Neural Differential Radiance Field: Learning the Differential Space Using a Neural Network

Saeed Hadadan(✉) and Matthias Zwicker

University of Maryland, College Park, USA
{saeedhd,zwicker}@cs.umd.edu

Abstract. We introduce an adjoint-based inverse rendering method using a Neural Differential Radiance Field, i.e. a neural network representation of the solution of the differential rendering equation. Inspired by neural radiosity techniques, we minimize the norm of the residual of the differential rendering equation to directly optimize our network. The network is capable of outputting continuous, view-independent gradients of the radiance field w.r.t 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 simultaneously train networks to represent radiance and differential radiance, and optimize the unknown scene parameters. Our method is not scalable to millions of scene parameters, but we propose future work directions that could make that happen in the future.

Keywords: Neural radiance fields · physically-based differentiable rendering · inverse rendering

1 Introduction

Differentiable rendering is the problem of computing derivatives of a rendering process w.r.t 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).

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. In the scope of this paper, we are only focused on a set of algorithms that use a neural network approximation of the radiance function. Such methods account for global illumination by querying a radiance cache instead of building complete Monte Carlo path integrals which could be very costly. Despite being

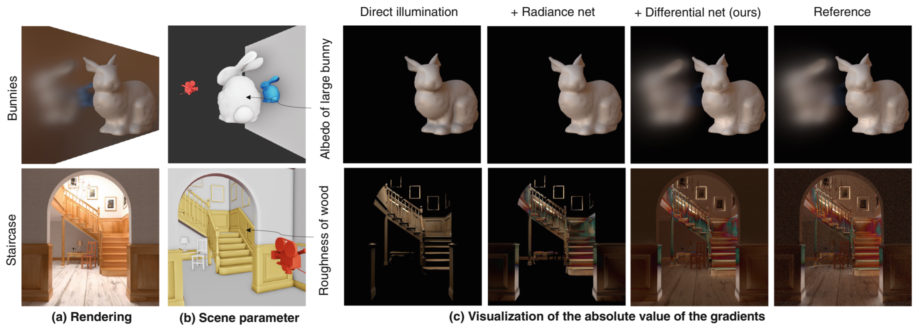


Fig. 1. Our differential radiance network can account for primary and secondary gradients. In (a) we showcase bunny scene where the reflection of bunnies in the mirror creates indirect effects, and staircase scene with significant levels of global illumination. (b) We pick a scalar non-spatially varying material parameter in each scene, and (c) visualize the gradient space for each method. The largest bias expectedly appears in the direct illumination solver; we observe improvement in gradients when we use a radiance cache on top of it. However, the inter-reflection of gradients in the scene is still totally missing. Next, our differential radiance network gives us the full differential solution, despite bouncing light only once, see Fig. 3a.

computationally simpler than path integrals, these methods tend to miss *indirect gradients* as a result of not differentiating the radiance cache w.r.t scene parameters. This could result in considerable bias in gradients (see the missing indirect effects in Fig. 1 under 'radiance net').

In this paper, we propose a novel technique to account for the missing indirect gradients. Our key idea is to represent the entire *differential radiance field*, that is, the derivative of the radiance field w.r.t a set of scene parameters, using a neural network, in addition to the radiance cache proposed by state-of-the-art. We then train this network to satisfy the *differential rendering equation* introduced by Nimier-David et al. [8]. Our method is inspired by the Neural Radiosity approach by Hadadan et al. [3] to solve the rendering equation; here we use the same training scheme to train our differential network as well. We show that we can account for indirect gradients more accurately than neural-network based methods that learn radiance caches such as [4,12]. In addition, our approach is the first that produces continuous view-independent differential radiance fields for given scene parameters, instead of only sampling derivatives for a discrete set of rays.

On the other hand, we can only optimize a limited number of parameters using our approach as the size of the output layer of our differential radiance network is equal to the number of parameters being optimized. Finally, even with the correction of indirect gradients using our differential radiance network, our gradients would not be perfectly unbiased due to the fact that they are approximated by a neural network.

