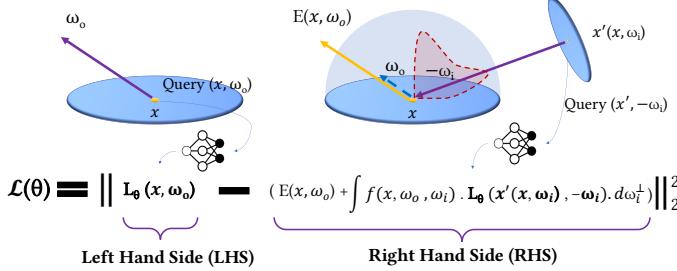
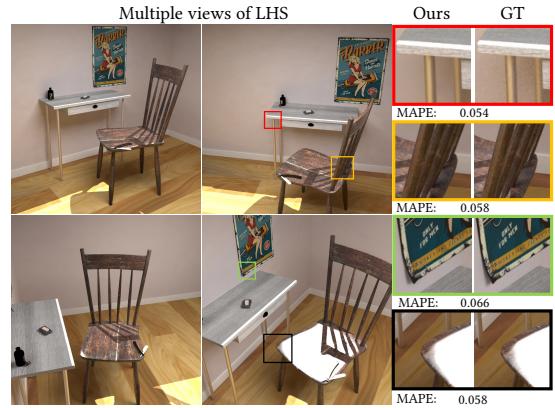


# Neural Radiosity

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



(a) Solving the rendering equation using a neural network by minimizing  $\mathcal{L}(\theta)$



(b) Multiple views of a solution of (a)

Fig. 1. (a) Neural Radiosity directly solves the rendering equation by minimizing the norm of its residual  $\mathcal{L}(\theta)$ , using a single neural network with parameters  $\theta$  to represent the radiance distribution  $L_\theta(x, \omega_o)$ . (b) After solving for the radiance  $L_\theta(x, \omega)$ , images from arbitrary viewpoints can be computed efficiently. Here, the left hand side (LHS) of the rendering equation represented by the network  $L_\theta(x, \omega_o)$  is visually indistinguishable from the ground truth (GT).

We introduce Neural Radiosity, an algorithm to solve the rendering equation by minimizing the norm of its residual, similar as in classical radiosity techniques. Traditional basis functions used in radiosity, such as piecewise polynomials or meshless basis functions are typically limited to representing isotropic scattering from diffuse surfaces. Instead, we propose to leverage neural networks to represent the full four-dimensional radiance distribution, directly optimizing network parameters to minimize the norm of the residual. Our approach decouples solving the rendering equation from rendering (perspective) images similar as in traditional radiosity techniques, and allows us to efficiently synthesize arbitrary views of a scene. In addition, we propose a network architecture using geometric learnable features that improves convergence of our solver compared to previous techniques. Our approach leads to an algorithm that is simple to implement, and we demonstrate its effectiveness on a variety of scenes with diffuse and non-diffuse surfaces.

Additional Key Words and Phrases: Neural Rendering, Neural Radiance Field

## 1 INTRODUCTION

In this paper<sup>1</sup>, we introduce a novel approach to solve the rendering equation [Kajiya 1986] using deep learning. We represent the radiance function, constituting the solution to the rendering equation, using a neural network. Our key idea is to optimize the parameters of this radiance function directly to minimize the residual of the rendering equation. This approach is conceptually similar to finite

<sup>1</sup>The published version available at <https://doi.org/10.1145/3478513.3480569>

Authors' addresses: Saeed Hadadan, University of Maryland, College Park, College Park, MD, 20740, USA, saeedhd@umd.edu; Shuhong Chen, University of Maryland, College Park, College Park, USA, shuhong@terpmail.umd.edu; Matthias Zwicker, University of Maryland, College Park, College Park, MD, USA, zwicker@cs.umd.edu.

element methods, which are known as radiosity algorithms in computer graphics. Therefore, we call our approach Neural Radiosity.

Neural networks are most commonly associated with applications in artificial intelligence, such as computer vision, natural language processing, or robotics. At the core of these applications, neural network training involves very high-dimensional optimization problems, which are addressed using gradient-based techniques with automatic differentiation (i.e., error backpropagation). Neural networks are attractive for these applications because they can represent complex high-dimensional functions with millions of parameters, while supporting effective gradient-based optimization. Yet, the powerful combination of a flexible, high-dimensional function representation and automatic differentiation has many potential applications outside of artificial intelligence. For example, neural network architectures have recently been proposed to represent radiance fields [Mildenhall et al. 2020], or to represent solutions of partial differential equations [Sitzmann et al. 2020].

Here, we propose using neural networks as a function representation to solve the rendering equation. Similarly as in radiosity techniques, we solve the rendering equation by minimizing the norm of its residual, which we achieve by optimizing over the parameters of the function representation as illustrated in Fig. 1(a). Our approach decouples solving for the radiance distribution in the scene from rendering perspective images, similar to traditional radiosity techniques. After obtaining the radiance distribution, arbitrary views can be rendered efficiently as shown in Fig. 1(b). A major challenge in traditional radiosity approaches is to represent solutions of particular scenes efficiently, by providing sufficient accuracy

while avoiding storage and computation of unnecessary degrees of freedom. This can be achieved with methods that adapt the function representation on the fly during the optimization process. Adaptive techniques, which include progressive meshing [Cohen et al. 1988; Lischinski et al. 1992], wavelet radiosity [Gortler et al. 1993], or hierarchical meshless basis functions [Lehtinen et al. 2008], are often cumbersome to implement, however. In addition, traditional radiosity approaches are typically limited to purely diffuse scenes in practice, although extensions to non-diffuse scenes have been proposed [Immel et al. 1986].

In contrast, our neural network-based approach represents the full four-dimensional radiance field and is not limited to diffuse scenes. Our approach also does not require meshing or adaptive techniques. These advantages are due to the non-linear nature of neural networks, which also implies that minimizing the residual norm as a function of the network parameters is a non-linear optimization problem. Yet this challenge can be solved robustly using mini-batch stochastic gradient descent, taking advantage of neural network frameworks that provide automatic differentiation (error backpropagation). Minimizing the norm of the residual using mini-batch stochastic gradient descent corresponds to Monte Carlo estimation of the gradient of the residual in each step. To improve convergence, we propose a network architecture including multi-resolution learnable features.

In summary, this paper makes the following contributions:

- We propose Neural Radiosity, a method to solve the rendering equation using neural networks. As in radiosity techniques, we minimize the norm of the residual of the rendering equation over the parameters of the function representation.
- We demonstrate a practical approach to solve the resulting non-linear optimization using mini-batch stochastic gradient descent. To improve convergence, we propose a network architecture that includes multi-resolution learnable features.
- We illustrate the feasibility of this approach experimentally by showing successful solutions for several scenes. In addition, we provide the source code of our approach.

## 2 RELATED WORK

*Radiosity Techniques.* The radiosity algorithm was proposed by Goral et al. [1984] to model indirect illumination in diffuse scenes. Similar to our approach, the key idea is to use a function space to represent the solution of the rendering equation; the original radiosity approach used a linear function space consisting of piecewise constant functions. Zatz [1993] generalized this approach by reformulating it using Galerkin finite element methods and introducing higher-order polynomial basis functions. Even with higher-order polynomials, however, it is challenging to represent solutions of complex scenes accurately and compactly. Hence, various techniques have been proposed to adapt the basis functions to the solution of a scene [Cohen et al. 1988; Gortler et al. 1993; Lehtinen et al. 2008; Lischinski et al. 1992]. While it is possible to extend such techniques to non-diffuse environments [Immel et al. 1986], high storage requirements make them unattractive for complex scenes. More recently, Dahm and Keller [2017] used reinforcement learning

to solve the rendering equation, which leads to a similar optimization problem as in our technique. They did not use neural networks in their implementation, however.

*Neural Representations of Radiance Fields.* Using neural networks to represent radiance fields has been popularized by Mildenhall et al.’s work on Neural Radiance Fields (NeRF) [2020]. They address the problem of novel view synthesis given a set of input photographs of a static scene. Their approach leverages a multilayer perceptron (MLP) as a regression function to interpolate the radiance samples given by the pixels of the input images. Our approach similarly represents radiance fields using a neural network, but we optimize the network parameters to minimize the norm of the residual of the rendering equation. This stands in contrast to solving a regression problem based on input samples that already contain global illumination.

