point-based approximation as [12]:

$$\mathrm{u}^\mathrm{f}(x) \equiv \int_\Gamma \mathrm{P}(x, y) \cdot \mathrm{f}(y) \,\mathrm{d}\sigma(y), \qquad \mathrm{u}^\mathrm{f}_\mathrm{pc}(x) \equiv \sum_{m=1}^M A_m \,\mathrm{P}(x, p_m) \cdot \mathrm{f}_m \,. \tag{3.4}$$

3.1.2 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:

$$\mathrm{P}(x, y) \equiv n(y) \cdot \nabla_x \mathrm{G}(x, y) \text{ where G satisfies } \triangle \mathrm{G}(x, y) = \delta(x - y), \tag{3.5}$$

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 3.6:

$$\mathrm{P}_\varepsilon(x, y) \equiv n(y) \cdot \nabla_x \mathrm{G}_\varepsilon(x, y) \text{ where } \mathrm{G}_\varepsilon \text{ satisfies } \triangle \mathrm{G}_\varepsilon(x, y) = \phi_\varepsilon(x - y), \tag{3.6}$$

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]:

$$\mathrm{P}_\varepsilon(x, y) \equiv \mathrm{P}(x, y) \cdot \mathrm{S}\!\left(\frac{\|x - y\|}{\varepsilon}\right), \text{ where } \mathrm{S}(t) \equiv \operatorname{erf}(t) - \frac{2}{\sqrt{\pi}} \cdot t \cdot \exp\!\left(-t^2\right). \tag{3.7}$$

We observe that $\mathrm{P}_\varepsilon(x, x) = 3^{-1}\varepsilon^{-3}\pi^{-3/2}$, which is finite for any $\varepsilon > 0$. 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). We set ε so that $\mathrm{P}_\varepsilon(x, x) = {}^1\!/_2$, a choice we justify in the next section.

3.1.3 Regularized dipole sum

Finally, we use the regularized Poisson kernel to approximate $\mathrm{u}^{\mathrm{f}}_{\mathrm{pc}}$ in 3.4 as:

$$\mathrm{u}^{\mathrm{f}}_{\mathrm{pc}}(x) \approx \mathrm{u}^{\mathrm{f}}_{\mathrm{pc},\varepsilon}(x) \equiv \sum_{m=1}^{M} A_m \, \mathrm{P}_\varepsilon(x, p_m) \cdot \mathrm{f}_m \,. \tag{3.8}$$

The point-cloud winding number is the solution for constant unit data, i.e., $\mathrm{w}_{\mathrm{pc}}(x) = \mathrm{u}^1_{\mathrm{pc}}(x)$.

3.2 Logarithmic complexity backpropagation

Chapter 4

Experimental evaluation

4.1 3D reconstruction

4.1.1 Implementation details

4.1.2 Comparisons

4.2 Multi-bounce rendering

4.2.1 Experimental setup

4.2.2 Results

4.3 Visualizations of various quantities

4.3.1 Fields

4.3.2 Point clouds

Chapter 5

Conclusion

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. 2.3.1, 2.3.2, 2.3.2, 2.3.3, 2.3.4, 2.3.5

[2] Josh Barnes and Piet Hut. A hierarchical o (n log n) force-calculation algorithm. *nature*, 324(6096):446–449, 1986. 2.3.3

[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. 3.1.2, 3.1.2

[4] Ricardo Cortez. The method of regularized stokeslets. *SIAM Journal on Scientific Computing*, 23(4):1204–1225, 2001. 3.1.2

[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. 3.1.2

[6] Nicole Feng, Mark Gillespie, and Keenan Crane. Winding numbers on discrete surfaces. *ACM Transactions on Graphics (TOG)*, 2023. 2.3.1

[7] John C Hart. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12(10):527–545, 1996. 2.3.4

[8] 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. 2.3.2

[9] 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

[10] Michael Kazhdan and Hugues Hoppe. Screened poisson surface reconstruction. *ACM Transactions on Graphics (ToG)*, 32(3):1–13, 2013. 2.3.5

[11] 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. 2.3.5

[12] Pavel A Krutitskii. The jump problem for the laplace equation. *Applied Mathematics*

Letters, 14(3):353–358, 2001. 2.3.1, 3.1.1

[13] 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

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

[15] Donald Meagher. Geometric modeling using octree encoding. *Computer graphics and image processing*, 19(2):129–147, 1982. 2.3.3

[16] 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. 2.2, 2.2.1

[17] 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, 2.3.4

[18] Jonathan Richard Shewchuk. Triangle: Engineering a 2d quality mesh generator and delaunay triangulator. In *Workshop on applied computational geometry*, pages 203–222. Springer, 1996. 2.3.2

[19] 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. 2.2.2

[20] 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. 2.2.2