Differentiable Signed Distance Function Rendering

DELIO VICINI, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
SÉBASTIEN SPEIERER, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
WENZEL JAKOB, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

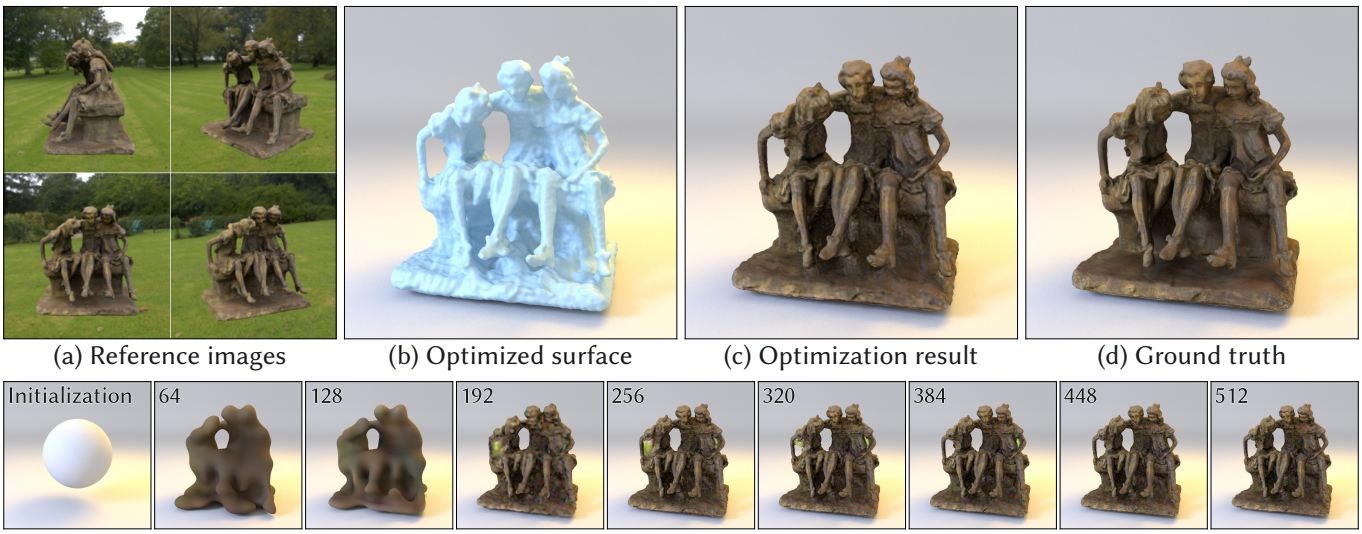


Fig. 1. Image-based shape and texture reconstruction of a statue given 32 (synthetic) reference images **(a)** and known environment illumination. We use differentiable rendering to jointly optimize a signed distance representation of the geometry and albedo texture by minimizing the L_1 loss between the rendered and the reference images. Our method correctly accounts for discontinuities and we therefore do not require ad-hoc object mask or silhouette supervision. We visualize the reconstructed surface **(b)** and the re-rendered textured object **(c)**. The view and illumination condition in **(b)** and **(c)** are different from the ones used during optimization. In **(d)** we render the ground truth triangle mesh.

Physically-based differentiable rendering has recently emerged as an attractive new technique for solving inverse problems that recover complete 3D scene representations from images. The inversion of shape parameters is of particular interest but also poses severe challenges: shapes are intertwined with visibility, whose discontinuous nature introduces severe bias in computed derivatives unless costly precautions are taken. Shape representations like triangle meshes suffer from additional difficulties, since the continuous optimization of mesh parameters cannot introduce topological changes.

One common solution to these difficulties entails representing shapes using signed distance functions (SDFs) and gradually adapting their zero level set during optimization. Previous differentiable rendering of SDFs did not fully account for visibility gradients and required the use of mask or silhouette supervision, or discretization into a triangle mesh.

In this article, we show how to extend the commonly used sphere tracing algorithm so that it additionally outputs a reparameterization that provides the means to compute accurate shape parameter derivatives. At a high level, this resembles techniques for differentiable mesh rendering, though we show that the SDF representation admits a particularly efficient reparameterization that outperforms prior work. Our experiments demonstrate the reconstruction of (synthetic) objects without complex regularization or priors, using only a per-pixel RGB loss.

CCS Concepts: • **Computing methodologies** → **Rendering**.

Additional Key Words and Phrases: differentiable rendering, inverse rendering, signed distance functions, gradient-based optimization, level set method, sphere tracing

ACM Reference Format:
Delio Vicini, Sébastien Speierer, and Wenzel Jakob. 2022. Differentiable Signed Distance Function Rendering. *ACM Trans. Graph.* 41, 4, Article 125 (July 2022), 18 pages. https://doi.org/10.1145/3528223.3530139

Authors' addresses: Delio Vicini, delio.vicini@epfl.ch, École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland; Sébastien Speierer, sebastien.speierer@epfl.ch, École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland; Wenzel Jakob, wenzel.jakob@epfl.ch, École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland.

1 INTRODUCTION

Methods for physically-based differentiable rendering (PBDR) are of increasing interest due to their ability to solve previously intractable inverse problems involving realistic material appearance, shadowing and interreflection. They differentiate physically-based rendering algorithms like path tracing [Kajiya 1986] and use the resulting gradients to minimize non-linear objective functions on high-dimensional domains.

A central challenge in these methods are *discontinuities* that arise at the silhouette of occluders. If not accounted for, these instantaneous changes severely bias derivatives when PBDR methods are used to optimize the scene geometry, which effectively breaks applications such as 3D reconstruction from images.

While rendering is intrinsically differentiable, it can be difficult to retain this property in the transition from equation to algorithm. For example, the direct application of automatic differentiation (AD) to a rendering algorithm will normally not produce usable gradients. Parameters that require special handling include triangle mesh vertices and variables that influence the zero level set of implicitly defined surfaces.

Besides the mathematics of gradient evaluation, a second important concern is the 3D representation underlying the scene. Many options exist, but not all are equally amenable to optimization. Triangle meshes have proven extremely popular in the setting of forward rendering, but inverse problems have considered a wider set of possibilities including point-based, implicit, or volumetric surface representations. Signed distance functions (SDF) are another possibility; renewed interest in them has led to specialized SDF reconstruction techniques that we review in Section 2. A signed distance function measures the signed distance to a surface that is defined by its zero level set. A key advantage of using SDFs as geometric representation is their ability to easily represent topological changes during the optimization process [Osher and Sethian 1988].

While there is a large body of work on using SDFs for inverse rendering, no existing technique can directly differentiate renderings of SDFs with respect to primary, secondary (shadows), or higher-order effects (global interreflections). We generally expect inverse problems to become more useful as their model evaluation and coupled steps like differentiation become increasingly representative of physical reality, hence this is highly desirable. A recent approach that can be adapted to SDFs *reparameterizes* the discontinuous image contribution function [Loubet et al. 2019; Bangaru et al. 2020]. While originally developed for triangle meshes, these methods are independent of the geometry representation and therefore also apply to SDFs. They expose a bias-versus-computation tradeoff, and to achieve high quality gradients, require tracing many costly additional rays.

We propose a specialized reparameterization technique for differentiable optimization of SDFs that addresses these drawbacks. Our method augments the sphere tracing technique [Hart 1996] commonly used for SDF rendering so that it collects a small amount of additional information while stepping through space. We use this information to cheaply instantiate a reparameterization that addresses issues with discontinuous integrals performed by the renderer, so that the remaining calculation can be handled by standard AD techniques. An important difference of our approach compared to prior work is that our parameterization does not need to trace auxiliary rays to sample the surrounding environment in search of occlusion boundaries, since equivalent information is naturally available in the signed distance value of the SDF representation. There are various subtleties that must be considered along the way, however; we show how careful derivation of gradients and Jacobian determinant leads to an effective and robust method that is both faster and more accurate than prior work.

Finally, we demonstrate the use of our method to reconstruct SDFs of complex objects with gradient-based optimization, eschewing ad-hoc silhouette losses or complex priors. An example result of such an optimization is shown in Figure 1. We do not pursue state-of-the art reconstruction from real-world data in this article; our focus is mainly on efficient derivative evaluation for SDFs. In summary, our contributions are the following:

- We propose a modification of sphere tracing that dynamically constructs reparameterizations enabling accurate differentiable SDF rendering.
- We demonstrate the use of our method for surface reconstruction without the need for complex priors or silhouette loss.
- We provide a rigorous derivation of our reparameterization and the resulting distortion of the integration domain.
- We draw a precise connection between using a reparameterization and applying the divergence theorem [Bangaru et al. 2020] for integrals over the unit sphere.

An open-source implementation of our method is available under https://github.com/rgl-epfl/differentiable-sdf-rendering.

2 RELATED WORK

In the following, we give a brief overview of the most closely related differentiable rendering methods.

Signed distance functions. Using a signed distance function representation for optimization is a form of a *level set method* [Osher and Sethian 1988]. Many works have used SDFs or general level set methods for 3D object reconstruction using laser scanners [Curless and Levoy 1996; Zhao et al. 2001] or RGB-D sensors [Newcombe et al. 2011; Nießner et al. 2013]. Level set methods have also been extended to account for the motion of occlusion boundaries [Tsai et al. 2004; Gargallo et al. 2007]. Recently, there has been renewed interest in SDFs with the aim of using them as a geometry representation for differentiable rendering. Concurrently, Jiang et al. [2020] and Liu et al. [2020] proposed to use SDFs for scene reconstruction using differentiable rendering. These works use *sphere tracing* [Hart 1996; Keinert et al. 2014; Seyb et al. 2019] to intersect rays with an SDF. To deal with visibility discontinuities, Liu et al. [2020] introduced a differentiable version of a silhouette loss, similar to later work by Yariv et al. [2020]. Niemeyer et al. [2020] optimize implicit surfaces by adding a 3D loss based on the visual hull [Laurentini 1994].

Relying on a silhouette loss can be problematic. Such approaches are difficult to generalize to shadows and higher order light bounces. Additionally, they cannot work in cases where silhouette information is not applicable (e.g., reconstructing a room from the inside). Cole et al.'s [2021] differentiable splatting method does not require silhouette information, but lacks the ability to generalize to shadows and interreflections. An alternative approach is to convert the SDF to a triangle mesh using marching cubes [Lorensen and Cline 1987] or marching tetrahedra [Doi and Koide 1991], to then fall back to using differentiable triangle mesh rendering methods [Remelli et al. 2020; Shen et al. 2021]. These methods can work well [Munkberg et al. 2022], but do not directly differentiate the process of SDF rendering. Our algorithm avoids meshing and leverages the SDF's global structure to obtain high quality gradients.

Alternative scene representations. While surface-based representations enable efficient rendering and are the standard for physically-based rendering, image-based optimization of surface geometry can

be challenging due to the inherent non-convexity of such an optimization. This problem is exacerbated by scenes containing fine detail that is often at the sub-pixel level (e.g., hair or foliage). Volumetric representations [Lombardi et al. 2019; Mildenhall et al. 2020; Vicini et al. 2021a; Yu et al. 2022] have been found to be a powerful tool to reach a desirable minimum more reliably. Recent hybrid methods [Yariv et al. 2021; Wang et al. 2021] define a volume density based on an underlying SDF to improve the convexity of the geometry reconstruction problem. Unlike our method, these approaches rely on differentiable volume rendering and do not directly differentiate the surface rendering process. Alternatively, point-based shape representations have also been shown to produce high quality scene reconstructions [Yifan et al. 2019; Rückert et al. 2021]. Another key tool for scalable scene representation are coordinate-based neural networks, also called *neural fields*, which push beyond the resolution limits of discretized grids, and which generalize to higher-dimensional signals like directional emission [Mildenhall et al. 2020]. We refer to the recent state of the art report by Xie et al. [2021] for details. A disadvantage of many non-surface-based representations is that they are difficult to reconcile with physically-based light transport, e.g., to account for interreflection.

Differentiable rasterization. Several works [Loper and Black 2014; Kato et al. 2018; Liu et al. 2019; Laine et al. 2020; Cole et al. 2021] have proposed differentiable versions of triangle mesh rasterization. These methods either slightly blur the image or build on the anti-aliasing operation to make the rendering process differentiable. In combination with preconditioned gradient descent, such methods can effectively optimize geometry with a fixed topology [Nicolet et al. 2021]. Rasterization is computationally efficient, but difficult to use to differentiably render soft shadows or indirect illumination.

Physically-based differentiable rendering. Physically-based rendering methods [Pharr et al. 2016] synthesize photorealistic images by explicitly simulating the full process of light transport and scattering via ray tracing. Efficient differentiable physically-based rendering [Gkioulekas et al. 2013; Khungurn et al. 2015; Gkioulekas et al. 2016; Li et al. 2018; Azinović et al. 2019; Nimier-David et al. 2019; Zhang et al. 2019] is an active area of research. One central challenge in differentiating such algorithms are visibility discontinuities (e.g., due to an object occluding the background, or casting a shadow). Under differentiation, these discontinuities introduce a derivative term that has to be integrated over object silhouettes. Standard path tracing algorithms do not sample this term and specialized differentiable rendering methods have been proposed that explicitly sample silhouette edges of triangle meshes [Li et al. 2018; Zhang et al. 2020, 2021]. Sampling these edges is challenging and doing so efficiently requires using an acceleration data structure. The problem becomes even more difficult when dealing with continuous implicit surfaces such as SDFs, where the silhouette does not consist of a discrete number of line segments. An alternative approach is to convert the boundary integral to an area integral by *reparameterizing* the integrand [Loubet et al. 2019; Bangaru et al. 2020]. Our work builds on the reparameterization method which we explain in detail in the next section. Alternative approaches are to build support for discontinuities into a domain specific language [Bangaru et al. 2021] or to use a custom data structure to compute analytic visibility for triangle meshes [Zhou et al. 2021].

Differentiating simulations that account for long light paths with many scattering events tends to be extremely memory-intensive. Recent methods [Nimier-David et al. 2020; Vicini et al. 2021b] address this issue by turning the differentiation into another simulation pass resembling path tracing. These methods can be combined with a reparameterization of the integrand to correctly deal with discontinuities [Zeltner et al. 2021].

3 BACKGROUND

Physically-based rendering [Pharr et al. 2016] computes the intensity of a pixel j as an integral over the space of light paths $\mathcal{P}$

$$I_j(\boldsymbol{\pi}) = \int_{\mathcal{P}} f_j(\mathbf{x}, \boldsymbol{\pi}) \, \mathrm{d}\mathbf{x}, \tag{1}$$

where $\mathbf{x}$ is a light path and $\boldsymbol{\pi}$ is a vector containing the scene parameters (e.g., shape parameters, texture values, etc.). The *image contribution function* f_j measures the contribution of a light path to pixel j. In practice, we estimate this high-dimensional integral using Monte Carlo integration [Kajiya 1986]. In the following, we will simplify the notation by writing all quantities and their derivatives using only a single scalar parameter π. However, all derivations generalize to the *reverse-mode* differentiation case that evaluates derivatives with respect to many parameters at once.