*Neural Network Techniques for Realistic Rendering.* Neural network techniques have been very successful for post-processing of rendered images, including denoising, antialiasing, and super-resolution [Vogels et al. 2018]. In these techniques, neural networks operate on images, and training is typically performed in a supervised manner using a dataset of high-quality ground truth images. Neural networks have also been leveraged for importance sampling in Monte Carlo rendering [Müller et al. 2019; Zheng and Zwicker 2018]. These techniques compute sets of Monte Carlo samples as training data to learn probability densities represented by neural networks. More generally, the rendering equation is a Fredholm equation of the second kind, and previous work has proposed neural networks to solve such equations [Effati and Buzhabadi 2012].

Recently, Müller et al. [2020] proposed neural networks as control variates in Monte Carlo integration (neural control variates, or NCV). This work has similarities with our approach, although we start from a different perspective. Control variates are a well-known variance reduction technique in Monte Carlo integration. In the basic formulation, the desired integral of an original function is expressed as the known integral of a “simpler” function (the control variate) plus the Monte Carlo estimate of the difference to the original integrand. Intuitively, if the difference between the original integrand and the control variate is closer to a constant, then the variance of this estimate is lower. The application of control variates to the rendering equation requires the computation of parameterized integrals; in other words, separate integrals representing the radiance for each pixel, or the outgoing radiance for each location and direction on a surface. Müller et al. therefore formulate parameterized control variates, which are factored into a scalar approximating the integral at each surface location and outgoing direction (the integral of the CV), and a normalized shape function (the shape of the CV). They then train separate neural networks to represent the integral of the CV, the shape of the CV, a sampling density proportional to the difference between the shape of the CV and the true integrand, and an additional weight for the control variate.

In contrast, we take a perspective akin to traditional radiosity techniques. We only use a single neural network representation of the outgoing radiance, and we directly optimize the network parameters to minimize the norm of the rendering equation residual. Our approach completely decouples solving the rendering equation from rendering (perspective) images using a given radiance field, as

in classical radiosity techniques. Finally, our approach is not based on the series expansion of the rendering equation, which forms the basis of path tracing and Metropolis light transport. We do not compute any path integrals, but instead solve for the outgoing radiance field directly. Our approach is similar to NCV in that we both use neural networks to represent the outgoing radiance distribution; however, we only train one neural network with a simple residual loss. Despite the simplicity of our method, we demonstrate successful results on a variety of scenes. Concurrent work by Müller et al. [2021] can be seen as a simplified version of NCV that is more similar to our approach since it also uses a single neural network, but it is geared towards real-time rendering and does not use our approach of minimizing the residual of the rendering equation.

*Neural Network Architectures.* Neural network architecture often has a significant influence on the convergence properties of network training. For example, NeRF [Mildenhall et al. 2020] proposed a positional encoding technique to embed the spatial input parameters into a higher dimensional vector, which led to improved results in practice. Tancik et al. [Tancik et al. 2020b] analyzed this strategy using neural tangent kernels [Jacot et al. 2020], showing that passing the input parameters through a Fourier transform allows an MLP network to capture high frequency details more effectively. As an alternative, Sitzman et al. [Sitzmann et al. 2020] introduce periodic activation functions. Using a principled initialization scheme, they show that their approach can better represent high-frequency detail compared to vanilla MLPs with ReLU activations. Our approach is inspired by the Neural Geometric Level of Detail technique by Takikawa et al. [2021] and similar techniques [Chabra et al. 2020; Chibane et al. 2020; Jiang et al. 2020; Liu et al. 2020; Peng et al. 2020]. They propose MLP enhancements using a hierarchy of local learnable features, and demonstrate how this can improve the accuracy of implicit representations of 3D geometry.

### 3 PROBLEM FORMULATION

Our goal is to compute solutions of the rendering equation, which can be expressed as

$$L(x, \omega_o) = E(x, \omega_o) + \int_{\mathcal{H}^2} f(x, \omega_i, \omega_o) L(x'(x, \omega_i), -\omega_i) d\omega_i^\perp, \quad (1)$$

where  $L(x, \omega_o)$  represents the unknown radiance distribution, which is a function of 3D locations  $x$  on surfaces and outgoing directions  $\omega_o$ . Light sources or emitters are represented by the emitted radiance distribution  $E(x, \omega_o)$ , and light scattering on surfaces is described by the integral over the incident directions  $\omega_i$  on the hemisphere  $\mathcal{H}^2$  using the projected solid angle measure  $d\omega_i^\perp$ . Here,  $f(x, \omega_i, \omega_o)$  is the bidirectional reflectance distribution function (BRDF). In addition,  $L(x'(x, \omega_i), -\omega_i)$  represents the incident radiance at  $x$  from incident direction  $\omega_i$ . Under the assumption that only surface scattering is considered, this is equivalent to the outgoing radiance in the direction  $-\omega_i$  at the surface location intersected by a ray starting at  $x$  in direction  $\omega_i$ , which is denoted by  $x'(x, \omega_i)$ .

*Neural network-based radiance fields.* In our approach, the unknown solution, that is the radiance distribution  $L(x, \omega_o)$ , is represented using a neural network. The variables in our problem are the set of neural network weights and bias values denoted by  $\theta$ ,

which are nonlinearly related to the output values of the network. In addition, the network is a monolithic function representation that does not consist of locally supported basis functions. Therefore, the typical finite element approach is not applicable.

Let us denote a radiance distribution given by a set of network parameters  $L_\theta(x, \omega_o) \in \mathcal{F}$ . Here  $\mathcal{F}$  denotes the space of functions that can be represented by the chosen network architecture. In addition, the residual of the rendering equation is

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

where the notation  $r_\theta$  indicates that the residual depends on the parameters  $\theta$  of the radiance function  $L_\theta$ . In our approach, we express a suitable norm of the residual (here the squared  $L_2$  norm for example) as a nonlinear function of the network parameters  $\theta$ ,

$$\begin{aligned} \mathcal{L}(\theta) &= \|r_\theta(x, \omega_o)\|_2^2 \\ &= \int_{\mathcal{M}} \int_{\mathcal{H}^2} r_\theta(x, \omega_o)^2 dx d\omega_o, \end{aligned} \quad (3)$$

where integration is over all scene surfaces  $\mathcal{M}$  and the hemisphere  $\mathcal{H}^2$ . Our desired solution is the radiance  $L_{\theta^*}(x, \omega_o)$ , where

$$\theta^* = \arg \min_{\theta} \mathcal{L}(\theta). \quad (4)$$

We propose to minimize the residual norm  $\mathcal{L}(\theta)$  using minibatch stochastic gradient descent, where the norm of the residual is repeatedly estimated using Monte Carlo integration, and gradients of the radiance field are computed using automatic differentiation.

*Monte Carlo Estimate.* The MC estimate of the residual norm is

$$\mathcal{L}(\theta) \approx \frac{1}{N} \sum_{j=1}^N \frac{r_\theta(x_j, \omega_{o,j})^2}{p(x_j, \omega_{o,j})}, \quad (5)$$

where  $N$  is the number of samples, and samples of surface locations  $x_j$  and outgoing directions  $\omega_{o,j}$  are distributed according to probability density  $p(x, \omega)$ .

*Gradients.* The MC approximation of the gradient  $\nabla_{\theta} \mathcal{L}(\theta)$  is

$$\nabla_{\theta} \mathcal{L}(\theta) \approx \frac{1}{N} \sum_{j=1}^N \frac{2r_\theta(x_j, \omega_{o,j}) \nabla_{\theta} r_\theta(x_j, \omega_{o,j})}{p(x_j, \omega_{o,j})}, \quad (6)$$

and the gradient of the residual is

$$\begin{aligned} \nabla_{\theta} r_\theta(x, \omega_o) &= \nabla_{\theta} L_\theta(x, \omega_o) - \\ &\quad \int_{\mathcal{H}^2} f(x, \omega_i, \omega_o) \nabla_{\theta} L_\theta(x'(x, \omega_i), -\omega_i) d\omega_i^\perp, \end{aligned} \quad (7)$$

This gradient will also be approximated by evaluating the hemispherical integral using Monte Carlo integration,

$$\begin{aligned} \nabla_{\theta} r_\theta(x_j, \omega_{o,j}) &= \nabla_{\theta} L_\theta(x_j, \omega_{o,j}) \\ &\quad - \frac{1}{M} \sum_{k=1}^M \frac{f(x_j, \omega_{i,j,k}, \omega_{o,j}) \nabla_{\theta} L_\theta(x'(x_j, \omega_{i,j,k}), -\omega_{i,j,k})}{p(\omega_{i,j,k})}. \end{aligned} \quad (8)$$

Here  $\omega_{i,j,k}, k \in \{1, \dots, M\}$  is a set of  $M$  samples of incident directions for each sample  $x_j, \omega_{o,j}$  ( $i$  stands for "incident", not an index).

