Differentiable Neural Radiosity

SAEED HADADAN, University of Maryland, College Park, USA
MATTHIAS ZWICKER, University of Maryland, College Park, USA

We introduce Differentiable Neural Radiosity, a novel method of representing differential global illumination effects while keeping memory and time complexity constant in path length. To solve inverse rendering problems, we use a pre-trained instance of our network that represents the differential rendering equation to directly optimize our network. The network is capable of outputting continuous, view-independent gradients of the radiance field with respect to scene parameters, taking into account differential global illumination effects while keeping memory and time complexity constant in path length. To solve inverse rendering problems, we leverage this to achieve faster and more accurate convergence compared to other techniques such as Automatic Differentiation, Radiative Backpropagation, and Path Replay Backpropagation.

Additional Key Words and Phrases: Photo-realistic rendering, Ray Tracing, Global Illumination, Differentiable Rendering, Neural Rendering, Neural Radiance Fields

1 INTRODUCTION

Differentiable rendering is the problem of computing derivatives of a rendering process with respect to scene parameters such as BRDFs, geometry, illumination, or camera parameters. Differentiable rendering is attractive because it enables gradient-based optimization in inverse rendering problems, where scene parameters are optimized such that the rendering process produces a target image (or multiple target images from different viewpoints). In combination with neural network-based 3D representations, differential rendering is popular because it enables end-to-end training of 3D neural representations with only images as training data.

Here we are focusing on differentiable rendering for algorithms that solve the rendering equation, with the ultimate goal to eventually enable inverse rendering based on real world photographs under large numbers of unknown scene parameters. Previous techniques build on Monte Carlo path tracing to solve the rendering equation, and differentiable rendering can be achieved using generic Automatic Differentiation (AD) techniques, or with specialized algorithms such as Radiative Backpropagation (RB) or Path Replay Backpropagation (PRB). However, these techniques can be computationally costly and they often cannot provide highly accurate results when computation time is limited as shown in Figure 1.

In this paper, we propose a novel technique that is inspired by the Neural Radiosity approach by Hadadan et al. [2021] to solve the rendering equation. Our key idea is to represent the entire differential radiance field, that is, the derivative of the radiance field with respect to a set of scene parameters, using a neural network. We then train this network to satisfy the differential rendering equation introduced by Nimier-David et al. [2020], and our training scheme is similar in spirit as in Neural Radiosity. We show that given a trained network for an initial scene, we can solve inverse rendering problems to fit BRDF parameters more efficiently and more accurately than previous work. In addition, our approach is the first that produces a continuous view-independent differential radiance field for given scene parameters, instead of only sampling derivatives for a discrete set of rays.

Fig. 1. Inverse rendering using Differentiable Neural Radiosity, compared to Automatic Differentiation, Radiative Backpropagation, and Path Replay Backpropagation. (a) We optimize one BRDF parameter per teapot, roughness for the left and middle teapots and diffuse albedo for the right teapot. (b) Renderings of the optimized materials using our method are more faithful to the target state than the previous state-of-the-art. (c) Given a pre-trained network, gradient computation time during inverse optimization experiences considerable speed-ups using our method. (d) The squared error of the BRDF parameters (here roughness parameter of the middle teapot) diminishes faster than other methods against time (details in Figure 6). Close-ups are only for visualization purpose; the target view is the image (a) bottom, using a more distant sensor.
2 RELATED WORK

Neural Network Techniques for Realistic Rendering. Neural networks can be leveraged in various ways for realistic rendering. Vogels et al. [2018] used neural networks for post-processing of rendered images to achieve denoising, anti-aliasing and super-resolution. Neural networks have also been utilized at the core of realistic rendering algorithms. For example, Müller et al. [2019] and Zhen and Zwicker [2019] use neural networks to learn importance sampling the radiance distribution during Monte Carlo rendering. In addition, Müller et al. [2020] train neural networks to learn control variates, which is a variance reduction technique for solving the rendering equation [Kajiya 1986]. Neural networks can also directly represent the solution of the rendering equation as proposed in two concurrent works, Neural Radiosity [Hadadan et al. 2021] and Neural Radiance Caching [Müller et al. 2021]. In these methods, the benefit of storing the radiance function in a network is that the network can be updated with limited computational effort under incremental changes in the rendered scenes.