2 Related Work

2.1 Neural Network Techniques for Realistic Rendering.

Neural networks can be leveraged in various ways for realistic rendering, from computational photography to medical imaging [10]. In particular, we are interested in the case where neural networks can directly represent the solution of the rendering equation as proposed in Neural Radiosity [3] and Neural Radiance Caching [7], where the radiance function is represented by neural networks as a form of a *radiance cache*.

This paper aims at leveraging neural networks to learn view-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.

2.2 Differentiable Rendering with Indirect Effects

Inverse rendering problems in computer vision and graphics heavily rely on differentiable rendering to reconstruct a set of scene parameters (geometry [1], reflectance properties, camera positions, etc.) from images. Most computer vision techniques on differentiable rendering simply ignore indirect illumination effects [5,6], but if we want to account for them, we require a full solution of the rendering equation. Techniques to differentiate the rendering equation while accounting for indirect effects have been proposed in prior work. The most naive approach would be to differentiate a path tracer using Automatic Differentiation (AD), which requires a transcription of the whole rendering process, thus suffering from prohibitive memory requirement. Instead, adjoint-based techniques take advantage of differential light properties to avoid the enormous transcript of AD. Specifically, *Radiative Backpropagation* (RB) [8] differentiates the rendering equation to obtain a *differential rendering equation*, which describes scattering and emission of *differential light*. The equation reveals that *differential light* travels through a scene similarly as regular light does. RB proposes an adjoint approach for differentiable rendering, which is more efficient than naive Automatic Differentiation of a Monte Carlo path tracer. RB's main shortcoming is that its time complexity is quadratic in path length, since at every scattering event during the adjoint phase, it requires to estimate incident radiance by building another complete light path. Although follow up works [11] have solved this issue, these algorithms still needs to build complete path integrals.

As opposed to the above methods that require building path integrals of arbitrary length, recent works such as Hadadan et al. [4] and Zhang et al. [12] propose to use a neural network to approximate the radiance function and query it during inverse rendering to account for global illumination effects. This approximation requires less computation memory and significantly less computation time than path tracing with arbitrary bounces. We will show that although this approximation can correctly account for global illumination in *primal* radiance space, it cannot account for global illumination effects in the *differential* radiance space. In other words, it misses the indirect gradients. Instead, in this paper we will propose a few alterations to the radiance-cache-based methods to correctly compute indirect gradients as well, with the help of a differential radiance network. Additionally, Hadadan et al. [4] and Zhang et al. [12] use automatic differentiation to compute the gradients of the whole process, while we propose to use an adjoint-based method. We validate our gradients compared to RB in Fig. 2.

3 Background

3.1 (Differential) Rendering Equation

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

$$I_k = \int_{\mathcal{A}} \int_{\mathcal{H}^2} W_k(x, \omega) L(x, \omega) dx d\omega_i^{\perp}, \tag{1}$$

where L is the incident radiance at location x and direction ω 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,

$$L(x, \omega_o) = E(x, \omega_o) + \int_{\mathcal{H}^2} f(x, \omega_i, \omega_o) L(x'(x, \omega_i), -\omega_i), d\omega_i^{\perp}. \tag{2}$$

Nimier-David et al. [8] differentiate the above equations w.r.t an arbitrary set of scene parameters $p = (p_1, ..., p_n)$. For simplicity, we use ∂_p to represent ∂/∂_p. Note that variables preceded by ∂_p imply a vectorized gradient w.r.t each parameter. By assuming a static camera where $\partial_p W_k = 0$, we can differentiate Eq. 1 as,

$$\partial_p I_k = \int_{\mathcal{A}} \int_{\mathcal{H}^2} W_k(x, \omega) \partial_p L(x, \omega) dx d\omega_i^{\perp}, \tag{3}$$

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

$$\begin{aligned}\partial_p L(x, \omega_o) = \partial_p E(x, \omega_o) &+ \int_{\mathcal{H}^2} f(x, \omega_i, \omega_o) \partial_p L(x'(x, \omega_i), -\omega_i) d\omega_i^{\perp} \\ &+ \int_{\mathcal{H}^2} \partial_p f(x, \omega_i, \omega_o) L(x'(x, \omega_i), -\omega_i) d\omega_i^{\perp},\end{aligned} \tag{4}$$

which is referred to as *differential rendering equation.* This equation explains the scattering of differential radiance in a similar manner to regular radiance. More