*Minibatch Stochastic Gradient Descent.* We minimize the residual norm using minibatch stochastic gradient descent, as described by the pseudocode in Algorithm 1. Because the network is used to evaluate both the left and right hand side of the rendering equation as shown in Equation 2, we could call this a “self-training” approach. Since the Monte Carlo gradient estimates are unbiased, this stochastic gradient descent is guaranteed to converge to a local minimum. If the network capacity is unlimited, it is guaranteed to converge to the exact solution where the residual norm vanishes. In practice, we improve convergence using adaptive momentum methods such as Adam [Kingma and Ba 2015].

**ALGORITHM 1:** Minibatch stochastic gradient descent, learning rate  $\eta$ .

---

```

initialize network parameters  $\theta$ ;
while not converged do
    sample a set of surface points  $\{x_j | j = 1 \dots N\}$  and outgoing
    directions  $\{\omega_{o,j} | j = 1 \dots N\}$ ;
    for each  $(x_j, \omega_{o,j})$ , sample a set of incident directions
     $\{\omega_{i,j,k} | k = 1 \dots M\}$ ;
    use the samples to evaluate the Monte Carlo estimate of
     $\nabla_\theta \mathcal{L}(\theta)$  using Equations 6 and 8;
     $\theta = \theta - \eta \nabla_\theta \mathcal{L}(\theta)$ ;
end
return  $\theta$ ;

```

---

*Image Synthesis.* The solution  $L_{\theta^*}(x, \omega_o)$  obtained from (4) represents the radiance field over the surfaces of the entire 3D scene. This allows us to synthesize 2D images rapidly using perspective projection or ray tracing, and by evaluating the left hand side (LHS) of the rendering equation  $L_{\theta^*}(x, \omega_o)$  for each surface location  $x$  corresponding to an image pixel, and the direction  $\omega_o$  that points from  $x$  to the center of projection (the virtual camera location). Alternatively, we can render images by evaluating the right hand side (RHS) using Monte Carlo integration of the hemispherical integral. A comparison video of LHS/RHS and path-traced renderings is presented as supplemental material.

*Evaluation Metric.* Throughout this paper, we use Mean Squared Error (MSE) and Mean Absolute Percentage Error (MAPE), defined as  $|\text{img} - \text{ref}| / (\text{ref} + 0.01)$ , to compare image-based error with respect to the path-traced ground truth.

## 4 IMPLEMENTATION

In this section we discuss several design choices of our implementation that are important to achieve accurate results in practice.

### 4.1 Relative Residual

The highly dynamic range of radiance values can introduce numerical instability to the training process, in particular with the  $L_2$  norm described in Equation 3. As observed in previous work [Müller et al. 2020], normalizing the loss values leads to a more stable training process, and facilitates learning darker parts of the scene. Thus, we

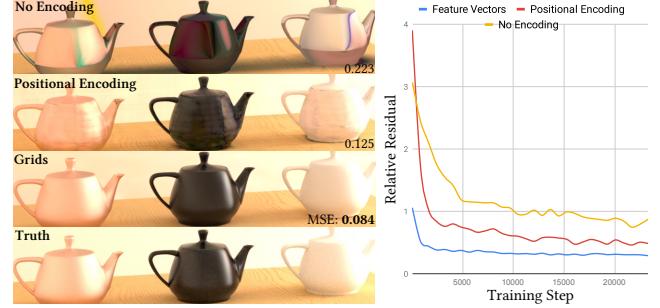


Fig. 2. Comparison of LHS renderings of *Veach Door* with different encoding techniques. For large scenes, the positional encoding approach produces artifacts in regions with more detailed geometry. LHS renderings are most faithful to the ground truth when multi-resolution feature grid vectors are used. From the training curves, we observe that multi-resolution learnable feature grids show superior convergence and training stability.

modify our loss function in Equation 3 to

$$\mathcal{L}(\theta) = \left\| \frac{r_\theta(x, \omega_o)}{sg(m_\theta(x, \omega_o)) + \epsilon} \right\|^2 \quad (9)$$

$$m_\theta(x, \omega_o) = \frac{L_\theta(x, \omega_o) + E(x, \omega_o) + T\{L_\theta\}(x, \omega_o)}{2} \quad (10)$$

$$T\{L_\theta\}(x, \omega_o) = \int_{\mathcal{H}^2} f(x, \omega_i, \omega_o) L_\theta(x'(x, \omega_i), -\omega_i) d\omega_i^\perp, \quad (11)$$

where  $\epsilon$  is a constant. The stop-gradient  $sg(\cdot)$  operator excludes the mean  $m_\theta$  from contributing to the gradient during optimization. We compare additional normalizations in our supplemental material.

### 4.2 Emission Reparameterization

The outgoing radiance in Equation 1 consists of the sum of the emitted and scattered radiance. In our implementation, we treat the emittance as a term given by the scene, and only learn to approximate the scattered radiance term. Hence we bypass artifacts that arise when trying to model high-contrast discontinuities at the boundaries of light sources. More formally, we reparameterize as follows:

$$L_\theta(x, \omega_o) = N_\theta(x, \omega_o) + E(x, \omega_o), \quad (12)$$

$$r_\theta(x, \omega_o) = L_\theta(x, \omega_o) - E(x, \omega_o) - T\{L_\theta\}(x, \omega_o) \quad (13)$$

$$= N_\theta(x, \omega_o) - T\{N_\theta + E\}(x, \omega_o), \quad (14)$$

where  $T\{L_\theta\}$  is the scattered radiance as defined in Eq. 11, and  $N_\theta(x, \omega_o)$  is now the network being optimized.

### 4.3 Multi-resolution Feature Grid

Our learned radiance field function  $L_\theta$  takes in a 3D point position  $x \in \mathbb{R}^3$  and direction  $\omega_o \in \mathcal{H}^2$ . A naive implementation may be a simple multi-layer perceptron with a 5D input  $[x, \omega_o]$ . However, it has been observed that the input encoding may play a large role in determining model performance. A popular mapping in previous work [Mildenhall et al. 2020] is sinusoidal positional encoding.

Instead, we saw significant performance gains from encoding the input using learned multi-resolution feature vectors, which has been observed similarly in other applications [Chabra et al. 2020;

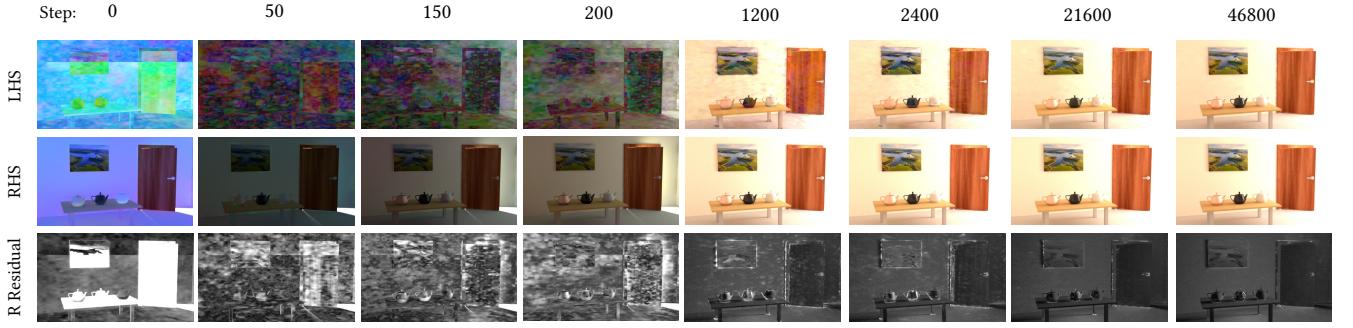


Fig. 3. Training milestones for *Veach Door*. RHS, LHS, and the relative residual for certain training steps are visualized. As training proceeds, the LHS gradually learns the radiance distribution, and the light coming from the room behind the door propagates to the RHS renderings. The residual visualisations show the relative residual. We use batch size  $N = 8192$ ,  $M = 32$ , and 48K steps  $S$ . Thus, the total number of samples is  $N \times M \times S = 12,580$  million.