This paper aims at leveraging neural networks to learn viewpoint-independent, continuous differential radiance fields, which is to the best of our knowledge unprecedented. We use a single neural network to represent the differential radiance function w.r.t arbitrary scene parameters. In a similar approach to Neural Radiosity, we optimize our network parameters directly by minimizing the norm of the residual of the differential rendering equation. We benefit from the re-renderability of the radiance network as in Neural Radiosity.

Visibility-related Discontinuities. Differentiating the rendering equation without taking into account discontinuities can lead to incorrect gradients, and discontinuities due to occlusions need special treatment. Li et al. [2018] tackle this issue by separating out the rendering equation into continuous and discontinuous parts and propose a silhouette edge sampling approach. Loubet et al. [2019] propose a reparametrization of the rendering integrals such that the positions of discontinuities do not depend on the scene parameters. For simplicity, in this paper we ignore discontinuities and focus on solving the differential rendering equation as a first step.

3 BACKGROUND

3.1 (Differential) Rendering Equation

Realistic rendering algorithms compute a set of measurements $l_k$ where $k$ corresponds to a pixel, given by the measurement equation

$$l_k = \int A \int_H W_k(x, \omega) L(x, \omega) dx d\omega^i,$$

where $L$ is the incident radiance at location $x$ and direction $\omega$ on the pixel, and $W_k$ is the importance of pixel $k$. As radiance $L$ remains constant along unoccluded rays, incident radiance at a pixel location and direction is equal to the outgoing radiance from the nearest surface along the ray. The outgoing radiance at surfaces can be computed using the rendering equation [Kajiya 1986],

$$L(x, \omega_i) = E(x, \omega_i) + \int_H f(x, \omega_i, \omega_o) L(x' (x, \omega_i), -\omega_i) d\omega_i^o,$$

which describes energy conservation in scattering events, i.e. the outgoing radiance at every location $x$ and direction $\omega_i$ can be described as emission of the surface $E(x, \omega_i)$ plus scattering of incident radiance. The integral on the right hand side describes light scattering at incidents directions $\omega_i$ over the projected solid angle hemisphere $H$. Function $f$ here represents bidirectional reflectance distribution function (BSDFs).

Nimier-David et al. [2020] differentiate the above equations w.r.t an arbitrary set of scene parameters $p = (p_1, ..., p_n)$. For simplicity, we use $\partial p$ to represent $\partial / \partial p$. Note that variables preceded by $\partial p$ imply a vectorized gradient w.r.t each parameter. By assuming a static camera where $\partial_p W_k = 0$, we can differentiate Equation 1 as,

$$\partial_p l_k = \int A \int_H W_k(x, \omega) \partial_p L(x, \omega) dx d\omega^i,$$

which describes the relationship between differential measurement $\partial_p l_k$ and differential radiance $\partial_p L$. Differential radiance $\partial_p L$ in turn can be found by differentiating Equation 2,

$$\partial_p L(x, \omega_i) = \partial_p E(x, \omega_i) + \int_H f(x, \omega_i, \omega_o) \partial_p L(x' (x, \omega_i), -\omega_i) d\omega_i^o + \int_H \partial_p f(x, \omega_i, \omega_o) L(x' (x, \omega_i), -\omega_o) d\omega_i^o,$$

where we call the differential rendering equation. This equation explains the scattering of differential radiance in a similar manner to regular radiance (Figure 2). More specifically, the first term describes how differential radiance is emitted from surfaces whose emission is dependent on the scene parameters $p$. The second term means that differential radiance scatters on surfaces based on their BRDFs, just
which is simply the difference of the left and right-hand sides of the rendering equation. The residual \( r_\theta \) is defined as the difference of the left and right hand side of Equation 4,

\[
r_\theta(x, \omega_o) = \partial_p L \phi(x, \omega_o) - \partial_p E(x, \omega_o)
\]

where \( r_\theta \) depends on the parameters \( \theta \) of the differential radiance function \( \partial_p L \phi \). Also, the primal radiance can be represented by a constant parameter set \( \phi \) in \( L_\phi \) which is independent of \( \theta \).

We define our loss as the L2 norm of the residual,

\[
L(\theta) = \|r_\theta(x, \omega_o)\|^2_
\]

Monte Carlo Estimation. The Monte Carlo estimation of the residual norm is

\[
L(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left( \sum_{i=1}^{M} \sum_{i=1}^{Z} \frac{r_\theta(x_i, \omega_{o,i,j})^2}{p(x_i, \omega_{o,i,j})} \right),
\]

where \( N \) is the number of samples, \( x_i \) and \( \omega_{o,i,j} \) are the surface location and the outgoing direction samples, taken from a distribution with density \( p(x, \omega) \).

