Differentiable Rendering of Neural SDFs through Reparameterization

Sai Praveen Bangaru
MIT CSAIL
USA
sbangaru@mit.edu

Michaël Gharbi
Adobe Research
USA
mgharbi@adobe.com

Tzu-Mao Li
UC San Diego
USA
tzli@ucsd.edu

Fujun Luan
Adobe Research
USA
fluan@adobe.com

Kalyan Sunkavalli
Adobe Research
USA
sunkaval@adobe.com

Miloš Hašan
Adobe Research
USA
mihasan@adobe.com

Sai Bi
Adobe Research
USA
sbi@adobe.com

Zexiang Xu
Adobe Research
USA
zexu@adobe.com

Gilbert Bernstein
MIT CSAIL &
UC Berkeley
USA
gilbo@berkeley.edu

Frédo Durand
MIT CSAIL
USA
fredo@mit.edu

Figure 1: We propose a novel method to correctly differentiate a neural SDF renderer by reparameterizing the pixel integral. Direct application of automatic differentiation to the renderer fails because of discontinuities like silhouette boundaries. In this work we show that, by carefully designing a discontinuity-aware warp function \( V(u; \theta) \) to reparameterize the pixel domain, we can remove these discontinuities, and the reparameterized integral is amenable to automatic differentiation. We demonstrate the benefits of our method on inverse rendering problems. Starting from a multiview dataset of real photos (a), our reparameterized renderer (c) can optimize a neural SDF that closely matches the input data, and generalizes to novel views. Our renderer matches or outperforms prior SDF renderers [Yariv et al. 2020] (b), while doing away with their need for additional geometric supervision in the form of per-view masks, which can be unreliable for real-world data. We show additional surface reconstructions obtained with our inverse renderer in (d).

ABSTRACT
We present a method to automatically compute correct gradients with respect to geometric scene parameters in neural SDF renderers. Recent physically-based differentiable rendering techniques for meshes have used edge-sampling to handle discontinuities, particularly at object silhouettes, but SDFs do not have a simple parametric form amenable to sampling. Instead, our approach builds on area-sampling techniques and develops a continuous warping function for SDFs to account for these discontinuities. Our method leverages the distance to surface encoded in an SDF and uses quadrature on sphere tracer points to compute this warping function. We further show that this can be done by subsampling the points to make the method tractable for neural SDFs. Our differentiable renderer can be used to optimize neural shapes from multi-view images and produces comparable 3D reconstructions to recent SDF-based inverse rendering methods, without the need for 2D segmentation masks to guide the geometry optimization and no volumetric approximations to the geometry.

CCS CONCEPTS
- Mathematics of computing → Differential calculus; Integral calculus; Calculus;
- Computing methodologies → Modeling and simulation; Computer vision; Computer vision representations; Rendering; Ray tracing.
1 INTRODUCTION
Differentiable rendering algorithms have become crucial tools in solving challenging inverse problems [Zhao et al. 2020], thanks to their ability to compute the derivatives of images with respect to arbitrary scene parameters. Naïve differentation of rendering algorithms does not handle discontinuities caused by visibility changes and object boundaries correctly. Previous work has observed that the discontinuities can be handled by properly handling the Dirac delta signals, and derived algorithms for explicit geometry representations like triangle meshes [Li et al. 2018; Zhang et al. 2020].

On the other hand, implicit representations like signed distance fields (SDFs) are appealing since they do not require the initialized geometry to have the right topology. Recent work has demonstrated the use of SDFs—usually parameterized using multi-layer perceptron networks—for the task of reconstructing shape and surface reflectance from images. However, these methods either require additional geometric supervision such as segmentation masks [Yariv et al. 2020; Zhang et al. 2021b] or make approximations to the geometry using volumetric models [Oechsle et al. 2021; Yariv et al. 2021] that limit their applicability.

In this paper, we derive an algorithm to automatically compute correct gradients with respect to geometric scene parameters in neural SDF renderers. Previous methods that rely on silhouette sampling are not directly applicable to SDFs since direct sampling of boundaries of implicit functions is challenging. Instead, we build on the reparameterization approaches [Bangaru et al. 2020; Loubet et al. 2019], which removes discontinuities through reparameterization while preserving the integral values. These methods do not require explicit sampling along discontinuities. Previous reparameterization methods focused on triangle meshes, and require new derivation for reparameterizing SDF rendering.

