Differentiable Monte Carlo Ray Tracing through Edge Sampling

TZU-MAO LI, MIT CSAIL
MIIKA AITTALA, MIT CSAIL
FRÉDO DURAND, MIT CSAIL
JAAKKO LEHTINEN, Aalto University & NVIDIA

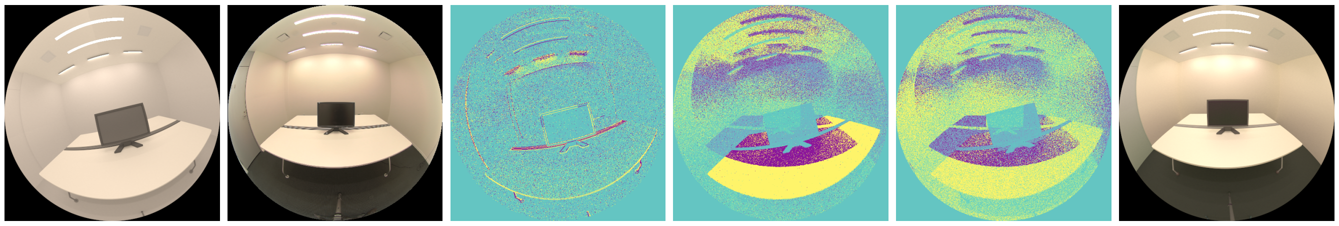

(a) initial guess
(b) real photograph
(c) camera gradient (per-pixel contribution)
(d) table albedo gradient (per-pixel contribution)
(e) light gradient (per-pixel contribution)
(f) our fitted result

Fig. 1. We develop a general-purpose differentiable renderer that is capable of handling general light transport phenomena. Our method generates gradients with respect to scene parameters, such as camera pose (c), material parameters (d), mesh vertex positions, and lighting parameters (e), from a scalar loss computed from the output image. (c) shows the per-pixel gradient contribution of the L^1 difference with respect to the camera moving into the screen. (d) shows the gradient with respect to the red channel of table albedo. (e) shows the gradient with respect to the green channel of the intensity of one light source. As one of our applications, we use our gradient to perform an inverse rendering task by matching a real photograph (b) starting from an initial configuration (a) with a manual geometric recreation of the scene. The scene contains a fisheye camera with strong indirect illumination and non-Lambertian materials. We optimize for camera pose, material parameters, and light source intensity. Despite slight inaccuracies due to geometry mismatch and lens distortion, our method generates image (f) that almost matches the photo reference.

Gradient-based methods are becoming increasingly important for computer graphics, machine learning, and computer vision. The ability to compute gradients is crucial to optimization, inverse problems, and deep learning. In rendering, the gradient is required with respect to variables such as camera parameters, light sources, scene geometry, or material appearance. However, computing the gradient of rendering is challenging because the rendering integral includes visibility terms that are not differentiable. Previous work on differentiable rendering has focused on approximate solutions. They often do not handle secondary effects such as shadows or global illumination, or they do not provide the gradient with respect to variables other than pixel coordinates.

We introduce a general-purpose differentiable ray tracer, which, to our knowledge, is the first comprehensive solution that is able to compute derivatives of scalar functions over a rendered image with respect to arbitrary scene parameters such as camera pose, scene geometry, materials, and lighting parameters. The key to our method is a novel edge sampling algorithm that directly samples the Dirac delta functions introduced by the derivatives of the discontinuous integrand. We also develop efficient importance sampling methods based on spatial hierarchies. Our method can generate gradients in times running from seconds to minutes depending on scene complexity and desired precision.

We interface our differentiable ray tracer with the deep learning library PyTorch and show prototype applications in inverse rendering and the generation of adversarial examples for neural networks.

Authors' addresses: Tzu-Mao Li, MIT CSAIL, tzumao@mit.edu; Miika Aittala, MIT CSAIL, miika@csail.mit.edu; Frédo Durand, MIT CSAIL, fredo@mit.edu; Jaakko Lehtinen, Aalto University & NVIDIA, jaakko.lehtinen@aalto.fi.

CCS Concepts: • **Computing methodologies → Ray tracing**; **Visibility**; *Reconstruction*;

Additional Key Words and Phrases: ray tracing, inverse rendering, differentiable programming

ACM Reference Format:
Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. 2018. Differentiable Monte Carlo Ray Tracing through Edge Sampling. *ACM Trans. Graph.* 37, 6, Article 222 (November 2018), 11 pages. https://doi.org/10.1145/3272127.3275109

1 INTRODUCTION

The computation of derivatives is increasingly central to many areas of computer graphics, computer vision, and machine learning. It is critical for the solution of optimization and inverse problems, and plays a major role in deep learning via backpropagation. This creates a need for rendering algorithms that can be differentiated with respect to arbitrary input parameters, such as camera location and direction, scene geometry, lights, material appearance, or texture values. Unfortunately, the rendering integral includes visibility terms that are not differentiable at object boundaries. Whereas the final image function is usually differentiable once radiance has been integrated over pixel prefilters, light source areas, etc., the integrand of rendering algorithms is not. In particular, the derivative of the integrand has Dirac delta terms at occlusion boundaries that cannot be handled by traditional sampling strategies.

Previous work in differentiable rendering [Kato et al. 2018; Loper and Black 2014] has focused on fast, approximate solutions using simpler rendering models that only handle primary visibility, and ignore secondary effects such as shadows and indirect light. Analytical solutions exist for diffuse interreflection [Arvo 1994] but are difficult to generalize for arbitrary material models. The work by Ramamoorthi et al. [2007] is an exception but it only differentiates with respect to image coordinates, whereas we want derivatives with respect to arbitrary scene parameters. Other previous work usually also relies on finite differences, with the usual limitation of these methods when the function is complex, namely that these methods work well for simple configurations but they do not propose a comprehensive solution to the full light transport equation.

In this work, we propose an algorithm that is, to the best of our knowledge, the first to compute derivatives of scalar functions over a physicially-based rendered image with respect to arbitrary input parameters (camera, light materials, geometry, etc.). Our solution is stochastic and builds on Monte Carlo ray tracing. For this, we introduce new techniques to explicitly sample edges of triangles in addition to the usual solid angle sampling of traditional approaches. This requires new spatial acceleration strategies and importance sampling to efficiently sample edges. Our method is general and can sample derivatives for arbitrary bounces of light transport. The running times we observed range from a second to a minute depending on the required precision, for an overhead of roughly $10\times-20\times$ compared to rendering an image alone.

We integrate our differentiable ray tracer with the automatic differentiation library PyTorch [Paszke et al. 2017] for efficient integration with optimization and learning approaches. The scene geometry, lighting, camera and materials are parameterized by PyTorch tensors, which enables a complex combination of 3D graphics, light transport, and neural networks. Backpropagation runs seamlessly across PyTorch and our renderer.

2 RELATED WORK

2.1 Inverse graphics

Inverse graphics techniques seek to find the scene parameters given observed images. Vision as inverse graphics has a long history in both computer graphics and vision (e.g. [Baumgart 1974; Patow and Pueyo 2003; Yu et al. 1999]). Many techniques in inverse graphics utilize derivatives of the rendering process for inference.

Blanz and Vetter [1999] optimized for shape and texture of a face. Shacked and Lischinski [2001] and Bousseau et al. [2011] optimized a perceptual metric for lighting design. Gkioulekas et al. [2016; 2013] focused on scattering parameters. Aittala et al. [2016; 2013; 2015] are interested in spatially varying material properties. Barron et al. [2015] proposed a solution to jointly optimize shape, illumination, and reflectance. Khungurn et al. [2015] aimed for matching photographs of fabrics. All of the approaches above utilize gradients for solving the inverse problem, and had to develop a specialized solver to compute the gradient of the specific light transport scenarios they were interested in.

Loper and Black [2014] and Kato et al. [2018] proposed two general differentiable rendering pipelines. Both of them focus on performance and approximate the primary visibility gradients when there are multiple triangles inside a pixel, and both of them assume Lambertian materials and do not compute shadows and global illumination.

Recently, it is increasingly popular for deep learning methods to incorporate a *differentiable rendering layer* in their architecture (e.g. [Liu et al. 2017; Richardson et al. 2017]). These rendering layers are usually special purpose and do not handle geometric discontinuities such as primary visibility and shadow.