The Monte Carlo estimation of the incident integral for any \( r_\theta(x_j, \omega_{o,j}) \) is

\[
r_\theta(x_j, \omega_{o,j}) = \partial_p L \phi(x_j, \omega_{o,j}) - \partial_p E(x_j, \omega_{o,j})
\]

Gradients. The Monte Carlo approximation of the gradient of the loss is

\[
\nabla_\theta L(\theta) \approx \frac{1}{N} \sum_{j=1}^{N} \frac{2r_\theta(x_j, \omega_{o,j}) \nabla \theta r_\theta(x_j, \omega_{o,j})}{p(x_j, \omega_{o,j})}.
\]
where the gradient of the residual is

\[
\nabla_\theta g(x, \omega_o) = \nabla_\theta (\partial_\theta L_\theta(x, \omega_o)) - \int_{\mathcal{H}} f(x, \omega_i, \omega_o) \nabla_\theta (\partial_\theta L_\theta(x'(x, \omega_i), -\omega_i)) d\omega_i^+ .
\]

This gradient will be approximated by the Monte Carlo estimation of the incident integral,

\[
\nabla_\theta g(x_j, \omega_{o,j}) = \nabla_\theta (\partial_\theta L_\theta(x_j, \omega_{o,j})) - \frac{1}{M} \sum_{k=1}^{M} \frac{f(x_j, \omega_{i,j,k}, \omega_{o,j}) \nabla_\theta (\partial_\theta L_\theta(x'(x_j, \omega_{i,j,k}), -\omega_{i,j,k}))}{p(\omega_{i,j,k})} .
\]

All the above gradients can be computed using a deep learning automatic differentiation framework during the optimization process, including the gradient terms \( \partial_\theta f \) and \( \partial_\theta E \).

5 INVERSE RENDERING USING OUR METHOD

For inverse rendering, the goal is to optimize a set of scene parameters \( \rho \) using an objective function \( z(.) \), which denotes the distance between a candidate image to the reference, and a rendering function \( g(.) \). To minimize \( z(g(\rho)) \), we need the gradient \( \frac{\partial z}{\partial \rho} \),

\[
\frac{\partial z}{\partial \rho} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial \rho} \tag{13}
\]

where \( y \) is a rendered image \( y = g(\rho) \). The term \( \frac{\partial y}{\partial \rho} \) can be interpreted as the gradient of the loss w.r.t pixel values of the candidate image. In most cases, computing this gradient is easy either manually (e.g. if it is \( L2 \) or \( L1 \)) or using AD (e.g. if it is a neural network). The more challenging part is \( \frac{\partial y}{\partial \rho} \), which is equivalent to the differential measurement vector [\( \partial_\rho l_0...\partial_\rho l_n \)], since we need to differentiate the rendering algorithm. Recall from Equation 3 that \( \partial_\rho l_k \) is the result of integrating the incident differential radiance \( \partial_\rho L \) over locations \( x \) and directions \( \omega \) on the hemisphere at pixel \( k \). Therefore, the task breaks down to finding \( \partial_\rho L(x, \omega) \). In fact, our proposal is to query our neural network \( \partial_\rho L_\theta(x, \omega) \) for the differential radiance distribution, as it can serve as a cached representation of the whole differential radiance distribution.

With the use of our network, inverse rendering breaks down into the following steps:

1. Compute a non-differentiable candidate primal rendering and its distance to the reference (\( L2 \), \( L1 \), etc.).
2. Find the derivatives of the loss w.r.t the pixels of the primal image (to get \( \frac{\partial y}{\partial \rho} \)).
3. Compute \( \frac{\partial y}{\partial \rho} \), which is equivalent to the measurement vector [\( \partial_\rho l_0...\partial_\rho l_n \)]. To do so, we trace rays from the sensor to find the first hit point and at that point, query our differential radiance network \( \partial_\rho L_\theta(x, \omega) \). More formally,

\[
\partial_\rho l_{k,\theta} = \int_{\mathcal{A}} \int_{\mathcal{H}} W_k(x, \omega) \partial_\rho L_\theta(x'(x, \omega)) dx d\omega^+ , \tag{14}
\]

4. Compute the gradient of the loss w.r.t the parameters \( \frac{\partial z}{\partial \rho} \) by multiplying the gradients from step (2) and (3) as in Equation 13.