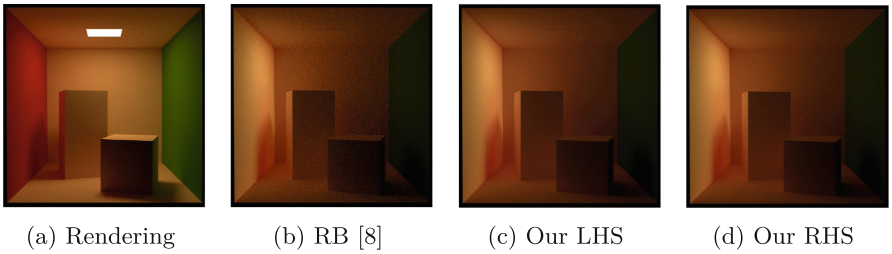


Fig. 2. Validation of our gradients compared to Radiative Backpropagation [8]. (a), (b) and (c) are the gradient images with respect to the red wall albedo.

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 differential radiance *scatters* on surfaces based on their BRDFs, similar to regular radiance in the rendering equation. The new third term represents additional differential *emission* from the surface if its BRDF function changes with perturbations of scene parameters p. This term is dependent on the incident radiance L at (x, ω_i), which implies computing $\partial_p L$ depends on computing L also.

3.2 Neural Radiosity

Neural Radiosity [3] is an algorithm to find a solution of the rendering equation (Eq. 2) using a single neural network. More formally, the radiance function $L(x, \omega_o)$ in Eq. 2 is represented by a neural network with a set of parameters ϕ (such as geometry, lighting, and material properties), as $L_\phi(x, \omega_o)$. The parameters ϕ of this network can be directly optimized in a self-training approach by minimizing the norm of the *residual* of the rendering equation. The residual $r_\phi(x, \omega_o)$ is

$$\begin{aligned} r_\phi(x, \omega_o) = L_\phi(x, \omega_o) - E(x, \omega_o) \\ - \int_{\mathcal{H}^2} f(x, \omega_i, \omega_o) L_\phi(x'(x, \omega_i), -\omega_i) d\omega_i^\perp, \end{aligned} \quad (5)$$

which is simply the difference of the left and right-hand sides of Eq. 2 when the radiance function L is substituted by L_ϕ. This neural network takes a location x and outgoing direction ω_o as input and returns the outgoing radiance. Such a pre-trained network serves as a compact, re-renderable, and view-independent solution of the rendering equation.

4 Solving the Differential Rendering Equation

Similar to Neural Radiosity, we propose to use neural network-based solvers to find the solution of the *differential rendering equation*. We call this Differentiable Neural Radiosity. Let us denote a differential radiance distribution $\partial_p L_\theta(x, \omega_o)$

as the unknown in Eq. 4, given by a set of network parameters θ. Additionally, we define a *residual* r_θ as the difference of the left and right hand side of Eq. 4,

$$\begin{aligned} r_\theta(x, \omega_o) = \partial_p L_\theta(x, \omega_o) - \partial_p E(x, \omega_o) - \int_{\mathcal{H}^2} f(x, \omega_i, \omega_o) \partial_p L_\theta(x'(x, \omega_i), -\omega_i) d\omega_i^\perp \\ - \int_{\mathcal{H}^2} \partial_p f(x, \omega_i, \omega_o) L_\phi(x'(x, \omega_i), -\omega_i) d\omega_i^\perp, \end{aligned} \tag{6}$$

where r_θ depends on the parameters θ of the differential radiance function $\partial_p L_\theta$. Also, the primal radiance can be represented by a constant parameter set ϕ in L_ϕ which is independent of θ.

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

$$\begin{aligned} \mathcal{L}(\theta) &= \|r_\theta(x, \omega_o)\|_2^2 \\ &= \int_{\mathcal{M}} \int_{\mathcal{H}^2} r_\theta(x, \omega_o)^2 dx d\omega_o, \end{aligned} \tag{7}$$

where $\mathcal{M}$ means integration over all scene surfaces. We propose to minimize $\mathcal{L}(\theta)$ using stochastic gradient descent.