3.1 Differentiable rendering

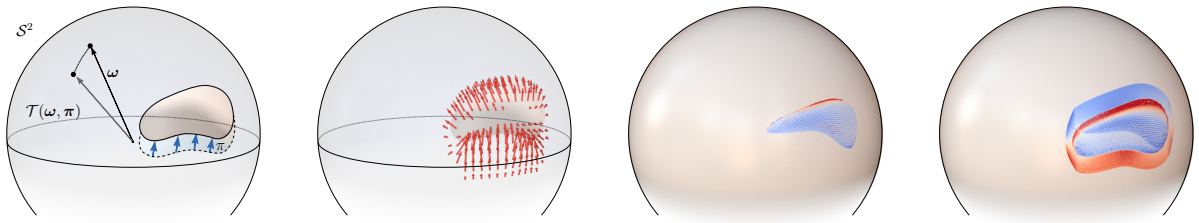
In differentiable rendering, our goal is to differentiate the value of this integral to minimize an image-based objective function over a large set of scene parameters (e.g., using gradient descent). Specifically, we want to estimate the following derivative:

$$\partial_\pi I_j(\pi) = \partial_\pi \int_{\mathcal{P}} f_j(\mathbf{x}, \pi) \, \mathrm{d}\mathbf{x}. \tag{2}$$

If the integrand does not contain any discontinuities that depend on π, we use the Leibniz rule to move the derivative operator inside the integral and apply Monte Carlo integration to estimate derivatives:

$$\partial_\pi I_j(\pi) = \int_{\mathcal{P}} \partial_\pi f_j(\mathbf{x}, \pi) \, \mathrm{d}\mathbf{x} \approx \frac{1}{N} \sum_{k=1}^{N} \frac{\partial_\pi f_j(\mathbf{x}_k, \pi)}{p(\mathbf{x}_k, \pi)}, \tag{3}$$

where N is the number of samples, x_k denote sampled light paths and p is the probability density function (PDF) of the used sampling strategy. The above estimator can, for example, be implemented by evaluating a unidirectional path tracer within a framework supporting automatic differentiation (AD) and differentiating its output [Nimier-David et al. 2019]. One important design decision here is whether the Monte Carlo sampling step itself, and consequently the PDF, are differentiated with respect to the scene parameters or not. It turns out that for most practical use cases, it is preferable to *detach* the sampling strategy and PDF from the differentiation, as done in Equation 3. This greatly simplifies efficient derivative computation [Zeltner et al. 2021; Vicini et al. 2021b] and is what we will do for the remainder of this paper. Detaching the sampling strategy can result in additional variance due to the mismatch between derivative integrand and PDF, but is usually preferable given the resulting simplification of the rendering process.



(a) Integrand (b) Vector field $\partial_\pi \mathcal{T}$ (c) Gradient of original integrand (d) Gradient of reparametrized integrand

Fig. 2. Illustration of a discontinuous integrand on the unit sphere $\mathcal{S}^2$. **(a)** We integrate the color of a shaded shape over a constant-colored background. The scene parameter π controls the translation of the object. **(b)** We then introduce a reparameterization $\mathcal{T}$, which is designed so that the vector field $\partial_\pi \mathcal{T}$ follows the motion of the object boundary and falls off continuously to 0 away from the object. **(c)** We further visualize the gradient of the original integrand and **(d)** the gradient after reparameterizing, including the area change. Blue and red colors indicate negative and positive values, respectively.

The previous estimator assumed that $f_j(\mathbf{x}, \pi)$ did not contain any discontinuities in $\mathbf{x}$ whose position depends on π. This generally ceases to be the case when π controls the shape or position of scene geometry. Such a dependence influences the position of visibility-related discontinuities in f_j, which requires an additional derivative term described by the Reynolds transport theorem [Reynolds 1903; Zhang et al. 2019]. Simply moving the derivative operator into the integral fails to account for this extra term and produces incorrect output [Li et al. 2018]. However, the Reynolds transport theorem is not the only way to handle parameter-dependent discontinuities.

3.2 Reparameterizing discontinuities

An alternative approach that is particularly well-suited for unidirectional path tracing is to *reparameterize* the integral [Loubet et al. 2019; Bangaru et al. 2020]. The idea is that this reparameterized integrand should then be free of parameter-dependent discontinuities. Following this step, it is legal to move the derivative operator inside the integral and estimate parameter derivatives by sampling light paths using standard path tracing. If we correctly account for the distortion due to the reparameterization, this can even result in an unbiased gradient estimator [Bangaru et al. 2020].

Unidirectional rendering algorithms can be expressed as the recursive solution of spherical integrals. In the following, we will therefore differentiate integrals over the unit sphere of directions $\mathcal{S}^2$, instead of the full path space:

$$\partial_\pi I(\pi) = \partial_\pi \int_{\mathcal{S}^2} f(\boldsymbol{\omega}, \pi)\, \mathrm{d}\boldsymbol{\omega}. \tag{4}$$

This formulation for now ignores effects due to interreflections, but that will be sufficient for most derivations. We will connect the results to the general case at the very end. Similar to prior work, we will not handle the very challenging special case of discontinuities observed through perfectly specular interactions (e.g., geometry observed through a water surface).

Given this spherical integral, we can now reformulate the problem using a change of variables $\mathcal{T}\colon \mathcal{S}^2 \to \mathcal{S}^2$ that maps the unit sphere onto itself. The reparameterization $\mathcal{T}$ has to be chosen so that the resulting integrand is free of discontinuities with respect to the scene parameters, which then allows moving the derivative operator into the integral:

$$\begin{aligned}
\partial_\pi I(\pi) &= \partial_\pi \int_{\mathcal{S}^2} f(\boldsymbol{\omega}, \pi)\, \mathrm{d}\boldsymbol{\omega} \\
&= \partial_\pi \int_{\mathcal{S}^2} f\left(\mathcal{T}(\boldsymbol{\omega}, \pi), \pi\right) \left\| \mathrm{D}\mathcal{T}_{\boldsymbol{\omega},\pi}(\mathbf{s}) \times \mathrm{D}\mathcal{T}_{\boldsymbol{\omega},\pi}(\mathbf{t}) \right\| \mathrm{d}\boldsymbol{\omega} \\
&= \int_{\mathcal{S}^2} \partial_\pi \left[f\left(\mathcal{T}(\boldsymbol{\omega}, \pi), \pi\right) \left\| \mathrm{D}\mathcal{T}_{\boldsymbol{\omega},\pi}(\mathbf{s}) \times \mathrm{D}\mathcal{T}_{\boldsymbol{\omega},\pi}(\mathbf{t}) \right\| \right] \mathrm{d}\boldsymbol{\omega}, \quad (5)
\end{aligned}$$

where $\mathbf{s}$ and $\mathbf{t}$ are orthonormal tangent vectors of the unit sphere at $\boldsymbol{\omega}$ and $\mathrm{D}\mathcal{T}_{\boldsymbol{\omega},\pi}$ is the differential of $\mathcal{T}$ with respect to the vector $\boldsymbol{\omega}$. The norm of the cross product of transformed tangent vectors accounts for the distortion in the integration domain, similar to the Jacobian determinant for a change of variables in ambient space.

Constructing a reparameterization. The key challenge in using this approach is constructing a suitable reparameterization $\mathcal{T}$. Bangaru et al. [2020] formalized its requirements: we need $\mathcal{T}$ to satisfy $\partial_\pi \mathcal{T}(\boldsymbol{\omega}_b, \pi) = \partial_\pi \boldsymbol{\omega}_b$ for all directions $\boldsymbol{\omega}_b$ that lie on the set of discontinuities of the original integrand. In other words, differentiating the reparameterization should result in a differential motion that perfectly matches the motion of the discontinuity on the unit sphere. Note that $\partial_\pi \mathcal{T}(\boldsymbol{\omega}_b, \pi)$ and $\partial_\pi \boldsymbol{\omega}_b$ are vectors that lie in the tangent space of the unit sphere. Aside from that, we need $\partial_\pi \mathcal{T}(\boldsymbol{\omega}, \pi)$ itself to be continuous for the reparameterization to be valid. Further, it is necessary that the integration domain either has no boundaries (e.g., the unit sphere), or the integrand goes to zero as it approaches the domain boundary.

Figure 2 illustrates a reparameterization of an integral over the unit sphere. We visualize the original integrand, the vector field $\partial_\pi \mathcal{T}(\boldsymbol{\omega}, \pi)$, and the derivatives of the original and reparameterized integrand. The gradient of the original integrand does not contain any terms related to the occlusion change on the silhouette of the moving object.

We construct $\mathcal{T}$ to be an identity ($\mathcal{T}(\boldsymbol{\omega}, \pi) = \boldsymbol{\omega}$) for the current parameter value π, while the derivative $\partial_\pi \mathcal{T}(\boldsymbol{\omega}, \pi)$ satisfies the requirements explained above. This approach yields the correct derivative integral without modifying the primal radiance estimator.

For triangle meshes, a suitable reparameterization can be obtained by first constructing an auxiliary reparameterization, which attains the correct motion at triangle boundaries. This reparameterization is then convolved with a filter kernel in the spherical domain, that removes discontinuities from the reparameterization itself [Bangaru

et al. 2020]. Since this blurring only affects the reparameterization, it can be used to construct an unbiased gradient estimator of the original integral.

Distortion of the integration domain. The area element in Equation 5, $\left\|\mathrm{D}\mathcal{T}_{\boldsymbol{\omega},\pi}(\mathbf{s}) \times \mathrm{D}\mathcal{T}_{\boldsymbol{\omega},\pi}(\mathbf{t})\right\|$, accounts for the distortion of the integration domain due to the reparameterization. If we were to reparameterize the 3D ambient space, we would simply use the Jacobian determinant of the mapping. However, here this would be incorrect as the reparameterization works on a manifold. We instead need to use the norm of the cross product of the tangent vectors, mapped through the differential of the reparameterization.

This formulation of the reparameterized integral using the cross product of the transformed tangent vectors is more explicit than what has been described in previous work. In Appendix A, we draw the connection between this formulation as a reparameterization over the unit sphere and the divergence formulation by Bangaru et al. [2020]. We show that we can evaluate either the cross product term or the divergence of the mapping and both will give the same results when differentiated, and correctly account for the fact that we reparameterize an integral over a manifold.

4 METHOD

In the following, we first briefly describe how we store and render SDFs. We then discuss differentiable SDF shading and then finally introduce our reparameterization method to handle visibility discontinuities.

4.1 Preliminaries

For a surface $\mathcal{M} \subset \mathbb{R}^3$, the signed distance function $\phi\colon \mathbb{R}^3 \to \mathbb{R}$ measures the distance of a point $\mathbf{x} \in \mathbb{R}^3$ to the surface. We assume the distance to be of negative sign inside and positive outside the surface. Figure 3 shows a slice through an example SDF. In the following, we will denote the value of the SDF as $\phi(\mathbf{x}, \boldsymbol{\pi})$, where $\boldsymbol{\pi}$ is the vector of parameters defining the SDF. A key property of SDFs is that they satisfy the eikonal equation:

$$\|\partial_{\mathbf{x}}\phi(\mathbf{x}, \boldsymbol{\pi})\| = 1, \tag{6}$$

i.e., the positional gradient has unit norm. For a point on the surface $\mathcal{M}$, its surface normal equals the positional gradient $\partial_{\mathbf{x}}\phi(\mathbf{x}, \boldsymbol{\pi})$.

SDF representation. In this paper, we store signed distance functions on a voxel grid. With this representation, $\boldsymbol{\pi}$ represents the list of stored values. During rendering, the grid values are interpolated using cubic B-spline basis functions. Sufficiently high-order interpolation is important, since normals are related to the derivative of the SDF. Simple trilinear interpolation would result in discontinuous shading producing an undesirable faceted appearance. We interpolate positional gradient and Hessian using analytic derivatives of the basis functions and leverage the continuity of the SDF's positional gradient to construct a reparameterization. Our method relies on the interpolation smoothing out any potential discontinuities in the positional gradient (including on the SDF's skeleton).

Ray intersection. We use the sphere tracing algorithm [Hart 1996] to render the SDF representation. Sphere tracing is an iterative procedure that efficiently skips through empty space. In each iteration, the step size is equal to the absolute value of the SDF, i.e., the unsigned distance to the surface, at the current location $\mathbf{x}_i$. This means the algorithm takes a step given the minimum distance to the surface, which ensures that we do not accidentally step over it. Eventually, the ray will either escape into the void, or the evaluated distance will fall below a specified convergence threshold ε, and the ray intersection location is returned. Our method then additionally uses the intermediate SDF evaluations to construct a reparameterization.

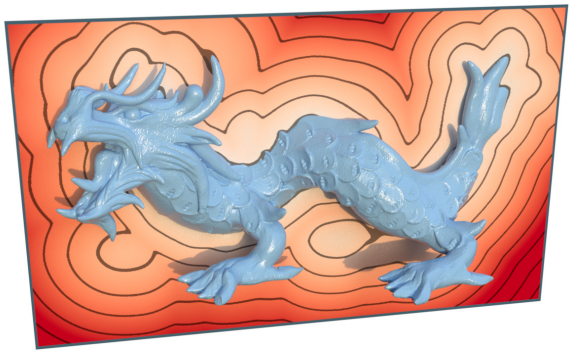


Fig. 3. A signed distance function is positive outside the object and negative inside. Here we visualize a slice of the SDF and its corresponding isolines. In our implementation, we store the values of the SDF on a regular grid and use B-spline interpolated lookups to ensure smooth normals.

Notation. As in Section 3, the following derivations use a single parameter π rather than the parameter vector $\boldsymbol{\pi}$. It will further be useful to distinguish between uses of π that are differentiated, or *attached* to the automatic differentiation graph, and uses that are *detached*. We will denote detached parameters as π_0.

4.2 Shading gradients

Aside from handling discontinuities, differentiable rendering of SDFs also requires the ability to differentiate the evaluation of the surface normal that is later used when evaluating the shape's reflectance model. If a ray intersects the surface defined by the SDF, the shading normal at the intersection location is given by

$$\mathbf{n}(\pi) = \frac{\partial_{\mathbf{x}}\phi(\mathbf{x}_{t(\pi)}, \pi)}{\left\|\partial_{\mathbf{x}}\phi(\mathbf{x}_{t(\pi)}, \pi)\right\|}, \tag{7}$$

where $t(\pi)$ is the intersection distance, $\mathbf{x}_{t(\pi)} \coloneqq \mathbf{x}_\mathrm{o} + t(\pi)\boldsymbol{\omega}$ the intersection location on the surface, with the ray origin $\mathbf{x}_\mathrm{o}$ and the ray direction $\boldsymbol{\omega}$. The normalization is needed, since the grid-interpolated SDF representation cannot guarantee that the eikonal constraint is perfectly satisfied. To differentiate $\mathbf{n}(\pi)$, special care is required, since the intersection distance t depends on π and is the result of sphere tracing, a numerical root finding procedure. Using the inverse function theorem [Niemeyer et al. 2020; Yariv et al. 2020], one can show that

$$\partial_\pi t(\pi) = -\frac{\partial_\pi \phi(\mathbf{x}_{t(\pi_0)}, \pi)}{\langle \partial_{\mathbf{x}}\phi(\mathbf{x}_{t(\pi_0)}, \pi_0), \boldsymbol{\omega}\rangle}, \tag{8}$$

where $\langle \cdot, \cdot \rangle$ denotes the dot product. Using this expression, we can differentiate the surface normal and correctly account for its dependency on the intersection distance, without the need to track parameter derivatives across sphere tracing iterations.

4.3 Reparameterizing discontinuities

We now turn to our reparameterization for differentiable SDF rendering, decomposing the problem into two steps: first, we define a vector field, whose derivative follows the motion of the SDF surface in 3D. We then show how evaluating this vector field along continuous positions in 3D space enables constructing a reparameterization on the unit sphere, which can correctly handle occlusion and self-occlusion by SDFs. Bangaru et al. [2020] followed a similar two-step strategy to define a reparameterization for rendering triangle meshes, but theirs is constructed from a set of auxiliary rays that must be separately traced.

Motion of implicit surfaces. Our eventual goal is to define a reparameterization on the unit sphere. We begin by defining an auxiliary 3D vector field $\mathcal{V}\colon \mathbb{R}^3 \to \mathbb{R}^3$. It is constructed so that the derivative $\partial_\pi \mathcal{V}(\mathbf{x}, \pi) \in \mathbb{R}^3$ matches the infinitesimal surface motion with respect to π when evaluated on the zero level set.