Chibane et al. 2020; Jiang et al. 2020; Liu et al. 2020; Peng et al. 2020; Takikawa et al. 2021]. Specifically, we define  $n$  regular 3D lattice grids containing the entire scene, each at a different resolution. At each point on each lattice, we initialize a feature vector of length  $l$ . To query the model at a given point, we trilinearly interpolate the features stored at the enclosing voxel’s eight corners. This is done for each lattice, and the resultant interpolations from each level of detail are averaged into a final feature vector. We finally feed this embedding to the downstream network in addition to the raw position coordinates. More formally,

$$L_\theta(x, \omega_o) = \text{MLP} \left( \begin{bmatrix} x \\ G(x) \\ \omega_o \end{bmatrix} \right), \quad G(x) = \frac{1}{n} \sum_{i=0}^{n-1} \text{trilinear}(x, V_i[x]), \quad (15)$$

where  $G : \mathbb{R}^3 \rightarrow \mathbb{R}^l$  is the multi-resolution grid embedding function, and  $V_i[x]$  represents the eight  $l$ -dimensional features at the corners of the voxel enclosing  $x$  on lattice  $i$ . Since the *trilinear* interpolation is differentiable, we may easily train the features. We compare this approach to positional encoding [Mildenhall et al. 2020] in Fig. 2.

#### 4.4 Scene Information as Additional Input

The BSDF  $f(x, \omega_i, \omega_o)$  plays a critical role when integrating for the residual loss; it is directly multiplied with incoming radiances  $L_\theta(x'(x, \omega_i), -\omega_i)$  sampled from the model in training. However, for complex materials such as micro-facets, the BSDF may be highly non-linear in  $(x, \omega_o)$ , significantly destabilizing training. To mitigate the effects of BSDF behavior on training, we follow practices in previous work [Müller et al. 2020; Ren et al. 2013], providing the network with given scene information such as surface normal, diffuse reflectance, and specular reflectance in addition to the position and angle  $(x, \omega_o)$ . Given this extra information, the network is successful at dealing with various materials with complex BSDFs. We report the network inputs in detail in our supplemental material.

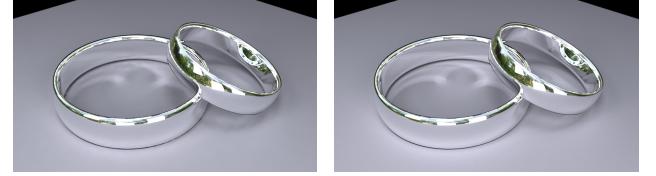


Fig. 4. *Rings* scene with caustics

#### 4.5 Specular BSDFs and Caustics

Perfect mirror BSDFs follow a delta-dirac distribution, which is not square-integrable [Müller et al. 2020] and cannot be learned effectively by the model. Instead, we trace each ray through (potentially many) specular reflections until it hits a non-specular surface where we can query radiance from our model. Since this removes the need to evaluate our model on specular surfaces, we do not sample points on specular surfaces during training. Fig. 4 shows that we can handle caustics formed by specular surfaces, and we demonstrate our model trained on the *Chair* scene with a specular chair in Fig. 12.

#### 4.6 Training

Figs. 3 and 5 illustrate the optimization process for the *Veach Door* scene. We use the Adam optimizer [Kingma and Ba 2015] with a learning rate of  $5e-4$ , decayed by a factor of 0.33 at every one-third of the training process. The experiments in this paper use from  $n = 5$  to  $n = 9$  resolution levels of feature grids, the first level always being  $2 \times 2 \times 2$ . Empirically, we found that model performance is stable with feature dimension  $l > 8$ , and so we fix  $l = 16$  in all our experiments.

#### 4.7 Sampling

Our sampling approach is different for the left hand and right hand sides. For the left hand side, the samples  $(x_j, \omega_{o,j})$  in Equation 8 are drawn from a uniform distribution. More specifically,  $x_j$  is sampled from a uniform distribution over all the surfaces in the scene;  $\omega_{o,j}$

Table 1. Training costs for the examples in Fig. 12. In practice, training times can be reduced significantly at the cost of small decreases in accuracy. \*The large number of objects in the *Bedroom* scene causes Mitsuba 2 [Nimier-David et al. 2019] to generate large CUDA kernels that require offloading of GPU registers to memory. This leads to slower training and memory leakage in the current implementation of Mitsuba 2. Hence we needed to use a smaller batch size, but still ran out of memory periodically due to memory leakage. We worked around this problem by resuming training three times.

Scene	$N$ (Batch Size)	$M$	Grid Resolution	$S$ (Training Steps)	$N \times M \times S$ (Total # of Samples)	Training Time	Peak Memory GiB
Living Room	16384	32	64	38 K	20128 M	5 h 22 m	9.43
Bedroom	8192	32	64	72 K	18880 M	7 h 1 m	10.69*
Dining Room	16384	32	512	36 K	18880 M	5h 23m	10.58
Veach Door	16384	32	128	24 K	12576 M	3h 1m	8.7
Chair	16384	32	64	24 K	12576 M	1h 38m	7.20
Mirror Chair	16384	32	64	22 K	11744 M	1h 47m	7.46
Rings	16384	32	128	40 K	20960 M	4h 50m	10.43
Copper Man - Rest	16384	32	32	8 k	4192 M	52m	10.10
Cornell Box	16384	32	32	4 K	2080 M	22 m	9.54

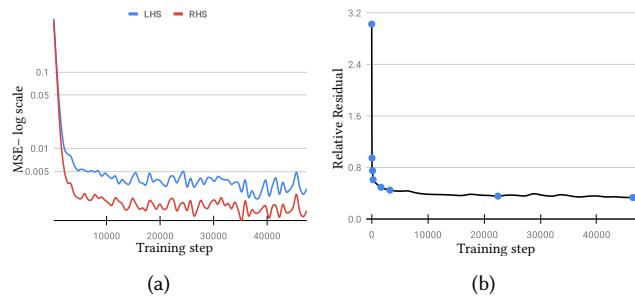


Fig. 5. (a) Image-based MSE of LHS/RHS compared to ground truth of the *Veach Door* scene. Note that the vertical axis is log-scale to show the two curves distinctively. As the chart illustrates, the RHS is always more faithful to the ground truth compared to the LHS. This is because RHS involves one more step of ray tracing that integrates multiple network outputs, as opposed to the LHS which computes one network output per sample. (b) Relative residual loss curve during training. The milestones marked on the line chart correspond to the training steps visualized in Fig. 3.

is sampled uniformly either on the local unit hemisphere (for one-sided BSDFs) or on the local unit sphere (for two-sided BSDFs).

To sample the transport operator in Equation 14, the directions  $\omega_{i,j,k}$  are sampled using BSDF and emitter sampling. Note that sampling  $\omega_{i,j,k}$  is repeated  $M$  times and averaged to achieve less noisy gradients.

#### 4.8 Architecture

The neural network used in our experiments consists of 6 linear layers of width 512 with ReLU activation functions. Thanks to the representational power of multi-level feature grids, we found that the MLP prediction head still delivered satisfying performance despite being relatively shallow. The model parameter initialization that worked best in our experiments was a uniform  $[-1, 1]$  initialization scheme, scaled by inverse root of weight size (default PyTorch [Paszke et al. 2017] linear layer initialization). Our network inputs are discussed in detail in our supplemental document.

#### 4.9 Renderer

Neural Radiosity requires calling several rendering methods during training. In particular, evaluating the right hand side of the rendering equation residual requires one ray-tracing step per sampled point to gather the predicted scattered emittance at intersected points; i.e. to calculate  $T\{L_\theta\}$ , we must use the model to predict many  $L_\theta(x'(x, \omega_i), -\omega_i)$ . We display RHS renderings for several scenes in Fig. 12, along with the relative residual loss objective. The left hand side is simpler to calculate, as we only need to feed sampled points to  $L_\theta$  once. Due to our reparameterization of emittance, both RHS and LHS evaluations require retrieving emitted radiance from the given scene properties.

We use Mitsuba 2 [Nimier-David et al. 2019] as our rendering engine, and implement the deep learning components with PyTorch [Paszke et al. 2017]. Mitsuba 2 provides Python bindings that facilitate communication between C++ data structures and PyTorch tensors. Both the rendering and training occur on a single GPU (Nvidia RTX 2080).