Specifically, we construct a silhouette-aware reparameterization similar to that of Loubet et al. [2019], but following the equivalent unbiased warp definition that Bangaru et al. [2020] used to produce correct gradients for triangle-meshes. We leverage the fact that SDFs naturally encode the distance to the surface, and develop a practical algorithm that uses a quadrature on sphere tracing [Hart 1996] samples to construct a reparameterization that removes the discontinuities. We further show that this can be computed using only a subset of sphere tracing samples, reducing the computational burden of the backward pass for bulky neural SDFs.

Our algorithm produces correct geometry gradients for SDFs. It does away with the segmentation masks and depth guidance required by previous techniques [Yariv et al. 2020], without making a volumetric approximation to the geometry [Oechsle et al. 2021; Yariv et al. 2021]. We show that our differentiable renderer can be used to optimize neural shapes from multi-view images, with no additional information beyond the RGB data and the corresponding camera parameters. Our focus is on occlusion discontinuities, so the rest of the paper assumes a differentiable shading model.

2 RELATED WORK
We focus on work that recovers the latent 3D scene from images through differentiable rendering. We categorize them by the type of scene representation.

Meshes. To account for discontinuities, earlier work focused on approximating the derivatives of mesh rendering by smoothing the geometry [de La Gorce et al. 2011; Kato et al. 2018; Liu et al. 2019; Loper and Black 2014; Rhodin et al. 2015]. Alternatively, some work derived correct analytical derivatives under simplified assumptions [Arvo 1994; Zhou et al. 2021]. Li et al. [2018] noticed that the differentiation of discontinuities caused by the visibility and geometric boundaries lead to Dirac delta signals, and can be integrated by the pixel antialiasing integral or the rendering equation. They proposed an edge sampling algorithm to explicitly sample the Dirac delta on triangle mesh silhouettes. Importance sampling the silhouettes can be difficult, therefore Loubet et al. [2019] and Bangaru et al. [2020] later proposed to convert the silhouette integral into an area integral. Loubet et al. formulated the conversion using a reparameterization, and derived an approximated reparameterization to remove discontinuities. Bangaru et al. built on Loubet et al.’s work and derived an unbiased estimator by showing the equivalence between the reparameterization and divergence theorem. On the other hand, Zhang et al. [2020] showed that directly sampling silhouette in path-space [Veach 1998] can also be done efficiently. Directly sampling the silhouette for SDFs is difficult. Our work extends the reparameterization approach to handle SDFs, including approximate SDFs defined by neural networks.

Level sets and signed distance fields. A level set defines a surface using the roots of a 3D implicit function. A signed distance field is a specific kind of level set where the implicit function defines the distance of a 3D point to the surfaces, where the sign is negative when the point is inside the object. SDFs can be represented using polynomials [Blinn 1982], voxels [Izadi et al. 2011], or neural networks [Park et al. 2019]. Differentiable rendering for SDFs has been discussed in computer vision and used for 3D surface reconstruction [Jiang et al. 2020; Kellhofer et al. 2021; Niemeyer et al. 2020; Yariv et al. 2020; Zhang et al. 2021b], but current methods all ignore the discontinuities when differentiating, and require 2D object masks to converge. An alternative way to render the signed distance field is to convert it to another format such as a thin participating medium [Oechsle et al. 2021; Wang et al. 2021; Yariv et al. 2021], a mesh [Remelli et al. 2020], or a point cloud [Cole et al. 2021]. These methods all introduce approximation. Instead, we focus on deriving accurate gradients without approximation. Our work is closely related to the concurrent work by Vicini et al. [2022]. They too build on the warped-area sampling formulation from Bangaru et al. [2020], and arrive at a similar harmonic weighting scheme applied to sphere tracer points. The main difference is that we apply our method to the neural SDF and radiance model proposed by Yariv et al. [2020] to remove their mask requirement in a principled
manner. Vicini et al. [2022], on the other hand, experiment with voxelized SDFs and analytical BRDFs. The higher memory requirement of neural SDFs motivated our top-k version of the harmonic weights. We also focus on primary visibility since our main goal is to lift the mask requirement in neural SDF rendering.