5. Update the scene parameters using the computed gradient using a learning rate and an optimizer (e.g. Adam [Kingma and Ba 2017]).
6. Fine-tune our networks \( \partial_\rho L_\theta \) and \( L_\theta \) after updating the scene parameters.

Figure 4b summarizes the steps of our pipeline.

6 IMPLEMENTATION

6.1 Training

Our training process has two steps: First, we train a neural network to learn the primal radiance distribution using Neural Radiosity [Hadadan et al. 2021], i.e., minimizing the norm of the residual of the rendering equation in Equation 5. The resulting pre-trained \( L_\theta \) then serves as an estimation of incident radiance in the residual of the differential rendering equation (Equations 4 and 6). In order to train \( \partial_\rho L_\theta \), the \( L2 \) norm of the differential residual is then minimized during training (see the loss function Equation 7).

Similar to Neural Radiosity, our training scheme is a self-training approach, that is, instead of providing noisy estimated data to our network as ground truth regression data, we compute both sides of the differential rendering equation using the same network and minimize the difference (residual) during training (Figure 4a).

Renderer. Our training scheme requires calls to rendering functions, automatic differentiable rendering functions, and deep learning optimization routines. For (differentiable) rendering purposes, we use Mitsuba2 [Nimier-David et al. 2019], which provides useful Python bindings for the underlying C++ implementations, the ability to work with wavefronts of rays, and a differentiable megakernel that provides AD in backward and forward mode. We use Mitsuba
AD to evaluate derivatives of $\partial_p E$ and $\partial_p f$ using forward mode. Mitsuba’s megakernel allows parallel forward propagation for multiple parameters.

6.2 Sampling

Norm of the Residual. To sample the norm of the residual, we need to sample locations $x_i$ and directions $\omega_{i, j}$ in Equation $8$. Both variables are sampled on uniform distributions, i.e. $x$ uniformly on all the surfaces in the scene and $\omega_{i, j}$ uniformly on the unit hemisphere (in case of two-sided BSDFs, we sample on the unit sphere).

Hemispherical Integral. Monte Carlo estimation of the hemispherical scattering integrals in Equation $9$ requires sampling of incident directions $w_{i, j, k}$ and $w_{i, j, l}$. Firstly, we sample both integrals using the same set of incident directions to get performance gain. Since the two integrals are summed, the correlation between the two integrals using the same samples does not produce bias. Secondly, we importance sample emitter directions and BSDF distributions in primal space to obtain incident directions. It is worth mentioning that our samples of $w_{i, j, k}$ are detached [Zeltner et al. 2021], meaning that they are static w.r.t perturbations in scene parameters.

6.3 Architecture

Each of our networks is an MLP with 6 fully-connected layers with 512 neurons per hidden layer. The input parameters are the location $x$ and direction $\omega_0$. Inspired by Hadadan et al. [2021], we also input extra surface properties such as the surface normal, and diffuse and specular reflectance at location $x$. Additionally, we also store multi-resolution learnable feature vectors in sparse grids in the scene and input them to the network, which empowers the network’s ability to learn high frequency functions, demonstrated by Hadadan et al. [2021]. We adopt this approach for both of our networks. Recently, Müller et al. [2022] have introduced a multi-resolution hash encoding as an augmentation to neural networks that enables them to train in order of seconds, in addition to allowing the use of smaller networks without loss of quality. Adopting such architecture can enhance the efficiency of our method to use smaller networks with significant inference and training speedups.

6.4 Updating Networks during Inverse Optimization

As stated in Section 5, the final step of inverse rendering involves fine-tuning after changes in the parameter space at every iteration. Ideally, each of our radiance and differential distributions should be updated to adapt to the scene parameters. That being said, we find fine-tuning a bottleneck during an optimization task. Instead, we propose to keep the networks intact in their initial state, but instead of fetching gradients from $\partial_p L_\theta$, we use the RHS of our solution (see the right hand side in Equation $4$). The intuition behind this choice is that our RHS more quickly adapts to the scene parameter changes, as computing RHS requires one extra ray-tracing step using the updated parameters. After every update, terms $\partial f$ and $f$ in our scattering integrals used to evaluate the RHS are updated and
current. While this leads to an additional bias in our gradients by keeping $\partial_p L_\phi$ and $L_\phi$ outdated, our experiments show that we can still provide convergence improvements compared to the previous state-of-the-art.