Since tangential motion does not affect discontinuities, we define $\mathcal{V}$ as a scaled multiple of the surface normal, specifically

$$\mathcal{V}(\mathbf{x}, \pi) = -\frac{\partial_{\mathbf{x}} \phi(\mathbf{x}, \pi_0)}{\|\partial_{\mathbf{x}} \phi(\mathbf{x}, \pi_0)\|^2} \phi(\mathbf{x}, \pi). \tag{9}$$

Here, only the value of the SDF $\phi(\mathbf{x}, \pi)$ depends on the differentiable parameter π, while the positional gradient and its squared norm are static and hence written using π_0. Similar expressions have been used in prior work on the motion of implicit surfaces [Stam and Schmidt 2011], neural level sets [Atzmon et al. 2019] and differentiable marching cubes [Remelli et al. 2020]. We can verify that the gradient of this vector field matches the surface motion by differentiating with respect to the parameter π:

$$\begin{aligned}
\partial_\pi \mathcal{V}(\mathbf{x}, \pi) &= -\frac{\partial_{\mathbf{x}} \phi(\mathbf{x}, \pi_0)}{\|\partial_{\mathbf{x}} \phi(\mathbf{x}, \pi_0)\|^2} \partial_\pi \phi(\mathbf{x}, \pi) \\
&= -\frac{\frac{\partial_{\mathbf{x}} \phi(\mathbf{x}, \pi_0)}{\|\partial_{\mathbf{x}} \phi(\mathbf{x}, \pi_0)\|}}{\left\langle \partial_{\mathbf{x}} \phi(\mathbf{x}, \pi_0), \frac{\partial_{\mathbf{x}} \phi(\mathbf{x}, \pi_0)}{\|\partial_{\mathbf{x}} \phi(\mathbf{x}, \pi_0)\|} \right\rangle} \partial_\pi \phi(\mathbf{x}, \pi) \\
&= \frac{-\mathbf{n}}{\langle \partial_{\mathbf{x}} \phi(\mathbf{x}, \pi_0), \mathbf{n} \rangle} \partial_\pi \phi(\mathbf{x}, \pi) \\
&= \partial_\pi \left[\mathbf{x}_o + t(\pi) \mathbf{n} \right] = \partial_\pi \mathbf{x}(\pi),
\end{aligned} \tag{10}$$

where $\mathbf{n} = \frac{\partial_{\mathbf{x}} \phi(\mathbf{x}, \pi_0)}{\|\partial_{\mathbf{x}} \phi(\mathbf{x}, \pi_0)\|}$ is the surface normal. The expression on the third line is exactly the motion of a surface point $\mathbf{x}$ that is the result of intersecting a ray in the normal direction $\mathbf{n}$ with the SDF. This follows from plugging $\boldsymbol{\omega} = \mathbf{n}$ into Equation 8, which describes the intersection distance gradient. In this case, the ray origin $\mathbf{x}_o$ just needs to be any point along this ray such that the ray intersects the SDF perpendicularly. Note that this is just a construction to prove that $\partial_\pi \mathcal{V}(\mathbf{x}, \pi)$ has the right direction and magnitude, we do not need to actually compute such a ray intersection.

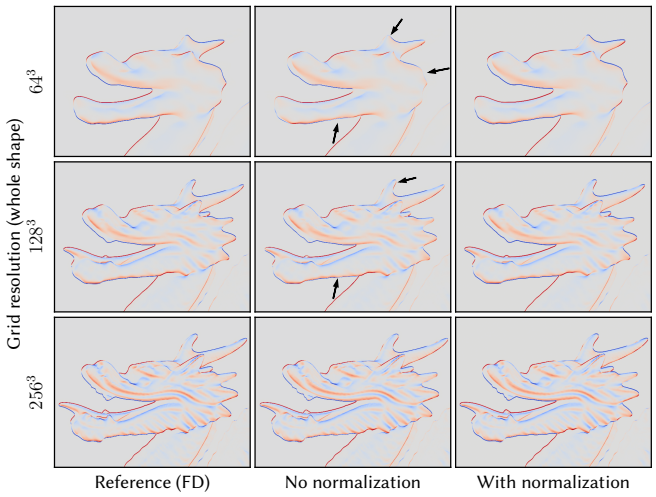
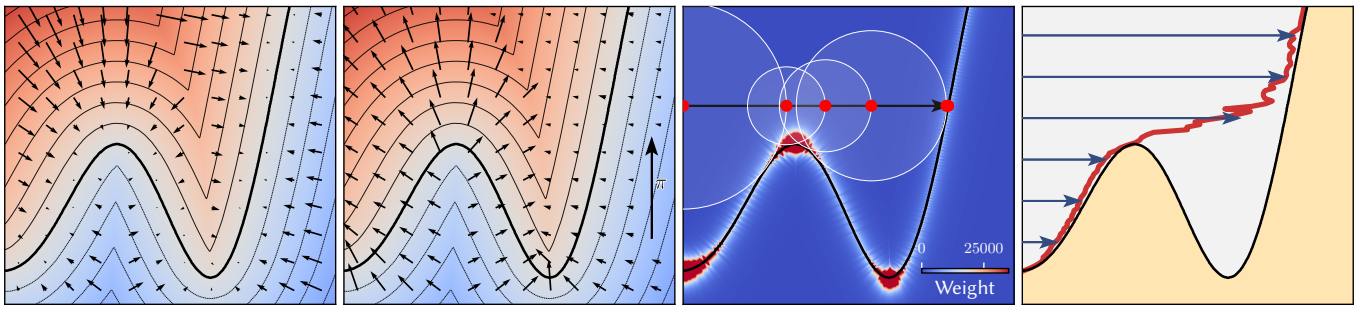


Fig. 4. We visualize the gradients of a rendered image with respect to a vertical translation of an object represented as an SDF. If we do not include the normalization term described in Equation 9, the magnitude of the gradient (**middle** column) does not quite match the reference. Including the normalization improves the accuracy of the estimated silhouette gradients (**right**). As the SDF grid resolution is increased (y-axis), the interpolated SDF more closely matches a true SDF, and the effect of the normalization is less pronounced.

So far, this derivation does not explicitly assume the function ϕ to be an SDF. If ϕ is indeed an SDF, the gradient norm in the denominator in Equation 9 equals 1. However, we found that including the normalization by the squared gradient norm makes our method more robust when working with approximate (e.g., grid-interpolated) SDFs. The effect of this is shown in Figure 4.

Reparameterization of the unit sphere. We now use this 3D vector field to define a reparameterization in the solid angle domain. Recall the primary requirement of the reparameterization: for a direction $\boldsymbol{\omega}_b$ on a discontinuity on the unit sphere, the reparameterization's gradient $\partial_\pi \mathcal{T}(\boldsymbol{\omega}_b, \pi)$ needs to exactly match the motion of that boundary direction.

The key idea is the following: we define an *evaluation distance function* $t(\boldsymbol{\omega}, \pi_0)$ that, as a boundary is approached, converges to the distance at which the edge of the SDF causing the discontinuity is located in 3D. One important detail is that this distance must itself be continuous in $\boldsymbol{\omega}$, or the requirements on the reparameterization would be violated. We compute this distance as a weighted combination of the distances that are encountered during sphere tracing, with weights chosen so that the weighted sum will have the right convergence characteristics as it approaches a boundary. Figure 5 illustrates the high-level idea. From an implementation point of view, this corresponds to computing not just the intersection distance during sphere tracing, but one additional distance that we can then use to define our reparameterization. One important property of this distance computation is that it does not depend on the differentiable parameter π. This means that we do not need to compute expensive parameter derivatives $\partial_\pi \phi(\mathbf{x}, \pi)$ within the sphere tracing loop.



(a) $\mathcal{V}(\mathbf{x}, \boldsymbol{\pi})$ (b) $\partial_\pi \mathcal{V}(\mathbf{x}, \boldsymbol{\pi})$ (c) Sphere tracing steps (d) Evaluation distance

Fig. 5. Our reparameterization builds on a vector field $\mathcal{V}(\mathbf{x}, \pi)$ that is defined everywhere in ambient space **(a)**. Taking the derivative of that vector field with respect to a scalar parameter π will result in a vector field $\partial_\pi \mathcal{V}$ that follows the parameter-dependent motion due to π. In **(b)**, we differentiate the vector field with respect to a global translation in the vertical direction. For a given ray direction, we then compute a weighted combination of positions encountered during sphere tracing to determine where to evaluate the vector field $\partial_\pi \mathcal{V}$. In **(c)** we visualize the intermediate sphere tracing steps and weights that influence the final evaluation point. In **(d)** we draw the computed vector field evaluation locations as a red line as the origin of the incident ray changes. The red line is continuous and coincides with the surface on silhouette edges. In an actual shape optimization, we differentiate $\mathcal{V}(\mathbf{x}, \boldsymbol{\pi})$ with respect to all SDF parameters $\boldsymbol{\pi}$ (i.e. all individual grid values) simultaneously using reverse-mode automatic differentiation.

For now, we assume this distance t to be given and we will define it precisely later. Given the distance, we construct a reparameterization on the unit sphere by first defining an auxiliary function:

$$\begin{aligned}\bar{\mathcal{T}}(\boldsymbol{\omega}, \pi) &= [\mathbf{x}_t + \mathcal{V}(\mathbf{x}_t, \pi) - \mathcal{V}(\mathbf{x}_t, \pi_0)] - \mathbf{x} \\ &= t\boldsymbol{\omega} + \mathcal{V}(\mathbf{x}_t, \pi) - \mathcal{V}(\mathbf{x}_t, \pi_0),\end{aligned} \tag{11}$$

where $\mathbf{x}$ is the ray origin of the ray along $\boldsymbol{\omega}$ and $\mathbf{x}_t := \mathbf{x} + t\boldsymbol{\omega}$. The idea here is to take the 3D location $\mathbf{x}_t$ and displace it using our vector field $\mathcal{V}$. We then subtract the ray origin $\mathbf{x}$ to turn this expression into a direction aligned with $\boldsymbol{\omega}$. The subtraction of $\mathcal{V}(\mathbf{x}_t, \pi_0)$ ensures that the map is simply scaling $\boldsymbol{\omega}$ when no derivative is being taken. Since $\bar{\mathcal{T}}(\boldsymbol{\omega}, \pi)$ is not yet a vector of unit length, we normalize to obtain a reparameterization of the unit sphere:

$$\mathcal{T}(\boldsymbol{\omega}, \pi) = \frac{\bar{\mathcal{T}}(\boldsymbol{\omega}, \pi)}{\left\|\bar{\mathcal{T}}(\boldsymbol{\omega}, \pi)\right\|}. \tag{12}$$

In the primal domain, this is now an identity map from the unit sphere onto itself. We further need the derivative of this map to follow the motion of the implicit surface over the unit sphere. We can show this by explicitly computing the derivative $\partial_\pi \mathcal{T}$:

$$\partial_\pi \mathcal{T}(\boldsymbol{\omega}, \pi) = \frac{1}{t}\left(\mathbb{I} - \boldsymbol{\omega} \cdot \boldsymbol{\omega}^T\right) \partial_\pi \mathcal{V}(\mathbf{x}_t, \pi), \tag{13}$$

where $\mathbb{I}$ is the 3 by 3 identity matrix and $\boldsymbol{\omega} \cdot \boldsymbol{\omega}^T$ is the outer product of $\boldsymbol{\omega}$ with itself (see Appendix B for the expanded derivation). This derivative expression has a simple geometric interpretation: we constructed $\mathcal{V}$ such that its gradient $\partial_\pi \mathcal{V}(\mathbf{x}_t, \pi)$ follows the motion of the surface in 3D space. We then assumed that t was computed such that if $\boldsymbol{\omega}$ approaches a discontinuity, the evaluation distance will converge to the distance to the SDF edge along the current ray. By multiplying the vector field gradient by $\mathbb{I} - \boldsymbol{\omega} \cdot \boldsymbol{\omega}^T$, we project it onto the unit sphere's tangent space. This means any motion in the direction of $\boldsymbol{\omega}$ will be removed. Additionally, the division by t can be interpreted as a form of a geometry term: the motion of the 3D surface over the sphere is decreasing linearly as the evaluation location t moves further away. Overall, this means that our reparameterization attains the right motion on the discontinuities. What remains is to define the evaluation distance t more precisely.

Reparameterization evaluation distance. We evaluate our 3D vector field $\mathcal{V}$ at a distance t along the current ray. We obtain a distance function that is both continuous and has the right characteristics on the discontinuities by computing a weighted sum of distances along the ray that are encountered during sphere tracing:

$$t = \frac{1}{\sum_{i=1}^{N} w(i)} \sum_{i=1}^{N} w(i) t_i, \tag{14}$$

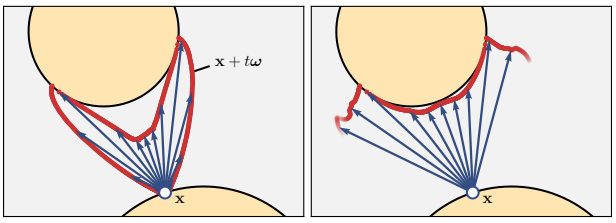
where t_i are the intermediate distances attained during sphere tracing and w is a weighting function. We define our weighting function as a product of three terms:

$$w(i) = w_{\text{edge}}(i) w_{\text{dist}}(i) w_{\text{bbox}}(i). \tag{15}$$

The first factor detects proximity to discontinuities:

$$w_{\text{edge}}(i) = \left(\varepsilon + |\phi(\mathbf{x}_t, \pi_0)| + \alpha \left\langle \frac{\partial_{\mathbf{x}} \phi(\mathbf{x}_t, \pi_0)}{\|\partial_{\mathbf{x}} \phi(\mathbf{x}_t, \pi_0)\|}, \boldsymbol{\omega} \right\rangle^2\right)^{-p}, \tag{16}$$

where we use $\varepsilon = 10^{-6}$, $\alpha = 0.1$, $p = 2$ and $\boldsymbol{\omega}$ is the ray direction. This weighting function is designed such that $w \to \infty$ as a surface is approached at a grazing angle (i.e., the sphere tracing reaches an edge causing a discontinuity). The dot product between the ray direction and the (normalized) gradient ensures that the weight only goes to infinity at grazing angles. Without this term, the evaluation distance would coincide with the ray intersection distance and not be continuous. It is crucial for this distance function to be continuous in $\boldsymbol{\omega}$, as otherwise the resulting gradients would be incorrect. This weighting scheme serves a similar purpose as the weights used by Bangaru et al. [2020]. In their method, the weights are used for a 2D convolution that has to be evaluated using Monte Carlo integration, whereas our implementation re-uses the locations that are already sampled during sphere tracing.



(a) $\mathbf{x} + t\boldsymbol{\omega}$ without w_{dist} (b) $\mathbf{x} + t\boldsymbol{\omega}$ with w_{dist}

Fig. 6. We compare the evaluation distance function t with and without the w_{dist} factor. Without this factor, the evaluation distance approaches zero for grazing outgoing ray directions. Including w_{dist} reduces the influence of the surface at the ray origin on the evaluation distance. This also implies that the evaluation distance is undefined for rays that never approach a surface, and we need to make sure our reparameterization continuously goes to zero before reaching this case.

This first weighting term is not quite sufficient to robustly estimate gradients however. For indirect rays starting on the SDF itself it will likely select an evaluation distance that is very close to the ray origin. This is in particular the case for rays leaving a surface at near grazing angles, causing unnecessary variance without improving gradient estimation. We therefore multiply by a second term:

$$w_{\text{dist}}(i) = \min\left[\sum_{j=1}^{i} \max\left(\frac{\left|\phi(x_{t_{j-1}}, \pi_0)\right| - \left|\phi(x_{t_j}, \pi_0)\right|}{\min\left(\beta, \left|\phi(x_{t_j}, \pi_0)\right|\right)}, 0\right), 1\right], \quad (17)$$

where j iterates over sphere tracing steps and $\beta = 0.05$. While the formula appears complicated, the intuition is simple: we only start considering sphere tracing locations as a surface is *approached*. This present formulation ensures that this is done in a way that remains continuous, since we want the final distance t to vary continuously as the ray direction changes. Moreover, the distance term in the denominator ensures that the weight is guaranteed to reach 1 as the surface is reached. There is little extra cost in adding this weighting term, as it just re-uses evaluations of the SDF that are already computed during sphere tracing. The effect of this weighting term is illustrated on a 2D example in Figure 6.