Volumetric. A scene can also be represented as participating media instead of solid surfaces. Gkiolekas et al. [2013] pioneered the use of differentiable volume rendering for inverse problems. Zhang et al. [2019; 2021c] tackled discontinuities at volumetric boundaries. Recently, there has been a surging interest in using volumetric representations—parameterized either as discretized grids or neural networks—for view synthesis [Liu et al. 2020; Lombardi et al. 2019; Mildenhall et al. 2020; Xie et al. 2022]. These volumetric representations allow for a trivially differentiable rendering model and can achieve high-quality novel view synthesis and appearance acquisition [Bi et al. 2020a,b]. However, it is still a challenge to extract high-quality surface geometry from these methods, and while the trade-offs between surface and volume representations is an interesting research topic, we focus on surface representations.

Light transport. In addition to handling discontinuities, recent work also studies the reduction of variance and memory consumption for Monte Carlo rendering [Nimier-David et al. 2020; Vicini et al. 2021; Zeltner et al. 2021; Zhang et al. 2021a]. Earlier rendering work used derivatives for forward rendering [Li et al. 2015; Luan et al. 2020; Ramamoorthi et al. 2007; Ward and Heckbert 1992]. Our work is largely orthogonal to these.

3 Method

Our method computes the correct gradient of a rendering function (i.e., the pixel integral of the radiance function on the camera image plane) with respect to geometric parameters, in the presence of primary visibility discontinuities, for scenes where the geometry is represented by a signed distance field \( f \), parameterized by \( \theta \) (e.g., the weights of neural networks). Our approach builds on Bangaru et al. [2020]. We show how to extend their warp function to SDFs in order to reparameterize an intractable boundary integral. We summarize the necessary background in \( \S 3.1 \). We then derive a warp function for SDFs that is continuous and boundary consistent (\( \S 3.2 \)) as an integral along camera rays, and show how to compute it via quadrature using sphere tracer points (\( \S 3.3 \)). In Section 3.4, we finally give an unbiased approximation for this warp that is tractable for use with neural SDFs. Section 3.5 provides details on how to use our approach to solve inverse rendering problems.

3.1 Background: boundary-aware warping

Without loss of generality, assume a box pixel filter, so that \( \mathcal{U} \subset \mathbb{R}^2 \) is the image plane region corresponding to the pixel of interest. Let \( L(u; \theta) \) denote the radiance along the ray from \( u \in \mathcal{U} \), a point on the image plane, and denote \( \theta \in \mathbb{R}^N \) the vector of geometric scene parameters (e.g. neural network weights). In matrix expressions below, we will assume vector quantities \( (u, x, \theta) \) to be row vectors, and gradients with respect to \( \theta \) to be column vectors.

We aim to compute the gradient of the rendering integral \( I \) with respect to parameters \( \theta \):

\[
\frac{\partial I}{\partial \theta} = \frac{\partial}{\partial \theta} \int_{\mathcal{U}} L(u; \theta) du.
\]

(1)

Primary visibility discontinuities make the radiance function non-differentiable along occlusion boundaries (Fig. 3). Denoting \( \mathcal{U}_{\text{sil}}(\theta) \subset \mathcal{U} \) the set of object silhouettes, for a point \( u_{\text{sil}} \in \mathcal{U}_{\text{sil}} \), the radiance \( L(u_{\text{sil}}; \theta) \) is discontinuous in \( \theta \). This makes naive automatic differentiation methods applied to the Monte Carlo sampling of \( I \) produce incorrect gradients since they ignore the Dirac delta that arises from the differentiation.

Li et al. [2019; 2018] and Zhang et al. [2019] showed that Eq. (1) can be split into two terms: an interior integral, for contributions away from discontinuities and a boundary integral, along discontinuities:

\[
\frac{\partial I}{\partial \theta} = \int_{\mathcal{U}} \frac{\partial}{\partial \theta} L(u; \theta) du + I_{\text{sil}}.
\]

(2)

The second integral \( I_{\text{sil}} \) is harder to compute because sampling the boundary is difficult. This is particularly true for SDFs whose surface boundaries admit no easy parametric form. We will not cover boundary sampling in detail, since we will not use it; instead, we will use a result from Bangaru et al. [2020], who showed, using the divergence theorem, that this boundary term can be turned into an integral over the interior \( \mathcal{U} \setminus \mathcal{U}_{\text{sil}}(\theta) \), which is easier to sample:

\[
I_{\text{sil}} = \int_{\mathcal{U} \setminus \mathcal{U}_{\text{sil}}(\theta)} \nabla_u \cdot (L(u; \theta) \mathcal{V}(u; \theta)) du.
\]

