Shape, Light and Neural Material Decomposition using Monte-Carlo Rendering and Diffusion Denoising

Ananth Kalyanasundaram Simeng Li Soumya Mondal Surya Prabhakaran Jonathan Schmidt
Technical University of Munich

Abstract

Differentiable rendering has made significant progress lately, allowing for the creation of high-quality 3D scene reconstructions from multiple images. One of the leading methods in differentiable rendering is NVDIFFRECMC [24], which employs a physically-based shading model that integrates ray tracing and Monte Carlo methods to achieve superior decomposition. However, this algorithm is limited by its reliance on a subset of available data and primitive bilateral filtering. We implemented a neural BRDF and denoising to improve simultaneously reconstructing geometry (in explicit triangle meshes), materials, and lighting. This results in substantial improvements in material and light separation over previous methods. We believe that incorporating neural denoising techniques and augmenting the training data can further enhance the quality of the inverse rendering pipeline.

1. Introduction

Differentiable rendering techniques have shown great potential for achieving accurate 3D reconstruction from multiple image observations. Various methods have been developed for this purpose, including NeRF [39] which uses differentiable volume rendering to create high-quality view interpolation through neural, density-based light fields. Other surface-based methods rely on signed distance [45, 47] or triangle meshes [43, 55] to capture high-quality geometry. More recent [5, 43, 65] work has further decomposed these representations into geometry, material, and environment light.

Despite the impressive results achieved by these methods, most rely on appearance baked into neural light fields or apply simple shading models that do not account for shadows or indirect illumination [5, 64]. While some methods do consider shadowing and indirect illumination, the deviations from physically-based shading models make it difficult for these methods to plausibly disentangle shape, material, and lighting.

While it is theoretically straightforward to replace the rendering engines of these 3D reconstruction methods with more photorealistic differentiable renderers [31, 36, 46] and optimize in a setting with more accurate light simulation, including global illumination effects, in practice, this poses a challenge. The noise in multi-bounce Monte Carlo rendering makes gradient-based optimization challenging, requiring very high sample counts which result in intractable iteration times.

Our work addresses the gap between existing multi-view 3D reconstruction methods and physically-based differentiable rendering, providing high-quality reconstructions with competitive runtime performance. We use a Neural BRDF model for material representation. To enhance visual fidelity, we employ Monte-Carlo integration with ray tracing to compute indirect illumination and use a generative model for denoising.

Our key contributions are:

- A modular framework for inverse rendering.
- A neural BRDF with a prior from material databases.
- A diffusion model for differentiable image denoising.

2. Related Work

2.1. Neural Reconstruction

Neural methods for multi-view reconstruction can be broadly classified into two groups based on the type of scene representation used: implicit and explicit. The implicit methods, such as NeRF [39] and its follow-up methods [16, 37, 40, 44, 49, 50, 59, 60, 62, 64], utilize volumetric representations and compute radiance by volumetric rendering of a 5D light field that is encoded in a neural network. Although these methods have shown remarkable success in novel view synthesis, they suffer from geometric quality limitations due to the ambiguity of volume rendering [64].

In contrast, surface-based rendering methods [45, 47, 57, 61]directly optimize the underlying surface using implicit differentiation or gradually morph from a volumetric representation into a surface representation. These methods aim

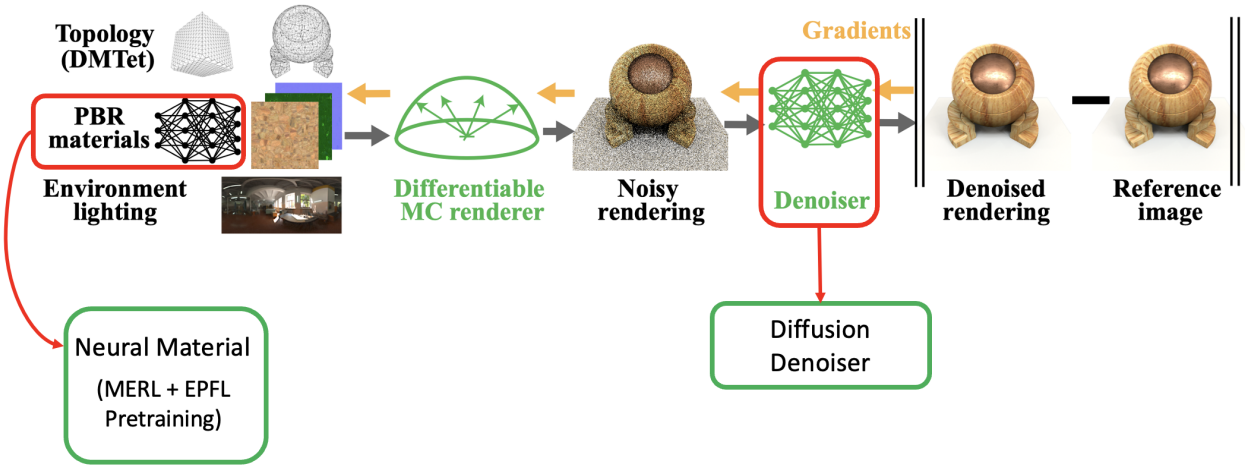


Figure 1. Overview of our contributions: We extend the *nvdiffrecmc* [24] pipeline with a neural material model and a diffusion-based denoiser.

to overcome the limitations of volume rendering and improve the quality of the rendered surfaces. Implicit representations, such as DeepSDF, and explicit representations, such as 3D meshes, are used in these methods. While explicit representation methods produce high-quality geometric results, they require more computational resources compared to implicit representations. Most of these methods assume a given mesh topology [10, 11, 35], although recent work in this area includes topology optimization [3, 15, 34, 55].

2.2. Material and Lighting Estimation

Estimating surface radiometric properties from images have relied on various methods, including special viewing configurations, lighting patterns, and complex capturing setups [17, 18, 20, 30]. Some recent approaches have utilized neural networks to predict the Bidirectional Radiance Distribution Function (BRDF) directly from images [14, 21, 32, 33], while others have used differentiable rendering methods [10, 35] to predict geometry, Spatially-varying BRDF, and lighting via photometric loss.

Some methods represent illumination using mixtures of spherical Gaussians [5, 43, 63, 66], pre-filtered approximations [6, 43], or low-resolution environment maps. Additionally, some methods account for the shadowing term, which requires a split optimization approach with geometry locked before the shadow term is sampled. Other approaches use neural networks [58, 66] to represent indirect illumination.

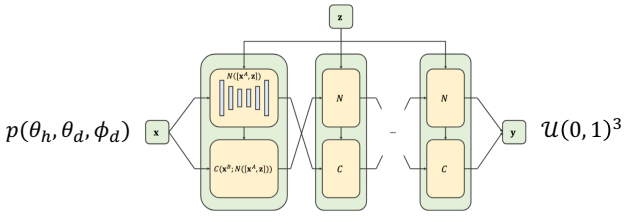
2.3. Denoising

Denoisers play a crucial role in real-time and production renderers. Cross-bilateral filters have traditionally been used [67], but require manual tuning for each scene. In recent years, neural denoisers [2, 9, 26] trained on large datasets have demonstrated impressive quality without manual adjustments and are now widely used in production renderers. Our approach incorporates differentiable versions of these denoisers directly into our pipeline. We believe that differentiable denoisers have great potential in physically-based inverse rendering in the future.