## 5 RESULTS AND ANALYSIS

### 5.1 Training and Rendering Results

We demonstrate rendering results for several scenes in Fig. 12, including diffuse, rough specular, plastic, and perfect mirror materials, and report training costs for all scenes in Table 1. As our model represents the entire radiance distribution, we may efficiently perform multi-view renderings of static scenes at inference time. Given a novel view, rendering the LHS consists of simply finding intersections of all rays from the camera pixels, and passing them to our learned neural network. Since our neural architecture is rather shallow, the cost of querying the network is small, and is already efficiently parallelized for GPUs by mini-batching. Multiple views of the chair scene are shown in Fig. 1b.

Fig. 8 compares LHS/RHS rendering based on a trained model using our method against path-traced results on the *Veach Door* scene given the same amount of render time. As expected, LHS rendering completes relatively quickly; we only need to ray-trace camera rays with few samples per pixel. On the other hand, path-tracing incurs the typical Monte Carlo noise artifacts. In equal rendering time,

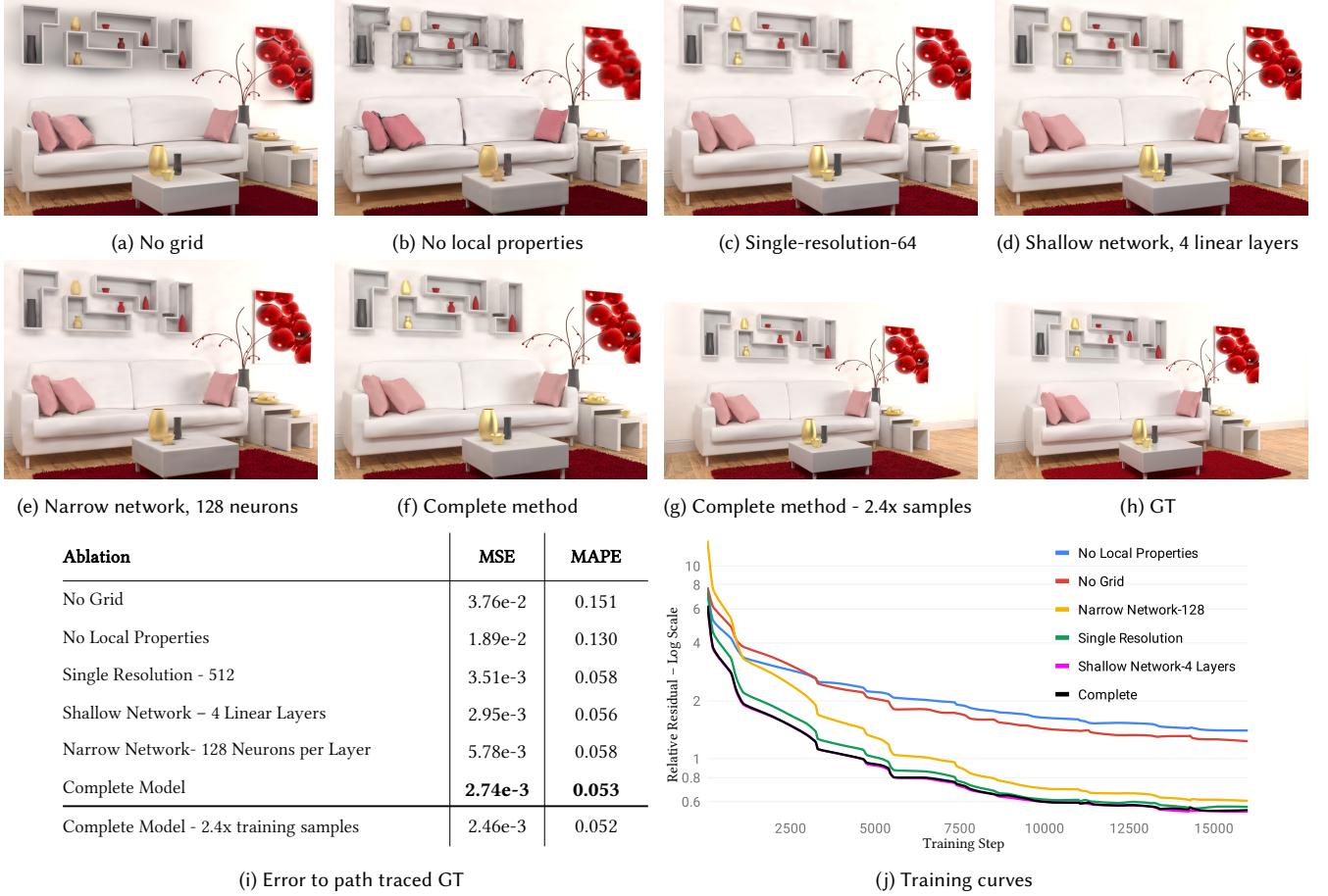


Fig. 6. Ablation Study. All renderings show the LHS. The complete model uses multi-resolution of 64, and an MLP with 6 layers and 512 width. We use a batch size of  $N = 16,384$ ,  $M = 32$ , and 16K steps of training for a total of 8,384M samples for all the experiments (except the one mentioned that has 2.4x more samples). The complete models are the ones with the best convergence and least error.



Fig. 7. Comparing our approach (a) to fitting the model to a noisy estimate of scattered radiance (b), where  $M$  is the number of paths cast per position sample to compute the scattering integral. The noisy estimate uses an equal number of samples  $M = 32$  as ours.

our LHS with only 8 spp produces images with much lower error than path-tracing with 16 spp. Similarly, our RHS with one additional ray-tracing step also produces better quality renderings than path-tracing with equal time and spp. In summary, our approach is competitive with path tracing for novel view synthesis using a trained model, but training takes on the order of minutes or hours.

## 5.2 Ablation Study

We conducted an ablation study on various system components to evaluate their effectiveness, and we show results using the *Living Room* scene (Fig. 6). We trained our model without feature grids, local property inputs, or multi-resolution grids, and also adjusted network width and depth. The study shows how every component contributes to a better LHS rendering with lower error and faster convergence. As the ablations' differences are less visible with RHS renderings, we only show LHS in the figure. Grids play an important role in learning shadows, and in general any high frequency changes in radiance along the geometry. If no local properties are inputted to the network, the textures are blurry, the glossy reflections are not learned properly, and sharp edges/corners (large surface normals changes) are blurred. We see that a narrower network with width 128 instead of 512 is insufficient to model the complexity of reflections, and converges much more slowly. The table also shows that our architecture is less affected by network depth, with our complete 6-layer model performing marginally better than a 4-layer network.



(a) Equal LHS rendering time of 14 seconds.

(b) Equal RHS ( $M = 16$ ) rendering time of 3.5 minutes

Fig. 8. Equal rendering time comparison between our pre-trained model and path-tracing. The training time for the model took about 3 hours (Table 1). As an implementation detail, note that the rendering time depends on parameters such as the GPU thread block size. To use larger spp, we must accommodate GPU memory availability by dividing the computation into smaller blocks. Thus, 128 spp path-tracing is not exactly 8 times as costly as 16 spp.

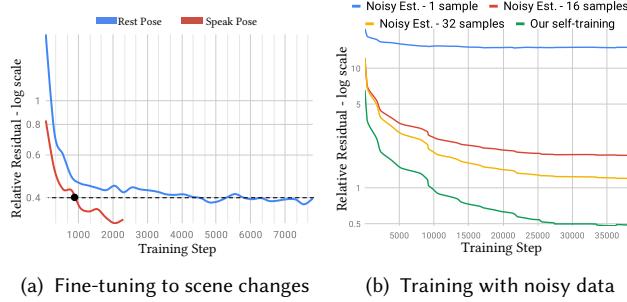


Fig. 9. (a) Training and fine-tuning the *Copper Man* scene. Our model trained on the *Rest* pose can be rapidly fine-tuned to the scene changes in *Speak*. (b) Comparing our self-training to fitting to noisy estimates. Our method ( $M = 32$ ) converges better than training on an equal number of path-traced estimates. Increasing  $M$  in the noisy estimate brings the training curve closer to ours (we cannot increase  $M$  above 32 due to memory limitations).