(3)

Here \( \nabla_u \) is the divergence operator, and \( \mathcal{V}(u; \theta) \in \mathbb{R}^{N \times 2} \) is a warping function required to satisfy two properties:

1. **continuity**: \( \mathcal{V}(\cdot; \theta) \) is continuous on \( \mathcal{U} \), and

Figure 2: Overview. A standard SDF rendering pipeline is generally discontinuous, which means there are points \( u \) where the rendering function \( L(u; \theta) \) is not differentiable in \( \theta \), highlighted in red (a). Our method uses intermediate points from a sphere tracer applied to an SDF \( f \), to compute a warp function \( \mathcal{V} \) (c). Using this warp, we reparameterize the integration domain to avoid discontinuities (d), which allows us to compute correct gradients of the rendering equation. The key to achieving this is to design the warp \( \mathcal{V} \) so it is continuous in \( u \) everywhere, and satisfies some consistency criterion on the geometric boundaries.
We show necessary and sufficient conditions on the weights to make point \( u \) consistency condition in Section 3.1, requires that, at a discontinuity 3.2.1 Boundary consistency for implicit functions.

\[ \begin{align*}
\text{2D pixel space side view of the 3D scene}
\end{align*} \]

Figure 3: As geometric parameters \( \theta \) vary, visibility creates discontinuities in the rendering function \( L(u; \theta) \) which traditional automatic differentiation cannot handle (left). These discontinuities correspond to changes in the weights to make \( \mathcal{U} \) through the inverse Jacobian.

(2) boundary consistency: \( \mathcal{V} \) agrees with the derivative of the discontinuity conditions when \( u \) approaches the discontinuity. That is, \( \lim_{u \to \text{discont.}} \mathcal{V}(u; \theta) = \delta \mathcal{V}(u; \theta) \) for \( u \in \mathcal{U}(\theta) \).

Bangaru et al. further show the area integral is equivalent to applying the change of variable [Loubet et al. 2019] \( u \mapsto T(u, \theta) = u + (\theta - \theta_0) \mathcal{V}(u; \theta) \) in Eq. (1), where the derivative is computed at \( \theta_0 \), but \( \delta \mathcal{V} = 0 \). After reparameterization we get:

\[ \delta \mathcal{V} = \int \delta \frac{\partial}{\partial \theta} \left[ L(T(u, \theta), \theta) \right] \left[ \mathcal{V}(T(u, \theta)) \right] du. \]  

(4)

Expanding \( T \) and using Eq. (3), one can show that Eq. (4) indeed computes \( \delta \mathcal{V} \). Intuitively, the reparameterization \( T \) moves each point on the boundary locally at the velocity of their derivatives, essentially removing the discontinuities, while the determinant term accounts for the change of measure.

The main goal of this paper is to derive a suitable form for \( \mathcal{V}(u; \theta) \) for SDFs, that can be tractably computed, so that we can evaluate Eq. (4) using Monte Carlo estimation.

Rendering. To render an SDF \( f \) and compute \( L(u; \theta) \), we need to find the closest intersection point \( x(u, t) \in \mathbb{R}^3 \) such that \( f(x; \theta) = 0 \), where \( t \) is the distance along the ray associated with pixel location \( u \). To find the intersection distance, we use space tracing [Hart 1996], which applies a fixed-point iteration to generate a sequence of points \( x_n \in \mathcal{T}(u) \), such that \( \lim_{n \to \infty} x_n = x \).

3.2 Continuous boundary-consistent warp for SDFs

In this section, we construct an idealized warp function \( \mathcal{W} \) that satisfies the continuity and boundary-consistency conditions of Section 3.1. First, we derive the boundary gradient \( \partial \mathcal{V}(u; \theta) \) with which the warp should agree at silhouette points (§ 3.2.1). We then smoothly extrapolate this gradient using a weighted integral along the primary ray passing through \( u \), to obtain our warp function (§ 3.2.2). We show necessary and sufficient conditions on the weights to make the warp continuous and boundary-consistent (§ 3.2.3).