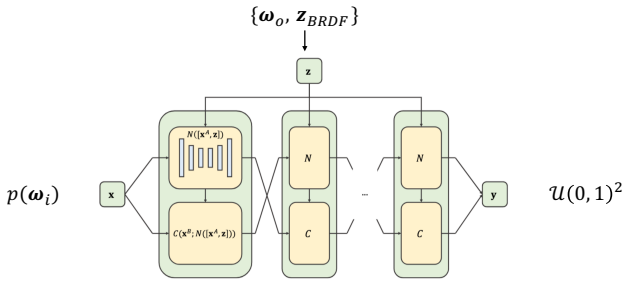
3. System Overview

In this work, we aim to tackle the complex problem of optimizing the shape, material, and environment lighting of an object using physically-based rendering. Our approach involves utilizing a set of multi-view images that come with known foreground segmentation masks and camera poses. Our ultimate objective is to enhance the intrinsic decomposition of lighting and materials, which will result in assets that can be utilized for various purposes such as relighting, editing, animation, or simulation. To demonstrate the effectiveness of our approach, we build upon the recent method NVDIFFREMC [23], which is designed to solve the same optimization task of shape, materials, and environment lighting.

Our system is summarized in Figure 1. The optimization of a triangular mesh with arbitrary topology from a set of images is achieved through 2D supervision. To represent geometry, a signed distance field is defined on a three-dimensional grid and then reduced to a triangular surface mesh using deep marching tetrahedra (DMTet) [55].Next, the extracted surface mesh is rendered in a differentiable renderer. In contrast to the NVDIFFRECMC, which employs a physically-based (PBR) material model from Disney [7] that merges a diffuse term with an isotropic, spec-



(a) iBRDF [12]



(b) Ours

Figure 2. Comparison of Normalizing Flow architectures for representing BRDFs.

ular GGX lobe [56], we leverage a Neural BRDF model, pre-trained on MERL and EPFL dataset, as elaborated in the section 5. Finally, the rendered image is evaluated against a reference image using a photometric loss. We utilize the same renderer as NVDIFFRECMC, which employs ray tracing and Monte Carlo integration to compute the direct lighting integral. The scene lighting is represented using a high dynamic range light probe stored as a floating point texture with a resolution of 256×256 texels. To mitigate the inherent variance of Monte Carlo integration, we employ a diffusion model-based denoising technique, as described in section 6.

In addition, unlike NVDIFFRECMC which was implemented using Pytorch, we developed our complete inverse rendering pipeline using Pytorch Lightning. This allowed for faster development with pre-built modules, reducing boilerplate code and improving readability. We were able to integrate the material, geometry and lighting modules efficiently in our pipeline.

4. Material Databases

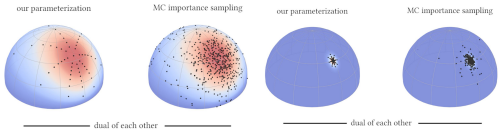
Introducing new BRDF datasets is considered a promising way of training on more kinds of materials and capture more complicated material properties. The MERL dataset contains 100 isotropic materials [38].

The UTIA and EPFL dataset have high enough resolution for synthesis [13]. The EPFL dataset is created by the Realistic Graphics Lab (RGL) in EPFL. The UTIA is a dataset containing 150 anisotropic materials [22], and the EPFL dataset contains 51 isotropic and 11 anisotropic materials (until February, 2023) [13].

The UTIA database is investigated, but as the project is focusing on isotropic materials, it is finally decided to focus on the isotropic parts of the EPFL dataset. EPFL dataset also has some advantages over the previous datasets by applying the new adaptive parameterization method. MERL and UTIA datasets convert RGB to spectra using the heuristic formula, and MERL has measurement artifacts such as the optical aberrations, the discontinuities and artifacts at smaller than 75 degrees [13].

4.1. EPFL Dataset

EPFL datasets reaches better compression compared to MERL by applying the adaptive parameterization, which is based on each material's BRDF properties. Rough materials, which has more diffuse effects, has a wider BRDF lobe, while shiny materials have more concentrated lobes as in Figure 3. The region including the lobe and surrounded region is the area of high interest in BRDF measurement. Adaptive parameterization automatically samples the high-interest area by applying monte-carlo importance sampling and several parameterization warps [13]. The monte-carlo sampling and resulting parameterization forms duality [13].



(a) Diffuse (b) Specular

Figure 3. Diffuse and Specular Material lobes and sampling. Figure from [13].

There are three main phases of the whole BRDF measurement process as in Figure 12. Acquisition, Sampling and Evaluation [13]. In acquisition, there are 2 steps: first step is to acquire the retro-reflective responses, which is used to determine the material's BRDF properties. After the individual properties are determined, the spectrometer is then used to acquire the parametric BRDF. During acquisition and sampling, 4 warps are applied. Thus the inverse of the warps are applied in the evaluation phase to determine the BRDF values corresponding to the incident and outgoing angles [13].

4.2. BRDF Format Conversion

The main structure of conversion from EPFL format to MERL format is based on the C++ implementation of evaluation phase of EPFL dataset. There are two types of evaluations functions, the RGB and spectro evaluations. The RGB one will take the incident and outgoing angles and evaluate the RGB results, while the spectro version output

the results for different wavelengths [13]. We only use the RGB component. based on different It provides an evaluation method from $powiteq\_rgb.cpp$ that evaluates the corresponding RGB values from the data files based on the given lighting and viewing direction [13].

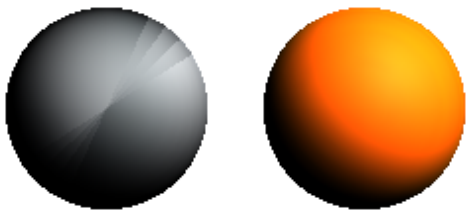
4.3. Comparisions of Different Parameterizations

This adaptive parameterization has several advantages over traditional methods. The traditional methods are mostly built around the half angle, such as the Rusinkiwiscz coordinates [13, 53]. They only have high sampling rates on high-frequency materials. The MERL dataset only contains specific set of materials and has high dimensionality, including the incident direction, outgoing direction and normal. The high dimensionality makes the measurement expensive to operate. However, the adaptive parameterizations is measured based on each material's individual properties, and the measurement is restricted to only 2D [13], which makes the process tractable.

We also need to transform the coordinate system, since MERL uses the Rusinkiwiscz Coordinates, while the evaluation procedure of the EPFL dataset uses standard direction vectors (ω_i, ω_o) [13, 53]. The conversion is based on the utility function from Nerfactor project [65].

4.4. Pre-training Results

For the pretraining procedure, the training dataset is the combination of MERL and EPFL dataset, and the validation dataset is the MERL dataset. Compared to the case of using the single MERL dataset, the validation result of combination of two datasets has a good convergence but slightly worse results than the result of using single MERL as in Figure 13. This has several reasons. The combination of two datasets leads to a more complicated feature space, which can cause the convergence on MERL to be slightly worse. Another reason can be due to the loss of precision during the conversion between MERL and EPFL as in figure. The rendered cube of converted EPFL dataset has some artifacts as in Figure 4. As EPFL dataset used importance sampling, the uniformly sampled $180 * 90 * 90$ angles can also lead to worse precision.



(a) EPFL Ceilling Gray Paint (b) MERL Orange Paint

Figure 4. Comparison of 2 rendered spheres in MERL format

4.5. Section Summary

As a result, there are some potential future works. To better utilize the power of adaptive parameterization, the new training and validation methods are important. It would also be an interesting topic to include anisotropic materials. To capture the more complicated materials, which means more complex feature space, new networks are left to be constructed.