### 6.5 Differential Emission Re-parametrization

Hadadan et al. [2021] show that taking the emission term in the residual out of the network ($L_{\phi}$) makes the training process more stable. This is due to the high dynamic range of emitter values. Instead, they compute the emission term separately as it is a simple known function, and add it to the network output. We use a similar approach in the differential context, and the equations below reparameterize $\partial_p L_\phi$ w.r.t. the differential emission term and substitute it in the operator form of the residual,

$$\partial_p L_\phi(x, \omega_o) = N_\phi(x, \omega_o) + \partial_p E(x, \omega_o),$$

$$r_\phi(x, \omega_o) = \partial_p L_\phi(x, \omega_o) - \partial_p E(x, \omega_o) - T(\partial_p L_\phi)(x, \omega_o) - T'(L_\phi)(x, \omega_o) + N_\phi(x, \omega_o) - T(N_\phi + \partial_p E)(x, \omega_o),$$

where the transport operators are,

$$T(\partial_p L_\phi)(x, \omega_o) = \int_{ab} f(x, \omega_i, \omega_o) \partial_p L_\phi(x'(x, \omega_i), -\omega_i) d\omega_i^+,$$

$$T'(L_\phi)(x, \omega_o) = \int_{ab} \partial_p f(x, \omega_i, \omega_o) L_\phi(x'(x, \omega_i), -\omega_i) d\omega_i^+.$$
Differentiable Neural Radiosity

7

(a) Copper teapot (left) – roughness parameter
(b) Plastic teapot (middle) – roughness parameter
(c) Diffuse teapot (right) – albedo

Fig. 6. Experiment 1. Optimization curves for parameter error vs. iteration count and time (results in Figure 1). Given a pre-trained network, our method achieves faster and better convergence compared to the baselines.

|
| Initial state | Target | Ours | AD | RB | PRB | MAPE |
|--------------|-------|------|----|----|-----|-------|
|              |       |      |    |    |     | 6.68e-4 |

Fig. 7. Experiment 2. Optimization for two BRDF parameters of one teapot. The initial state is designed such that the material behaves similar to a diffuse BRDF. Complete iteration time (higher number on each bar) and gradient computation time per iteration (lower number) are reported.

Table 1. Details of inverse optimization experiments. All experiments use path tracing primal rendering at 16 spp and adjoint image sampling at 4 spp. We used different views of the sensor to render primal images in Experiments 1 and 2. A far view of Veach Door (in Figure 1a) was used at resolution 480x270 in Experiment 1. A close-up view (in Figure 1a) at resolution 480x120 was used for Experiment 2.

| Exp. | Fig. | Optimized parameters | Initial state | Target state | lr | Inverse optimization peak memory-GiB | Our model training time |
|------|------|----------------------|---------------|--------------|----|-------------------------------------|------------------------|
| 1    | 1 & 6| Roughness            | .4            | .05          | .0015 | AD: 7.09, RB: 3.51, PRB: 3.00 Ours: 9.58 | \(L_\phi\): 135m, \(\partial_p L_\theta\): 55m |
|      |      | Roughness            | .1            | 0.4          | .002 |                                      |                        |
|      |      | Albedo               | .8, .8, .8    | 3, 3, 9      | .004 |                                      |                        |
| 2    | 7    | Roughness            | 1             | 0.1          | .01  | AD: 4.53, RB: 2.52, PRB: 2.46 Ours: 6.42 | \(L_\phi\): 150m, \(\partial_p L_\theta\): 146m |
|      |      | Index of refraction (imaginary part) | 1, 1, 1 | 3.9, 2.4, 2.2 | .02 |                                      |                        |

We conduct two inverse rendering optimizations using our method to find BRDF material properties in the Veach door scene. This scene is of particular interest because the room is mostly lit by global illumination. Maximum path length is set to 15 for the baselines. The three BRDF functions of the three teapots are rough conductor, rough plastic, and diffuse, from left to right. Please note that for optimization, we used path tracing for primal rendering across all baselines as well as in our method. Our models were only trained for the initial states in each experiment.