3.2.1 Boundary consistency for implicit functions.

The boundary consistency condition in Section 3.1, requires that, at a discontinuity point \( u \) the warp agrees with \( \partial \mathcal{V}(u; \theta) \). The derivation proposed by Bangaru et al. [2020] does not apply directly to implicit surfaces,

\[ \begin{align*} 
\text{so we derive this boundary derivative using the implicit function theorem. Specifically, the derivative of a scene point } x \in \mathbb{R}^3 \text{ on the surface, i.e., } f(x; \theta) = 0, \text{ w.r.t. parameters } \theta \in \mathbb{R}^N \text{ is given by:} \\
G(x; \theta) := \partial \mathcal{V} x = -\partial \mathcal{V} \int \frac{\partial f}{\partial \mathcal{V} x} T \partial \mathcal{V} x, \\
\text{where } \mathcal{V} x \in \mathbb{R}^N. 
\end{align*} \]  

(5)

The above directly follows from the implicit function theorem applied to \( f(x; \theta) = 0 \). This implicit velocity formulation also appears in previous work [Remelli et al. 2020; Stam and Schmidt 2011; Vicini et al. 2022]. To get the derivative in pixel coordinates \( \partial \mathcal{V} u = \partial \mathcal{V} x \cdot \partial \mathcal{V} u \in \mathbb{R}^N \), we need to project this derivative by the Jacobian \( \partial \mathcal{V} u \in \mathbb{R}^N \), which for a perspective camera can be easily derived by hand. For more generality, we can obtain this Jacobian as the pseudo-inverse \( \delta \) of the forward Jacobian:

\[ \partial \mathcal{V} u = (\partial \mathcal{V} x, u, t)^\dagger. \]  

(6)

Taken together, the derivative at a silhouette point \( u \), with corresponding 3D position \( x_u = x(u; \mathcal{V}, \theta) \), is then:

\[ \partial \mathcal{V} u = G(x_u; \theta) \partial \mathcal{V} u. \]  

(7)

Figure 3 illustrates the geometric configuration.

3.2.2 Extending to a smooth warp \( \mathcal{W}(u; \theta) \) by integration along the ray. Now that we have an expression for the warp at silhouette points, we extend it to all points, by smoothing this term in a consistent manner. Our method leverages the fact that our implicit SDF \( f(x; \theta) \) is continuous in 3D space and achieves smoothing by convolving along the ray (Fig. 4(b)). This avoids casting expensive additional rays which are needed by Bangaru et al. [2020], and also propagates gradients to points in free space near the boundary points. While investigating the free space gradient is outside the scope of this paper, previous work [Oechsle et al. 2021; Wang et al. 2021] have noted that this can have a stabilizing effect on the optimization of neural SDFs. Note that, while they adapt a volumetric rendering model for better convergence, we do so while computing correct boundary gradients for a surface-based representation.

Our proposed warp function smoothly extends Eq. (7) to non-boundary points as follows:

\[ \mathcal{W}(u; \theta) = \int_{t=0}^{t_0} w(x(u, t)) G(x; \theta) \partial \mathcal{V} u dt. \]  

(8)

with \( t_0 \) the distance to the closest intersection, \( t_0 = \infty \) when the ray does not intersect.

3.2.3 Choice of weights. In order to satisfy the boundary consistency criteria, the weights need to asymptotically satisfy the limit:

\[ \lim_{u \to \text{sil}} \frac{w(x(u, t))}{t_0 - t} = \delta(t - t_0), \]  

(9)

where \( \delta \) is the Dirac delta operator.

From Eq. (9), we see that our weights have to depend on some notion of distance to the silhouette. For an implicit function \( f \) that is at least \( C_1 \) continuous, the following constraints implicitly characterize the silhouette points [Gargarlo et al. 2007]:

\[ \begin{align*} 
f(x(u, t); \theta) &= 0, \\
\partial x f(x(u, t); \theta) T \partial x x(u, t) &= 0. 
\end{align*} \]  

(10)
We now have a clear form for our warp function that can be used to compute weights for our integral along any ray.

\[
S(x) = |f(x; \theta)| + \lambda_d |\partial_x f(x; \theta)^T \partial_x x|,
\]

where \(\lambda_d > 0\). This characteristic function is similar to the boundary test function used by Bangaru et al. [2020] for meshes. However, unlike their boundary test, \(S(x)\) is defined everywhere in the SDF’s 3D domain, not just the surface points. This allows us to use these weights for our integral along any ray.