To our knowledge our method is the first that is able to differentiate through a full path tracer, while taking the geometric discontinuities into account.

2.2 Derivatives in rendering

Analytical derivatives have been used for computing the footprint of light paths [Igehy 1999; Shinya et al. 1987; Suykens and Willems 2001] and predicting the changes of specular light paths [Chen and Arvo 2000; Jakob and Marschner 2012; Kaplanyan et al. 2014]. The derivatives are usually manually derived for the particular type of light paths the work focused on, making it difficult to generalize to arbitrary material models or lighting effects. Unlike these methods, we compute the gradients using a hybrid approach that mixes automatic differentiation and manually derived derivatives focusing on the discontinuous integrand.

Arvo [1994] proposed an analytical method for computing the spatial gradients for irradiance. The method requires clipping of triangle meshes in order to correctly integrate the form factor, and does not scale well to scenes with large complexity. It is also difficult or impossible to compute closed-form integration for arbitrary materials.

Ramamoorthi et al.'s work on first order analysis of light transport [2007] is highly related to our method. Their method is a special case of ours. Our derivation generalizes their method to differentiate with respect to any scene parameters. Furthermore we handle primary visibility, secondary visibility and global illumination.

Irradiance or radiance caching [Jarosz et al. 2012; Krivanek et al. 2005; Ward and Heckbert 1992] numerically computes the gradient of interreflection with respect to spatial position and orientation of the receiver. To take discontinuities into account, these methods resort to stratified sampling. Unlike these methods, we estimate the gradient integral directly by automatic differentiation and edge sampling.

Li et al. [2015] proposed a variant of the Metropolis light transport [Veach and Guibas 1997] algorithm by computing the Hessian of a light path contribution with respect to the path parameters using automatic differentiation [Griewank and Walther 2008]. Their method does not take geometric discontinuities into account.

3 METHOD

Our task is the following: given a 3D scene with a continuous parameter set Φ (including camera pose, scene geometry, material and lighting parameters), we generate an image using the path tracing algorithm [Kajiya 1986]. Given a scalar function computed from the image (e.g. a loss function we want to optimize), our goal is to backpropagate the gradient of the scalar with respect to all scene parameters Φ.

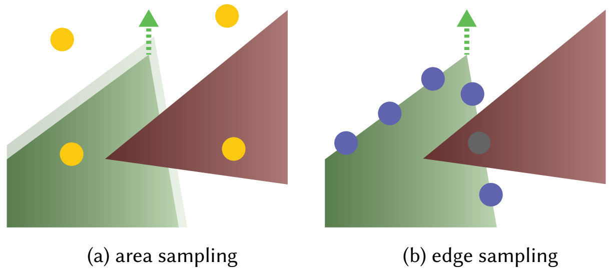


Fig. 2. (a) The figure shows a pixel overlapped with two triangles. We are interested in computing the derivative of pixel color with respect to the green triangle moving up. Since the area covered by the green triangle increases, the final pixel color will contain more green area and less white background. Traditional area sampling (yellow samples) even instrumented with automatic differentiation does not account for the change in covered area. (b) In addition to traditional area sampling, we propose a novel edge sampling algorithm (blue samples) to sample the differential area on the edges. Our method computes unbiased gradients and correctly takes occlusion into account.

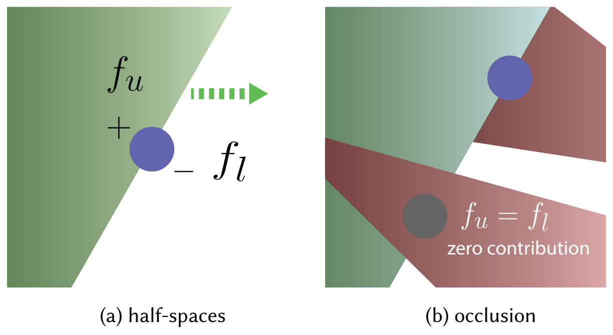


Fig. 3. (a) An edge splits the space into two half-spaces f_u and f_l. If the edge moves right, the green area increases while the white area decreases. We integrate over edges to compute gradients by taking into account the change in areas. To compute the integration, we sample a point (the blue point) on the edge and compute the difference between the half-spaces by computing the color on the two sides of the edge. (b) Our method handles occlusion correctly since an occluded sample will land on the continuous part of the path contribution function, thus having the exact same contribution on the two sides (for example, the grey sample has zero contribution to the gradient).

The pixel color is formalized as an integration over all light paths that pass through the pixel filter. We use Monte Carlo sampling to estimate both the integral and the gradient of the integral. However, since the integrand is discontinuous due to edges of geometry and occlusion, traditional area sampling does not correctly capture the changes due to camera parameters or triangle vertex movement (Figure 2 (a)). Mathematically, the gradient of the discontinuous integrand is a Dirac delta function, therefore traditional sampling has zero probability of capturing the Dirac deltas.

Our strategy for computing the gradient integral is to split it into smooth and discontinuous regions (Figure 2). For the smooth part of the integrand (e.g. Phong shading or bilinear texture reconstruction) we employ traditional area sampling with automatic differentiation. For the discontinuous part we use a novel edge sampling method to capture the changes at boundaries. In the following subsection, we first focus on primary visibility where we only integrate over the 2D screen-space domain (Section 3.1). We then generalize the method to handle shadow and global illumination (Section 3.2)

We focus on triangle meshes and we assume the meshes have been preprocessed such that there is no interpenetration. We also assume no point light sources and no perfectly specular surfaces. We approximate these with area light sources and BRDFs with very low roughness. We also focus on static scenes and leave integration over the time dimension for motion blur as future work.

3.1 Primary visibility

We start by focusing on the 2D pixel filter integral for each pixel that integrates over the pixel filter k and the radiance L, where the radiance itself can be another integral that integrates over light sources or the hemisphere. We will generalize the method to handle discontinuities inside the radiance integral in Section 3.2. The pixel color I can be written as:

$$I = \iint k(x,y)L(x,y)dxdy. \tag{1}$$

For notational convenience we will combine the pixel filter and radiance and call them scene function $f(x,y) = k(x,y)L(x,y)$. We are interested in the gradients of the integral with respect to some parameters Φ in the scene function $f(x,y;\Phi)$, such as the position of a mesh vertex:

$$\nabla I = \nabla \iint f(x,y;\Phi)dxdy. \tag{2}$$

The integral usually does not have a closed-form solution, especially when more complex effects such as non-Lambertian BRDFs are involved. Therefore we rely on Monte Carlo integration to estimate the pixel value I. However, we cannot take the naive approach of applying the same Monte Carlo sampler to estimate the gradient ∇I, since the scene function f is not necessarily differentiable with respect to the scene parameters (Figure 2a).

A key observation is that all the discontinuities happen at triangle edges. This allows us to explicitly integrate over the discontinuities. A 2D triangle edge splits the space into two half-spaces (f_u and f_l in Figure 3). We can model it as a Heaviside step function θ:

$$\theta(\alpha(x,y))f_u(x,y) + \theta(-\alpha(x,y))f_l(x,y), \tag{3}$$

where f_u represents the upper half-space, f_l represents the lower half-space, and α defines the edge equation formed by the triangles. For each edge with two endpoints $(a_x,a_y),(b_x,b_y)$, we can construct the edge equation by forming the line $\alpha(x,y) = Ax+By+C$. If $\alpha(x,y) > 0$ then the point is at the upper half-space, and vice versa. For the two endpoints of the edge $\alpha(x,y) = 0$. Thus by plugging in the two endpoints we obtain:

$$\alpha(x,y) = (a_y - b_y)x + (b_x - a_x)y + (a_x b_y - b_x a_y). \tag{4}$$

We can rewrite the scene function f as a summation of Heaviside step functions θ with edge equation α_i multiplied by an arbitrary function f_i:

$$\iint f(x,y)dxdy = \sum_i \iint \theta(\alpha_i(x,y))f_i(x,y)dxdy. \tag{5}$$

f_i itself can contain Heaviside step functions, for example a triangle defines a multiplication of three Heaviside step functions. f_i can even be an integral over light sources or the hemisphere. This fact is also crucial for our later generalization to secondary visibility.