4.1 Monte Carlo Estimation

The Monte Carlo estimation of the residual norm is

$$\mathcal{L}(\theta) \approx \frac{1}{N} \sum_{j=1}^{N} \frac{r_\theta(x_j, \omega_{o,j})^2}{p(x_j, \omega_{o,j})}, \tag{8}$$

where N is the number of samples, x_j and $\omega_{o,j}$ 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

$$\begin{aligned} r_\theta(x_j, \omega_{o,j}) &= \partial_p L_\theta(x_j, \omega_{o,j}) - \partial_p E(x_j, \omega_{o,j}) \\ &- \frac{1}{M} \sum_{k=1}^{M} \frac{f(x_j, \omega_{i,j,k}, \omega_{o,j}) \partial_p L_\theta(x'(x_j, \omega_{i,j,k}), -\omega_{i,j,k})}{p(\omega_{i,j,k})} \\ &- \frac{1}{Z} \sum_{l=1}^{Z} \frac{\partial_p f(x_j, \omega_{i,j,l}, \omega_{o,j}) L_\phi(x'(x_j, \omega_{i,j,l}), -\omega_{i,j,l})}{p(\omega_{i,j,l})}. \end{aligned} \tag{9}$$

The notation $\omega_{i,j,k}$ and $\omega_{i,j,l}$ indicates that each sample $x_j, \omega_{o,j}$ has its own set of samples of M and Z incident directions $\omega_{i,j,k}$ and $\omega_{i,j,l}$ (i stands for "incident", it is not an index).

5 Inverse Rendering Using Our Method

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

$$\frac{\partial z}{\partial p} = \frac{\partial z}{\partial y} . \frac{\partial y}{\partial p}, \tag{10}$$

where y is a rendered image $y = g(p)$. The term $\frac{\partial z}{\partial y}$ 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 $\frac{\partial y}{\partial p}$, which is equivalent to the differential measurement vector $[\partial_p I_0 ... \partial_p I_n]$, since we need to differentiate the rendering algorithm. Recall from Eq. 3 that $\partial_p I_k$ is the result of integrating the incident *differential radiance* $\partial_p L$ over locations x and directions ω on the hemisphere at pixel k. Therefore, the task breaks down to finding $\partial_p L(x, w)$. In our approach we query our neural network $\partial_p L_\theta(x, w)$ for the differential radiance, as it represents the entire differential radiance distribution in the scene.

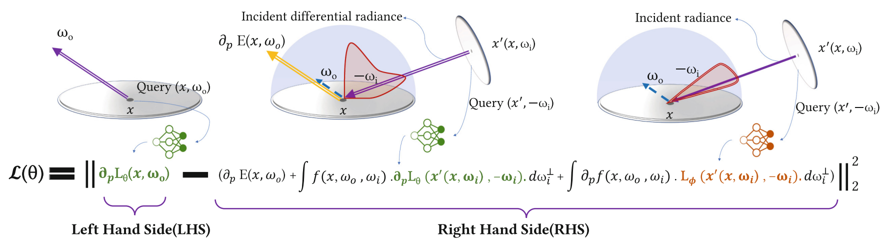
With the use of our network, inverse rendering breaks down into an iteration over the following steps:

1. Train (or fine-tune) our networks $\partial_p L_\theta$ and L_ϕ with the current state of the scene parameters (minimize Eqs. 5 and 7).
2. Compute a non-differentiable candidate primal rendering and its distance to the reference $(L2, L1$, etc.). In case of multi-view optimization, the losses are summed.
3. Find the derivatives of the loss w.r.t the pixels of the primal image (to get $\frac{\partial z}{\partial y}$).
4. Compute $\frac{\partial y}{\partial p}$, which is equivalent to the measurement vector $[\partial_p I_0 ... \partial_p I_n]$. To do so, we trace rays from the sensor to find the first hit point and at that point, query our differential radiance network $\partial_p L_\theta$. More formally,