Our final harmonic weights are given by:

\[
w(x) = S(x)^{-\gamma}, \gamma > 2.
\]

For \(\gamma \geq 2\), our weights satisfy the limit in Eq. (9). Intuitively, this is because the \(w(x) \rightarrow \delta(x - x_{sil})\) as \(x \rightarrow x_{sil}\). See our supplementary material for a discussion of correctness, and derivation of \(\gamma \geq 2\).

Fig. 5(a) shows our weight distribution along the ray for all \(u\) in an 1D example sphere tracer.

3.3 Estimating the warp through its quadrature \(V^q\)

We now have a clear form for our warp function that can be used to reparameterize and differentiate the rendering function. Unfortunately, the asymptotical sharpness of our weights required to obtain valid warp, also makes the integral (8) very difficult to sample. For \(u\) close to the silhouette \(U_{sil}\), the weights become very concentrated at the surface boundary, presenting a tricky integrand if we were to uniformly sample along the ray.

Careful importance sampling of areas near the boundary could remedy this, but there is unfortunately no straightforward way to implement this: the weight distribution depends heavily on the configuration of silhouettes near \(u\), dictated by the SDF.

Our approach foregoes stochastic sampling altogether. We construct a trapezoidal quadrature on the series of intermediate points \(x_n \in T(u)\) generated by the sphere tracer, shown in Fig. 5(b). This quadrature estimator for the warp is given by:

\[
V^q(u; \theta) = \frac{\sum_{x_i \in T(u)} w^q(x_i) G(x_i; \theta) \partial_x u}{\sum_{x_i \in T(u)} w^q(x_i)},
\]

where \(w^q(x_i) = w(x_i) (t_i - t_{i-1})/2\), and \(t_i\) is the distance along the ray to sphere tracer point \(x_i\). Assuming the underlying SDF \(f(x; \theta)\) is \(C_1\) continuous, the intermediate points of the sphere tracer are continuous at all \(u \in U_{sil}\). By composition of continuous functions, \(V^q(\cdot; \theta)\) is also continuous.

Our quadrature warp \(V^q\) satisfies the continuity and boundary consistency condition (§3.1). Since we apply trapezoidal quadrature, \(V^q(u; \theta)\) is in general a biased estimator of integral \(V^{int}(u; \theta)\). However, the two terms are equal in the limit as \(u\) approaches the silhouette, i.e., for \(u_{sil} \in U_{sil}\), \(\lim_{u \rightarrow u_{sil}} V^q(u; \theta) = \lim_{u \rightarrow u_{sil}} V^{int}(u; \theta)\), and since the right-hand side is boundary consistent, so is our quadrature warp \(V^q\). See supplemental for a sketch proof of correctness.

3.4 Top-k subset weighting \(\tilde{w}_k\) to reduce memory use

For complex SDFs such as a neural network, our quadrature warp \(V^q\) has the caveat that it requires back-propagating through every sphere tracer point. Previous work like IDR [Yariv et al. 2020] do not have this issue since their (biased) gradient is only computed at the intersection point, and they exclude other points from the gradient computation. Our approach, on the other hand, uses a weighted sum, so we cannot discard intermediate points.

However, as shown in Fig. 5(b), the vast majority of sphere tracer points have negligible weight, and most of the mass is concentrated close to the silhouette. We exploit this by only using the subset of points with the highest weight in our warp estimation. That is, instead of using all of \(T(u)\), we can instead use a top-k subset \(T_k(u)\). Selecting the top-k weights requires adjusting them to ensure that they remain continuous. For a subset size of \(k\), our weights are:

\[
\tilde{w}_k(x) = \begin{cases} w(x) - \min_{x \in T_k(u)} w(x), & \text{if } x \in T_k(u) \\ 0, & \text{otherwise.} \end{cases}
\]

The weights \(\tilde{w}_k(x)\) still produce a continuous warp field (see supplemental for a sketch proof). Intuitively, even though the set of points change as a function of \(u\), whenever this change occurs, the points that swap in or out of the set always have weight 0.

3.5 Inverse Rendering Details

In this section, we briefly discuss some details that make our inverse rendering pipeline tractable.