We want to analytically differentiate the Heaviside step function θ and explicitly integrate over its derivative – the Dirac delta function δ. To do this we first swap the gradient operator inside the integral, then we use product rule to separate the integral into two:

$$\begin{aligned}&\nabla \iint \theta(\alpha_i(x,y))f_i(x,y)dxdy\\ =&\iint \delta(\alpha_i(x,y))\nabla\alpha_i(x,y)f_i(x,y)dxdy\\ +&\iint \nabla f_i(x,y)\theta(\alpha_i(x,y))dxdy.\end{aligned} \tag{6}$$

Equation 6 shows that we can estimate the gradient using two Monte Carlo estimators. The first one estimates the integral over the edges of triangles containing the Dirac delta functions, and the second estimates the original pixel integral except the smooth function f_i is replaced by its gradient, which can be computed through automatic differentiation.

To estimate the integral containing Dirac delta functions, we eliminate the Dirac function by performing variable substitution to rewrite the first term containing the Dirac delta function to an integral that integrates over the edge, that is, over the regions where $\alpha_i(x,y) = 0$:

$$\begin{aligned}&\iint \delta(\alpha_i(x,y))\nabla\alpha_i(x,y)f_i(x,y)dxdy\\ =&\int_{\alpha_i(x,y)=0} \frac{\nabla\alpha_i(x,y)}{\left\|\nabla_{x,y}\alpha_i(x,y)\right\|} f_i(x,y)d\sigma(x,y),\end{aligned} \tag{7}$$

where $\left\|\nabla_{x,y}\alpha_i(x,y)\right\|$ is the L^2 length of the gradient of the edge equations α_i with respect to x, y, which takes the Jacobian of the variable substitution into account. $\sigma(x,y)$ is the measure of the length on the edge [Hörmander 1983].

The gradients of the edge equations α_i are:

$$\begin{aligned}&\left\|\nabla_{x,y}\alpha_i\right\| = \sqrt{(a_x - b_x)^2 + (a_y - b_y)^2}\\ &\frac{\partial\alpha_i}{\partial a_x} = b_y - y,\ \frac{\partial\alpha_i}{\partial a_y} = x - b_x\\ &\frac{\partial\alpha_i}{\partial b_x} = y - a_y,\ \frac{\partial\alpha_i}{\partial b_y} = a_x - x\\ &\frac{\partial\alpha_i}{\partial x} = a_y - b_y,\ \frac{\partial\alpha_i}{\partial y} = b_x - a_x.\end{aligned} \tag{8}$$

As a byproduct of the derivation, we also obtain the screen space gradients $\frac{\partial}{\partial x}$ and $\frac{\partial}{\partial y}$, which can potentially facilitate adaptive sampling as shown in Ramamoorthi et al.'s first-order analysis [2007]. We can obtain the gradient with respect to other parameters, such as camera parameters, 3D vertex positions, or vertex normals by propagating the derivatives from the projected triangle vertices using the chain rule:

$$\frac{\partial\alpha}{\partial p} = \sum_{k\in\{x,y\}} \frac{\partial\alpha}{\partial a_k}\frac{\partial a_k}{\partial p} + \frac{\partial\alpha}{\partial b_k}\frac{\partial b_k}{\partial p}, \tag{9}$$

where p is the desired parameter.

We use Monte Carlo sampling to estimate the Dirac integral (Equation 7). Recall that a triangle edge defines two half-spaces (Equation 3), therefore we need to compute the two values $f_l(x,y)$ and $f_u(x,y)$ on the edge (Figure 3). By combining Equation 3 and Equation 7, our Monte Carlo estimation of the Dirac integral for a single edge E on a triangle can be written as:

$$\frac{1}{N}\sum_{j=1}^{N} \frac{\|E\|\nabla\alpha_i(x_j,y_j)(f_u(x_j,y_j) - f_l(x_j,y_j))}{P(E)\left\|\nabla_{x_j,y_j}\alpha_i(x_j,y_j)\right\|}, \tag{10}$$

where $\|E\|$ is the length of the edge and $P(E)$ is the probability of selecting edge E.

In practice, if we employ smooth shading, most of the triangle edges are in the continuous regions and the Dirac integral is zero for these edges since by definition of continuity $f_u(x,y) = f_l(x,y)$. Only the *silhouette edges* (e.g. [Hertzmann 1999]) have non-zero contribution to the gradients. We select the edges by projecting all triangle meshes to screen space and clip them against the camera frustrum. We select one silhouette edge with probability proportional to the screen space lengths. We then uniformly pick a point on the selected edge.

Our method handles occlusion correctly, since if the sample is blocked by another surface, (x,y) will always land on the continuous part of the contribution function $f(x,y)$. Such samples will have zero contribution to the gradients. Figure 3b illustrates the process.

To recap, we use two sampling strategies to estimate the gradient integral of pixel filter (Equation 2): one for the discontinuous regions of the integrand (first term of Equation 6), one for the continuous regions (second term of Equation 6). To compute the gradient for discontinuous regions, we need to explicitly sample the edges. We compute the difference between two sides of the edges using Monte Carlo sampling (Equation 10).

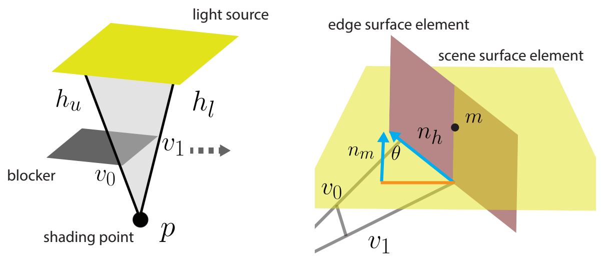
3.2 Secondary visibility

Our method can be easily generalized to handle effects such as shadow and global illumination by integrating over the 3D scene. Figure 4 illustrates the idea.

We focus on a single shading point p since we can propagate the derivative back to screen space and camera parameters using Equation 6. Given the shading point, the shading equation involves an integration over all points m on the scene manifold $\mathcal{M}$:

$$g(p) = \int_{\mathcal{M}} h(p,m)dA(m), \tag{11}$$

where A is the area measure of point m, and h is the scene function including material response, geometric factor, incoming radiance, and visibility. Note that $g(p)$ can itself be part of the pixel integrand $f(x,y)$ in the previous section (Equation 1). Therefore we can propagate the gradient of $g(p)$ using the chain rule or automatic differentiation with Equation 6.



(a) secondary visibility (b) width correction

Fig. 4. (a) Our method can be easily generalized to handle shadow and global illumination. Similar to the primary visibility case (Figure 3), a geometry edge (v_0, v_1) and the shading point p splits the 3D space into two half-spaces f_u and f_l and introduces discontinuity. Assuming the blocker is moving right, we integrate over the edge to compute the difference. By doing so we take account of the increase in blocker area and the decrease in light source area looking from the shading point. The integration over edge is defined on the intersection between the scene manifold and the plane formed by the shading point and the edge (the semi-transparent triangle). (b) The orientation of the infinitesmal width of the edge differs from the scene surface element the edge intersects with. During integration we need to project the scene surface element width onto the edge surface element. The ratio of the widths between the two is determined by $\frac{1}{\sin\theta}$, which is one over the length of the cross product between the normal of the edge plane and the scene surface.

Similar to the primary visibility case, an edge (v_0, v_1) in 3D introduces a step function into the scene function h:

$$\theta(\alpha(p,m))h_u(p,m) + \theta(-\alpha(p,m))h_l(p,m). \tag{12}$$

We can derive the edge function $\alpha(m)$ by forming a plane using the shading point p and the two points on the edge. The sign of the dot product between the vector $m - p$ and the plane normal determines the two half-spaces. The edge equation $\alpha(m)$ can therefore be defined by

$$\alpha(p,m) = (m-p) \cdot (v_0 - p) \times (v_1 - p). \tag{13}$$

To compute the gradients, we analogously apply the derivation used for primary visibility, using the 3D version of Equation 6 and Equation 7 with x, y replaced by p, m. The edge integral integrating over the line on the scene surface, analogous to Equation 7 is:

$$\begin{aligned} &\int_{\alpha(p,m)=0} \frac{\nabla\alpha(p,m)}{\|\nabla_m\alpha(p,m)\|} h(p,m) \frac{1}{\|n_m \times n_h\|} d\sigma'(m) \\ &n_h = \frac{(v_0-p)\times(v_1-p)}{\|(v_0-p)\times(v_1-p)\|}, \end{aligned} \tag{14}$$

where n_m is the surface normal on point m. There are two crucial differences between the 3D edge integral (Equation 14) and the previous screen space edge integral (Equation 7). First, while the measure of the screen space edge integral $\sigma(x,y)$ coincides with the unit length of the 2D edge, the measure of the 3D edge integral $\sigma'(m)$ is the length of projection of a point on the edge from the shading point p to a point m on the scene manifold (the semi-transparent triangle in Figure 4a illustrates the projection). Second, there is an extra area correction term $\|n_m \times n_h\|$, since we need to project the scene surface element onto the infinitesimal width of the edge (Figure 4b).