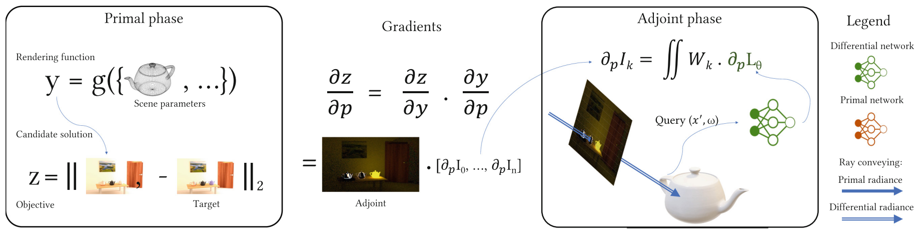
$$\partial_p I_{k,\theta} = \int_{\mathcal{A}} \int_{\mathcal{H}^2} W_k(x, \omega) \partial_p L_\theta(x'(x, \omega)) dx d\omega_i^{\perp}. \tag{11}$$

5. Compute the gradient of the loss w.r.t the parameters $\frac{\partial z}{\partial p}$ by multiplying the gradients from Step (2) and (3) as in Eq. 10.
6. Update the scene parameters using the computed gradient using an optimizer such as Adam).

Figure 3b summarizes the steps of our pipeline.


Incident differential radiance
$\partial_p \mathrm{E}(x, \omega_o)$
ω_o
$-\omega_i$
$x'(x, \omega_i)$
Query (x, ω_o)
x
Query $(x', -\omega_i)$
Incident radiance
$\mathcal{L}(\theta) = \| \partial_p \mathrm{L}_\theta(x, \omega_o) - (\partial_p \mathrm{E}(x, \omega_o) + \int f(x, \omega_o, \omega_i) . \partial_p \mathrm{L}_\theta(x'(x, \omega_i), -\omega_i) . d\omega_i^\perp + \int \partial_p f(x, \omega_o, \omega_i) . \mathrm{L}_\phi(x'(x, \omega_i), -\omega_i) . d\omega_i^\perp) \|_2^2$
Left Hand Side(LHS)
Right Hand Side(RHS)

(a) Sampling our loss function during training



(b) Our inverse rendering pipeline

Fig. 3. Pipeline schematics illustrating our training scheme and inverse optimizations steps.

6 Implementation

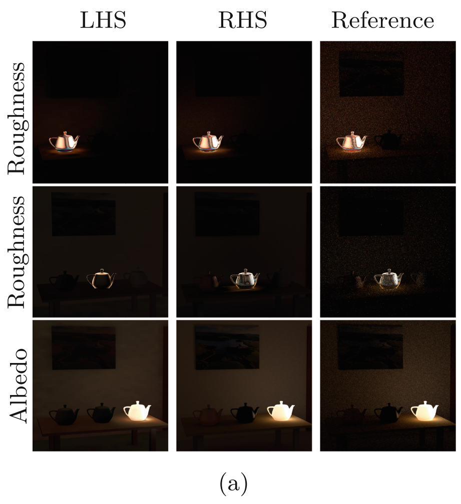
6.1 Training

The training process for our networks is end-to-end and it occurs simultaneously with the parameter optimization, i.e., during every optimization step, we take a certain number of training steps for our networks to adapt to the most recent changes in the scene parameters. In each training step, we minimize the norm of the residual of the rendering equation and the differential rendering equation simultaneously as

$$\mathcal{L}(\theta, \phi) = \|r_\phi(x, \omega_o)\|_2^2 + \|r_\theta(x, \omega_o)\|_2^2 ,$$

using a separate set of samples for location and direction in each residual term. Please note that the network L_ϕ is also present in the differential residual term r_θ (see the loss function Eq. 7). In practice, we use a $sg(.)$ to prevent our primal network from adapting itself to the differential loss.

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 (Fig. 3a).



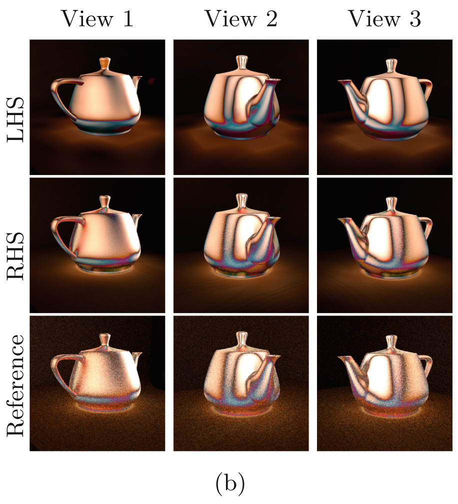


Fig. 4. (a) Separate renderings of our differential network w.r.t to the BRDF parameters of each teapot in the Fig. 3b. Our network is capable of learning non-diffuse gradients and global illumination effects in differential space. Using 8 spp for LHS and 2048 for RHS and reference (b) Multi-view renderings of our view-independent solution of the differential rendering equation compared to reference. We show the derivative with respect to the copper teapot's roughness (the first row in Fig. 4a).