5. Neural BRDF

5.1. Preliminaries

The spatially-varying BRDF is a 7-dimensional function, which maps incident light direction ω_i, outgoing light direction ω_o and the surface location $x \in \mathbb{R}^3$ to an RGB reflectance value $r \in \mathbb{R}^3$. It can be interpreted as the fraction of light coming from ω_i that gets reflected to ω_o at postion x. Following [65], we decompose the BRDF into a diffuse and a specular component, where the diffuse part only depends on the surface location x. Thus, our BRDF can be written as:

$$f_r(\omega_i, \omega_o, x) = a(x) + f_s(\omega_i, \omega_o, x) \tag{1}$$

where $a(x)$ refers to the diffuse color and $f_s(\cdot)$ determines the specular reflection. For simplicity, we assume the diffuse and specular color to be identical, which allows us to model f_s independently from any color. For the remainder of this section, we focus on modeling the specular reflection function.

Explicit BRDF models like [8] can be used for inverse rendering by optimizing its parameters until they produce the desired rendering. However, these parameters were designed to be artist-friendly and some parameter configurations are more likely and common than others. Such prior information could make the optimization more robust as it constrains the highly ill-posed problem, but cannot be incorporated into analytical models.

We instead use a neural network to represent the specular part of the BRDF. Therefore, we train a model on the material datasets (Sec. 4) along with a low dimensional latent code for each material. The latent code holds all material related properties, while the model decodes it into reflection values given viewing directions.

5.2. MLP BRDF

Following [65] we model the problem with a simple multi-layer perceptron (MLP) consisting of 4 fully-connected layers with ReLU activation functions. After the last layer we apply the Softplus function, since reflection values cannot be negative. The input to this model are the Rusinkiewicz coordinates [53] of the viewing directions

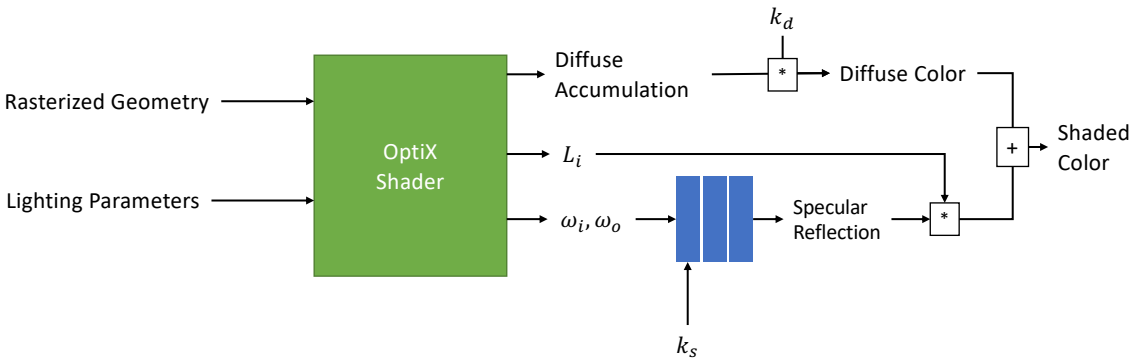


Figure 5. Our neural shading pipeline using a custom OptiX [48] routine for lighting and shadow computations followed by our neural BRDF model.

and the latent code of the material. We encode the coordinates using the multi-frequency sine-cosine embedding from NeRF [39] and concatenate it with the latent code. The output of the network is the scalar specular reflection value. The latent codes are optimized directly using Generative Latent Code Optimization [4].

In our inverse rendering setting, we usually evaluate many BRDF samples in a single step, meaning that we have a large batch size (>1M) compared to a low input and output data dimension (6D and 1D). In common deep learning frameworks like PyTorch, each MLP layer would perform a single matrix multiplication on the GPU. However, due to the size of the matrix, this requires a lot of communication between GPU thread blocks via high-level caches. Müller et al. [42] leverage the fact that the elements in the batch are independent of each other and divide the batch into smaller chunks that fit into a single thread block. Thus, the entire forward pass can be performed without any inter-thread-block synchronization, leading to its name *fully-fused MLP* [42].

We implement our MLP BRDF model explained previously using both ordinary PyTorch layers and the fully-fused MLP [42] to compare their performance. Details of which are further described in Sec. 7.1.

5.3. Normalizing Flow BRDF

In the previous section we modeled the problem as a discriminative task, meaning that we directly learn a mapping from the parameters $(\mathbf{z}, \omega_i, \omega_o)$ to the reflection value. This allows us to evaluate the reflection but does not provide access to the underlying reflectance distribution. Further, it does not enforce physical properties such as energy conservation.

These drawbacks can be tackled by using a generative model instead. Since our target distribution is low dimensional, we use Normalizing Flows. Therefore, we adopt the model from [12], which uses the NICE architecture [51] with piecewise-quadratic coupling layers [41]. The target distribution is the reflectence over ω_i, ω_o given a latent code $\mathbf{z}$ and the base distribution is a 3-dimensional uniform distribution. From a probabilistic perspective, [12] models the reflection as:

$$f_s(\omega_i, \omega_o, \mathbf{z}) = p(\omega_i, \omega_o | \mathbf{z}) \tag{2}$$

We found a significant issue with this parameterization as it models the joint distribution over incident and outgoing light direction, while in Monte-Carlo Rendering, ω_o is usually firm and we only want to sample incident directions.

Our Normalizing Flow models the problem as:

$$f_s(\omega_i, \omega_o, \mathbf{z}) = p(\omega_i | \omega_o, \mathbf{z}) \tag{3}$$

While this parameterization enables importance sampling, it disqualifies the usage of Rusinkiewicz coordinates [53] due to the separation of incident and outgoing light direction. The consequences of this are outlined in Sec. 7.1. We also found, that the number of layers and model parameters can be reduced significantly. Details can be found in the appendix.

5.4. Integration

We integrate our neural BRDF models into the rendering pipeline of *nvdiffrecmc* [24]. *nvdiffrecmc* applies a deferred rendering scheme with a differentiable rasterizer [29] followed by a custom OptiX ray tracing routine to perform shadow tests and pixel color computations using Disney's physically-based shading model [8]. OptiX requires any computations to be defined as per-pixel CUDA functions, which contradicts PyTorch's batch execution model. Further, OptiX code is not captured by the autograd engine, which entails manual gradient computations.

We therefore decided to implement most shading logic in Python, and only use OptiX for ray tracing routines to

leverage hardware acceleration. Our modified OptiX kernel samples incident light directions ω_i and computes the corresponding light values L_i. We then evaluate our neural BRDF model and get the specular reflection values, which we add to the diffuse base color.

A major drawback of this solution is that it's memory footprint grows linearly with the number of samples per pixel (spp).

6. Diffusion Denoiser

Integrated Monte Carlo rendering produces noisy outputs [24] at low samples per pixel (spp) values. Authors use a bilateral denoiser [54] to tackle this issue. However this denoiser works through bilinear interpolation which often causes blur and other artificats in the image. The authors tried using a neural denoiser [1, 52]; however this did not yield better results than the bilateral denoiser and was hence discarded. Moreover, the UNet denoiser treats this task as a deterministic problem.