**Implementation.** Our method requires 3 nested derivative passes to (i) compute normals \(\partial_n f\), (ii) compute Jacobian of the transformation \(\partial_n T\) and (iii) to compute derivatives of the full pipeline \(\partial_\theta [L(T(u, \theta)) \text{det}(\partial_n T(u, \theta))]\). We use the Python JAX automatic differentiation system [Bradbury et al. 2018], which supports nested forward-backward differentiation. We use forward-mode for (i) and (ii), and reverse-mode for (iii). Note that since (i) and (ii) both involve differentiating w.r.t. spatial coordinates, the SDF model must be \(C_1\) continuous in \(x\) (but only \(C_0\) continuous in \(\theta\)). We enforce \(C_1\) continuity through softplus non-linearities instead of using ReLUs.

**Network architecture.** For our inverse rendering results, we use the network architecture shown in Fig. 11 of Yariv et al. [2020]. Since our method is slightly more memory-intensive (even with

\[\text{Note that even though we consider the top } k \text{ weights, only } k - 1 \text{ weights actually have a non-zero contribution to the } \theta\text{-derivative.}\]
Figure 5: **Weight visualization.** A contour plot of a sample 2D SDF (first row). We use an orthographic camera for illustration, so camera rays are parallel to the horizontal axis. We show our three weighting schemes in unnormalized (second row) and normalized (third row) form. Our proposed harmonic weights (a) for $\gamma = 4.0, \lambda_d = 1e-1$ are well approximated by a trapezoidal quadrature on the sphere tracer points (b). The blank regions with no weight can be excluded from the computation, which leads to our proposed top-$k$ subset weights (c), for $k = 8$. This reduces both the compute and memory burden of the backward pass. We visualize the weight in a symlog plot, values are linear in $[0, 10^1]$ and $[0, 10^{-3}]$ for the unnormalized and normalized weights, respectively.

(a) harmonic weight
(b) quadrature approximation
(c) top-$k$ subset weight

Figure 6: **Gradient quality.** We compare the image gradients computed naïvely without reparameterization [Yariv et al. 2020] and with our method against the “ground truth” gradient computed with finite differences for three scenes. Our method properly handles boundary discontinuities both due to object edges (in purple insets) and self-occlusions (in green insets).

Pixel sampling. Similar to Yariv et al. [2020] and other neural methods, we sample a subset of pixels for each iteration since it can be computationally prohibitive to trace the entire image when using a deep neural representation. However, unlike Yariv et al. [2020], which works with a single ray at the center of the pixel, our approach must integrate the spatially-varying warp $V$ over each pixel. We achieve this by Monte-Carlo sampling within each pixel.

Multi-level optimization. Since we only use a subset of pixels, the likelihood of sampling a pixel with silhouette gradient is fairly low. For unbiased derivatives, only pixels that are partially covered by a surface have a non-zero boundary contribution. This is in contrast to approximate derivatives (e.g., [Liu et al. 2019], [Yariv et al. 2020]) that have a wider spatial footprint. To alleviate this issue, we use a multi-scale pyramid of the target image throughout our optimization.

Initialization. We use the geometric network initialization [Atzmon and Lipman 2020] which approximately produces a spherical SDF. We also initializes the weights of the positional encoding layer to 0 [Yariv et al. 2020]. We found this subtle modification implicitly enforces a coarse-to-fine mechanism that yields significantly better generalization to novel views.

Eikonal constraint. We represent our SDF $f$ using a neural network, which does not necessarily satisfy the distance property. We adopt the Eikonal regularization loss [Gropp et al. 2020] to explicitly enforce this. Since the resulting $f$ is only an approximation of an SDF, we pad our weights with a small $\epsilon$ to avoid infinities.
Figure 7: Neural SDF reconstruction. We compare with IDR [Yariv et al. 2020] on three synthetic scenes (top three rows) and two real captured scenes (bottom rows). IDR requires 2D mask supervision, without which it completely diverges. Thanks to our accurate gradient computation, our reconstructions are on par with IDR, without requiring any additional supervision beyond the input images and camera poses. In fact, on the real scenes, our reconstructions (without masks) outperform IDR with masks (see the head of the pony, or the legs, tails and wings of the dragon) because of errors in automatic 2D segmentation.

4 RESULTS

4.1 Ground truth gradient comparisons

We first evaluate the correctness of our gradient by visualizing gradients on three different scenes (illustrated in Fig. 6). For Torus—a analytical torus model textured with a diffuse Perlin noise albedo—we visualize the gradients w.r.t the outer radius (distance from the center to the center of the ring). (Santa and Kitty) are 3D models that we represent as neural SDFs. We take the parameters of the neural SDF from an intermediate iteration during an inverse rendering optimization, and visualize the gradient w.r.t the bias parameter of the last layer output (i.e. the level set perturbation). We also compute the gradient without reparameterization; this is similar to the gradient used in previous SDF-based inverse rendering methods [Yariv et al. 2020]. Note that the interior gradient is largely unaffected by reparameterization, with the gradient at the silhouettes being the largest benefit of our method, especially at self-occlusions. In the next subsection, we show that this boundary gradient is critical and without it, the inverse rendering diverges.