6.2 Sampling and Architecture

To sample the norm of the residual, we uniformly sample locations x_i and directions $w_{o,i}$ in Eq. 8. The incident direction samples are taken using MIS of emitter and BSDF. Each of our networks is an MLP with 3 fully-connected layers with 256 neurons per hidden layer, preceded by multi-resolution sparse grid encoding of location x.

6.3 LHS Vs. RHS for Differential Radiance

As stated in Sect. 5, Step (4) of inverse rendering requires a query to $\partial_p L_\theta$ to compute gradients of pixels w.r.t. scene parameters. Equivalently, one could query our RHS to compute these gradients. Ideally, if the residual is zero everywhere, there should be no difference between the LHS and RHS; in practice, however, the is always a nonzero residual. We find that our RHS more quickly adapts to scene parameter changes, as computing the RHS requires one extra ray-tracing step using the updated parameters. Hence, we use the RHS to query the gradients in the experiments in this paper.

7 Results and Analysis

7.1 Comparison to Previous Work

In Sect. 2.2, we mentioned that automatic differentiation of multi-bounce path integrals could be memory intensive and time-consuming when dealing with complex scenes. A solution to alleviate the memory and time complexity of building path integrals is to use a radiance cache; it can provide global illumination effects while removing the need to trace further bounces. Such a radiance cache can be solely trained from input images [12], and/or using a global illumination solver [4] based on Neural Radiosity [3].

More formally, in Eq. (4), a radiance cache would provide an approximation of the term L in the second integral term which would make it more accurate than a direct illumination solver (see Fig. 1). The issue is, having a network representation of L would not yield $\partial_p L$, since the network is not differentiable with respect to the scene parameters. Therefore, using a radiance-cache-based method results in $\partial_p L = 0$. That is the motivation for our method, to account for the term $\partial_p L$ and L at the same time using separate networks; Fig. 1 shows that our method yields the least biased gradients, accounting for the global illumination effects in both primal and differential spaces, needless of tracing further bounces.

Here we provide an analysis of our method:

- **Smooth gradients:** All path integral based methods compute gradients using Monte Carlo sampling which results in noisy gradients. Instead, our networks $\partial_p L_\theta$ and L_ϕ produce smooth gradients that could enable a faster and more robust optimization process.
- **Constant time complexity:** Our time complexity is *constant* in path length, similar to other radiance-cache-based methods that avoid computing path integrals. Our method requires tracing only one bounce to compute a full global illumination solution in both differential and primal space.
- **View-independence:** Our method provides view-independent solutions to the differential rendering equation (Fig. 4b). 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 helpful for multi-view optimization tasks.
- **Memory complexity** The inference and training memory complexity of our differential network (LHS) with batch size of k samples is $O(k * n)$ where n is the number of the parameters of our networks; for the RHS with N surface samples and M samples for the hemispherical integral, the complexity would be $O(N * M * l)$. As our differential network requires an output channel for each scene parameter included in the gradient, assuming a fixed network size except the last layer, the number of network parameters l grows linearly with the number of scene parameters in gradient.

7.2 Inverse Rendering Experiments

We conduct an inverse rendering using our method to find the albedo of the small bunny in Fig. 1 from a single view; the bunny is not directly visible by the camera, but only the reflection off the camera can provide information about its albedo. Results in Fig. 5 show that radiance-cache based methods completely fail to account for indirect gradients, while our method can successfully optimize for the parameter using the differential radiance field.