Lastly, we need to ensure continuity of the evaluation distance even as the number N of sphere tracing steps changes. Such a change can either be caused by the algorithm traversing beyond the bounding box, or by a different number of iterations being needed to converge to the surface intersection. The first case is handled by the bounding box weight term, which attenuates the influence of positions as the bounding box of the SDF is approached:

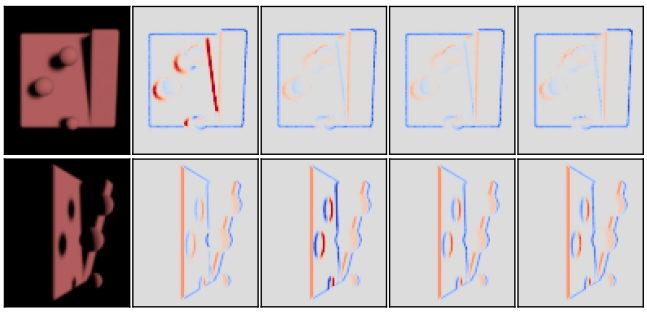
$$w_{\text{bbox}}(i) = \min\left(\text{dist}(x_{t_i}, \text{bbox})/0.01, 1\right). \quad (18)$$

To deal with the second case, we additionally multiply each weight by the mean of the previous and current sphere tracing step length. This works because the iteration count increases or decreases due to samples either being right below or newly above the distance threshold used to terminate a ray intersection. In these cases, the distance between subsequent samples will approach zero. Conceptually, this weighting scheme can be interpreted as replacing the summation in Equation 14 by integration and applying a trapezoidal quadrature to both integrals:

$$t = \frac{1}{\int w(t)\,\mathrm{d}t} \int w(t) t\,\mathrm{d}t. \quad (19)$$

A key insight here is that we do not need these integrals to be evaluated in an unbiased way. We just need the resulting function $t(\boldsymbol{\omega}, \pi_0)$ to satisfy the necessary conditions and the resulting gradient estimator will be unbiased.

While our algorithm here is designed to work with SDFs, the high level idea could potentially also be applied to more general implicit functions, where, instead of sphere tracing, ray marching and bisection are used to compute ray intersections. Such a generalization would still require the implicit function to be continuous and exhibit sufficient global structure to enable a suitable evaluation distance computation (e.g., an indicator function would not work).



Scene (a) Detached $\mathbf{x}$ (b) Decoupled reparam. (c) Correct nesting (d) Reference (FD)

Fig. 7. This figure shows the subtleties of nesting reparameterizations. We render the same object from two different viewpoints and compute gradients with respect to a translation. If we fully detach the origin $\mathbf{x}$ of the shadow ray **(a)** the result is wrong. If we do not track the effect of the reparameterization of the primary rays, we also get wrong results **(b)**. Only if we carefully account for these effects we get output **(c)** that matches the reference **(d)**.

Weighted reparameterization. We can further reduce the variance of the gradient estimator by attenuating the effect of the reparameterization for directions that are further away from an actual discontinuity. In general, we can multiply our 3D vector field with any continuous weighting function $w_{\mathcal{V}}(\mathbf{x}, \pi_0)$, as long as its value approaches 1 as it approaches a discontinuity (with respect to the ray direction $\boldsymbol{\omega}$). We write a weighted version of our vector field as:

$$\bar{\mathcal{V}}(\mathbf{x}, \pi) = w_{\mathcal{V}}(\mathbf{x}, \pi_0)\mathcal{V}(\mathbf{x}, \pi). \quad (20)$$

The weighting function itself depends on the detached scene parameter, which ensures that the scene parameter gradient remains unchanged. This approach could even be generalized to a weighted sum of vector fields, similar to multiple importance sampling [Veach and Guibas 1995]. The weights would only need to sum to 1 on an actual discontinuity. A similar idea has been used by Zeltner et al. [2021] to handle discontinuities that occur when computing BSDF derivatives through the BSDF sampling routine. We use the following weighting function to attenuate the vector field $\mathcal{V}(\mathbf{x}, \pi)$:

$$w_{\mathcal{V}}(\mathbf{x}, \pi_0) = \max\left(0, 1 - \frac{\phi(\mathbf{x}_t, \pi_0)}{t \cdot \varepsilon(\mathbf{x})}\right) \quad (21)$$

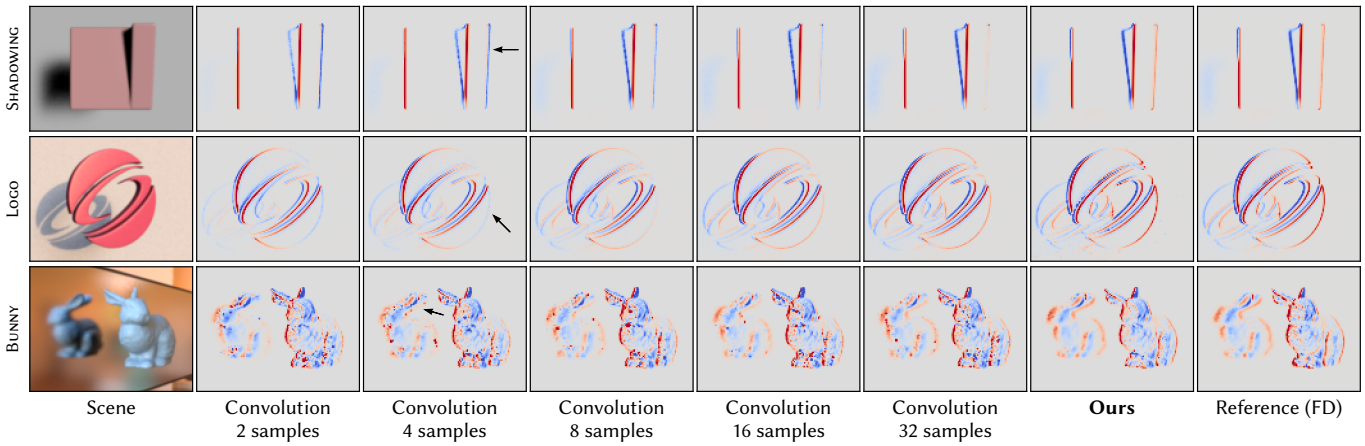


Fig. 8. We compare the gradients obtained using our method to the ground truth and an SDF version of the convolution method by Bangaru et al [2020]. The gradient images are computed by using forward-mode differentiation with respect to a translation of the entire object. For the convolution method, we show the results using varying numbers of auxiliary rays estimating the convolution integral. Increasing the number of rays improves the accuracy of the gradient estimate, at the cost of increased computation time. All gradient images are rendered using 1024 samples per pixel. The Shadowing and Logo scene are using direct illumination, and the Bunny scene is rendered with one bounce of indirect illumination.

where $\varepsilon(\mathbf{x})$ is the minimum of 0.01 and the distance of $\mathbf{x}$ to the SDF's bounding box. This weighting effectively restricts the reparameterization to only have an impact as $\mathbf{x}$ is approaching a surface and therefore possibly a discontinuity. Additionally, by considering the bounding box it ensures that our reparameterization continuously drops off to zero as the evaluation location approaches the border of the SDF volume. Finally, we also multiply this weight by the sum of sphere tracing weights (clamped to at most 1) to handle the degenerate case of the evaluation location being undefined.

Area element. On top of evaluating the reparameterization $\mathcal{T}$ itself, we need to evaluate the area element as defined in Equation 5. To do so, we first compute the Jacobian matrix $\partial_{\boldsymbol{\omega}}\mathcal{T}$ analytically. We then simply evaluate the trace of this Jacobian to account for the area change, as it is equivalent to the cross product formulation under differentiation (see Appendix A). The trace requires slightly less computation than explicitly computing the cross product of the transformed tangent vectors.

Since our reparameterization depends on the distance $t(\boldsymbol{\omega}, \pi_0)$, we need to evaluate the derivative $\partial_{\boldsymbol{\omega}} t(\boldsymbol{\omega}, \pi_0) \in \mathbb{R}^3$ to compute the Jacobian. We analytically compute this term during sphere tracing, and return it alongside the distance t and, if applicable, the intersection distance. None of these terms require tracking a differentiable dependency on π through the sphere tracing loop, hence there is no need to build an AD graph over it, which would be an expensive operation. The complete derivation of the Jacobian is laborious and done in Appendix C.

Variance reduction. Reparameterizing the integral can cause undesirable gradient variance in regions of the image without discontinuities. Based on the observation that the majority of that noise is caused by the differentiable evaluation of the pixel filter, prior work suggested using antithetic sampling and control variates [Loubet et al. 2019; Bangaru et al. 2020]. We find that steps like antithetic sampling and control variates add a significant amount of implementation complexity, and that a similar variance reduction can be achieved by easier means. Renderers usually divide the accumulated radiance in each pixel by the sum of accumulated pixel filter weights to reduce variance [Pharr et al. 2016, Section 13.9]. Interestingly, we found that by tracking derivatives through this normalization step, most of the noise caused by the differentiable pixel filter evaluation can be eliminated.

Nested reparameterization. When building a differentiable rendering algorithm, we have to correctly handle the subtleties due to *nested* application of our reparameterization. Related to that, the surface point that results from a ray intersection might be parameter dependent due to the motion of the surface itself, as explained in Section 4.2. Both these factors will have the consequence that the ray origin $\mathbf{x}$, and hence $\mathbf{x}_t$, can differentiably depend on the SDF parameter π. This is relevant for example when applying our reparameterization to a shadow ray. In the derivations so far, we have not considered this potential dependency. Completely ignoring the parameter dependence of $\mathbf{x}_t$ when handling the shadow ray will produce wrong results. If we ignore the dependency on the reparameterization of the primary ray when reparameterizing the shadow ray, the gradient is also not correct. Only if we track the dependency on the primary ray's reparameterization all the way through the secondary reparameterization, we get correct gradients, as shown in Figure 7. More precisely, we need to make sure to evaluate $\phi(\mathbf{x}_t, \pi)$ in Equation 9 using the parameter-dependent $\mathbf{x}_t(\pi) = \mathbf{x}(\pi) + t\boldsymbol{\omega}$. All other terms of the reparameterization can remain detached as before. This then ensures that our vector field produces the right relative motion between occluder and ray origin. A more detailed discussion and derivation is provided in Appendix D.

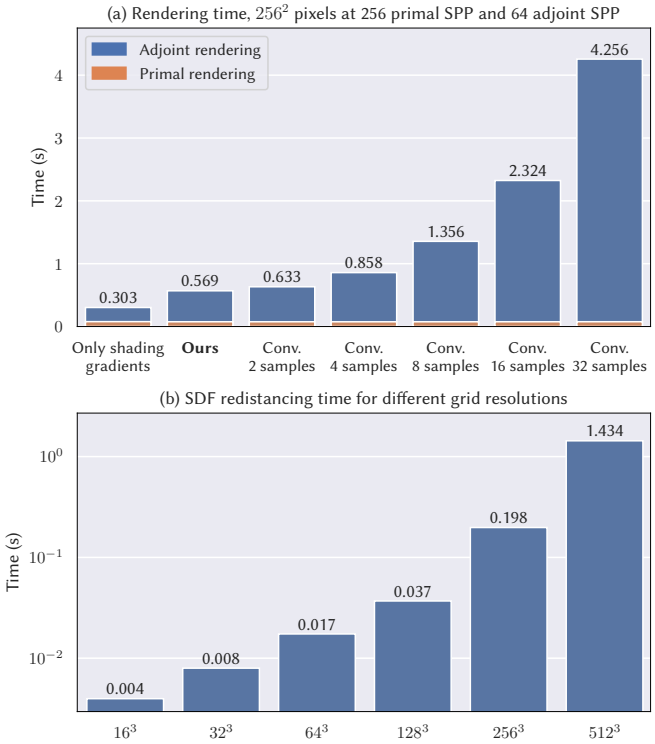


Fig. 9. We measure the rendering time **(a)** and the SDF redistancing time **(b)**. The rendering time plot reports both the time to render the primal image and the time required to compute SDF gradients in reverse-mode.

5 SHAPE OPTIMIZATION

The reparameterization introduced in the previous section enables computing accurate gradients for renderings of signed distance functions. An important use of such gradients is 3D shape reconstruction given a set of observed views. In this section, we will describe the overall pipeline and settings we use to produce the optimization examples in this paper. Our goal is to reconstruct a shape by solving

$$\boldsymbol{\pi}^* = \arg\min_{\boldsymbol{\pi}} \sum_{i=1}^{N} \ell\left(I^i(\boldsymbol{\pi}), I^i_{ref}\right), \tag{22}$$

where N is the number of views and $\boldsymbol{\pi}$ contains both the SDF parameters, as well as any optimized parameters used in its bidirectional scattering distribution function (BSDF). The loss function ℓ measures the differences between images. In the following, we will describe the specific parameters and heuristics that we use to make this optimization practical.

Loss function. We use an L_1 loss on linear RGB pixel values for all optimizations. We evaluate it both at the original image resolution, as well as on a 3-level pyramid of downsampled reference and rendered images. This helps increasing the spatial support of the loss function. If we were to only evaluate the L_1 loss at the original resolution, the optimization might more easily get stuck in a local minimum representing a low-quality solution.

Optimizing SDFs. During optimization, the differentiable renderer backpropagates gradients to the SDF grid values. Even after a single iteration of gradient descent, the values stored in the grid might not represent a valid SDF anymore [Gomes and Faugeras 2000]. In particular, the SDF will violate the eikonal constraint and in general $\|\partial_{\mathbf{x}}\phi(\mathbf{x}, \boldsymbol{\pi})\| \neq 1$. A common approach to reduce the deviation from a true SDF is adding an eikonal regularization term to the optimization, that penalizes deviations of the gradient norm from 1 [Li et al. 2005]. This is particularly useful when the SDF is not stored explicitly, but rather is the output of a neural network. The disadvantage of this regularization approach is that it does not yield an exact SDF and introduces another hyperparameter in the form of a regularization weight. Since we are directly storing the SDF values on a grid, we found it more convenient to explicitly *redistance* the SDF after every iteration of the optimization. The high-level idea is to reconstruct the distance function values by marching outwards from the current zero level set [Adalsteinsson and Sethian 1995; Sethian 1996, 1999]. In practice, we use a CUDA implementation of the parallel fast sweeping method [Zhao 2004; Detrixhe et al. 2013] and redistance the SDF after every iteration of the optimization. To further improve performance, it could be interesting to use a sparse SDF representation in conjunction with a sparse version of fast sweeping [Museth 2017]. Another approach would be to use *velocity extension* [Adalsteinsson and Sethian 1999], that infers SDF-compatible grid value updates from the speed of the surface itself. We experimented with simple versions of such an idea, but in the end found the redistancing to work well enough. Alternatively, one could investigate updating the SDF using the adjoint state method, similar to work on art-directable fluid simulations [McNamara et al. 2004] and travel-time tomography [Leung and Qian 2006].

Regularization. On top of the image-based loss, we found it beneficial to slightly regularize the SDF using a Laplacian regularization. While the optimization itself is robust, a small amount of regularization can reduce the noise on unobserved regions or due to variance in the gradients and the rendered images. This primarily improves the surface appearance when re-rendering under new illumination conditions. We use a simple discrete Laplacian kernel, which penalizes differences between a voxel and its directly adjacent neighbors. We use a regularization weight of 10^{-5}, which is small enough to not oversmooth the surface.

Multiscale optimization. It is further advantageous to initially optimize the SDF at a lower resolution than the target resolution. We start optimizing at a resolution of 16^3 voxels and then double the resolution several times during the optimization until the desired target resolution is reached.

Texture optimization. We also optionally optimize albedo and roughness parameters, stored on trilinearly interpolated grids. Optimization of these quantities is straightforward and only requires clamping to valid parameter ranges. When optimizing roughness, we use the Disney BSDF [Burley 2012], but turn off all lobes except for diffuse and specular lobe.