### 5.3 Dynamic Scenes

Neural Radiosity can be extended to dynamic scenes, in which object movements cause changes in the radiance distribution. Instead of retraining the entire network from scratch at every new scene configuration, we use transfer learning to fine-tune weights of one scene to those of another. We demonstrate this in Figs. 9a and 10, where a *Copper Man* is reposed from a *Rest* into a *Speak* pose. While directly reusing the pretrained *Rest* pose network on the modified pose performs poorly (Fig. 10) as expected, the network is able to quickly adapt after only a few steps of fine-tuning (Fig. 9a).

### 5.4 Training with Noisy Unbiased Data

As an alternative to optimization by “self-training” (using the model to evaluate the RHS), we compare results to training on noisy unbiased estimates of the RHS. That is, we compare a loss calculated with respect to path-traced samples of the RHS (“noisy ground truth”)

to our loss, which queries our model. In Fig. 7, we show an LHS rendering of our method self-trained with  $M = 32$  model queries per sample, next to LHS renderings of our model trained with path-traced samples of varying noise levels (including  $M = 32$  for direct comparison). As we see from Fig. 7 and the training curves in Fig. 9b, self-training outperforms noisy estimate training. The path-traced estimates suffer from a fixed amount of noise that destabilizes training, whereas radiance estimates queried from self-training improve across iterations, leading to better convergence.

### 5.5 Study of Multi-resolution Feature Grids

We study the effect of multi-resolution feature grids on convergence and rendering quality in Fig. 11. We use the *Dining Room* scene with various levels of single- and multi-resolution grids. Fig. 11 shows that increasing the multi-resolution level reduces error, and that the absence of multi-resolution grids causes dotty artifacts. The training curves in Fig. 11 further quantify the superior convergence and accuracy using multi-resolution. We use an efficient sparse grid implementation with quadratic space complexity by only storing those grid vertices that contribute to the trilinear interpolation at surface samples. We report memory requirements in Table 2.

### 5.6 Limitations and Future Work

A main limitation of our current implementation is that the training times are typically not competitive with state of the art Monte Carlo integration techniques. However, we can amortize initial training time when rendering multiple images in animations. A promising direction of future work is improved adaptation for dynamic scenes. Our *Copper Man* demonstration (Fig. 9a) constitutes a form of transfer learning, whereby we transfer learned weights from a pretrained network to train on a novel domain, in this case a different scene. Other methods of handling dynamic scenes include meta-learning, in which we would optimize a model initialization that can quickly adapt to any general scene; similar meta-learning formulations have

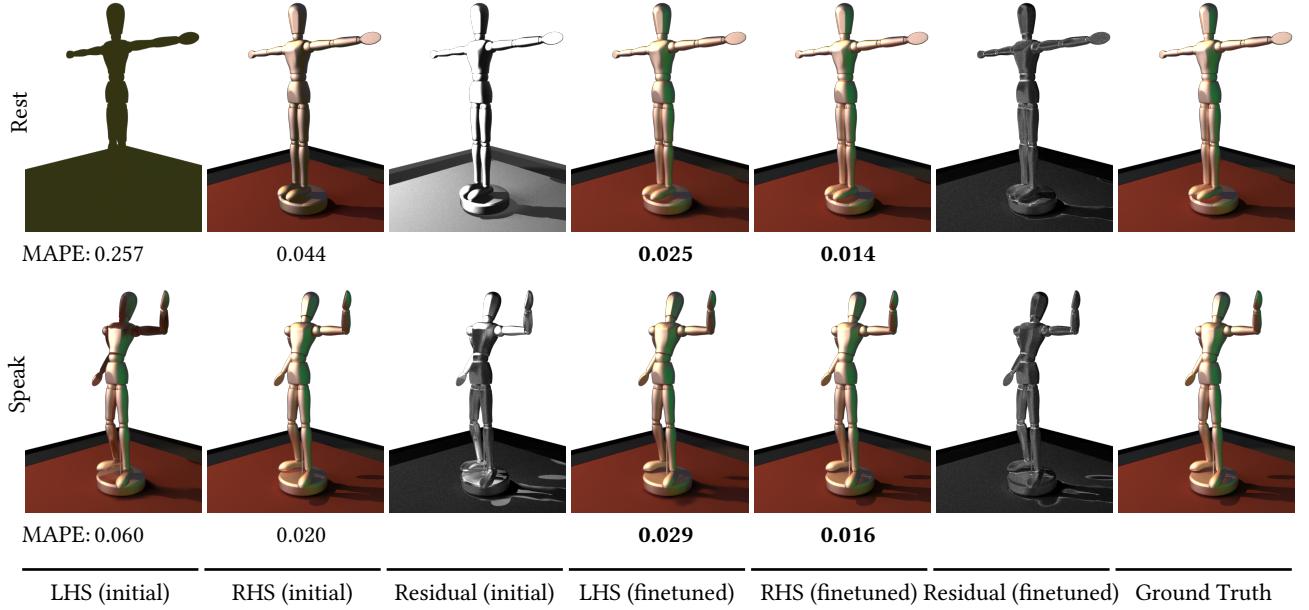


Fig. 10. Dynamic scenes with fine-tuning. The Rest Pose renderings are generated by standard training of our Neural Radiosity solver. We then use the trained neural network weights as the initialization for solving the Speak pose. The network is able to very quickly adapt to the new scene configuration through fine-tuning. See also Figure 9a for a comparison of training convergence.

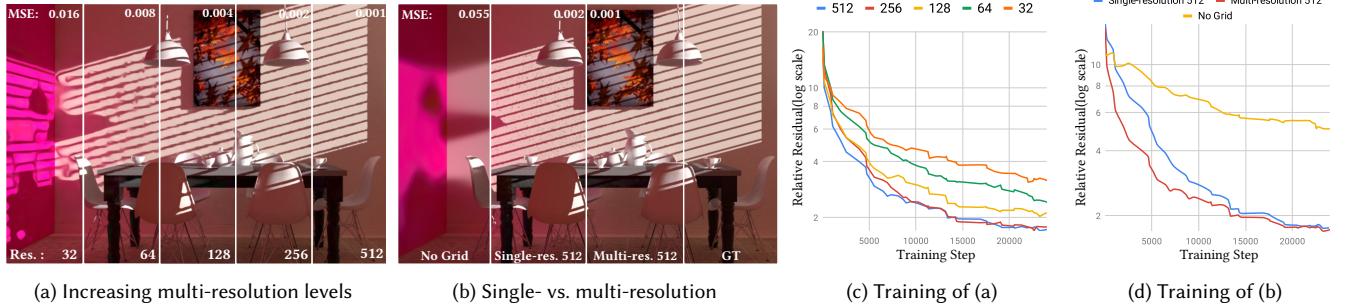


Fig. 11. Impact of grid resolution and our multi-resolution approach on LHS renderings. (a) Our multi-resolution approach leads to high quality results. (b) Omitting our feature grid produces blurry artifacts. Using a single-resolution grid results in dotty renderings and higher MSE. Our approach is most faithful to the ground truth. (c,d) Faster and better convergence occurs when using multi-resolution with higher levels.

already been proposed [Tancik et al. 2020a]. Other directions include few-shot learning, which would entail fine-tuning to a novel scene configuration given a very limited sample budget; in the context of our problem, we may design specialized sampling methods targeted at regions with changed geometry. For real-time rendering applications, we envision systems where the neural network training process to solve the rendering equation runs continuously and concurrently with a real-time image synthesis process.

Additionally, Neural Radiosity could be coupled with an adaptive sampling approach for faster convergence while maintaining the unbiased nature of the gradient Monte Carlo estimates. More future work may consist of network designs to accommodate scenes containing participating media, and improvements to positional encoding techniques such as multi-level feature octrees.

## 6 CONCLUSIONS

We propose Neural Radiosity, a novel method leveraging deep neural networks to solve the rendering equation. Inspired by traditional radiosity techniques, our approach formulates the full radiance distribution as a learnable network architecture, and is optimized by minimizing the norm of the rendering equation residual. By taking advantage of the representational capacity of neural models, we are able to solve for global illumination on scenes with complex lighting and non-diffuse surfaces. Our implementation incorporates key architectural design decisions, such as multi-resolution feature grids, into an accurate and compact scene representation. Finally, we demonstrate that our system is capable of efficient multi-view rendering, and can be extended to handle complex dynamic scenes.

**Table 2.** Space and training time costs for the resolution experiments in Fig. 11. The voxel count and density columns indicate the number and percentage of allocated grid vertices, and the feature size column shows the storage needed for each resolution level. Training time for a given resolution in each row involves all levels up to that row. That is, for 512 multi-resolution training, we train using all the levels from 4 to 512, and we provide the total feature vector storage in the last row. The training runs use 12,576M samples.