Our key idea is to treat the task as a stochastic problem and use generative models to solve this. We have seen the success of GANs [19] in the previous decade which beat several deterministic neural networks in the tasks of segmentation [28], denoising and super-resolution.

Denoising diffusion models [27] are a class of deep generative models that learn to generate data samples from noise by reversing a Markov chain of diffusion steps. They are trained using variational inference to maximize the likelihood of recovering the original data from noisy versions, using UNets [52] in general. Diffusion models can capture complex data distributions and generate high-quality samples for various tasks such as image synthesis. Diffusion models have several advantages over other generative models like GANs, such as being stable to train, and also producing better quality of results in general.

6.1. Dataset

Using Blender, we render 1000 views per spp of the object at 20 spp values. As a result, we create a dataset of 19000 pairs with 4096 as the target spp. For validation, we render 200 novel views of the bob object. This diversity in the dataset ensures robustness of the model to a good range of spp and views.

7. Experiments

7.1. BRDF Pretraining

We qualitatively and quantitatively evaluate our BRDF models on the MERL dataset [38]. For the qualitative comparison, we take the latent codes that were learned during training and use them to render spheres with the respective model. The results are depicted in Fig. 8. All models can capture a wide variety of materials from the database. Still, we observe minor artifacts in our Normalizing Flow model, since we cannot use Rusinkiewicz coordinates [53] here.

Table 1 shows the quantitative evaluation of our BRDF models when trained on both the MERL and EPFL dataset, compared to the MERL dataset alone. In the first experiment, we compare the mean-squared error (MSE) of our validation split of the MERL dataset. The validation split consists of 0.1% randomly sampled values from each material and is consistent across all experiments. Here, the Normalizing Flow models perform much better than the MLPs. It is important to note, however, that both NF models have vastly more parameters (500k - 1M) than the MLP model (50k).

Next, we compare the SSIM of the spheres from Fig. 8. For each model, we compare the rendering generated with only the latent code with the rendering generated with the reference reflection values from the MERL database. Again, the Normalizing Flow models perform better, which we attribute to the larger model size.

Finally, we compare the runtime it takes to evaluate 1 million BRDF queries. Here the fully-fused MLP [42] outperforms the Normalizing Flows by an order of magnitude. One reason is the smaller model size of the MLP but the main performance gain is the custom GPU implementation of the MLP. It would be interesting to see, whether a fully-fused Normalizing Flow with less parameters would achieve similar speeds. We leave that part to future work.

7.2. Diffusion denoiser

We train the diffusion model for 110k iterations with a learning rate of 1e-4 and a batch size of 1 due to memory limitations. The training was done on a NVIDIA 3090 GPU. For the loss curve, refer to Fig. 14

From Tab. 2, we can observe that our diffusion denoiser performs significantly better than the Bilateral denoiser in terms of SSIM and PSNR. The increase in SSIM can be attributed to the diffusion model enhancing more details whereas there is loss of structure caused by the blurring of the bilateral denoiser.

7.3. Inverse Rendering

We compare our inverse rendering architecture with the baseline *nvdiffrecmc* [24]. The quantitative comparison (Tab. 3) shows that our neural material model does not perform as well as the Disney model used by the baseline. We attribute this to the missing importance sampling in the Monte-Carlo rendering, which is particularly essential for glossy and metallic materials. The Normalizing Flows models would probably perform better at the cost of higher runtimes.

The qualitative results in Fig. 10 show this lack of specular highlights in the re-renderings of our model. However, the reconstructed material properties in the middle row

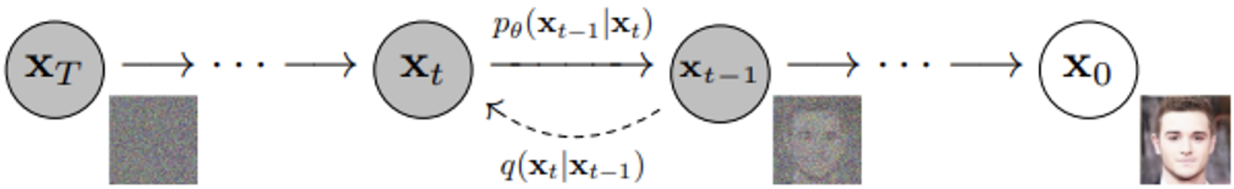
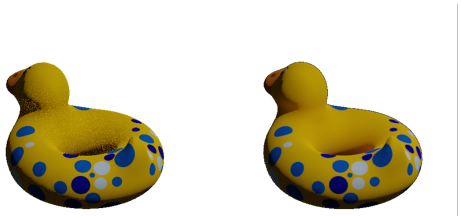


Figure 6. Overview of the diffusion model [27]. The forward process is a Markov chain where data is destroyed by adding noise in steps. The reverse process uses a neural network to predict the probability density in the previous step.

Method	MERL [38]			MERL [38] + EPFL [13]		
	MSE	SSIM	Runtime(ms / 1M queries)	MSE	SSIM	Runtime(ms / 1M queries)
MLP (PyTorch)	0.030	0.971	11	0.112	0.972	11
MLP (TCNN [42])	0.029	0.973	**1**	0.117	0.975	**1**
NF (iBRDF [12])	**0.006**	**0.990**	1007	0.371	0.643	1007
NF (Ours)	0.019	0.977	569	**0.021**	**0.990**	569

Table 1. Quantitative Evaluation of our BRDF models on the MERL dataset, when trained on different datasets.



(a) Input (b) Ground Truth

Figure 7. Example of a low spp - high spp image pair from the dataset

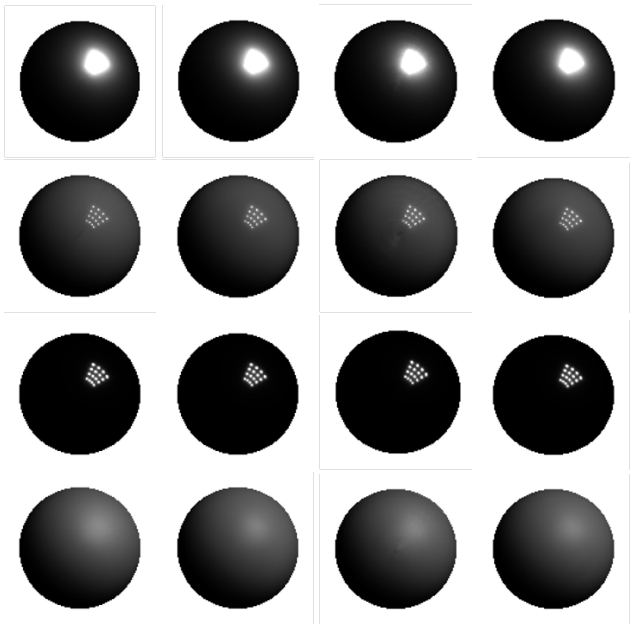
Method	PSNR (dB)	SSIM
Bilateral Denoiser	14.49	0.7401
Diffusion Model (ours)	**15.88**	**0.8035**

Table 2. Quantitative Results of the diffusion denoiser training.