To integrate the 3D edge integral using Monte Carlo sampling we substitute the variable again from the point m on the surface to the line parameter t on the edge $v_0 + t(v_1 - v_0)$:

$$\int_0^1 \frac{\nabla\alpha(p,m(t))}{\|\nabla_m\alpha(p,m(t))\|} h(p,m(t)) \frac{\|J_m(t)\|}{\|n_m \times n_h\|} dt, \tag{15}$$

where the Jacobian $J_m(t)$ is a 3D vector describing the projection of edge (v_0, v_1) onto the scene manifold with respect to the line parameter. We derive the Jacobian in Appendix A.1.

The derivatives for $\alpha(p,m)$ needed to compute the edge integral are:

$$\begin{aligned} &\|\nabla_m\alpha(p,m)\| = \|(v_0-p)\times(v_1-p)\| \\ &\nabla_{v_0}\alpha(p,m) = v_1 \times m, \ \nabla_{v_1}\alpha(p,m) = m \times v_0 \\ &\nabla_p\alpha(p,m) = (v_0-p)\times(v_1-p). \end{aligned} \tag{16}$$

Efficient Monte Carlo sampling of secondary edges is more involved. Unlike primary visibility where the viewpoint does not change much, shading point p can be anywhere in the scene. The consequence is that we need a more sophisticated data structure to prune the edges with zero contribution. Section 4 describes the process for importance sampling edges.

4 IMPORTANCE SAMPLING THE EDGES

Our edge sampling method described in Section 3 requires us to sample an edge from hundreds of thousands, or even millions of triangles in the scene. The problem is two-fold: we need to sample an edge and then sample a point on the edge efficiently. Typically only a tiny fraction of these edges contribute to the gradients, since most of the edges are not silhouette (e.g. [Hertzmann 1999]), and many of them may have small solid angle. Naive sampling methods fail to select important edges. Even if the number of edges is small, it is often the case that only a small region on the edge has non-zero contribution, especially when there exists highly-specular materials.

As mentioned in Section 3.1, the case for primary visibility is easier since the viewpoint is the camera. We project all edges onto the screen in a preprocessing pass, and test whether they are silhouette with respect to the camera position. We sample an edge based on the distance of two points on the screen and uniformly sample in screen space. For secondary visibility the problem is much more complicated. The viewpoint can be anywhere in the scene, and we need to take the material response between the viewpoint and the point on the edge into account.

In this section we describe a scalable implementation for sampling edges given arbitrary viewpoint. Our solution is inspired by previous methods for sampling many light sources using hierarchical data structures (e.g. [Estevez and Kulla 2018; Paquette et al. 1998; Walter et al. 2005]), efficient data structures for selecting silhouette edges [Hertzmann and Zorin 2000; Sander et al. 2000], and the more recent closed-form solution for linear light sources [Heitz et al. 2016; Heitz and Hill 2017].

4.1 Hierarchical edge sampling

Given a shading point, our first task is to importance sample one or more triangle edges. There are several factors to take into account

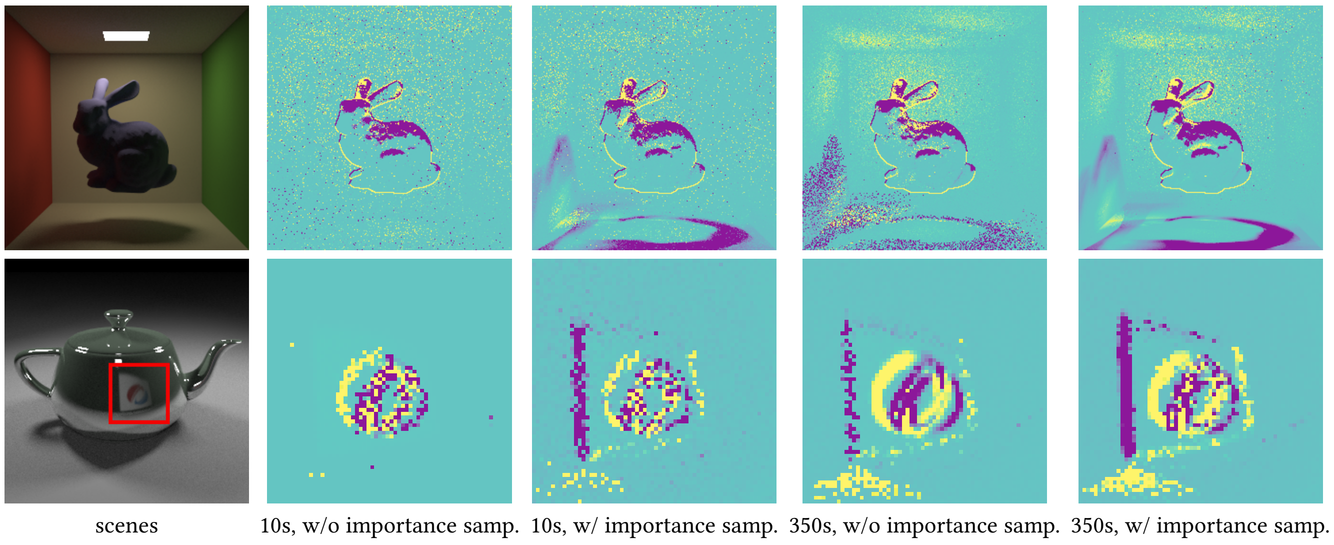

scenes
10s, w/o importance samp.
10s, w/ importance samp.
350s, w/o importance samp.
350s, w/ importance samp.

Fig. 5. Equal time comparison between sampling with and without our importance sampling method. We tested our algorithm on scenes with soft shadow, global illumination, and specular reflection. We show the per-pixel derivatives of average color with respect to the bunny moving up in the top row, and the derivatives with respect to the reflected plane with the SIGGRAPH logo moving right in the second row. For the second row we only show the gradients in the red inset. The texture derivatives are resolved better without importance sampling since it has less overhead and more samples. However, without importance sampling it is difficult to capture rare events such as shadows cast by a small area light or very specular reflection of edges, causing extremely high variance in the gradients.

when selecting the edges: the geometric foreshortening factor proportional to the inverse squared distance to the edge, the material response between the shading point and the point on the edge, and the radiance incoming from the edge direction (e.g. whether it hits a light source or not).

We build two hierarchies. The first contains the triangle edges that associate with only one face and meshes that do not use smooth shading normals. The second contains the remaining edges. For the first set of edges we build a 3D bounding volume hierarchy using the 3D positions of the two endpoints of an edge. For the second set of edges we build a 6D bounding volume hierarchy using the two endpoint positions and the two normals associated with the two faces of an edge. For quick rejection of non-silhouette edges, for each node in the hierarchy we store a cone direction and an angle covering all possible normal directions [Sander et al. 2000]. An alternative might be a 3D hierarchy similar to the ones used by Sander et al. [2000] or Hertzmann and Zorin [2000], but we opt for simplicity here. Similar to previous work [Walter et al. 2006], we scale the directional components with $\frac{1}{8}$ the diagonal of the scene's bounding box. During the construction we split the dimension with longest extent.

We traverse the hierarchies to sample edges. The edges blocking light sources are usually the most significant source of contribution. Therefore we traverse the hierarchy twice. The first traversal focuses on edges that overlap with the cone subtended by the light source at the shading point, and the second traversal samples all edges. We combine the two sets of samples using multiple importance sampling [Veach and Guibas 1995]. We use a box-cone intersection to quickly discard the edges that do not intersect the light sources.