Res.	Voxel Count	Density %	Features Size	Train Time
32	3045	9.3	190 KB	2h 52m
64	11 K	4.4	719 KB	3h 17m
128	51 K	2.4	3.11 MB	2h 59m
256	214 K	1.3	13.1 MB	3h 8m
512	863 K	0.64	52.7 MB	3h 11m
Total			<b>69.9 MB</b>	

## ACKNOWLEDGMENTS

We would like to thank Wenzel Jakob for insightful initial discussions and providing the Mitsuba 2 system. We also thank Sébastien Speierer for helping us troubleshoot and modify Mitsuba 2. Baptiste Nicolet’s script to convert blender files to Mitsuba is highly appreciated. Finally, we want to thank the scene authors Benedikt Bitterli [2016] and Blendswap user “barcin” [2020].

## REFERENCES

- Barcin. 2020. At a Barber Scene. <https://blendswap.com/blend/25730>.
- Benedikt Bitterli. 2016. Rendering resources. <https://benedikt-bitterli.me/resources/>.
- Rohan Chabra, Jan Eric Lenssen, Eddy Ilg, Tanner Schmidt, Julian Straub, Steven Lovegrove, and Richard Newcombe. 2020. Deep Local Shapes: Learning Local SDF Priors for Detailed 3D Reconstruction. arXiv:2003.10983 [cs.CV]
- Julian Chibane, Thimo Alldieck, and Gerard Pons-Moll. 2020. Implicit Functions in Feature Space for 3D Shape Reconstruction and Completion. arXiv:2003.01456 [cs.CV]
- Michael F. Cohen, Shenchang Eric Chen, John R. Wallace, and Donald P. Greenberg. 1988. A Progressive Refinement Approach to Fast Radiosity Image Generation. *SIGGRAPH Comput. Graph.* 22, 4 (June 1988), 75–84. <https://doi.org/10.1145/378456.378487>
- Ken Dahm and Alexander Keller. 2017. Learning Light Transport the Reinforced Way. arXiv:1701.07403 [cs.LG]
- Sohrab Eftati and Reza Buzhabadi. 2012. A neural network approach for solving Fredholm integral equations of the second kind. *Neural Comput. & Appl.* 21 (07 2012), 1–10. <https://doi.org/10.1007/s00521-010-0489-y>
- Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battail. 1984. Modeling the Interaction of Light between Diffuse Surfaces. *SIGGRAPH Comput. Graph.* 18, 3 (Jan. 1984), 213–222. <https://doi.org/10.1145/964965.808601>
- Steven J. Gortler, Peter Schröder, Michael F. Cohen, and Pat Hanrahan. 1993. Wavelet Radiosity. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques* (Anaheim, CA) (SIGGRAPH ’93). Association for Computing Machinery, New York, NY, USA, 221–230. <https://doi.org/10.1145/166117.166146>
- David S. Immel, Michael F. Cohen, and Donald P. Greenberg. 1986. A Radiosity Method for Non-Diffuse Environments. *SIGGRAPH Comput. Graph.* 20, 4 (Aug. 1986), 133–142. <https://doi.org/10.1145/15886.15901>
- Arthur Jacot, Franck Gabriel, and Clément Hongler. 2020. Neural Tangent Kernel: Convergence and Generalization in Neural Networks. arXiv:1806.07572 [cs.LG]
- Chiyu Max Jiang, Avneesh Sud, Ameesh Makadia, Jingwei Huang, Matthias Nießner, and Thomas Funkhouser. 2020. Local Implicit Grid Representations for 3D Scenes. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
- James T. Kajiya. 1986. The Rendering Equation. *SIGGRAPH Comput. Graph.* 20, 4 (Aug. 1986), 143–150. <https://doi.org/10.1145/15886.15902>
- Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Yoshua Bengio and Yann LeCun (Eds.). <http://arxiv.org/abs/1412.6980>
- Jaakko Lehtinen, Matthias Zwicker, Emmanuel Turquin, Janne Kontkanen, Frédéric Durand, François X. Sillion, and Timo Aila. 2008. A Meshless Hierarchical Representation for Light Transport. *ACM Trans. Graph.* 27, 3 (Aug. 2008), 1–9. <https://doi.org/10.1145/1360612.1360636>
- Dani Lischinski, Filippo Tampieri, and Donald P. Greenberg. 1992. Discontinuity Meshing for Accurate Radiosity. *IEEE Comput. Graph. Appl.* 12, 6 (Nov. 1992), 25–39. <https://doi.org/10.1109/38.163622>
- Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. 2020. Neural Sparse Voxel Fields. *NeurIPS* (2020).
- Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2020. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *ECCV*.
- Thomas Müller, Brian McWilliams, Fabrice Rousselle, 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 Rousselle, 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 Rousselle, Jan Novák, and Alexander Keller. 2021. Real-Time Neural Radiance Caching for Path Tracing. 40, 4 (2021). <https://doi.org/10.1145/3450626.3459812>
- Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. 2019. Mitsuba 2: A Retargetable Forward and Inverse Renderer. *Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 38, 6 (Dec. 2019). <https://doi.org/10.1145/3355089.3356498>
- 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).
- Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. 2020. Convolutional Occupancy Networks. In *European Conference on Computer Vision (ECCV)*.
- Peiran Ren, Jiaping Wang, Minmin Gong, Stephen Lin, Xin Tong, and Baining Guo. 2013. Global Illumination with Radiance Regression Functions. *ACM Trans. Graph.* 32, 4, Article 130 (July 2013), 12 pages. <https://doi.org/10.1145/2461912.2462009>
- Vincent Sitzmann, Julien N.P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. 2020. Implicit Neural Representations with Periodic Activation Functions. In *Proc. NeurIPS*.
- Towaki Takikawa, Joey Lititalen, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. 2021. Neural Geometric Level of Detail: Real-time Rendering with Implicit 3D Shapes. *CoRR* abs/2101.10994 (2021). arXiv:2101.10994 <https://arxiv.org/abs/2101.10994>
- Matthew Tancik, Ben Mildenhall, Terrance Wang, Dovi Schmidt, Pratul P. Srinivasan, Jonathan T. Barron, and Ren Ng. 2020a. Learned Initializations for Optimizing Coordinate-Based Neural Representations. *arXiv preprint arXiv:2012.02189* (2020).
- Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nitin Raghavan, Utkarsh Singh, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. 2020b. Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains. arXiv:2006.10739 [cs.CV]
- Thijs Vogels, Fabrice Rousselle, Brian McWilliams, Gerhard Röthlin, 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>
- Harold R. Zatz. 1993. Galerkin Radiosity: A Higher Order Solution Method for Global Illumination. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques* (Anaheim, CA) (SIGGRAPH ’93). Association for Computing Machinery, New York, NY, USA, 213–220. <https://doi.org/10.1145/166117.166145>
- Quan Zheng and Matthias Zwicker. 2018. Learning to Importance Sample in Primary Sample Space. *CoRR* abs/1808.07840 (2018). arXiv:1808.07840 <http://arxiv.org/abs/1808.07840>

## A SUPPLEMENTARY MATERIALS

### A.1 Visualization of Feature Grids

We provide a visualization of the learned feature grids in Fig. 13 on the *Veach Door* scene. To generate each visualization, we first find camera ray intersections, and collect the interpolated multi-level grid features (levels 4-128) at each hit point, resulting in a 16-channel feature image (one channel per feature dimension). We then perform PCA across the 16-D pixels, and reduce the dimensionality to the three top principal components. The transformed 3-channel pixels are then normalized to span the *Lab* color space for visualization.

From the visualizations, we see that although the network started with random initializations, network training encourages locations with similar radiance properties to have similar weights. In particular, note how each of the teapots has different PCA colors, indicating that the model has learned different features for handling various material types. Another interesting observation is that the network

learns different features for areas with shadows; see the alternative PCA colors for shadows of the table and teapots.

### A.2 Residual Normalization

We compare the use of different residuals as loss functions, and conclude that the relative residual proposed in the main paper performs the best. The comparison is made by optimizing four different loss functions: relative residual normalized by average of LHS and RHS as proposed in the paper, by LHS only, by RHS only, or with no normalization. Figure 14 illustrates different renderings of the *Bedroom* scene after optimizing the above loss terms separately. We found that our approach using normalization of the average of the LHS and RHS leads to the most accurate solutions. We observed that normalizing by the LHS often creates numerical instabilities and exploding gradients during training.