Method	MSE	PSNR (dB)	SSIM
nvdiffrecmc	**0.0006**	**32.455**	**0.969**
Ours w/ neural BRDF	0.0014	28.407	0.949

Table 3. Quantitative evaluation of our inverse rendering pipeline.

show, that our neural BRDF correctly learned that the parameters are constant. Note, that the reconstructed material properties of our method depict a latent code, which is why we cannot compare the colors with the ground-truth



(a) MLP BRDF (b) iBRDF [12] (c) NF BRDF (d) Ground Truth

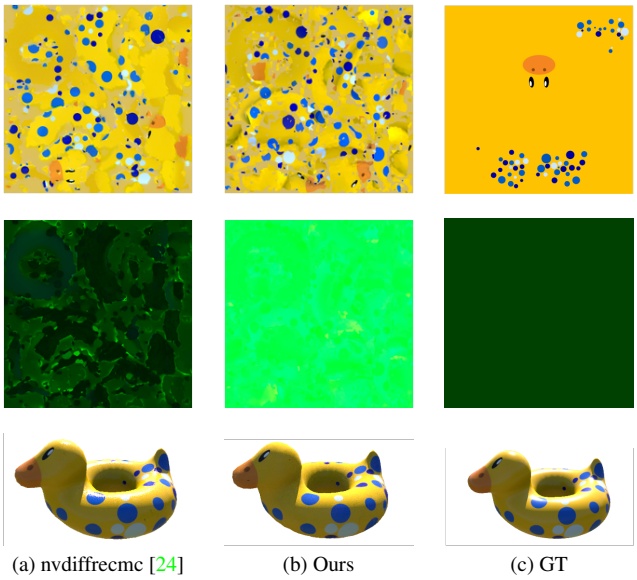
Figure 8. Qualitative comparison of our BRDF models evaluation on different materials from the MERL dataset.

nor *nvdiffrecmc* [24].

In addition to the neural BRDF model, we also study the results of using the diffusion denoiser at concurrence. Due to memory limitations, we render our images at a resolution of 128x128 during training and inference times. From the qualitative comparison in Fig. 11, one can see that the

(a) Input (b) Bilateral (c) Diffusion (d) Ground Truth

Figure 9. Qualitative Results of the denoisers



(a) nvdiffrecmc [24] (b) Ours (c) GT

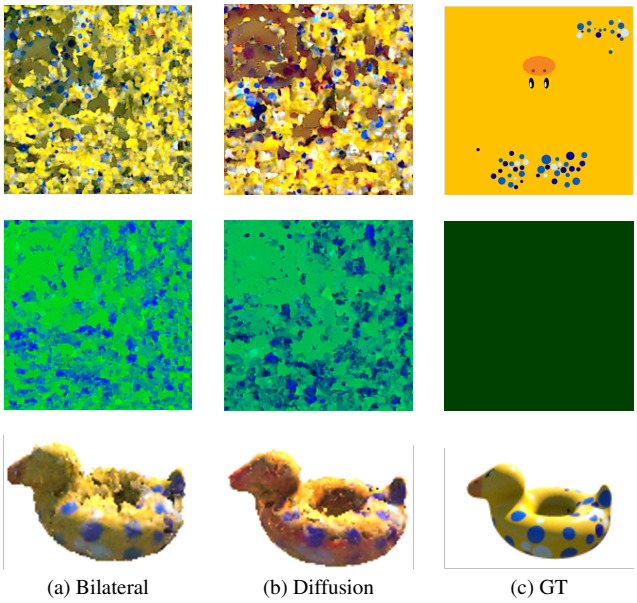
Figure 10. Qualitative comparison of our inverse rendering results with nvdiffrecmc [24]: top row: reconstructed diffuse texture, mid row: reconstructed material properties, bottom row: re-rendering of reconstructed object.

Method	MSE	PSNR (dB)	SSIM
Ours w/ Bilateral	0.0177	17.515	0.7633
Ours w/ Diffusion	**0.0148**	**18.273**	**0.7859**

Table 4. Quantitative evaluation of denoiser at a render resolution of 128x128.

output mesh generated by the pipeline using the diffusion denoiser is more complete and has significantly less artifacts as compared to the ones using the bilateral denoiser. We also note that our diffusion model might be overfitting on the dataset generated using Blender.

Quantitatively from Tab. 4, we observe that even at low render resolutions, our pipeline with diffusion denoiser outperforms the one with the bilateral denoiser in all the metrics used for comparison.



(a) Bilateral (b) Diffusion (c) GT

Figure 11. Ablation study of different denoisers in our inverse rendering pipeline. Top row: reconstructed diffuse texture, mid row: reconstructed material properties, bottom row: re-rendering of reconstructed object.

8. Conclusion

We extended *nvdiffrecmc* [24] by a neural material model and a diffusion based denoiser. While the neural material models promise to bring prior knowledge to the vastly ill-posed problem of inverse rendering, we were facing issues regarding their runtime and a possible lack of importance sampling, which needs to be addressed in future work. Our diffusion denoiser demonstrated better results than the bilateral denoiser. Further improvements can be made if we also vary the lighting in the data. This would make the diffusion model invariant to lighting. The diffusion model increases inference time and memory requirements, which is why we suggest the usage of latent diffusion models for better performance and reduced computation requirements.

References

[1] Attila T. Afra. Open image denoise, 2022. https://www.openimagedenoise.org/. 6

[2] Steve Bako, Thijs Vogels, Brian McWilliams, Mark Meyer, Jan Novák, Alex Harvill, Pradeep Sen, Tony Derose, and Fabrice Rousselle. Kernel-predicting convolutional networks for denoising monte carlo renderings. *ACM Trans. Graph.*, 36(4):97–1, 2017. 2

[3] Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neu-

ral radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5855–5864, 2021. 2

[4] Piotr Bojanowski, Armand Joulin, David Lopez-Pas, and Arthur Szlam. Optimizing the latent space of generative networks. In *International Conference on Machine Learning*, pages 600–609. PMLR, 2018. 5

[5] Mark Boss, Raphael Braun, Varun Jampani, Jonathan T Barron, Ce Liu, and Hendrik Lensch. Nerd: Neural reflectance decomposition from image collections. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12684–12694, 2021. 1, 2

[6] Mark Boss, Varun Jampani, Raphael Braun, Ce Liu, Jonathan Barron, and Hendrik Lensch. Neural-pil: Neural pre-integrated lighting for reflectance decomposition. *Advances in Neural Information Processing Systems*, 34:10691–10704, 2021. 2

[7] Brent Burley and Walt Disney Animation Studios. Physically-based shading at disney. In *Acm Siggraph*, volume 2012, pages 1–7. vol. 2012, 2012. 2

[8] Brent Burley and Walt Disney Animation Studios. Physically based shading at disney. 2012. 4, 5

[9] Chakravarty R Alla Chaitanya, Anton S Kaplanyan, Christoph Schied, Marco Salvi, Aaron Lefohn, Derek Nowrouzezahrai, and Timo Aila. Interactive reconstruction of monte carlo image sequences using a recurrent denoising autoencoder. *ACM Transactions on Graphics (TOG)*, 36(4):1–12, 2017. 2

[10] Wenzheng Chen, Huan Ling, Jun Gao, Edward Smith, Jaakko Lehtinen, Alec Jacobson, and Sanja Fidler. Learning to predict 3d objects with an interpolation-based differentiable renderer. *Advances in neural information processing systems*, 32, 2019. 2

[11] Wenzheng Chen, Joey Litalien, Jun Gao, Zian Wang, Clement Fuji Tsang, Sameh Khamis, Or Litany, and Sanja Fidler. Dib-r++: learning to predict lighting and material with a hybrid differentiable renderer. *Advances in Neural Information Processing Systems*, 34:22834–22848, 2021. 2