During the traversal, for each node in the hierarchy we compute an importance value for selecting which child to traverse next, based on an upper bound estimation of the contribution, similar to the lightcuts algorithm [Walter et al. 2005]. We estimate the bound using the total length of edges times inverse squared distance times a Blinn-Phong BRDF (using the method described in Walter's note [2005]). We set the importance to zero if the node does not contain any silhouette. We traverse into both children if the shading point is inside both of their bounding boxes, or when the BRDF bound is higher than a certain threshold (for all examples in the paper we set it to 1), or when the angle subtended by the light cone is smaller than a threshold (we set it to $\cos^{-1}(0.95)$).

4.2 Importance sampling a single edge.

After we select a set of edges, we need to choose a point on the edge we selected. Oftentimes with a highly-specular BRDF, only a small portion of the edge has significant contribution. We employ the recent technique on integrating linear light sources over Linearly Transformed Cosine Distribution [Heitz et al. 2016; Heitz and Hill 2017]. Heitz and Hill's work provides a closed-form solution of the integral between a point and a linear light source, weighted by BRDF and geometric foreshortening. We numerically invert the integrated cumulative distribution function using Newton's method for importance sampling. We precompute a table of fitted linearly transformed cosine for our BRDFs.

We evaluate our sampling method using equal-time comparison and show the results in Figure 5. We compare against the baseline approach of uniformly sampling all edges by length. The baseline

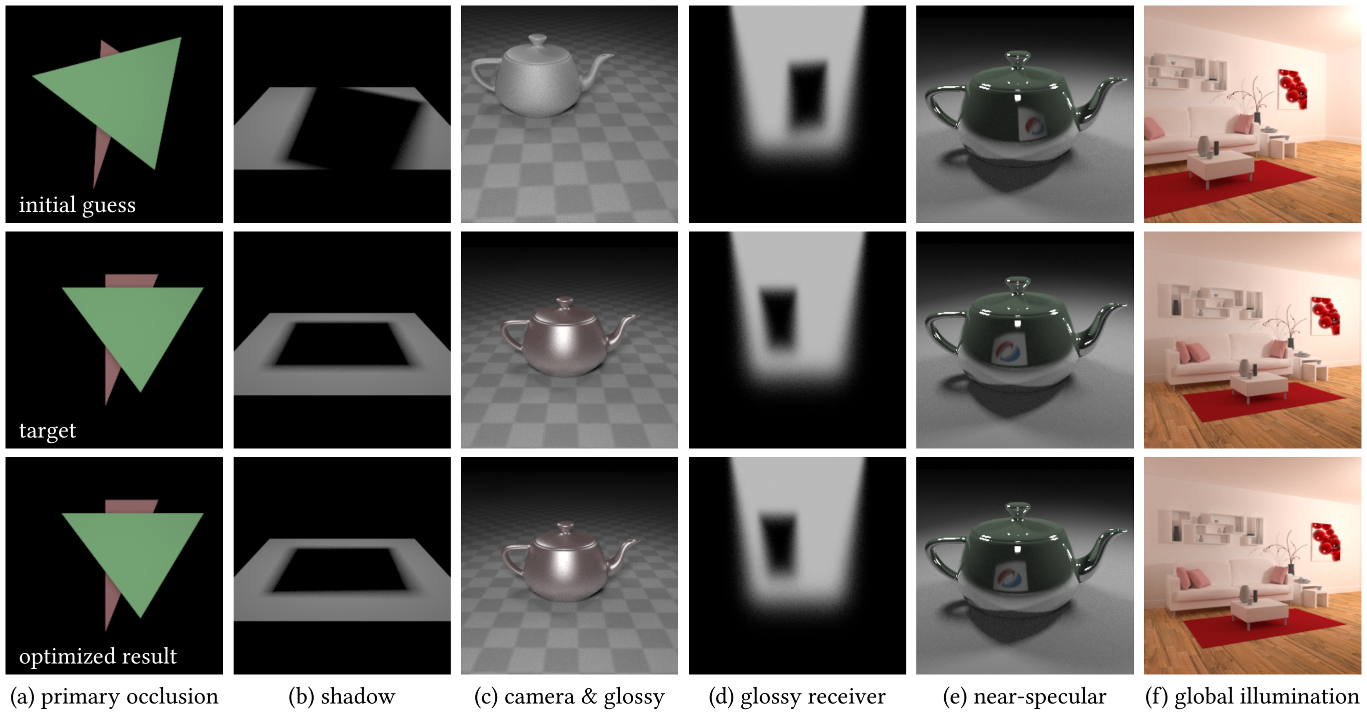


Fig. 6. We verify our renderer by matching a range of synthetic scenes with different light transport configurations. For each scene, we start from an initial parameter (first row) and attempt to set scene parameters so that the rendering matches the target (second row) using gradient-based optimization. Each scene is intended to test a different aspect of the renderer. (a) optimizes triangle positions under the presence of occlusion. (b) optimizes blocker position for shadow. (c) optimizes camera pose and material parameters over textured and glossy surfaces. (d) optimizes the blocker position where the shadow receiver is highly glossy. (e) optimizes an almost specular reflection of a plane behind the camera; the free parameter is the plane position. (f) optimizes camera pose under the presence of global illumination and soft shadow. Our method is able to generate gradients for these scenes and to optimize the parameters correctly, resulting in minimal difference between the optimized result (final row) and target (second row). All the scenes are rendered with 4 samples per pixel during optimization. The final renderings are produced with 625 samples per pixel, except for (f) we use 4096 samples. We encourage the reader to refer to the supplementary materials for videos and more scenes.

approach is not able to efficiently sample rare events such as shadows cast by a small light source or very specular reflection of edges, while our importance sampling generates images with much lower variance.

5 RESULTS

We implement our method in a stand-alone C++ renderer with an interface to the automatic differentiation library PyTorch [Paszke et al. 2017]. To use our system, the user constructs their scenes using lists of PyTorch tensors. For example, triangle vertices and indices are represented by floating point and integer tensors. Our renderer in the forward pass outputs an image which is also a PyTorch tensor. The user can then compute a scalar loss on the output image and obtain the gradient by backpropagating to the scene parameters. Inside our C++ renderer, we use an operator overloading approach for automatic differentiation. We use the Embree library [Wald et al. 2014] for our ray casting operations. The renderer supports a pinhole camera with planar and equiangular spherical projection, Lambertian and Blinn-Phong BRDFs with Schlick approximation for Fresnel reflection, bilinear reconstruction of textures for diffuse and specular reflectance and roughness, and area light sources with triangle meshes.

5.1 Verification of the method

We tested our method on several synthetic scenes covering a variety of effects, including occlusion, non-Lambertian materials, and global illumination. Figure 6 shows the scenes. We start from an initial parameter, and try to optimize the parameters to minimize the L^2 difference between the rendered image and target image using gradients generated by our method (except for the living room scene in Figure 6 (f) where we optimize for the L^2 difference between the Gaussian pyramids of the rendered image and target image). Our PyTorch interface allows us to apply their in-stock optimizers, and backpropagate to all scene parameters easily. We use the Adam [Kingma and Ba 2015] algorithm for optimization. The number of parameters ranges from 6 to 30. The experiment shows that our renderer is able to generate correct gradients for the optimizer to infer the correct scenes. It also shows that we are able to handle many different light transport scenarios, including cases where a triangle vertex is blocked but we still need to optimize it into the correct position, optimization of blocker position when we only see

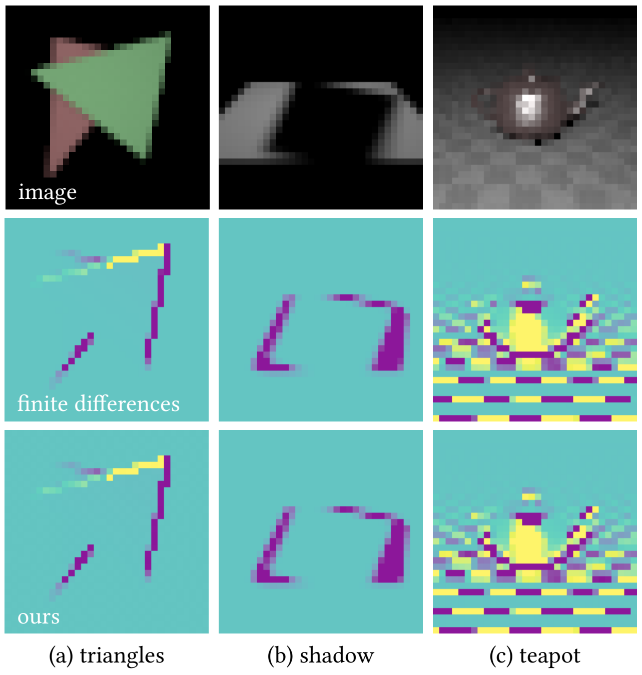


Fig. 7. We compare with central finite differences by rendering the scenes in Figure 6 at 32×32. The scenes are slightly adjusted to make the per-pixel gradient look clearer in the image. The derivatives are with respect to (a) each rightmost vertex of the two triangles moving left (b) the shadow blocker moving up (c) the camera moving into the screen. Our derivatives match the finite differences within an error of 1% relative to the L^1 norm of the gradients. Finite differences usually take two or three orders of magnitude more samples to reach the same error. For our method, we use 16 thousand samples per pixel for the scene with two triangles and 32 thousand samples per pixel for the other two scenes. For finite differences, we use 1 million samples per pixel for the triangles scene and 10 million samples per pixel for the rest.

the shadow, joint optimization of camera and material parameters, pose estimation in presence of global illumination, optimizing blockers occluding highly-glossy reflection, and inverting near specular reflection. See the supplementary materials for more results.