### A.3 Network Inputs

We input the normalized position  $x \in \mathbb{R}^3$  along with the vectors obtained from our multi-level feature grid. As mentioned in the main paper, we also provide normalized directional inputs including the outgoing direction as  $\omega_o$  and surface normals. The other inputs are surface roughness in  $\mathbb{R}^2$ , and diffuse or specular reflectance values in  $\mathbb{R}^3$ . Table 3 lists all the network inputs used in our method.

Table 3. Network Inputs

Input	Symbol	Range
Position	$x \in \mathbb{R}^3$	$[0, 1]$
Direction	$\omega \in \mathbb{R}^3$	$[-1, 1]$
Surface Normal	$n \in \mathbb{R}^3$	$[-1, 1]$
Diffuse Reflectance	$f_d \in \mathbb{R}^3$	$[-1, 1]$
Specular Reflectance	$f_r \in \mathbb{R}^3$	$[-1, 1]$
Interpolated features	$G(x) \in \mathbb{R}^{16}$	-

### A.4 Implementation of Sparse Feature Grids

Storing dense feature vectors incurs cubic storage costs in the grid resolution; the space inefficiency comes from the fact that many voxels do not contain any scene surfaces. By using a fast pre-processing step, we find and store only the voxels with some surface inside them. The algorithm begins by finding voxels that contain at least one triangle; next we store a hash value for each relevant voxel in a sorted *array of hashes*. An array of feature vectors with the same size as the *array of hashes* is then allocated. With this scheme, the hash of the queried voxel is computed in  $O(1)$ , and look-up of the feature vector array can be done efficiently with PyTorch’s GPU-accelerated sorted search. In our implementation, the hash value we used is simply  $x + y \times \text{gridsize} + z \times \text{gridsize}^2$ , where  $(x, y, z)$  is the voxel coordinate.

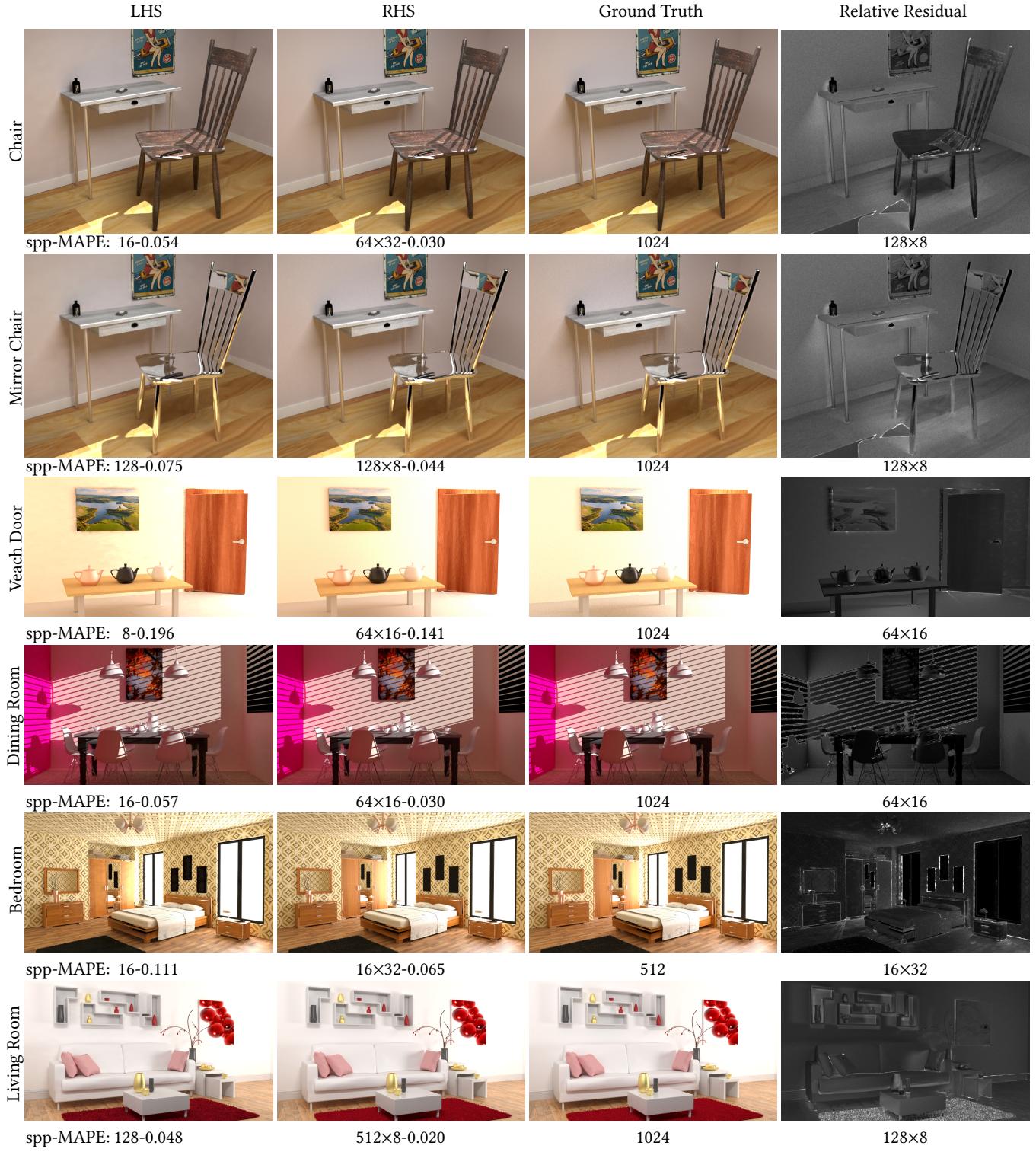
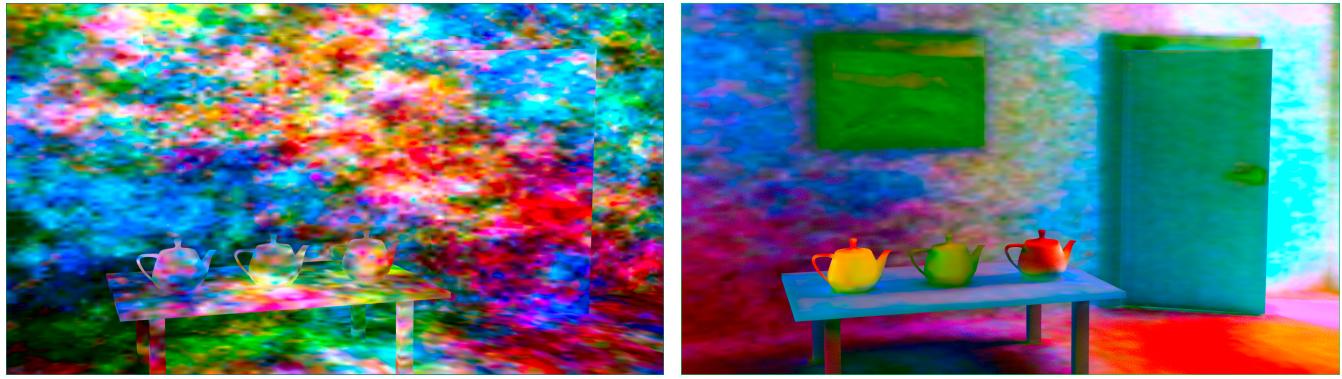


Fig. 12. Results for example scenes. The number of sample rays shot per camera pixel is denoted by “spp”, multiplied by our RHS hyperparameter  $M$ , denoting the number of rays cast to compute the scattering integral. *Dining Room*, *Bedroom*, *Veach Room*, *Living Room* are provided by Bitterli [2016]. *Chair* and *Mirror Chair* are a modified versions of [Barcin 2020].



(a) Before training

(b) After training

Fig. 13. Visualization of the grids in the *Veach Door* scene. The interpolated features are rendered into 16-D pixels, which then are reduced by PCA to 3-D *Lab* colors.



(a) No normalization, MSE = 0.013, MAPE = 0.208

(b) Our normalization, MSE = 0.018, MAPE: 0.111

(c) Normalize only by LHS, MSE = 0.030, MAPE: 0.147

(d) Normalize only by RHS, MSE = 0.061, MAPE: 0.173

Fig. 14. Comparing results using residuals with different normalizations.