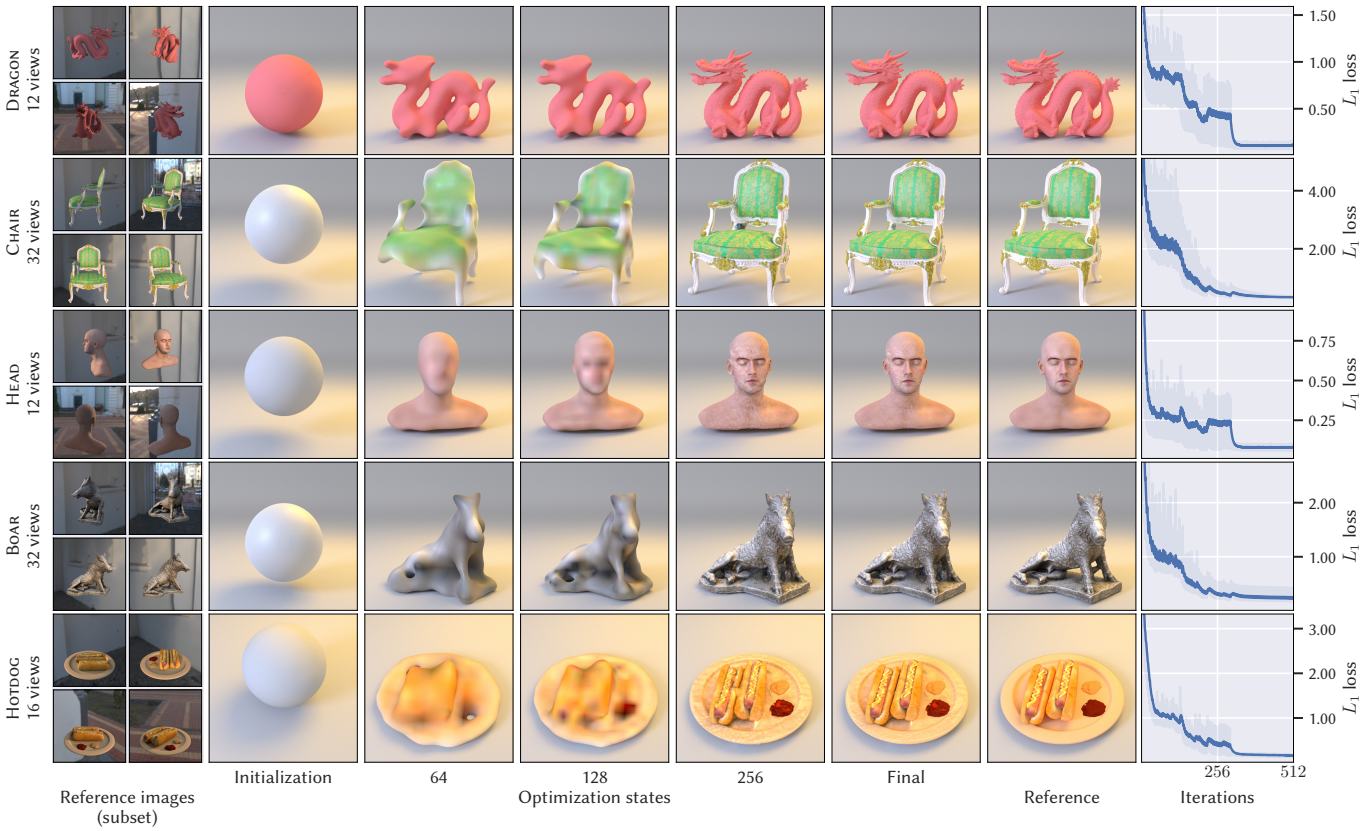


Fig. 10. Results using our method on a few challenging example objects. All these examples use 512 gradient descent iterations (using batches of 6 views). For the Dragon scene, the BSDF parameters are assumed to be fully known. For the other scenes we optimize albedo textures, and for Chair and Boar additionally surface roughness.

6 RESULTS

In the following, we evaluate the correctness, performance and optimization results using our method. Our implementation runs using NVIDIA CUDA and all timings that are reported in the following have been measured on a NVIDIA TITAN RTX graphics card (24 GB of RAM).

Implementation. We implemented our differentiable SDF renderer on top of Mitsuba 2 [Nimier-David et al. 2019] and use reverse-mode AD to propagate derivatives to the SDF parameters. We implemented the majority of our pipeline using Mitsuba 2's Python API. Interpreting our reparameterization as a differentiable ray tracing operation, we absorb its logic into the ray intersection function, which can then be used as follows inside an integrator:

```
si, ray.d, area_element = ray_intersect(ray)
# Evaluate terms in the integrand using the now reparameterized "ray.d"
throughput *= bsdf.eval(si, si.to_local(ray.d))
# ... and multiply by area element
throughput *= area_element
```

The ray intersection routine returns a surface interaction record, the reparameterized ray direction and the area element. This abstraction allows to cleanly implement different integrators leveraging the same reparameterization logic.

We use the same hyperparameters for all our results and did not find our method to be particularly sensitive to the various parameters used to define the weights in Section 4. Unless stated otherwise, our optimizations use a direct illumination integrator with emitter sampling. All optimizations use the Adam optimizer [Kingma and Ba 2015] with a learning rate that is proportional to the current grid resolution. We optimize SDFs up to a resolution of 256^3 voxels by differentiably rendering 512^2 pixel images. Initial optimization iterations use both a lower resolution SDF and lower rendered image resolutions to improve performance. Similar to previous work, we decorrelate the estimation of the primal image and the gradients [Gkioulekas et al. 2016]. For our optimizations, we use 256 primal and 64 adjoint samples per pixel. These sample counts are chosen conservatively to reduce the impact of Monte Carlo noise on the optimization results. For the highest performance optimization, adaptively sampling both temporal and spatial dimensions could be effective at reducing the overall optimization time.

Gradient validation. We validate the gradients computed using our reparameterization in Figure 8. We use forward-mode differentiation to obtain gradients of the pixel values with respect to a translation of the SDF. The reference gradients are obtained using finite differences (with $h = 10^{-3}$). For the scene using indirect

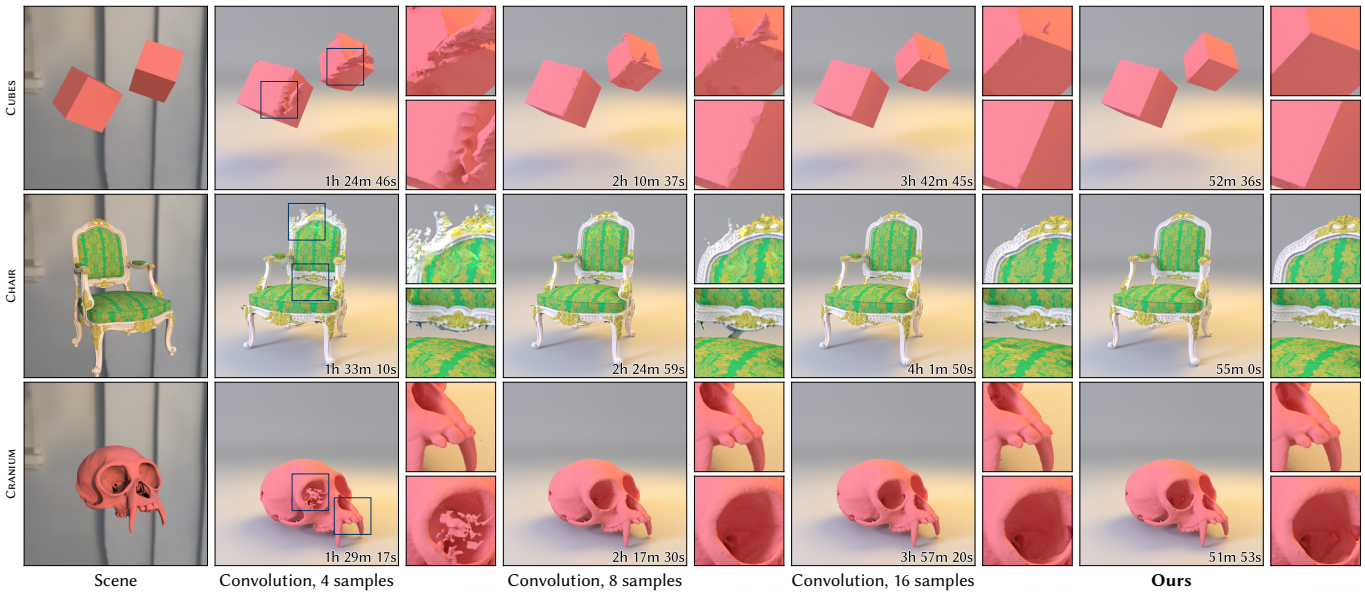


Fig. 11. We compare the reconstruction results using our reparameterization and the convolution method [Bangaru et al. 2020] at equal iteration count. Increasing the number of samples to estimate the convolution integral improves the quality of results, but at the cost of drastically increasing the total time for the optimization. Our method both results in the most accurate reconstruction and the shortest runtime.

illumination, we use a reparameterized version of path replay backpropagation [Vicini et al. 2021b], following the work by Zeltner et al. [2021]. We also implemented an SDF version of the convolution method by Bangaru et al. [2020]. This is the only prior algorithm that can be used to differentiate physically-based renderings of implicit shapes without explicit meshing. We use their convolution kernel and apply it to our 3D vector field defined in Equation 9. We set the concentration parameter κ of the spherical von Mises-Fisher distribution to 10^5 for all our experiments. This worked better than the default of 10^4 suggested in the original paper. Higher values (e.g., $\kappa = 10^6$) then again seemed to reduce the quality, in particular of shadow gradients. In Figure 8, we estimate the convolution integral using varying numbers of auxiliary rays. We can see that using 4 rays already provides some edge gradients, but we oftentimes need up to 16 to get a more accurate estimate with the correct sign. This appears to be consistent with the observations made for triangle meshes in the original paper. Our method on the other hand produces gradients that closely match the finite difference reference.

Benchmarks. We benchmark the different gradient computation methods in Figure 9. We show both the time required to render the primal image and the time used to estimate gradients. We use our direct illumination integrator for these benchmarks. Compared to only considering shading gradients, our method is around 1.9× slower, since it evaluates additional terms during sphere tracing and also needs to compute the area element. It is faster than the convolution method, in particular as the number of auxiliary rays increases. The primal rendering time remains constant across all methods, since we are careful to only reparameterize when computing gradients. We also benchmark the SDF redistancing implementation to show the potential overhead caused by the redistancing. We only need to redistance once per iteration, but each iteration of gradient descent might require rendering multiple images. As the SDF resolution increases, we also need to increase the resolution of the rendered images to effectively use the additional surface resolution. Overall, we found the overhead due to redistancing negligible compared to the differentiable rendering itself.

Optimization results. In Figure 10 we show different reconstructions obtained using our reparameterization and shape optimization scheme. We always initialize our SDF to a sphere of a constant color. For all scenes, we re-render the optimization result from a novel view using a new illumination condition that was not part of the optimization. Our method can reconstruct these various objects without the use of a silhouette constraint. One common issue in shape optimizations are pieces of geometry that are detached from the main shape and are not removed by the optimization. We did not observe such issues with our method. The combination of multiscale optimization and noise in the optimization process even allows to overcome some of the local minima inherent in this setting. For example, this allows to correctly reconstruct the complex topology of the Chair scene. However, a purely surface-based optimization routine can also get stuck in local minima, where the loss will not provide any useful gradients to improve further. In the combination with optimizing albedo textures this can cause some holes of shapes to be erroneously filled in, e.g., between the legs of the Boar statue. This non-convexity is a known difficulty of the optimization problem and addressing it is beyond the scope of this paper.

Comparison to the convolution method. While we have already seen that our reparameterization produces more accurate gradients than the SDF version of the method by Bangaru et al. [2020], we can also validate that this actually yields better optimization results. In

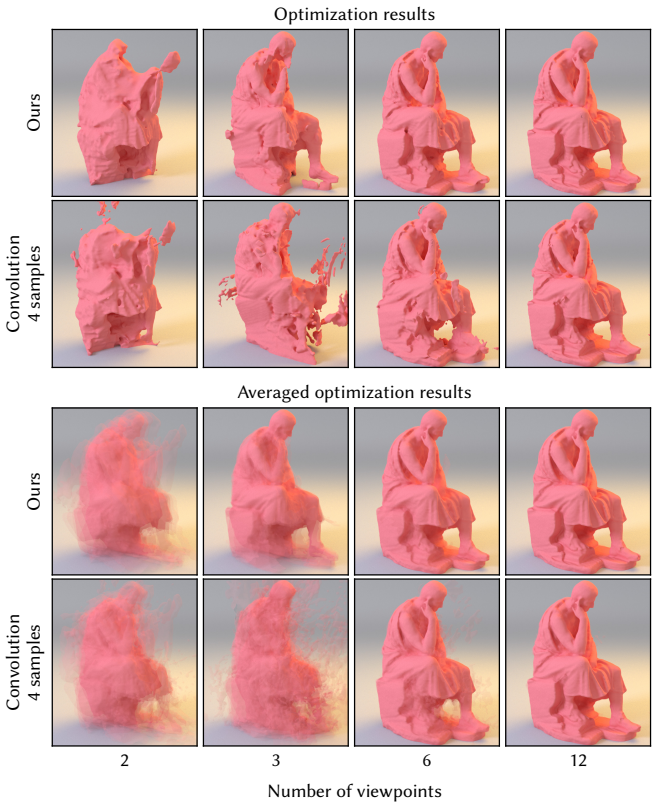


Fig. 12. The robustness of the shape reconstruction depends on the number of input views. **Top row:** we show an optimization result using a varying number of reference views for both our method and the convolution method. **Bottom row:** we average renderings of optimized shapes for 8 different sets of reference viewpoints. We set up evenly spaced virtual cameras around the object and then rotated them around the vertical axis of the object by varying amounts. The haze around the object in the averaged image is caused by variance in the reconstructed geometry.

Figure 11 we show optimizations both using our and the convolution method using varying numbers of auxiliary rays. Overall, we can see that using a low number of auxiliary rays often results in artifacts in the reconstructed geometry. The inaccurate edge gradients in particular seem to cause problems in scenes with sharp edges, as illustrated in the Cubes scene. As the number of auxiliary rays increases, the convolution method manages to produce better results, at the cost of an increase in overall runtime. Similar issues would likely occur if one were to apply the convolution method to a finely tesselated triangle mesh obtained, e.g., using MeshSDF [Remelli et al. 2020].

Influcence of the number of viewpoints. The quality of the optimization results depends on the number of reference images of the given scene. Studying the behavior of this effect can provide additional insight in the stability of gradient estimation and optimization methods. In Figure 12, we compare reconstructions obtained both using our method and the convolution method using 4 auxiliary rays. We show both results as the number of views increases and also show images blending results of 8 separate runs over different

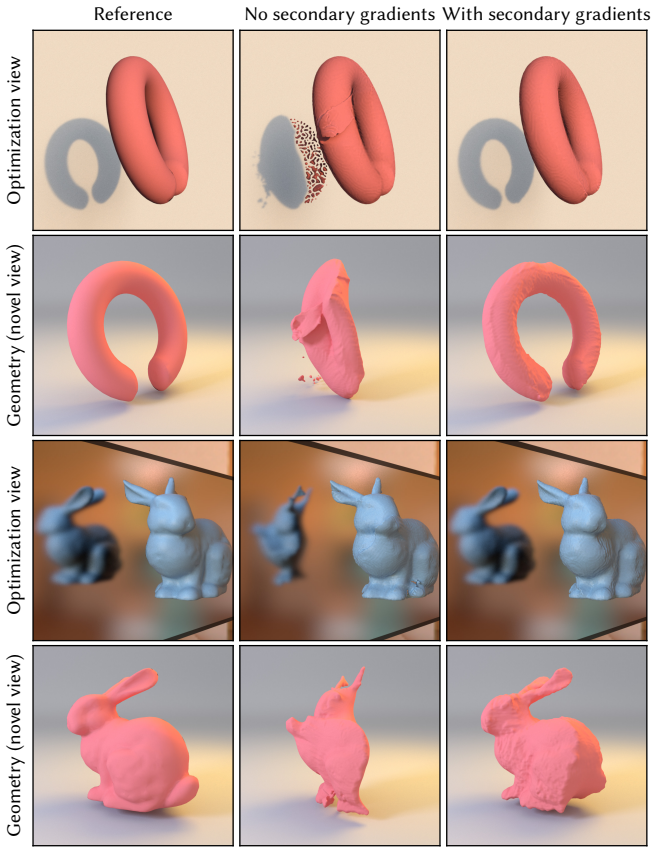


Fig. 13. We compare single view reconstructions both without using any secondary gradients (middle column) and using secondary gradients (right column). Accounting for indirect effects improves the reconstruction quality when the number of observations is limited.