We also compare our method to central finite differences on a lower resolution version of the synthetic scenes in Figure 7. Our derivatives match the finite difference within an error of 1% relative to the L^1 norm of the gradients. The comparison is roughly equal quality. We increase the number of samples for the finite differences until the error is low enough. In general finite difference requires a small step size to measure the visibility gradient correctly, thus they usually take two or three orders of magnitude more samples to reach the same error as our result. In addition, finite differences do not scale with the number of parameters, making them impractical for most optimization tasks.

Figure 8 demonstrates the convergence of our method by visualizing the gradients of the bunny scene in Figure 5 over different numbers of samples per pixel. We show the gradients of the average of pixel colors with respect to the bunny moving right on the screen.

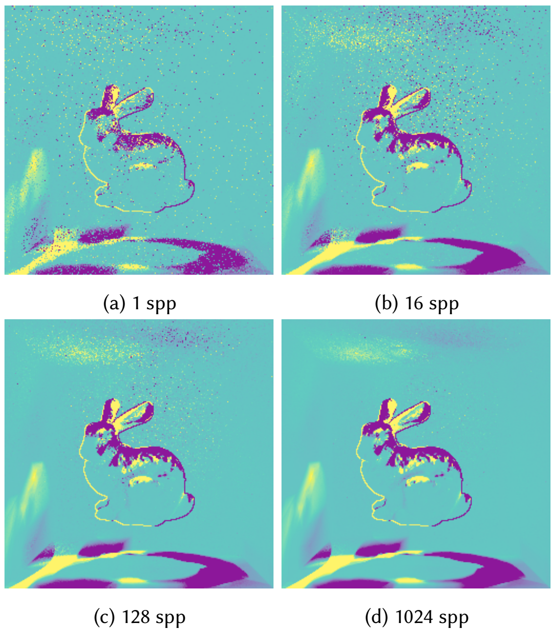


Fig. 8. We visualize the per-pixel gradient contribution generated by our method over different numbers of the samples per pixel. We take the bunny scene from Figure 5. The gradient is the average of color with respect to the bunny moving right. The 1024 samples per pixel image took around 40 minutes to compute on a 6-core machine. In practice we usually use 4 samples per pixel for inverse rendering.

Generating the near-converged 1024 samples per pixel image takes around 40 minutes on a 6-core machine. In practice we don't render converged images for optimization. We utilize stochastic gradient descent and render a low sample count image (usually 4).

5.2 Comparison with previous differentiable renderers

In this subsection we compare our method with two previously proposed differentiable renderers: OpenDR [Loper and Black 2014] and Neural 3D Mesh renderer [Kato et al. 2018]. Both previous methods focus on speed and approximate the gradients even under Lambertian materials with unshadowed direct lighting. In contrast, our method outputs unbiased gradients and supports arbitrary non-Dirac materials, shadow, and global illumination, as shown in Figure 6.

Both OpenDR and Neural 3D Mesh renderer follow the approach of first rendering into a color buffer using a traditional rasterizer with z-buffer. They then approximate the derivatives with respect to screen-space triangle vertex positions using the rendered color buffer. OpenDR performs a screen-space filtering approach based on a brightness constancy assumption [Jones and Poggio 1996]. The shape of the filter is determined by boundary detection using triangle ID. For the horizontal derivatives of a pixel neighboring an occlusion boundary on the left, they use the kernel $[0, -1, 1]$. For pixels that are not neighboring any boundaries, or are intersecting with boundaries, or are neighboring more than one occlusion

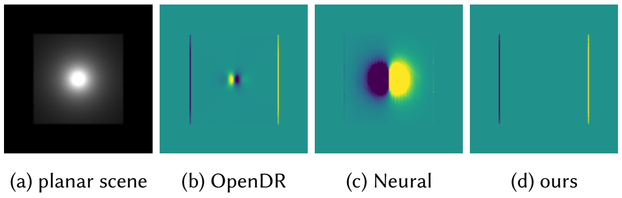


Fig. 9. (a) A plane lit by a point light close to the plane. We are interested in the derivative of the image with respect to the plane moving right. Since the point light stays static, the derivatives should be zero except for the boundary. (b) (c) Previous work uses color buffer differences to approximate the derivatives, making them unable to take large variation between pixels into account and output non zero derivatives at the center. (d) Our method outputs the correct derivatives.

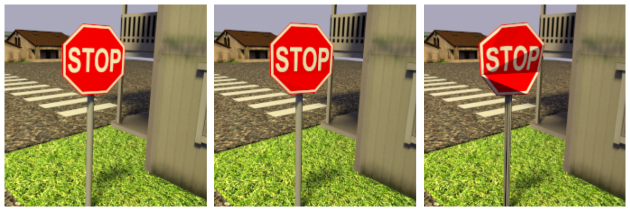
boundary, they use the kernel $\frac{1}{2}[-1, 0, 1]$. The Neural 3D Mesh renderer performs an extra edge rasterization pass of the triangle edges and accumulates the derivatives by computing the difference between the color difference on the color buffer around the edge. The derivative responses are modified by applying a smooth falloff.

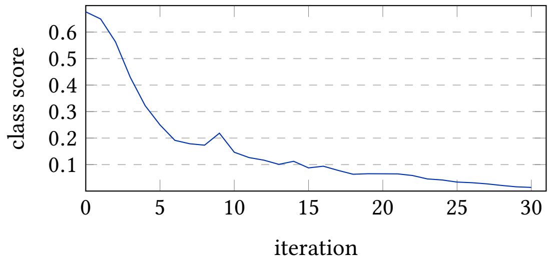
Both previous differentiable renderers output incorrect gradients in the case where there is brightness variation between pixels due to lighting. Figure 9 shows an example of a plane lit by a point light with inverse squared distance falloff. We ask the two renderers and ours to compute the derivatives of the pixel color with respect to the plane moving right. Since the light source does not move, the illumination on the plane remains static and the derivatives should be zero except for the boundaries of the plane. Since both previous renderers use the differences between color buffer pixels to approximate derivatives, they incorrectly take the illumination variation as the changes that would happen if the plane moves right, and output non-zero derivatives around the highlights. On the other hand, since we sample *on* the edges, our method correctly outputs zero derivatives on continuous regions.

OpenDR's point light does not have distance falloff and the Neural 3D mesh renderer does not support point lights so we modified their renderers. Our renderer does not support pure point lights so we use a small planar area light to approximate a point light. We also tessellate the plane into 256×256 grids as both previous renderers use Gouraud shading.

5.3 Inverse rendering application

We apply our method on an inverse rendering task for fitting camera pose, material parameters, and light source intensity. Figure 1 shows the result. We take the scene photo and geometry data from the thesis work of Jones [2017], where the scene was used for validating daylight simulation. The scene contains strong indirect illumination and has non-Lambertian materials. We assign most of the materials to white except for plastic or metal-like objects, and choose an arbitrary camera pose as an initial guess. There are in total 177 parameters for this scene. We then use gradient-based optimizer Adam and the gradients generated by our method, to find the correct camera pose and material/lighting parameters. In order to avoid getting stuck in local minima, we perform the optimization in a multi-scale fashion, starting from 64×64 and linearly increasing to the final resolution 512×512 through 8 stages. For each scale we use an L^1 loss and perform 50 iterations. We exclude the light source in the loss function by setting the weights of pixels with radiance larger than 5 to 0.