[12] Zhe Chen, Shohei Nobuhara, and Ko Nishino. Invertible neural brdf for object inverse rendering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(12):9380–9395, 2021. 3, 5, 7, 13

[13] Jonathan Dupuy and Wenzel Jakob. An adaptive parameterization for efficient material acquisition and rendering. *Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 37(6):274:1–274:18, Nov. 2018. 3, 4, 7, 13

[14] Duan Gao, Xiao Li, Yue Dong, Pieter Peers, Kun Xu, and Xin Tong. Deep inverse rendering for high-resolution svbrdf estimation from an arbitrary number of images. *ACM Trans. Graph.*, 38(4):134–1, 2019. 2

[15] Jun Gao, Wenzheng Chen, Tommy Xiang, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Learning deformable tetrahedral meshes for 3d reconstruction. *Advances In Neural Information Processing Systems*, 33:9936–9947, 2020. 2

[16] Stephan J Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin. Fastnerf: High-fidelity neural rendering at 200fps. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14346–14355, 2021. 1

[17] Andrew Gardner, Chris Tchou, Tim Hawkins, and Paul Debevec. Linear light source reflectometry. *ACM Transactions on Graphics (TOG)*, 22(3):749–758, 2003. 2

[18] Abhijeet Ghosh, Tongbo Chen, Pieter Peers, Cyrus A Wilson, and Paul Debevec. Estimating specular roughness and anisotropy from second order spherical gradient illumination. In *Computer Graphics Forum*, volume 28, pages 1161–1170. Wiley Online Library, 2009. 2

[19] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020. 6

[20] Darya Guarnera, Giuseppe Claudio Guarnera, Abhijeet Ghosh, Cornelia Denk, and Mashhuda Glencross. Brdf representation and acquisition. In *Computer Graphics Forum*, volume 35, pages 625–650. Wiley Online Library, 2016. 2

[21] Yu Guo, Cameron Smith, Miloš Hašan, Kalyan Sunkavalli, and Shuang Zhao. Materialgan: reflectance capture using a generative svbrdf model. *arXiv preprint arXiv:2010.00114*, 2020. 2

[22] Vávra R. Haindl M., Filip J. Digital material appearance: the curse of tera-bytes. *ERCIM News*, (90):49–50, 2012. 3

[23] Jon Hasselgren, Nikolai Hofmann, and Jacob Munkberg. Shape, light & material decomposition from images using monte carlo rendering and denoising. *arXiv preprint arXiv:2206.03380*, 2022. 2

[24] Jon Hasselgren, Nikolai Hofmann, and Jacob Munkberg. Shape, Light, and Material Decomposition from Images using Monte Carlo Rendering and Denoising. *arXiv:2206.03380*, 2022. 1, 2, 5, 6, 7, 8

[25] Jon Hasselgren, Jacob Munkberg, Jaakko Lehtinen, Miika Aittala, and Samuli Laine. Appearance-driven automatic 3d model simplification. In *EGSR (DL)*, pages 85–97, 2021. 12

[26] Jon Hasselgren, Jacob Munkberg, Marco Salvi, Anjul Patney, and Aaron Lefohn. Neural temporal adaptive sampling and denoising. In *Computer Graphics Forum*, volume 39, pages 147–155. Wiley Online Library, 2020. 2

[27] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020. 6, 7

[28] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *CVPR*, 2017. 6

[29] Samuli Laine, Janne Hellsten, Tero Karras, Yeongho Seol, Jaakko Lehtinen, and Timo Aila. Modular primitives for high-performance differentiable rendering. *ACM Transactions on Graphics*, 39(6), 2020. 5

[30] Hendrik PA Lensch, Jochen Lang, Asla M Sá, and Hans-Peter Seidel. Planned sampling of spatially varying brdfs. In *Computer graphics forum*, volume 22, pages 473–482. Wiley Online Library, 2003. 2

[31] Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. Differentiable monte carlo ray tracing through edge sampling. *ACM Transactions on Graphics (TOG)*, 37(6):1–11, 2018. 1

[32] Zhengqin Li, Mohammad Shafiei, Ravi Ramamoorthi, Kalyan Sunkavalli, and Manmohan Chandraker. Inverse rendering for complex indoor scenes: Shape, spatially-varying lighting and svbrdf from a single image. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2475–2484, 2020. 2

[33] Zhengqin Li, Zexiang Xu, Ravi Ramamoorthi, Kalyan Sunkavalli, and Manmohan Chandraker. Learning to reconstruct shape and spatially-varying reflectance from a single image. *ACM Transactions on Graphics (TOG)*, 37(6):1–11, 2018. 2

[34] Yiyi Liao, Simon Donne, and Andreas Geiger. Deep marching cubes: Learning explicit surface representations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2916–2925, 2018. 2

[35] Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. Soft rasterizer: A differentiable renderer for image-based 3d reasoning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7708–7717, 2019. 2

[36] Guillaume Loubet, Nicolas Holzschuch, and Wenzel Jakob. Reparameterizing discontinuous integrands for differentiable rendering. *ACM Transactions on Graphics (TOG)*, 38(6):1–14, 2019. 1

[37] Ricardo Martin-Brualla, Noha Radwan, Mehdi SM Sajjadi, Jonathan T Barron, Alexey Dosovitskiy, and Daniel Duckworth. Nerf in the wild: Neural radiance fields for unconstrained photo collections. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7210–7219, 2021. 1

[38] Wojciech Matusik, Hanspeter Pfister, Matt Brand, and Leonard McMillan. A data-driven reflectance model. *ACM Transactions on Graphics*, 22(3):759–769, July 2003. 3, 6, 7

[39] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021. 1, 5

[40] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics (ToG)*, 41(4):1–15, 2022. 1

[41] Thomas Müller, Brian McWilliams, Fabrice Rousselle, Markus Gross, and Jan Novák. Neural importance sampling. *ACM Transactions on Graphics (ToG)*, 38(5):1–19, 2019. 5

[42] Thomas Müller, Fabrice Rousselle, Jan Novák, and Alexander Keller. Real-time neural radiance caching for path tracing. *arXiv preprint arXiv:2106.12372*, 2021. 5, 6, 7

[43] Jacob Munkberg, Jon Hasselgren, Tianchang Shen, Jun Gao, Wenzheng Chen, Alex Evans, Thomas Müller, and Sanja Fidler. Extracting triangular 3d models, materials, and lighting from images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8280–8290, 2022. 1, 2, 12

[44] Michael Niemeyer and Andreas Geiger. Giraffe: Representing scenes as compositional generative neural feature fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11453–11464, 2021. 1

[45] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3504–3515, 2020. 1

[46] Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. Mitsuba 2: A retargetable forward and inverse renderer. *ACM Transactions on Graphics (TOG)*, 38(6):1–17, 2019. 1

[47] Michael Oechsle, Songyou Peng, and Andreas Geiger. Unisurf: Unifying neural implicit surfaces and radiance fields for multi-view reconstruction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5589–5599, 2021. 1

[48] Steven G Parker, Heiko Friedrich, David Luebke, Keith Morley, James Bigler, Jared Hoberock, David McAllister, Austin Robison, Andreas Dietrich, Greg Humphreys, et al. Gpu ray tracing. *Communications of the ACM*, 56(5):93–101, 2013. 5

[49] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-nerf: Neural radiance fields for dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10318–10327, 2021. 1

[50] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14335–14345, 2021. 1

[51] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International conference on machine learning*, pages 1530–1538. PMLR, 2015. 5, 13