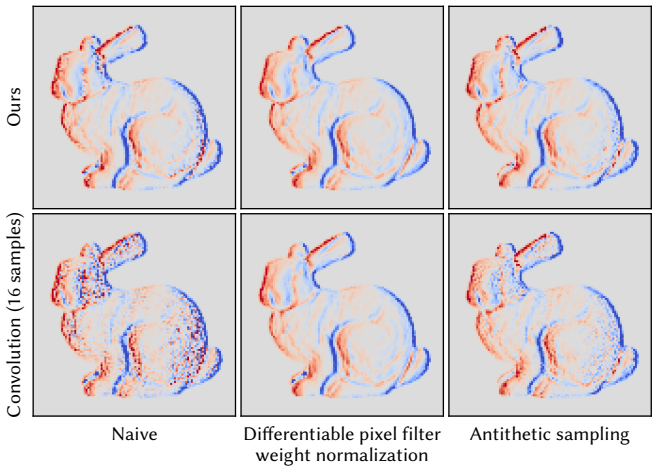


Fig. 14. This figure shows the image gradients computed using 4 samples per pixels both using our method and the convolution method [Bangaru et al. 2020], the latter using 16 samples to estimate the inner convolution integral. We can reduce the noise of the gradients by using a differentiable pixel filter weight normalization or using antithetic sampling.

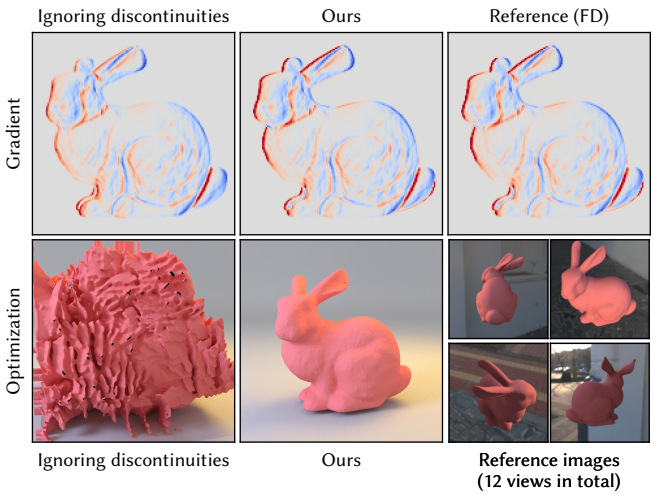


Fig. 15. **Top row:** Ignoring discontinuities misses important gradient contributions due to occlusion effects. **Bottom row:** Using such strongly biased gradients in an optimization is very likely to completely diverge.

configurations to illustrate the stability of the optimization. These results further confirm that our method is more robust at a lower number of reference views than the convolution method. Increasing the number of auxiliary rays would again increase its runtime. At a higher number of viewpoints the optimization is more constrained and the results become more similar.

Benefits of diffentiating secondary effects. One key advantage of our method is that it can differentiate secondary effects such as shadows and indirect illumination. Figure 13 showcases two example optimizations where accounting for those gradients improves the reconstructed geometry. Both examples use only a single reference view and in both cases differentiating secondary effects helps to reduce ambiguities by leveraging additional shape cues in the form of shadows and reflections. In the first example, the optimization that ignores secondary gradients converges to an undesirable local minimum, where part of the shadow on the background plane ends up being approximated by many small disconnected components. In the second example, we optimize accounting for indirect illumination, which allows to use the shape's mirror reflection as an additional constraint.

Gradient variance. In Figure 14, we visualize forward-mode gradients computed using a low number of samples per pixel. We compare a naive implementation to one that uses antithetic sampling of the pixel filter, and one that simply keeps the pixel filter normalization weights attached to the differentiation, as discussed in Section 4.3. We can see that the differentiable normalization weights reduce the variance both for ours and prior work, and seem to perform slightly better than antithetic sampling. We therefore use the differentiable weight normalization for all implemented methods.

Comparison to using only the shading gradient. While from a theoretical point of view it is clear that we need to account for visibility discontinuities, it is worth validating that not doing so does not work. In Figure 15 we run a simple optimization both using our method and an implementation that does not reparameterize discontinuities. While at first glance the gradient images look quite similar, the missing edge gradients make the optimization diverge completely.

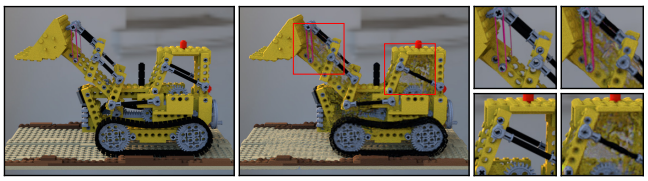


Fig. 16. The presence of complex topology and spatially varying albedo textures can result in challenging, non-convex optimization problems with many undesirable local minima. In this example, we optimized SDF and albedo texture at a resolution of 256^3 using 40 input images. The result is rendered using the same environment map that is used during optimization.

Non-convexity of surface-based reconstruction. Directly optimizing surfaces represented as SDFs can work surprisingly well in many cases. However, the surface reconstruction problem itself can exhibit undesirable local minima, in particular in the presence of complex topology (i.e., objects with holes). This sometimes causes undesirable connections between object parts that should remain disjoint. The gradient information is then not always sufficient to infer that an opening must be created. Figure 16 shows such an example, where gradient descent is unable to correctly reconstruct the topology of a complex object.

7 CONCLUSION AND FUTURE WORK

We presented a novel approach to the problem of differentiable rendering of signed distance functions. Our method efficiently computes accurate gradients for image-based optimization of SDFs. We exploit the computational structure of sphere tracing, convenient properties of the SDF representation, and the flexibility admitted by the reparameterization framework.

Our reparameterization handles the discontinuities caused by SDFs, but supporting efficient combination with other shape representations remains future work (e.g., a triangle mesh occluding an SDF). It would be interesting to combine our method with a sparse data structure storing the SDF values to allow scaling the SDF resolution more adaptively. While our method works well with minimal regularization, investigating more tailored regularization methods could be worthwhile to further improve results.

For practical shape reconstruction, the main limitation is the inherent non-convexity of direct surface optimization. This could be remedied by either using a more sophisticated method to initialize the SDF, or by using some form of semi-transparency.

Lastly, it would be interesting to apply a similar reparameterization approach to triangle meshes. Instead of using a convolution, one could construct a reparameterization during traversal of the ray acceleration data structure. This could reduce both variance and bias of the gradient estimator.

ACKNOWLEDGMENTS

We thank Damien Martin for helpful interactions during an MSc. project focused on SDFs. We also thank Miguel Crespo, Baptiste Nicolet and Dongqing Wang for proofreading. This research was supported by the Swiss National Science Foundation (SNSF) as part of grant 200021_184629.

REFERENCES

David Adalsteinsson and James A. Sethian. 1995. A Fast Level Set Method for Propagating Interfaces. *J. Comput. Phys.* 118, 2 (1995), 269–277.

David Adalsteinsson and James A. Sethian. 1999. The Fast Construction of Extension Velocities in Level Set Methods. *J. Comput. Phys.* 148, 1 (1999), 2–22.

Matan Atzmon, Niv Haim, Lior Yariv, Ofer Israelov, Haggai Maron, and Yaron Lipman. 2019. Controlling neural level sets. In *Advances in Neural Information Processing Systems (NeurIPS)*. 2032–2041.

Dejan Azinović, Tzu-Mao Li, Anton Kaplanyan, and Matthias Nießner. 2019. Inverse Path Tracing for Joint Material and Lighting Estimation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Sai Bangaru, Tzu-Mao Li, and Frédo Durand. 2020. Unbiased Warped-Area Sampling for Differentiable Rendering. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 39, 6 (2020), 245:1–245:18.

Sai Bangaru, Jesse Michel, Kevin Mu, Gilbert Bernstein, Tzu-Mao Li, and Jonathan Ragan-Kelley. 2021. Systematically Differentiating Parametric Discontinuities. *ACM Trans. Graph. (Proc. SIGGRAPH)* 40, 107 (2021), 107:1–107:17.

Brent Burley. 2012. Physically-based shading at Disney. *SIGGRAPH Course Notes. Practical physically-based shading in film and game production.* (2012), 1–27.

Forrester Cole, Kyle Genova, Avneesh Sud, Daniel Vlasic, and Zhoutong Zhang. 2021. Differentiable Surface Rendering via Non-Differentiable Sampling. (2021). arXiv:2108.04886

Brian Curless and Marc Levoy. 1996. A Volumetric Method for Building Complex Models from Range Images. In *SIGGRAPH Comput. Graph.* 303–312.

Miles Detrixhe, Frédéric Gibou, and Chohong Min. 2013. A parallel fast sweeping method for the eikonal equation. *J. Comput. Phys.* 237 (2013), 46–55.

Akio Doi and Akio Koide. 1991. An Efficient Method of Triangulating Equi-Valued Surfaces by Using Tetrahedral Cells. *IEICE Transactions on Information and Systems* 1, 74 (1991), 214–224.

Pau Gargallo, Emmanuel Prados, and Peter Sturm. 2007. Minimizing the Reprojection Error in Surface Reconstruction from Images. *The IEEE International Conference on Computer Vision (ICCV)* (2007), 1–8.

Ioannis Gkioulekas, Anat Levin, and Todd Zickler. 2016. An evaluation of computational imaging techniques for heterogeneous inverse scattering. In *European Conference on Computer Vision*. Springer, 685–701.

Ioannis Gkioulekas, Shuang Zhao, Kavita Bala, Todd Zickler, and Anat Levin. 2013. Inverse Volume Rendering with Material Dictionaries. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 32, 6, Article 162 (Nov. 2013), 13 pages.

José Gomes and Olivier Faugeras. 2000. Reconciling Distance Functions and Level Sets. *Journal of Visual Communication and Image Representation* 11, 2 (2000), 209–223.

John C. Hart. 1996. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer* 12, 10 (1 Jan. 1996), 527–545.

Yue Jiang, Dantong Ji, Zhizhong Han, and Matthias Zwicker. 2020. SDFDiff: Differentiable Rendering of Signed Distance Fields for 3D Shape Optimization. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

James T. Kajiya. 1986. The Rendering Equation. In *SIGGRAPH Comput. Graph.* 143–150.

Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. 2018. Neural 3D Mesh Renderer. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Benjamin Keinert, Henry Schäfer, Johann Korndörfer, Urs Ganse, and Marc Stamminger. 2014. Enhanced Sphere Tracing. In *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference*. The Eurographics Association.

Pramook Khungurn, Daniel Schroeder, Shuang Zhao, Kavita Bala, and Steve Marschner. 2015. Matching Real Fabrics with Micro-Appearance Models. *ACM Trans. Graph.* 35, 1, Article 1 (Dec. 2015), 26 pages.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations (ICLR)*.

Samuli Laine, Janne Hellsten, Tero Karras, Yeongho Seol, Jaakko Lehtinen, and Timo Aila. 2020. Modular Primitives for High-Performance Differentiable Rendering. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 39, 6 (2020).

A. Laurentini. 1994. The visual hull concept for silhouette-based image understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 16, 2 (1994), 150–162.

Shingyu Leung and Jianliang Qian. 2006. An adjoint state method for three-dimensional transmission traveltime tomography using first-arrivals. *Communications in Mathematical Sciences* 4, 1 (2006), 249 – 266.

Chunming Li, Chenyang Xu, Changfeng Gui, and M.D. Fox. 2005. Level set evolution without re-initialization: a new variational formulation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 430–436.

Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. 2018. Differentiable Monte Carlo Ray Tracing through Edge Sampling. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 37, 6 (2018), 222:1–222:11.

Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. 2019. Soft Rasterizer: A Differentiable Renderer for Image-based 3D Reasoning. *The IEEE International Conference on Computer Vision (ICCV)* (Oct. 2019).

Shaohui Liu, Yinda Zhang, Songyou Peng, Boxin Shi, Marc Pollefeys, and Zhaopeng Cui. 2020. DIST: Rendering Deep Implicit Signed Distance Function with Differentiable Sphere Tracing. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. 2019. Neural Volumes: Learning Dynamic Renderable Volumes from Images. *ACM Trans. Graph. (Proc. SIGGRAPH)* 38, 4, Article 65 (July 2019), 14 pages.

Matthew M. Loper and Michael J. Black. 2014. OpenDR: An Approximate Differentiable Renderer. In *European Conference on Computer Vision (ECCV)*. Springer, 154–169.

William E. Lorensen and Harvey E. Cline. 1987. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In *SIGGRAPH Comput. Graph.* 163–169.

Guillaume Loubet, Nicolas Holzschuch, and Wenzel Jakob. 2019. Reparameterizing discontinuous integrands for differentiable rendering. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 38, 6 (Dec. 2019).

Antoine McNamara, Adrien Treuille, Zoran Popović, and Jos Stam. 2004. Fluid control using the adjoint method. In *ACM Trans. Graph. (Proc. SIGGRAPH)*, Vol. 23. ACM, 449–456.

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2020. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *European Conference on Computer Vision (ECCV)*.

Jacob Munkberg, Jon Hasselgren, Tianchang Shen, Jun Gao, Wenzheng Chen, Alex Evans, Thomas Mueller, and Sanja Fidler. 2022. Extracting Triangular 3D Models, Materials, and Lighting From Images. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Ken Museth. 2017. Novel Algorithm for Sparse and Parallel Fast Sweeping: Efficient Computation of Sparse Signed Distance Fields. In *ACM SIGGRAPH Talks*. Article 74.

Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. 2011. KinectFusion: Real-time dense surface mapping and tracking. In *2011 10th IEEE International Symposium on Mixed and Augmented Reality*. 127–136.

Baptiste Nicolet, Alec Jacobson, and Wenzel Jakob. 2021. Large Steps in Inverse Rendering of Geometry. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 40, 6 (Dec. 2021).

Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. 2020. Differentiable Volumetric Rendering: Learning Implicit 3D Representations without 3D Supervision. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Marc Stamminger. 2013. Real-Time 3D Reconstruction at Scale Using Voxel Hashing. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 32, 6, Article 169 (Nov. 2013), 11 pages.

Merlin Nimier-David, Sébastien Speierer, Benoît Ruiz, and Wenzel Jakob. 2020. Radiative Backpropagation: An Adjoint Method for Lightning-Fast Differentiable Rendering. *ACM Trans. Graph. (Proc. SIGGRAPH)* 39, 4, Article 146 (July 2020), 15 pages.

Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. 2019. Mitsuba 2: A Retargetable Forward and Inverse Renderer. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 38, 6 (Nov. 2019), 17 pages.

Stanley Osher and James A. Sethian. 1988. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *J. Comput. Phys.* 79, 1 (1988), 12–49.

Matt Pharr, Wenzel Jakob, and Greg Humphreys. 2016. *Physically Based Rendering: From Theory to Implementation* (3rd ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Edoardo Remelli, Artem Lukoianov, Stephan Richter, Benoit Guillard, Timur Bagautdinov, Pierre Baque, and Pascal Fua. 2020. MeshSDF: Differentiable Iso-Surface Extraction. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 33.

Osborne Reynolds. 1903. *Papers on mechanical and physical subjects: the sub-mechanics of the universe, Vol. 3.* The University Press.

Darius Rückert, Linus Franke, and Marc Stamminger. 2021. Adop: Approximate differentiable one-pixel point rendering. (2021). arXiv:2110.06635

James A. Sethian. 1996. A fast marching level set method for monotonically advancing fronts. *Proceedings of the National Academy of Sciences* 93, 4 (1996), 1591–1595.