(d) combined class score of street sign and traffic light

Fig. 10. Our method can be used for finding 3D scenes as adversarial examples for neural networks. We use the gradient generated by our method to optimize for the geometry of the stop sign, camera pose, light intensity and direction to minimize the class scores of street sign and traffic light classes. After 5 iterations the network classifies the stop sign as a handrail, and after 25 iterations both street sign and traffic light are out of the top 5 prediction. In (d) we plot the sum of street sign and traffic light class scores as a function of iteration. As we optimize scene parameters such as the stop sign shape, gradient descent tries to find the geometries that minimizes the class scores, thus we see decreasing of the score.

5.4 3D adversarial example

Recently, it has been shown that gradient-based optimization can also be used for finding adversarial examples for neural networks (e.g. [Goodfellow et al. 2015; Szegedy et al. 2014]) for analysis or mining training data. The idea is to take an image that was originally labelled correctly by a neural network classifier, and use backpropagation to find an image that minimizes the network's output with respect to the correct output. Our system can be used for mining adversarial examples of 3D scenes, since it provides the ability to backpropagate from image to scene parameters. A similar idea has been explored by Zeng et al. [2017], but we use a more general renderer.

We demonstrate this in Figure 10. We show a stop sign classified correctly as a street sign by the VGG16 classifier [Simonyan and Zisserman 2014]. We then optimize for 2256 parameters including camera pose, light intensity, sun position, global translation, rotation, and vertex displacement of the stop sign. We perform stochastic gradient descent to minimize the network's output of the classes street sign and traffic light, using 256 samples per pixel. After 5 iterations the network starts to output handrail as the most probable class. After 23 iterations both the street sign class and traffic light class are out of the top-5 predictions and the sum of the two has less than 5% probability.

We do not claim this is a robust way to break or to attack neural networks, since the CG scene we use has different statistics compared to real world images. Nevertheless this demonstrates that our gradient can be used for finding interesting scene configurations and can be potentially used for mining training data.

5.5 Limitations

Performance. Our current CPU implementation takes seconds to minutes to generate a small resolution image (say 256×256) with a small number of samples (say 4). Note though that when using stochastic gradient descent it is usually not necessary to use high sample counts.

We have found that, depending on the type of scene, the bottleneck can be at the edge sampling phase or during automatic differentiation of the light paths. Developing better sampling algorithms such as incorporating bidirectional path tracing could be an interesting avenue of future work. Developing better compiler techniques for optimizing automatic differentiation code and supporting GPU backends is also an important task.

Other light transport phenomena. We assume static scenes with no participating media. Differentiating motion blur requires sampling on 4D edges with an extra time dimension. Combining our method with Gkioulekas et al.'s work [2013] for handling participating media is left as future work.

Interpenetrating geometries and parallel edges. Dealing with the derivatives of interpenetration of triangles requires a mesh splitting process and its derivatives. Interpeneration can happen if the mesh is generated by some simulation process. Our method also does not handle the case where two edges are perfectly aligned as seen from the center of projection (camera or shadow ray origin). However, these are zero-measure sets in path space, and as long as the two edges are not perfectly aligned to the viewport, we will be able to converge to the correct solution.

Shader discontinuities. We assume our BRDF models and shaders are differentiable and do not handle discontinuities in the shaders. We handle textures correctly by differentiating through the smooth reconstruction, and many widely-used reflection models such as GGX [Walter et al. 2007] (with Smith masking) or Disney's principled BRDF [Burley 2012] are differentiable. However, we do not handle the discontinuities at total internal reflection and some other BRDFs relying on discrete operations, such as the discrete stochastic microfacet model of Jakob et al. [2014]. Compiler techniques for band-limiting BRDFs can be applied to mitigate the shader discontinuity issue [Yang and Barnes 2018].

6 CONCLUSION

We have introduced a differentiable Monte Carlo ray tracing algorithm that is capable of generating correct and unbiased gradients with respect to arbitrary input parameters such as scene geometry, camera, lights and materials. For this, we have introduced a novel edge sampling algorithm to take the geometric discontinuities into consideration, and derived the appropriate measure conversion. For increased efficiency, we use a new discrete hierarchical sampling method to focus on relevant edges as well as continuous edge importance sampling. We believe this method and the software that we will release will have an impact in inverse rendering and deep learning.

A APPENDIX

A.1 Derivation of the 3D edge Jacobian

We derive the Jacobian $J_m(t)$ in Equation 15. The goal is to compute the derivatives of point $m(t)$ with respect to the line parameter t. The relation between $m(t)$ and t is described by a ray-plane intersection. That is, we are intersecting a plane at point m with normal n_m with a ray of origin p and unnormalized direction $\omega(t)$:

$$\begin{aligned} \omega(t) &= v_0 + (v_1 - v_0)t - p \\ \tau(t) &= \frac{(m-p)\cdot n_m}{\omega(t)\cdot n_m} \\ m(t) &= \tau(t)\omega(t). \end{aligned} \tag{17}$$

We can then derive the derivative $J_m(t) = \frac{\partial m(t)}{\partial t}$ as:

$$J_m(t) = \tau(t)\left((v_1 - v_0) - \omega(t)\frac{(v_1 - v_0)\cdot n_m}{\omega(t)\cdot n_m}\right) \tag{18}$$

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their detailed comments (especially reviewer #1). The work started as an internship project at NVIDIA, where Marco Salvi and Aaron Lefohn provided immensely helpful advice. Luke Anderson and Prafull Sharma helped proofread the drafts. Nathaniel Jones modelled the conference scene in Figure 1 and gave helpful comments on the applications of inverse rendering for architectural design. The teapot model in Figure 5 was modelled by Martin Newell and the bunny in the same figure was created by Brian Curless and Marc Levoy. Both of them were downloaded from Morgan McGuire's website [2017]. The living room scene in Figure 6 (f) was modelled by Wig42 and ported to Mitsuba scene format by Benedikt Bitterli [2016]. The stop sign in Figure 10 was modelled by Elijah Rai, and the street in the same figure was modelled by Pabong. Both of them were downloaded from Free3D.com. The work is funded by Toyota Research Institute.

REFERENCES

Miika Aittala, Timo Aila, and Jaakko Lehtinen. 2016. Reflectance modeling by neural texture synthesis. *ACM Trans. Graph. (Proc. SIGGRAPH)* 35, 4 (2016), 65:1–65:13.

Miika Aittala, Tim Weyrich, and Jaakko Lehtinen. 2013. Practical SVBRDF Capture In The Frequency Domain. *ACM Trans. Graph. (Proc. SIGGRAPH)* 32, 4 (2013), 110:1–110:12.

Miika Aittala, Tim Weyrich, and Jaakko Lehtinen. 2015. Two-shot SVBRDF Capture for Stationary Materials. *ACM Trans. Graph. (Proc. SIGGRAPH)* 34, 4 (2015), 110:1–110:13.

James Arvo. 1994. The Irradiance Jacobian for Partially Occluded Polyhedral Sources. In *SIGGRAPH*. 343–350.

Jonathan T Barron and Jitendra Malik. 2015. Shape, Illumination, and Reflectance from Shading. *Transactions on Pattern Analysis and Machine Intelligence* 37, 8 (2015), 1670–1687.

Bruce Guenther Baumgart. 1974. *Geometric modeling for computer vision.* Technical Report. Stanford University Department of Computer Science.

Benedikt Bitterli. 2016. Rendering resources. https://benedikt-bitterli.me/resources/.

Volker Blanz and Thomas Vetter. 1999. A morphable model for the synthesis of 3D faces. In *SIGGRAPH*. 187–194.

Adrien Bousseau, Emmanuelle Chapoulie, Ravi Ramamoorthi, and Maneesh Agrawala. 2011. Optimizing environment maps for material depiction. *Computer Graphics Forum (Proc. EGSR)* 30, 4 (2011), 1171–1180.

Brent Burley. 2012. Physically-based shading at Disney. In *SIGGRAPH Course Notes. Practical physically-based shading in film and game production.*, Vol. 2012. 1–7.

Min Chen and James Arvo. 2000. Theory and application of specular path perturbation. *ACM Trans. Graph.* 19, 4 (2000), 246–278.