7.2 Inverse rendering optimization

We conduct two inverse rendering optimizations using our method to find BRDF material properties in the Veach door scene. This scene is of particular interest because the room is mostly lit by global illumination. Maximum path length is set to 15 for the baselines. The three BRDF functions of the three teapots are rough conductor, rough plastic, and diffuse, from left to right. Please note that for optimization, we used path tracing for primal rendering across all baselines as well as in our method. Our models were only trained for the initial states in each experiment.

Experiment 1. In our first experiment, we optimize one BRDF parameter per teapot. Table 1 lists the optimized parameters in this experiment, and their initial and target states. The solution of our differential network for the initial state can be found in Figure 5. From Figure 1, we can observe that our optimization method is

take an adjoint method that consumes constant memory in path length along with RB and PRB.

- **Constant time complexity**: Given pre-trained networks, our time complexity is constant in path length, since we do not build the complete path integral. Instead, we need to cheaply query our networks at the first intersection. Conversely, PRB and AD build complete light paths and are linear in path length. Compared to RB, since we have incident primal radiance cached in \(L_\phi\), our time complexity is not quadratic in path length.

- **View-independence**: Unlike all three methods above, our method provides view-independent solutions to the differential rendering equation (Figure 3). This means our solutions need not be recomputed/updated under changes of sensor parameters – except if sensor parameters are in the set of parameters that are being optimized. This property can be a key improvement for multi-view optimization tasks.

Fig. 6. Experiment 1. Optimization curves for parameter error vs. iteration count and time (results in Figure 1). Given a pre-trained network, our method achieves faster and better convergence compared to the baselines.
capable of finding near specular target states (for the left teapot), as well as the opposite direction for the middle teapot. Also, it shows that our method improves both gradient computation speed and accuracy of the final results compared to the baselines. Figure 6 plots the optimization curves for this optimization, indicating that we achieve faster and better convergence than the other methods.

Experiment 2. In the second experiment, we optimize two BRDF parameters for the left teapot, its roughness and the imaginary part of the index of refraction. The initial state in this experiment is designed such that the BRDF initial state appears similar to a diffuse material. Figure 7 shows that we can successfully achieve more accurate final results compared to the reference target state, in addition to faster convergence and speed up in gradient computation time in comparison with baselines. We refer to Table 1 for details of this experiment.

8 CONCLUSIONS AND FUTURE WORK
In this paper, we introduced a new method to solve the differential rendering equation using a single neural network. Our network parameters are optimized directly by minimizing the norm of the residual of the differential rendering equation. Our learnable network architecture is capable of representing the full continuous, view-independent differential radiance distribution and accounts for global differential illumination. A pre-trained instance of our network can be leveraged in inverse optimization problems to re-construct complex non-diffuse BRDF properties and achieve faster convergence with better time and memory complexity than the state of the art. Finally, the view-independence of our solution has the potential to produce further speedups in multi-view optimizations.

Our method has a number of limitations. First, our differential network requires an output channel for each scene parameter included in the gradient. As we use fully-connected layers, the number of connections in the last layer grows exponentially with the number of outputs and this generates a memory constraint for differentiable rendering tasks that require optimizing w.r.t millions of parameters. Additionally, our training/fine-tuning process is currently time consuming, but could be improved in many ways. For example, our residual sampling uses a uniform distribution. A more efficient implementation could use importance sampling instead. Further, we use importance sampling of emitters and BSDFs in the primal space. A more efficient method would be to importance sample incident directions according to the underlying differential emission ($\partial p_E$) and BSDF ($\partial p_f$) distribution. Such a modification would produce less noise when evaluating the RHS, which would result in smoother signals during both training and inverse optimization. Finally, we could use high-performance implementations of fully connected networks instead of standard Pytorch functionality, which can provide large speedups as shown in previous work.

REFERENCES
Dejan Azinovic, Tzu-Mao Li, Anton Kaplanyan, and Matthias Nießner. 2019. Inverse Path Tracing for Joint Material and Lighting Estimation. CoRR abs/1903.07145 (2019). arXiv:1903.07145 http://arxiv.org/abs/1903.07145

Saeed Hadadan, Shuhong Chen, and Matthias Zwicker. 2021. Neural Radiosity. ACM Trans. Graph. 40, 6, Article 236 (Dec 2021), 11 pages. https://doi.org/10.1145/3478513.3480569

James T. Kajiya. 1986. The Rendering Equation. SIGGRAPH Comput. Graph. 20, 4 (Aug 1986), 143–150. https://doi.org/10.1145/15886.15902