James A. Sethian. 1999. Fast Marching Methods. *SIAM Rev.* 41, 2 (1999), 199–235.

Dario Seyb, Alec Jacobson, Derek Nowrouzezahrai, and Wojciech Jarosz. 2019. Nonlinear sphere tracing for rendering deformed signed distance fields. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 38, 6 (Nov. 2019).

Tianchang Shen, Jun Gao, Kangxue Yin, Ming-Yu Liu, and Sanja Fidler. 2021. Deep Marching Tetrahedra: a Hybrid Representation for High-Resolution 3D Shape Synthesis. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Jos Stam and Ryan Schmidt. 2011. On the Velocity of an Implicit Surface. *ACM Trans. Graph.* 30, 3, Article 21 (May 2011), 7 pages.

Yen-Hsi Richard Tsai, Li-Tien Cheng, Stanley Osher, Paul Burchard, and Guillermo Sapiro. 2004. Visibility and its dynamics in a PDE based implicit framework. *J. Comput. Phys.* 199, 1 (2004), 260–290.
Eric Veach and Leonidas J. Guibas. 1995. Optimally Combining Sampling Techniques for Monte Carlo Rendering. In *SIGGRAPH Comput. Graph.* Association for Computing Machinery, New York, NY, USA, 419–428.
Delio Vicini, Wenzel Jakob, and Anton Kaplanyan. 2021a. A Non-Exponential Transmittance Model for Volumetric Scene Representations. *ACM Trans. Graph. (Proc. SIGGRAPH)* 40, 4 (Aug. 2021), 136:1–136:16.
Delio Vicini, Sébastien Speierer, and Wenzel Jakob. 2021b. Path Replay Backpropagation: Differentiating Light Paths using Constant Memory and Linear Time. *ACM Trans. Graph. (Proc. SIGGRAPH)* 40, 4 (Aug. 2021), 108:1–108:14.
Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. 2021. NeuS: Learning Neural Implicit Surfaces by Volume Rendering for Multi-view Reconstruction. In *Advances in Neural Information Processing Systems (NeurIPS).*
Yiheng Xie, Towaki Takikawa, Shunsuke Saito, Or Litany, Shiqin Yan, Numair Khan, Federico Tombari, James Tompkin, Vincent Sitzmann, and Srinath Sridhar. 2021. Neural Fields in Visual Computing and Beyond. arXiv:2111.11426
Lior Yariv, Jiatao Gu, Yoni Kasten, and Yaron Lipman. 2021. Volume Rendering of Neural Implicit Surfaces. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 34.
Lior Yariv, Yoni Kasten, Dror Moran, Meirav Galun, Matan Atzmon, Basri Ronen, and Yaron Lipman. 2020. Multiview Neural Surface Reconstruction by Disentangling Geometry and Appearance. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 33.
Wang Yifan, Felice Serena, Shihao Wu, Cengiz Öztireli, and Olga Sorkine-Hornung. 2019. Differentiable Surface Splatting for Point-based Geometry Processing. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 38, 6 (2019).
Alex Yu, Sara Fridovich-Keil, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. 2022. Plenoxels: Radiance Fields without Neural Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*
Tizian Zeltner, Sébastien Speierer, Iliyan Georgiev, and Wenzel Jakob. 2021. Monte Carlo Estimators for Differential Light Transport. *ACM Trans. Graph. (Proc. SIGGRAPH)* 40, 4 (Aug. 2021), 78:1–78:16.
Cheng Zhang, Bailey Miller, Kai Yan, Ioannis Gkioulekas, and Shuang Zhao. 2020. Path-Space Differentiable Rendering. *ACM Trans. Graph. (Proc. SIGGRAPH)* 39, 4, Article 143 (July 2020), 19 pages.
Cheng Zhang, Lifan Wu, Changxi Zheng, Ioannis Gkioulekas, Ravi Ramamoorthi, and Shuang Zhao. 2019. A Differential Theory of Radiative Transfer. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 38, 6, Article 227 (Nov. 2019), 16 pages.
Cheng Zhang, Zihan Yu, and Shuang Zhao. 2021. Path-Space Differentiable Rendering of Participating Media. *ACM Trans. Graph. (Proc. SIGGRAPH)* 40, 4 (2021), 76:1–76:15.
Hongkai Zhao. 2004. A fast sweeping method for eikonal equations. *Math. Comp.* 74 (2004), 603–627. Issue 250.
Hong-Kai Zhao, S. Osher, and R. Fedkiw. 2001. Fast surface reconstruction using the level set method. In *Proceedings IEEE Workshop on Variational and Level Set Methods in Computer Vision.* 194–201.
Yang Zhou, Lifan Wu, Ravi Ramamoorthi, and Ling-Qi Yan. 2021. Vectorization for Fast, Analytic, and Differentiable Visibility. *ACM Trans. Graph.* 40, 3, Article 27 (July 2021), 21 pages.

A EQUIVALENCE OF AREA ELEMENT AND DIVERGENCE DERIVATIVES

In the following, we prove that we can either evaluate the area element (using the cross product) or the divergence of the mapping computed in ambient space. Under differentiation with respect to the scene parameter π, these are equivalent. Since we reparameterize the manifold of the unit sphere, we found this is to be not immediately obvious. We would like to show:

$$\partial_\pi \left\| \mathrm{D}\mathcal{T}_{\boldsymbol{\omega},\pi}(\mathbf{s}) \times \mathrm{D}\mathcal{T}_{\boldsymbol{\omega},\pi}(\mathbf{t}) \right\| = \partial_\pi \mathrm{div}\, \mathcal{T}(\boldsymbol{\omega}, \pi). \tag{23}$$

The differential $\mathrm{D}\mathcal{T}_{\boldsymbol{\omega},\pi}$ is computed in ambient space and in the following we write it as a parameter dependent Jacobian matrix $\mathbf{J}(\pi)$. We then prove the original statement by simplifying the derivative of the area element:

$$\begin{aligned}\partial_\pi \left\| \mathbf{J}(\pi)\mathbf{s} \times \mathbf{J}(\pi)\mathbf{t} \right\| &= \boldsymbol{\omega} \cdot \left(\partial_\pi \mathbf{J}(\pi)\mathbf{s} \times \mathbf{J}(\pi)\mathbf{t} + \mathbf{J}(\pi)\mathbf{s} \times \partial_\pi \mathbf{J}(\pi)\mathbf{t}\right) \\ &= \boldsymbol{\omega} \cdot \left(\partial_\pi \mathbf{J}(\pi)\mathbf{s} \times \mathbf{t} + \mathbf{s} \times \partial_\pi \mathbf{J}(\pi)\mathbf{t}\right),\end{aligned}$$

where we differentiated the vector norm and moved the derivative operator inside the cross product and "·" denotes the dot product. We also used that $\mathbf{J}(\pi)$ maps $\mathbf{s}$ and $\mathbf{t}$ onto themselves and therefore $\mathbf{J}(\pi)\mathbf{s} \times \mathbf{J}(\pi)\mathbf{t} = \boldsymbol{\omega}$. We simplify further by using the fact that the triple product $\mathbf{a} \cdot (\mathbf{b} \times \mathbf{c})$ is invariant under circular shifts of its arguments:

$$\begin{aligned}\boldsymbol{\omega} \cdot \left(\partial_\pi \mathbf{J}(\pi)\mathbf{s} \times \mathbf{t} + \mathbf{s} \times \partial_\pi \mathbf{J}(\pi)\mathbf{t}\right) &= \partial_\pi \mathbf{J}(\pi)\mathbf{s} \cdot \mathbf{t} \times \boldsymbol{\omega} + \partial_\pi \mathbf{J}(\pi)\mathbf{t} \cdot \boldsymbol{\omega} \times \mathbf{s} \\ &= \partial_\pi \mathbf{J}(\pi)\mathbf{s} \cdot \mathbf{s} + \partial_\pi \mathbf{J}(\pi)\mathbf{t} \cdot \mathbf{t},\end{aligned}$$

where we additionally used that $\mathbf{t} \times \boldsymbol{\omega} = \mathbf{s}$ and $\boldsymbol{\omega} \times \mathbf{s} = \mathbf{t}$. We then add the zero-valued term $\partial_\pi \mathbf{J}(\pi)\boldsymbol{\omega} \cdot \boldsymbol{\omega}$ to the expression above and use that the sum of inner products of basis vectors with $\partial_\pi \mathbf{J}(\pi)$ is exactly its trace:

$$\begin{aligned}&= \partial_\pi \mathbf{J}(\pi)\mathbf{s} \cdot \mathbf{s} + \partial_\pi \mathbf{J}(\pi)\mathbf{t} \cdot \mathbf{t} + \partial_\pi \mathbf{J}(\pi)\boldsymbol{\omega} \cdot \boldsymbol{\omega} \\ &= \mathrm{tr}(\partial_\pi \mathbf{J}(\pi)) = \partial_\pi \mathrm{tr}(\mathbf{J}(\pi)) = \partial_\pi \mathrm{div}\, \mathcal{T}(\boldsymbol{\omega}, \pi).\end{aligned}$$

With this, Equation 23 has been proven. This makes the equivalence of reparameterization and divergence formulation explicit and shows that in practice either formulation yields the same result. Both formulations correctly account for the geometry of the unit sphere.

To complete the proof, we still need to show the following:

$$\partial_\pi \mathbf{J}(\pi)\boldsymbol{\omega} \cdot \boldsymbol{\omega} = 0. \tag{24}$$

We do so assuming that we can write $\mathcal{T}(\boldsymbol{\omega}, \pi) = \bar{\mathcal{T}}(\boldsymbol{\omega}, \pi)/\left\|\bar{\mathcal{T}}(\boldsymbol{\omega}, \pi)\right\|$, where $\bar{\mathcal{T}}(\boldsymbol{\omega}, \pi)$ might not have unit norm, but preserves the direction, i.e. $\bar{\mathcal{T}}(\boldsymbol{\omega}, \pi) = \lambda\boldsymbol{\omega}$. Our reparameterization keeps the primal value of $\boldsymbol{\omega}$ fixed, but there can still be an arbitrarily complex differentiable relation to the parameter π. We then explicitly compute the parameter derivative of the Jacobian matrix $\mathbf{J}(\pi) = \partial_{\boldsymbol{\omega}} \mathcal{T}$:

$$\begin{aligned}\partial_\pi \mathbf{J}(\pi) &= \partial_\pi \left[\partial_{\boldsymbol{\omega}} \left(\frac{\bar{\mathcal{T}}(\boldsymbol{\omega}, \pi)}{\left\|\bar{\mathcal{T}}(\boldsymbol{\omega}, \pi)\right\|} \right) \right] = \partial_\pi \left[\left(\frac{1}{\left\|\bar{\mathcal{T}}\right\|}\mathbb{I} - \frac{1}{\left\|\bar{\mathcal{T}}\right\|^3}\bar{\mathcal{T}}\bar{\mathcal{T}}^T \right) \partial_{\boldsymbol{\omega}} \bar{\mathcal{T}} \right] \\ &= \underbrace{\left(\frac{1}{\left\|\bar{\mathcal{T}}\right\|}\mathbb{I} - \frac{1}{\left\|\bar{\mathcal{T}}\right\|^3}\bar{\mathcal{T}}\bar{\mathcal{T}}^T \right) \partial_\pi \partial_{\boldsymbol{\omega}} \bar{\mathcal{T}}}_{(1)} + \underbrace{\partial_\pi \left[\frac{1}{\left\|\bar{\mathcal{T}}\right\|}\mathbb{I} - \frac{1}{\left\|\bar{\mathcal{T}}\right\|^3}\bar{\mathcal{T}}\bar{\mathcal{T}}^T \right] \partial_{\boldsymbol{\omega}} \bar{\mathcal{T}}^T}_{(2)},\end{aligned}$$

where $\mathbb{I}$ is the 3 by 3 identity matrix and $\bar{\mathcal{T}}\bar{\mathcal{T}}^T$ is an outer product. We omit the arguments of $\bar{\mathcal{T}}(\boldsymbol{\omega}, \pi)$ for brevity. In order for Equation 24 to hold, we need to show that multiplying $\boldsymbol{\omega}$ with this matrix is a projection into tangent space of the unit sphere at $\boldsymbol{\omega}$. For the term (1) this follows immediately, since the terms inside the parentheses are such a projection (recall that $\bar{\mathcal{T}}$ is simply a scaled $\boldsymbol{\omega}$). For (2) this is a bit more laborious to show, but can be done by computing the derivative of the terms inside the brackets:

$$\begin{aligned}\partial_\pi \left[\frac{1}{\left\|\bar{\mathcal{T}}\right\|}\mathbb{I} - \frac{1}{\left\|\bar{\mathcal{T}}\right\|^3}\bar{\mathcal{T}}\bar{\mathcal{T}}^T \right] = -\frac{1}{\left\|\bar{\mathcal{T}}\right\|^3} \Bigg(&\underbrace{\bar{\mathcal{T}}^T \partial_\pi \bar{\mathcal{T}} \left(\mathbb{I} - \frac{1}{\left\|\bar{\mathcal{T}}\right\|^2}\bar{\mathcal{T}}\bar{\mathcal{T}}^T \right)}_{(1)} \\ &\underbrace{+ \partial_\pi \bar{\mathcal{T}}\bar{\mathcal{T}}^T - \frac{\bar{\mathcal{T}}^T \partial_\pi \bar{\mathcal{T}}}{\left\|\bar{\mathcal{T}}\right\|^2}\bar{\mathcal{T}}\bar{\mathcal{T}}^T}_{(2)} + \underbrace{\bar{\mathcal{T}}\partial_\pi \bar{\mathcal{T}}^T - \frac{\bar{\mathcal{T}}^T \partial_\pi \bar{\mathcal{T}}}{\left\|\bar{\mathcal{T}}\right\|^2}\bar{\mathcal{T}}\bar{\mathcal{T}}^T}_{(3)} \Bigg)\end{aligned}$$

Here, we can again see that (1) is a projection (scalar factors do not matter), and (2) and (3) require a few additional steps. We can simplify (2) to again see that it indeed projects onto tangent space:

$$\partial_\pi \bar{\mathcal{T}}\bar{\mathcal{T}}^T - \frac{\bar{\mathcal{T}}^T\partial_\pi\bar{\mathcal{T}}}{\left\|\bar{\mathcal{T}}\right\|^2}\bar{\mathcal{T}}\bar{\mathcal{T}}^T = \left(\partial_\pi\bar{\mathcal{T}} - \frac{\bar{\mathcal{T}}^T\partial_\pi\bar{\mathcal{T}}}{\left\|\bar{\mathcal{T}}\right\|^2}\bar{\mathcal{T}}\right)\bar{\mathcal{T}}^T$$
$$= \left(\mathbb{I} - \frac{1}{\left\|\bar{\mathcal{T}}\right\|^2}\bar{\mathcal{T}}\bar{\mathcal{T}}^T\right)\partial_\pi\bar{\mathcal{T}}\bar{\mathcal{T}}^T.$$

The second equality used that $\bar{\mathcal{T}}^T\partial_\pi\bar{\mathcal{T}}$ is a scalar to commute it with $\bar{\mathcal{T}}$. We skip the derivation for the term (3), as it is analogous. This shows that Equation 24 indeed holds.