Alejandro Conty Estevez and Christopher Kulla. 2018. Importance Sampling of Many Lights with Adaptive Tree Splitting. *ACM Comput. Graph. Interact. Tech. (Proc. HPG)* 1, 2 (2018), 25:1–25:17.

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.* 32, 6 (2013), 162:1–162:13.

Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *International Conference on Learning Representations*.

Andreas Griewank and Andrea Walther. 2008. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation* (second ed.). Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

Eric Heitz, Jonathan Dupuy, Stephen Hill, and David Neubelt. 2016. Real-time polygonal-light shading with linearly transformed cosines. *ACM Trans. Graph. (Proc. SIGGRAPH)* 35, 4 (2016), 41:1–41:8.

Eric Heitz and Stephen Hill. 2017. Linear-Light Shading with Linearly Transformed Cosines. In *GPU Zen*.

Aaron Hertzmann. 1999. Introduction to 3D Non-Photorealistic Rendering: Silhouettes and Outlines. In *SIGGRAPH Course Notes. Course on Non-Photorelistic Rendering*, Stuart Green (Ed.). ACM Press/ACM SIGGRAPH, New York.

Aaron Hertzmann and Denis Zorin. 2000. Illustrating smooth surfaces. In *SIGGRAPH*. 517–526.

Lars Hörmander. 1983. *The analysis of linear partial differential operators I: Distribution theory and Fourier analysis*.

Homan Igehy. 1999. Tracing Ray Differentials. *SIGGRAPH*, 179–186.

Wenzel Jakob, Miloš Hašan, Ling-Qi Yan, Jason Lawrence, Ravi Ramamoorthi, and Steve Marschner. 2014. Discrete stochastic microfacet models. *ACM Transs Graph. (Proc. SIGGRAPH)* 33, 4 (2014), 115:1–115:10.

Wenzel Jakob and Steve Marschner. 2012. Manifold exploration: a Markov Chain Monte Carlo technique for rendering scenes with difficult specular transport. *ACM Trans. Graph. (Proc. SIGGRAPH)* 31, 4 (2012), 58:1–58:13.

Wojciech Jarosz, Volker Schönefeld, Leif Kobbelt, and Henrik Wann Jensen. 2012. Theory, analysis and applications of 2D global illumination. *ACM Trans. Graph.* 31, 5 (2012), 125:1–125:21.

Michael J. Jones and Tomaso Poggio. 1996. *Model-Based Matching by Linear Combinations of Prototypes.* Technical Report.

Nathaniel Louis Jones. 2017. *Validated interactive daylighting analysis for architectural design.* Ph.D. Dissertation. Massachusetts Institute of Technology.

James T. Kajiya. 1986. The Rendering Equation. *Computer Graphics (Proc. SIGGRAPH)* 20, 4 (1986), 143–150.

Anton S Kaplanyan, Johannes Hanika, and Carsten Dachsbacher. 2014. The natural-constraint representation of the path space for efficient light transport simulation. *ACM Trans. Graph. (Proc. SIGGRAPH)* 33, 4 (2014), 102:1–102:13.

Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. 2018. Neural 3D Mesh Renderer. In *Conference on Computer Vision and Pattern Recognition*. 3907–3916.

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

Diederick P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

Jaroslav Krivanek, Pascal Gautron, Sumanta Pattanaik, and Kadi Bouatouch. 2005. Radiance Caching for Efficient Global Illumination. (2005), 550–561.

Tzu-Mao Li, Jaakko Lehtinen, Ravi Ramamoorthi, Wenzel Jakob, and Frédo Durand. 2015. Anisotropic Gaussian Mutations for Metropolis Light Transport through Hessian-Hamiltonian Dynamics. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)* 34, 6 (2015), 209:1–209:13.

Guilin Liu, Duygu Ceylan, Ersin Yumer, Jimei Yang, and Jyh-Ming Lien. 2017. Material Editing Using a Physically Based Rendering Network. In *International Conference on Computer Vision*. 2280–2288.

Matthew M. Loper and Michael J. Black. 2014. OpenDR: An Approximate Differentiable Renderer. In *European Conference on Computer Vision*, Vol. 8695. 154–169.

Morgan McGuire. 2017. Computer Graphics Archive. https://casual-effects.com/data

Eric Paquette, Pierre Poulin, and George Drettakis. 1998. A Light Hierarchy for Fast Rendering of Scenes with Many Lights. *Computer Graphics Forum (Proc. Eurographics)* (1998), 63–74.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. (2017).

Gustavo Patow and Xavier Pueyo. 2003. A survey of inverse rendering problems. *Computer Graphics Forum* 22, 4 (2003), 663–687.

Ravi Ramamoorthi, Dhruv Mahajan, and Peter Belhumeur. 2007. A First-order Analysis of Lighting, Shading, and Shadows. *ACM Trans. Graph.* 26, 1 (2007), 2:1–2:21.

Elad Richardson, Matan Sela, Roy Or-El, and Ron Kimmel. 2017. Learning detailed face reconstruction from a single image. In *Conference on Computer Vision and Pattern Recognition*. 5553–5562.

Pedro V. Sander, Xianfeng Gu, Steven J. Gortler, Hugues Hoppe, and John Snyder. 2000. Silhouette Clipping. In *SIGGRAPH*. 327–334.

Ram Shacked and Dani Lischinski. 2001. Automatic lighting design using a perceptual quality metric. *Computer Graphics Forum* 20, 3 (2001), 215–227.

Mikio Shinya, T. Takahashi, and Seiichiro Naito. 1987. Principles and Applications of Pencil Tracing. *Comput. Graph. (Proc. SIGGRAPH)* 21, 4 (1987), 45–54.

K. Simonyan and A. Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv preprint arXiv:1409.1556* (2014).

Frank Suykens and Yves D. Willems. 2001. Path Differentials and Applications. In *Eurographics Workshop on Rendering Techniques*. 257–268.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *International Conference on Learning Representations*.

Eric Veach and Leonidas J. Guibas. 1995. Optimally Combining Sampling Techniques for Monte Carlo Rendering. In *SIGGRAPH*. 419–428.

Eric Veach and Leonidas J. Guibas. 1997. Metropolis Light Transport. In *SIGGRAPH*. 65–76.

Ingo Wald, Sven Woop, Carsten Benthin, Gregory S Johnson, and Manfred Ernst. 2014. Embree: a kernel framework for efficient CPU ray tracing. *ACM Trans. on Graph. (Proc. SIGGRAPH)* 33, 4 (2014), 143.

Bruce Walter. 2005. Notes on the Ward BRDF. *Program of Computer Graphics, Cornell University, Technical report PCG-05* 6 (2005).

Bruce Walter, Adam Arbree, Kavita Bala, and Donald P Greenberg. 2006. Multidimensional lightcuts. *ACM Trans. Graph. (Proc. SIGGRAPH)* 25, 3 (2006), 1081–1088.

Bruce Walter, Sebastian Fernandez, Adam Arbree, Kavita Bala, Michael Donikian, and Donald P Greenberg. 2005. Lightcuts: a scalable approach to illumination. *ACM Trans. Graph. (Proc. SIGGRAPH)* 24, 3 (2005), 1098–1107.

Bruce Walter, Stephen R Marschner, Hongsong Li, and Kenneth E Torrance. 2007. Microfacet models for refraction through rough surfaces. *Rendering Techniques (Proc. EGSR)* (2007), 195–206.

Greg Ward and Paul Heckbert. 1992. Irradiance Gradients. In *Eurographics Rendering Workshop*. 85–98.

Y. Yang and C. Barnes. 2018. Approximate Program Smoothing Using Mean-Variance Statistics, with Application to Procedural Shader Bandlimiting. *Computer Graphics Forum (Proc. Eurographics)* 37, 2 (2018), 443–454.

Yizhou Yu, Paul Debevec, Jitendra Malik, and Tim Hawkins. 1999. Inverse global illumination: Recovering reflectance models of real scenes from photographs. In *SIGGRAPH*. 215–224.

Xiaohui Zeng, Chenxi Liu, Weichao Qiu, Lingxi Xie, Yu-Wing Tai, Chi Keung Tang, and Alan L Yuille. 2017. Adversarial Attacks Beyond the Image Space. *arXiv preprint arXiv:1711.07183* (2017).