[52] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015. 6

[53] Szymon M Rusinkiewicz. A new change of variables for efficient brdf representation. *Rendering techniques*, 98:11–22, 1998. 4, 5, 6

[54] Christoph Schied, Anton Kaplanyan, Chris Wyman, Anjul Patney, Chakravarty R Alla Chaitanya, John Burgess, Shiqiu Liu, Carsten Dachsbacher, Aaron Lefohn, and Marco Salvi. Spatiotemporal variance-guided filtering: real-time reconstruction for path-traced global illumination. In *Proceedings of High Performance Graphics*, pages 1–12. 2017. 6

[55] Tianchang Shen, Jun Gao, Kangxue Yin, Ming-Yu Liu, and Sanja Fidler. Deep marching tetrahedra: a hybrid representation for high-resolution 3d shape synthesis. *Advances in Neural Information Processing Systems*, 34:6087–6101, 2021. 1, 2

[56] Bruce Walter, Stephen R Marschner, Hongsong Li, and Kenneth E Torrance. Microfacet models for refraction through rough surfaces. In *Proceedings of the 18th Eurographics conference on Rendering Techniques*, pages 195–206, 2007. 3

[57] Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. *arXiv preprint arXiv:2106.10689*, 2021. 1

[58] Zian Wang, Jonah Philion, Sanja Fidler, and Jan Kautz. Learning indoor inverse rendering with 3d spatially-varying lighting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12538–12547, 2021. 2

[59] Zirui Wang, Shangzhe Wu, Weidi Xie, Min Chen, and Victor Adrian Prisacariu. Nerf–: Neural radiance fields without known camera parameters. *arXiv preprint arXiv:2102.07064*, 2021. 1, 13

[60] Suttisak Wizadwongsa, Pakkapon Phongthawee, Jiraphon Yenphraphai, and Supasorn Suwajanakorn. Nex: Real-time view synthesis with neural basis expansion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8534–8543, 2021. 1

[61] Lior Yariv, Yoni Kasten, Dror Moran, Meirav Galun, Matan Atzmon, Basri Ronen, and Yaron Lipman. Multiview neural surface reconstruction by disentangling geometry and appearance. *Advances in Neural Information Processing Systems*, 33:2492–2502, 2020. 1

[62] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. Plenoctrees for real-time rendering of neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5752–5761, 2021. 1

[63] Kai Zhang, Fujun Luan, Qianqian Wang, Kavita Bala, and Noah Snavely. Physg: Inverse rendering with spherical gaussians for physics-based material editing and relighting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5453–5462, 2021. 2

[64] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. Nerf++: Analyzing and improving neural radiance fields. *arXiv preprint arXiv:2010.07492*, 2020. 1

[65] Xiuming Zhang, Pratul P Srinivasan, Boyang Deng, Paul Debevec, William T Freeman, and Jonathan T Barron. Nerfactor: Neural factorization of shape and reflectance under an unknown illumination. *ACM Transactions on Graphics (TOG)*, 40(6):1–18, 2021. 1, 4

[66] Yuanqing Zhang, Jiaming Sun, Xingyi He, Huan Fu, Rongfei Jia, and Xiaowei Zhou. Modeling indirect illumination for inverse rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18643–18652, 2022. 2

[67] Matthias Zwicker, Wojciech Jarosz, Jaakko Lehtinen, Bochang Moon, Ravi Ramamoorthi, Fabrice Rousselle, Pradeep Sen, Cyril Soler, and S-E Yoon. Recent advances in adaptive sampling and reconstruction for monte carlo rendering. In *Computer graphics forum*, volume 34, pages 667–681. Wiley Online Library, 2015. 2

A. Optimization and Regularization

A.1. Image Loss

Our renderer employs physically-based shading techniques to create high dynamic range images. Therefore, the objective function needs to be able to handle a broad spectrum of floating-point values. To achieve this, we have adopted the approach of recent research in differentiable rendering [25,43]. Our image space loss, L_{image}, calculates the L_1 norm on tonemapped colors. We use the tone map operator, T, to transform linear radiance values, x, which follows the formula $T(x) = \tau(log(x+1))$. Here, $\tau(x)$ represents the sRGB transfer function.

$$\tau(x) = \begin{cases} 12.92x & x \leq 0.0031308 \\ (1+a)x^{1/2.4} - a & x > 0.0031308 \end{cases} \quad (4)$$
$$a = 0.055$$

A.2. Regularizers

In our particular setting, which involves multi-view images captured under constant lighting, it is necessary to use several prior assumptions in order to guide the optimization process towards achieving a satisfactory separation of the underlying factors such as geometry, materials, and lighting. While we would prefer to minimize the use of regularization, we rely on various smoothness regularizers for parameters such as albedo, specular parameters, and geometric surface normal, as mentioned next:

$$L_{\boldsymbol{k}_d} = \frac{1}{|\mathbf{x}_{\text{surf}}|} \sum_{\mathbf{x}_{\text{surf}}} |\boldsymbol{k}_d(\mathbf{x}_{\text{surf}}) - \boldsymbol{k}_d(\mathbf{x}_{\text{surf}} + \boldsymbol{\epsilon})| \quad (5)$$

The term x_{surf} refers to a position in world space located on the surface of the object, and a small random displacement vector denoted by $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma = 0.01)$ and following a normal distribution with mean 0 and standard deviation of 0.01 is added to this position. k_d refers to the albedo texture map. Regularizing the geometric surface normal, prior to the application of normal map perturbations, is a novel approach compared to [43] and serves to promote smoother geometry, particularly during the early stages of training.

Additionally, we note that normal mapping, which involves perturbing surface normals through a texture lookup, also benefits from regularization. While normal mapping can be a powerful tool for simulating local micro-geometry, the decorrelation between geometry and surface normal can sometimes pose problems. We observed that with normal mapping enabled, the environment light was incorrectly used as a color dictionary, where normal perturbations reoriented the normals of a geometric element in order to look up a desired color. To mitigate this issue, we employ the following regularizer. Given a normal perturbation n' in tangent space, our loss function is defined as:

$$L_{\mathbf{n}'} = \frac{1}{|\mathbf{x}_{\text{surf}}|} \sum_{\mathbf{x}_{\text{surf}}} 1 - \underbrace{\frac{\mathbf{n}'(\mathbf{x}_{\text{surf}}) + \mathbf{n}'(\mathbf{x}_{\text{surf}} + \boldsymbol{\epsilon})}{|\mathbf{n}'(\mathbf{x}_{\text{surf}}) + \mathbf{n}'(\mathbf{x}_{\text{surf}} + \boldsymbol{\epsilon})|}}_{\text{Half-angle vector}} \cdot (0,0,1) \quad (6)$$

Intuitively, we enforce that normal perturbations, modeling micro-geometry, randomly selected in a small local area, have an expected value of the unperturbed tangent space surface normal, (0, 0, 1). As mentioned in the paper, we additionally regularize based on monochrome image loss between the demodulated lighting terms and the reference image:

$$L_{\text{light}} = |Y(T(\mathbf{c}_d + \mathbf{c}_s)) - V(T(I_{\text{ref}}))|_1 \quad (7)$$