B PARAMETER DERIVATIVE OF THE REPARAMETERIZATION $\mathcal{T}$

The parameter derivative of the reparameterization $\mathcal{T}$ should follow the motion of the visible surface boundary on the unit sphere. We validate this by explicitly computing this derivative. In practice, the following computation will be carried out by automatic differentiation. Taking the derivative $\partial_\pi\mathcal{T}$ of our reparameterization we obtain:

$$\begin{aligned}
\partial_\pi\mathcal{T}(\boldsymbol{\omega},\pi) &= \partial_\pi\left[\frac{\bar{\mathcal{T}}(\boldsymbol{\omega},\pi)}{\left\|\bar{\mathcal{T}}(\boldsymbol{\omega},\pi)\right\|}\right]\\
&= \left(\frac{1}{\left\|\bar{\mathcal{T}}(\boldsymbol{\omega},\pi_0)\right\|}\mathbb{I} - \frac{1}{\left\|\bar{\mathcal{T}}(\boldsymbol{\omega},\pi_0)\right\|^3}\bar{\mathcal{T}}(\boldsymbol{\omega},\pi_0)\cdot\bar{\mathcal{T}}(\boldsymbol{\omega},\pi_0)^T\right)\partial_\pi\bar{\mathcal{T}}(\boldsymbol{\omega},\pi)\\
&= \left(\frac{1}{t}\mathbb{I} - \frac{1}{t^3}\bar{\mathcal{T}}(\boldsymbol{\omega},\pi_0)\cdot\bar{\mathcal{T}}(\boldsymbol{\omega},\pi_0)^T\right)\partial_\pi\bar{\mathcal{T}}(\boldsymbol{\omega},\pi)\\
&= \left(\frac{1}{t}\mathbb{I} - \frac{1}{t}\boldsymbol{\omega}\cdot\boldsymbol{\omega}^T\right)\partial_\pi\bar{\mathcal{T}}(\boldsymbol{\omega},\pi)\\
&= \frac{1}{t}\left(\mathbb{I} - \boldsymbol{\omega}\cdot\boldsymbol{\omega}^T\right)\partial_\pi\bar{\mathcal{T}}(\boldsymbol{\omega},\pi)\\
&= \frac{1}{t}\left(\mathbb{I} - \boldsymbol{\omega}\cdot\boldsymbol{\omega}^T\right)\partial_\pi\mathcal{V}(\mathbf{x}_t,\pi).
\end{aligned} \tag{25}$$

The first equality used the definition of $\mathcal{T}$. In the second equality we evaluate the derivative of the division by the norm and use the chain rule. Here, $\mathbb{I}$ is the 3 by 3 identity matrix and $\bar{\mathcal{T}}(\boldsymbol{\omega},\pi_0)\cdot\bar{\mathcal{T}}(\boldsymbol{\omega},\pi_0)^T$ is the outer product. For the terms inside the parentheses, we do not need to track parameter derivatives. In other words, we have $\pi = \pi_0$ and the term $V(\mathbf{x}+t\boldsymbol{\omega},\pi) - V(\mathbf{x}+t\boldsymbol{\omega},\pi_0)$ in the definition of $\bar{\mathcal{T}}$ is then equal to 0. This allows to simplify further to arrive at the final expression, where we use the shorthand notation $\mathbf{x}_t \coloneqq \mathbf{x}+t\boldsymbol{\omega}$. This shows that the normalization projects the derivative vector $\partial_\pi\mathcal{V}(\mathbf{x}_t,\pi)$ to tangent space and divides by the distance, producing a vector correctly matching the motion of the discontinuity.

C JACOBIAN OF THE REPARAMETERIZATION $\mathcal{T}$

Accounting for the distortion of the integration domain requires computing the Jacobian $\partial_{\boldsymbol{\omega}}\mathcal{T}(\boldsymbol{\omega},\pi) \in \mathbb{R}^{3\times3}$. To do so, we first compute the positional Jacobian $\partial_{\mathbf{x}}\mathcal{V}$. The vector field $\mathcal{V}$ is used to define the reparameterization $\mathcal{T}$ and was defined as:

$$\mathcal{V}(\mathbf{x},\pi) = -\frac{\partial_{\mathbf{x}}\phi(\mathbf{x},\pi_0)}{\left\|\partial_{\mathbf{x}}\phi(\mathbf{x},\pi_0)\right\|^2}\phi(\mathbf{x},\pi) \tag{26}$$

Note that only $\phi(\mathbf{x},\pi)$ is differentiable with respect to the scene parameter π. Taking the derivative with respect to the position $\mathbf{x}$ we get:

$$\partial_{\mathbf{x}}\mathcal{V}(\mathbf{x},\pi) = -\mathbf{A}\cdot\partial_{\mathbf{x}}^2\phi(\mathbf{x},\pi_0)\phi(\mathbf{x},\pi) - \frac{\partial_{\mathbf{x}}\phi(\mathbf{x},\pi_0)}{\left\|\partial_{\mathbf{x}}\phi(\mathbf{x},\pi_0)\right\|^2}\cdot\partial_{\mathbf{x}}\phi(\mathbf{x},\pi)^T \tag{27}$$

with $\mathbf{A}$ being the Jacobian of the division by the squared norm:

$$\mathbf{A} = \frac{1}{\left\|\partial_{\mathbf{x}}\phi(\mathbf{x},\pi_0)\right\|^2}\left(\mathbb{I} - \frac{2}{\left\|\partial_{\mathbf{x}}\phi(\mathbf{x},\pi_0)\right\|^2}\partial_{\mathbf{x}}\phi(\mathbf{x},\pi_0)\cdot\partial_{\mathbf{x}}\phi(\mathbf{x},\pi_0)^T\right)$$

Accounting for the additional weighting factor $w_{\mathcal{V}}$, we compute the weighted vector field's Jacobian using the product rule:

$$\partial_{\mathbf{x}}\bar{\mathcal{V}}(\mathbf{x},\pi) = \partial_{\boldsymbol{\omega}}\left[w_{\mathcal{V}}(\mathbf{x})\mathcal{V}(\mathbf{x},\pi)\right] = \mathcal{V}(\mathbf{x},\pi)\cdot\partial_{\boldsymbol{\omega}}w_{\mathcal{V}}(\mathbf{x})^T + w_{\mathcal{V}}(\mathbf{x})\partial_{\boldsymbol{\omega}}\mathcal{V}(\mathbf{x},\pi) \tag{28}$$

So far, this Jacobian assumes we vary the 3D position $\mathbf{x}$. We need to convert it to a Jacobian with respect to the ray direction $\boldsymbol{\omega}$:

$$\partial_{\boldsymbol{\omega}}\bar{\mathcal{V}}(\mathbf{x}_t,\pi) = \partial_{\mathbf{x}}\bar{\mathcal{V}}(\mathbf{x},\pi)\partial_{\boldsymbol{\omega}}\mathbf{x}_t, \tag{29}$$

where $\partial_{\boldsymbol{\omega}}\mathbf{x}_t = \partial_{\boldsymbol{\omega}}\left[\mathbf{x}_o + t\boldsymbol{\omega}\right] = \mathbb{I}\cdot t + \boldsymbol{\omega}\cdot\partial_{\boldsymbol{\omega}}t^T$. And finally, accounting for the conversion to a reparameterization on the unit sphere:

$$\partial_{\boldsymbol{\omega}}\mathcal{T}(\boldsymbol{\omega},\pi) = \mathbf{B}\cdot\left(\partial_{\boldsymbol{\omega}}\mathbf{x}_t + \partial_{\boldsymbol{\omega}}\bar{\mathcal{V}}(\mathbf{x}_t,\pi) - \partial_{\boldsymbol{\omega}}\bar{\mathcal{V}}(\mathbf{x}_t,\pi_0)\right), \tag{30}$$

where $\mathbf{B}$ is the normalization Jacobian which we used before:

$$\mathbf{B} = \frac{1}{\left\|\bar{\mathcal{T}}(\boldsymbol{\omega},\pi)\right\|}\mathbb{I} - \frac{1}{\left\|\bar{\mathcal{T}}(\boldsymbol{\omega},\pi)\right\|^3}\bar{\mathcal{T}}(\boldsymbol{\omega},\pi)\cdot\bar{\mathcal{T}}(\boldsymbol{\omega},\pi)^T.$$

Finally, we then simply evaluate $\mathrm{tr}\left(\partial_{\boldsymbol{\omega}}\mathcal{T}(\boldsymbol{\omega},\pi)\right)$. We can further reduce the number of 3×3 matrix products by analyzing this expression more closely. First of all, we know that $\left\|\bar{\mathcal{T}}(\boldsymbol{\omega},\pi)\right\| = t$ and does not depend on π in a differentiable way (t is computed using π_0). We will also drop any term that is additive and does not depend on π directly, as its derivative will be zero. In the end, we only need the derivative of the trace of the Jacobian:

$$\begin{aligned}
\partial_\pi\mathrm{tr}\left(\partial_{\boldsymbol{\omega}}\mathcal{T}(\boldsymbol{\omega},\pi)\right) &= \partial_\pi\mathrm{tr}\left(\mathbf{B}\cdot\left(\partial_{\boldsymbol{\omega}}\mathbf{x}_t + \partial_{\boldsymbol{\omega}}\bar{\mathcal{V}}(\mathbf{x}_t,\pi) - \partial_{\boldsymbol{\omega}}\bar{\mathcal{V}}(\mathbf{x}_t,\pi_0)\right)\right)\\
&= \partial_\pi\mathrm{tr}\left(\mathbf{B}\cdot\partial_{\boldsymbol{\omega}}\mathbf{x}_t\right) + \partial_\pi\mathrm{tr}\left(\mathbf{B}\cdot\partial_{\boldsymbol{\omega}}\bar{\mathcal{V}}(\mathbf{x}_t,\pi)\right)
\end{aligned}$$

We can now show that $\partial_\pi\mathrm{tr}\left(\mathbf{B}\cdot\partial_{\boldsymbol{\omega}}\mathbf{x}_t\right) = 0$:

$$\begin{aligned}
\partial_\pi\mathrm{tr}\left(\mathbf{B}\cdot\partial_{\boldsymbol{\omega}}\mathbf{x}_t\right) &= \partial_\pi\mathrm{tr}\left(\mathbf{B}\cdot\left[\mathbb{I}\cdot t + \boldsymbol{\omega}\cdot\partial_{\boldsymbol{\omega}}t^T\right]\right)\\
&= t\cdot\partial_\pi\mathrm{tr}\left(\mathbf{B}\right) + \partial_\pi\mathrm{tr}\left(\mathbf{B}\cdot\boldsymbol{\omega}\cdot\partial_{\boldsymbol{\omega}}t^T\right)
\end{aligned}$$

Since $\mathbf{B}$ is simply the projection onto tangent space, we know that $\mathrm{tr}(\mathbf{B}) = \mathrm{rank}(\mathbf{B}) = 2$ and therefore its derivative is zero. Additionally, $\mathbf{B}\cdot\boldsymbol{\omega}\cdot\partial_{\boldsymbol{\omega}}t^T$ is zero, since $\mathbf{B}$ removes any component in the direction of $\boldsymbol{\omega}$. Hence, $\partial_\pi\mathrm{tr}\left(\mathbf{B}\cdot\partial_{\boldsymbol{\omega}}\mathbf{x}_t\right) = 0$ and the derivative simplifies to:

$$\partial_\pi\mathrm{tr}\left(\partial_{\boldsymbol{\omega}}\mathcal{T}(\boldsymbol{\omega},\pi)\right) = \partial_\pi\mathrm{tr}\left(\mathbf{B}\cdot\partial_{\boldsymbol{\omega}}\bar{\mathcal{V}}(\mathbf{x}_t,\pi)\right)$$

D NESTING REPARAMETERIZATIONS

When reparameterizing a rendering algorithm, we need to correctly nest reparameterizations of the solid angle domain. For example, when rendering an image with direct illumination, we solve the following nested integration for each pixel:

$$I(\pi) = \int_{\mathcal{S}^2} f_0(\boldsymbol{\omega}_0, \pi) \int_{\mathcal{S}^2} f_1(\boldsymbol{\omega}_0, \boldsymbol{\omega}_1, \mathbf{x}_1(\boldsymbol{\omega}_0, \pi), \pi) \, \mathrm{d}\boldsymbol{\omega}_1 \, \mathrm{d}\boldsymbol{\omega}_0, \quad (31)$$

where f_0 and f_1 contain all relevant importance, visibility, BSDF and incident radiance terms. We use this simplified notation to reduce notational clutter. We explicitly write the dependency of f_1 on the ray intersection $\mathbf{x}_1(\boldsymbol{\omega}_0, \pi)$ of the camera ray in direction $\boldsymbol{\omega}_0$. The inner integration is over reflected directions $\boldsymbol{\omega}_1$.

If we now want to differentiate this nested integral, we need to introduce two reparameterizations. The first one reparameterizes the primary ray and the second one the secondary ray (or put more simply, the shadow ray). First, applying a reparameterization $\mathcal{T}_0$ on the primary ray:

$$I(\pi) = \int_{\mathcal{S}^2} f_0(\mathcal{T}_0(\boldsymbol{\omega}_0, \pi), \pi) \, |\mathcal{T}_0| \int_{\mathcal{S}^2} f_1(\mathcal{T}_0(\boldsymbol{\omega}_0, \pi), \boldsymbol{\omega}_1, \mathbf{x}_1(\mathcal{T}_0(\boldsymbol{\omega}_0, \pi), \pi), \pi) \, \mathrm{d}\boldsymbol{\omega}_1 \, \mathrm{d}\boldsymbol{\omega}_0, \quad (32)$$

where in a slight abuse of notation $|\mathcal{T}_0|$ denotes the area element. Assuming a static sensor, we drop the dependency of $\mathcal{T}_0$ on $\mathbf{x}_0$ for clarity. We further reparameterize the second ray to obtain:

$$I(\pi) = \int_{\mathcal{S}^2} f_0(\mathcal{T}_0(\boldsymbol{\omega}_0, \pi), \pi) \, |\mathcal{T}_0| \int_{\mathcal{S}^2} f_1\Big(\mathcal{T}_0(\boldsymbol{\omega}_0, \pi), \mathcal{T}_1(\boldsymbol{\omega}_1, \mathbf{x}_1(\mathcal{T}_0(\boldsymbol{\omega}_0, \pi), \pi), \pi), \mathbf{x}_1(\mathcal{T}_0(\boldsymbol{\omega}_0, \pi), \pi), \pi\Big) \, |\mathcal{T}_1| \, \mathrm{d}\boldsymbol{\omega}_1 \, \mathrm{d}\boldsymbol{\omega}_0. \quad (33)$$

The second reparameterization can now potentially depend on the first reparameterization, and hence π, through the position $\mathbf{x}_1$. On top of that, $\mathbf{x}_1$ might also directly depend on the parameters, e.g., if π controls the distance of the object from the sensor, $\mathbf{x}_1$ will move away or closer to the sensor as π changes.

The remaining question is whether our reparameterization $\mathcal{T}_1$ still evaluates to the correct motion over the unit sphere when differentiated. This can be validated by plugging the parameter-dependent position into the definition of our reparameterization. Equation 11 now needs to evaluate $\mathcal{V}$ using a parameter-dependent position $\mathbf{x}$:

$$\partial_\pi \mathcal{V}(\mathbf{x}_t(\pi), \pi) = \partial_\pi \mathcal{V}(\mathbf{x}(\pi) + t\boldsymbol{\omega}, \pi). \quad (34)$$

We previously assumed the ray origin to be fixed and did not consider the parameter-dependence of $\mathbf{x}_t$. Here we simplified the notation by omitting the explicit dependency of $\mathbf{x}$ on $\mathcal{T}_0$.

We can now use the following fact: when solving the inner integral and integrating over secondary rays, $\mathbf{x}(\pi)$ is fixed. That means that the dependency of $\mathbf{x}$ on π can be re-interpreted as simply another way in which the SDF values depend on π. When evaluating $\mathcal{V}$ we therefore need to evaluate

$$\mathcal{V}(\mathbf{x}_t(\pi), \pi) = -\frac{\partial_{\mathbf{x}} \phi(\mathbf{x}_t(\pi_0), \pi_0)}{\|\partial_{\mathbf{x}} \phi(\mathbf{x}_t(\pi_0), \pi_0)\|^2} \phi(\mathbf{x}_t(\pi), \pi). \quad (35)$$

It is important to detach the evaluation of the positional gradient from the parameter dependence introduced through $\mathbf{x}$. With this we moved the parameter dependency of $\mathbf{x}$ into the vector field evaluation, which then in turn guarantees the correct (relative) motion of the surface, as our formulation of $\mathcal{V}$ does not assume any specific relation of π and ϕ in order to produce a valid reparameterization. For future work, it could be interesting to generalize these derivations to path space, similar to the work by Zhang et al. [2019].