4.2 Comparisons with IDR

We compare our reconstructions against the SDF-based inverse rendering method of IDR [Yariv et al. 2020]. IDR does not correctly account for the boundary term of gradient of the rendering integral and requires additional supervision, in the form of accurate 2D segmentation masks. We implement IDR in our pipeline to ensure that the only difference is our reparameterization. We use the same network architecture for both methods (See Sec. 3.5 for details), and report results after roughly 25,000 network updates. Note that our method uses more samples (2 in the interior + 4 on each pixel boundary) since we use a Monte-Carlo approach to estimate the warp. IDR only requires one sample, fixed at the center of the pixel.
Figure 7 shows that, on three synthetic scenes (Santa, Kitty and Duck), our method without any 2D masks supervision obtains comparable depth and RGB reconstruction as IDR with (perfect) mask supervision. We also show reconstructions of a captured real scene (Pony from Bi et al. [2020b]). Here, we provide IDR with 2D masks derived from a COLMAP reconstruction, which has errors. As a result, our reconstruction outperforms IDR on this scene. Table 1 establishes this result using mean reconstructed PSNR.

We also tried to compare with IDR without mask supervision. In most cases, IDR without masks diverges completely because of the lack of gradients from the silhouette. This is similar to the observation made by Oechsle et al. [2021].

Our pipeline takes 10.1s per iteration on a system with 2× RTX 2080Ti, at 24,576 rays per iteration.

Table 1: PSNR comparison. Our method performs on-par with IDR on all datasets in spite of having no mask supervision.

|          | IDR (with masks) | Ours (w/o masks) |
|----------|------------------|------------------|
| Santa    | 34.86 dB         | 33.59 dB         |
| Kitty    | 36.96 dB         | 33.71 dB         |
| Duck     | 37.30 dB         | 34.14 dB         |
| Pony     | 32.06 dB         | 31.56 dB         |
| Dragon   | 31.64 dB         | 32.72 dB         |

4.3 Ablation study: Subset size

Our top-k weighting scheme reduces the memory footprint of our optimization, but this comes at a cost. The smaller $k$, the sharper the weight landscape. This can cause high variance that can impede the optimization of fine details. We explore this through an ablation study on the Santa dataset, varying $k$ shown below. We use 36 views for this study, and report results after 20,000 network updates. Details are resolved for $k \geq 14$, as shown in Fig. 8.

![Figure 8: Set Size Ablation. We compare reconstructions (at 25,000 updates) using different top-k subset sizes from $k = 3$ to $k = 21$, out of a total of 22 sphere tracer steps. We find that beyond $k = 7$, we obtain diminishing returns on reconstruction quality. In this scenario, that translates to roughly 65% fewer network evaluations stored in memory for the backward pass.](image)

5 CONCLUSION

We have presented a novel method to correctly differentiate neural SDFs rendering. Unlike prior work that relies on accurate masks or biased approximations of the boundary gradients, we reparameterize the pixel filter integral to account for the discontinuities. We have validated the correctness of our approach using finite difference ground truth, and demonstrated superior optimization convergence comparing state-of-the-art neural SDF renderers.

While we have focused on primary visibility in this work, our formulation can be extended to handle global illumination. In particular, we expect to be able to model light rays and jointly optimize for SDF geometry as well as surface reflectance and illumination. Modeling full global illumination with neural SDFs may require extensions or approximations to be computationally tractable. Finally, inverse rendering under unknown, natural illumination is ill-posed and it would be interesting to explore geometry, material and illumination priors that can be combined with our differentiable rendering formulation.

ACKNOWLEDGMENTS

This work was partially completed during an internship at Adobe Research and subsequently funded by the Toyota Research Institute and the National Science Foundation (NSF 2105806). We acknowledge the MIT SuperCloud and the Lincoln Laboratory Supercomputing Center for providing HPC resources. We also thank Shuang Zhao for early discussions, and Yash Belhe for proof-reading.