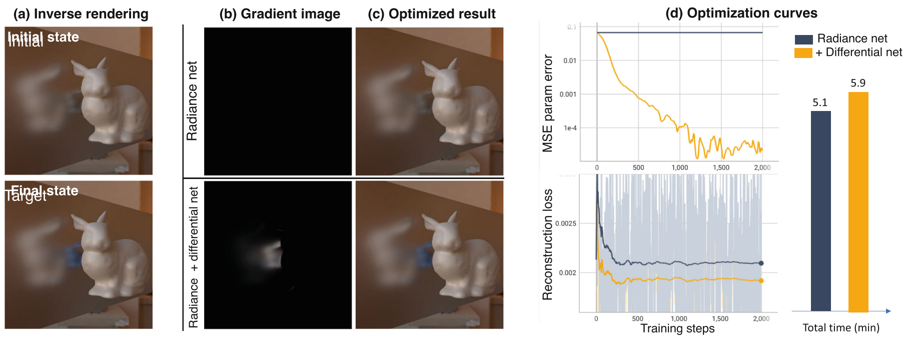


Fig. 5. Inverse rendering using our method compared to when a radiance cache was trained based on the input images similar in spirit to [4,12]. The parameter being optimized for is the albedo of the small bunny.

8 Limitation and Future Work

Our method has a key limitation: 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 linearly with the number of outputs (assuming fixed-sized network except the output layer) and this generates a memory constraint for differentiable rendering tasks that require optimizing w.r.t millions of parameters. One could use techniques such as low rank factorization [9] to reduce the output dimension of the neural network; another approach could be using hypernetworks [2] to have a neural network learn the weights of a small network $\partial_p L$ with millions of outputs.

9 Conclusion

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. Such a network can be utilized on top of a radiance-cache based method to fix the bias issue of missing indirect gradients.

Acknowledgments. We would like to thank professor Ming C. Lin from the University of Maryland, College Park for the insightful discussion, as the idea of this project formed during her Differentiable Programming course.

References

1. An, H., Lee, W., Moon, B.: Adaptively weighted discrete Laplacian for inverse rendering. Vis. Comput. **39**(8), 3211–3220 (2023). https://doi.org/10.1007/s00371-023-02955-2
2. Ha, D., Dai, A.M., Le, Q.V.: Hypernetworks. CoRR **abs/1609.09106** (2016). http://arxiv.org/abs/1609.09106
3. Hadadan, S., Chen, S., Zwicker, M.: Neural radiosity. ACM Trans. Graph. **40**(6) (2021). https://doi.org/10.1145/3478513.3480569
4. Hadadan, S., Lin, G., Novák, J., Rousselle, F., Zwicker, M.: Inverse global illumination using a neural radiometric prior (2023)
5. Li, J., Li, H.: Self-calibrating photometric stereo by neural inverse rendering. In: Avidan, S., Brostow, G., Cissé, M., Farinella, G.M., Hassner, T. (eds.) ECCV 2022. LNCS, vol. 13662, pp. 166–183. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-20086-1_10
6. Li, Z., Shen, X., Hu, Y., Zhou, X.: High-resolution SVBRDF estimation based on deep inverse rendering from two-shot images. Vis. Comput. (2022). https://doi.org/10.1007/s00371-022-02612-0
7. Müller, T., Rousselle, F., Novák, J., Keller, A.: Real-time neural radiance caching for path tracing **40**(4) (2021). https://doi.org/10.1145/3450626.3459812
8. Nimier-David, M., Speierer, S., Ruiz, B., Jakob, W.: Radiative backpropagation: an adjoint method for lightning-fast differentiable rendering. ACM Trans. Graph. **39**(4) (2020). https://doi.org/10.1145/3386569.3392406
9. Sainath, T.N., Kingsbury, B., Sindhwani, V., Arisoy, E., Ramabhadran, B.: Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 6655–6659 (2013). https://doi.org/10.1109/ICASSP.2013.6638949
10. Shetty, K., et al.: Deep learning compatible differentiable x-ray projections for inverse rendering (2021)
11. Vicini, D., Speierer, S., Jakob, W.: Path replay backpropagation: differentiating light paths using constant memory and linear time. ACM Trans. Graph. **40**(4) (2021). https://doi.org/10.1145/3450626.3459804
12. Zhang, Y., Sun, J., He, X., Fu, H., Jia, R., Zhou, X.: Modeling indirect illumination for inverse rendering. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 18643–18652 (2022)