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

Diederik P. Kingma and Jimmy Ba. 2017. Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs.LG]

Samuli Laine, Janne Hellsten, Tero Karras, Yeongho Seol, Jaakko Lehtinen, and Timo Aila. 2020. Modular Primitives for High-Performance Differentiable Rendering. arXiv:2011.02977 [cs.GR]

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

Shichen Liu, Weikai Chen, Tianye Li, and Hao Li. 2019. Soft Rasterizer: Differentiable Rendering for Unsupervised Single-View Mesh Reconstruction. arXiv:1903.05567 [cs.CV]

Matthew M. Loper and Michael J. Black. 2014. OpenDR: An Approximate Differentiable Renderer. In Computer Vision – ECCV 2014. David Fleet, Tomas Pfajdla, Bernt Schiele, and Tinne Tuytelaars (Eds.). Springer International Publishing, Cham, 154–169.

Guillaume Loubet, Nicolas Holzschuch, and Wenzel Jakob. 2019. Reparameterizing Discontinuous Integrands for Differentiable Rendering. ACM Trans. Graph. 38, 6, Article 228 (Nov 2019), 14 pages. https://doi.org/10.1145/3355089.3356510

Thomas Müller, Brian Mcwilliams, Fabrice Rouselle, Markus Gross, and Jan Novák. 2019. Neural Importance Sampling. ACM Trans. Graph. 38, 5, Article 145 (Oct. 2019), 19 pages. https://doi.org/10.1145/3341156

Thomas Müller, Fabrice Rouselle, Alexander Keller, and Jan Novák. 2020. Neural Control Variates. ACM Trans. Graph. 39, 6, Article 243 (Nov 2020), 19 pages. https://doi.org/10.1145/3414685.3417804

Thomas Müller, Fabrice Rouselle, Jan Novák, and Alexander Keller. 2021. Real-Time Neural Radiance Caching for Path Tracing. 40, 4 (2021). https://doi.org/10.1145/3456265.3459812

Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. arXiv:2001.05989 [cs.CV]

Merlin Nimier-David, Sébastien Speierer, Benoît Ruiz, and Wenzel Jakob. 2020. Radiative Backpropagation: An Adjacent Method for Lighting-Fast Differentiable Rendering. ACM Trans. Graph. 39, 4, Article 146 (July 2020), 15 pages. https://doi.org/10.1145/3386569.3392406

Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. 2019. Mitsuba 2: A Retargetable Forward and Inverse Renderer. ACM Trans. Graph. 38, 6, Article 203 (Nov 2019), 17 pages. https://doi.org/10.1145/3355089.3356498

Felix Petersen, Amit H. Bermano, Oliver Deussen, and Daniel Cohen-Or. 2019. Pxic2Vex: Image-to-Geometry Reconstruction using a Smooth Differentiable Renderer. arXiv:1903.11149 [cs.CV]

Helge Rhodin, Nadia Robertini, Christian Richard, Hans-Peter Seidel, and Christian Theobalt. 2016. A Versatile Scene Model with Differentiable Visibility Applied to Generative Pose Estimation. CoRR abs/1602.03725 (2016). arXiv:1602.03725 http://arxiv.org/abs/1602.03725

Delio Vicini, Sébastien Speierer, and Wenzel Jakob. 2021. Path Replay Backpropagation: Differentiating Light Paths Using Constant Memory and Linear Time. ACM Trans. Graph. 40, 4, Article 108 (Jul 2021), 14 pages. https://doi.org/10.1145/3450625.3459804

Thijs Vogels, Fabrice Rouselle, Brian Mcwilliams, Gerhard Röhlin, Alex Harvill, David Adler, Mark Meyer, and Jan Novák. 2018. Denoising with Kernel Prediction and Asymmetric Loss Functions. ACM Trans. Graph. 37, 4, Article 124 (July 2018), 15 pages. https://doi.org/10.1145/3197517.3201388

Tizian Zeltner, Sébastien Speierer, Iliaan Georgiev, and Wenzel Jakob. 2021. Monte Carlo Estimators for Differential Light Transport. ACM Trans. Graph. 40, 4, Article 78 (Jul 2021), 16 pages. https://doi.org/10.1145/3450625.3459807

Quan Zheng and Matthias Zwicker. 2019. Learning to Importance Sample in Primary Sample Space. Computer Graphics Forum, 38, 2 (2019), 169–179. https://doi.org/10.1111/cgf.13628