Here, T is the tonemap operator described in the previous paragraph. c_d is demodulated diffuse lighting, c_s is specular lighting, and I_ref is the target reference image. Monochrome images are computed through the simple luminance operator, Y (x) = $(x_r + x_g + x_b)$ /3, and HSV-value, V (x) = max (x_r, x_g, x_b). While the regularizer is limited by our inability to demodulate the reference image, it greatly increases lighting detail, which is particularly effective in datasets with high frequency lighting. We compute the final loss as a weighted combination of the image loss and regularizer terms:

$$L = L_{\text{image}} + \underbrace{\lambda_{\boldsymbol{k}_d}}_{=0.1} L_{\boldsymbol{k}_d} + \underbrace{\lambda_{\boldsymbol{k}_{\text{orm}}}}_{=0.05} L_{\boldsymbol{k}_{\text{orm}}} + \underbrace{\lambda_{\mathbf{n}}}_{=0.025} L_{\mathbf{n}} + \underbrace{\lambda_{\mathbf{n}'}}_{=0.25} L_{\mathbf{n}'} + \underbrace{\lambda_{\text{light}}}_{=0.15} L_{\text{light}} \quad (8)$$

A.3. Optimization Details

To optimize the parameters for geometry, material, and lighting, we use the Adam optimizer with default settings. Our approach for parameterizing geometry involves representing SDF values on a three-dimensional grid, with a perturbation vector assigned to each grid-vertex. Material and lighting are encoded in textures or high-frequency functions that are encoded through a neural network such as an MLP with positional encoding. We typically use varying learning rates for geometry, material, and lighting, with lighting having the highest learning rate and material having the lowest. Our approach is based on the publically available codebase of [43], and we closely follow their methodology for details. Note that we can capture high-frequency lighting details for specular objects.

It's worth noting that the initial stages of optimization, which begin with random geometry and involve significant

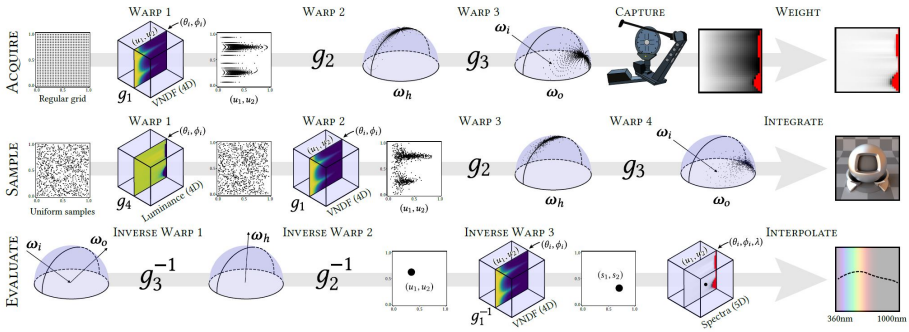


Figure 12. Adaptive Parameterization [13]

topology changes, are crucial in obtaining a satisfactory final result. However, the shadow test poses a challenge, as modifications to the geometry can drastically alter the lighting intensity in previously shadowed areas, leading to the optimization process becoming stuck in bad local minima. However, there is a problem, where the optimizer is unable to carve out the shape of the object. To address this issue, we progressively introduce the shadow term as optimization progresses.

We compute the color according to the rendering equation:

$$L(\omega_o) = \int_\Omega L_i(\omega_i) f(\omega_i, \omega_o)(\omega_i \cdot \mathbf{n}) d\omega_i \tag{9}$$

where $L_i(\omega_i) = L_i^{'}(\omega_i) H(\omega_i)$ can be separated into a lighting term, $L_i^{'}(\omega_i)$, and visibility term,$H(\omega_i)$. Rather than using binary visibility, we introduce a light leakage term, τ , and linearly fade in the shadow contributions over the first 1750 iterations:

$$H(\omega_i, \tau) = \begin{cases} 1 - \tau & \text{if intersect ray}(\omega_i) \\ 1 & \text{otherwise} \end{cases} \tag{10}$$

It is noted that gradually blending in the shadow term has a large impact on early convergence. In particular, since we start from a random topology, carving out empty space or adding geometry may have a large impact on overall shading, causing spiky and noisy gradients. The denoiser may also interfere with early topology optimization (blurred visibility gradients). This is particularly prominent when the denoiser parameters are trained along with scene parameters. Therefore, for neural denoisers, which have no easily configurable filter width, we instead linearly blend between the noisy and denoised images to create a smooth progression.

B. BRDF Model Details

MLP As mentioned in the main paper, the MLP consists of 4 fully-connected layers with ReLU activation and a hidden dimension of 128. We use 2 frequencies of the sine-cosine encoding [59] to encode our viewing angles. The latent code is 3-dimensional and initialized with a zero-mean Gaussian distribution and a standard deviation of 0.01.

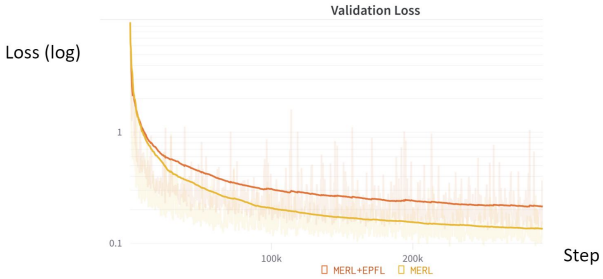


Figure 13. Pretrain result of EPFL+MERL

Normalizing Flow For details about the iBRDF model [12], please refer to the respective paper. In our adaptation, we use 4 NICE layers [51] and a latent dimension of 16, which is initialized with the same Gaussian as the MLP. We normalize ω_i in the $[0, 1]$ range, while ω_o is encoded with 8 frequencies of sine and cosine. To further reduce the model parameters, we shrink the size of the U-Net within each layer. While the iBRDF model [12] consists of around 1 million parameters, ours has only about 500k, while still achieving similar quantitative results.

Optimization To train the BRDF models, we use the Adam optimizer with an initial learning rate of 0.001. The learning rate is reduced every 10,000 steps by a factor of 0.1. A single batch consists of 4 different materials with 8192 samples each and we train for 5,000 epochs in total.

C. Denoising

Diffusion models are latent variable models of the form $p_\theta(\mathbf{x}_0) := \int p_\theta(\mathbf{x}_{0:T})\, d\mathbf{x}_{1:T}$, where $x_1, ..., x_T$ are latents of the same dimensionality as the data x_0 $q(x_0)$. The joint distribution $p_\theta(x_{0:T})$ is called the reverse process, and is

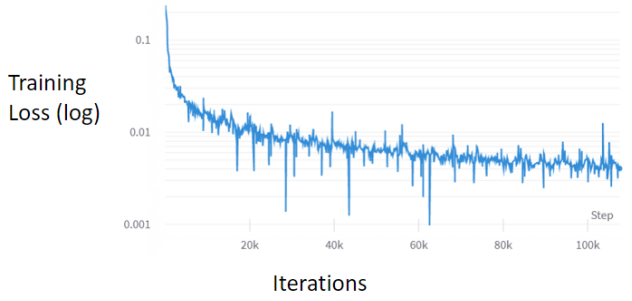


Figure 14. Loss curve of the diffusion model.

defined as a Markov chain with learned Gaussian transitions starting at
$p\left(\mathbf{x}_T\right)=\mathcal{N}\left(\mathbf{x}_T;\mathbf{0},\mathbf{I}\right)$:

$$p_\theta\left(\mathbf{x}_{0:T}\right):=p\left(\mathbf{x}_T\right)\prod_{t=1}^{T}p_\theta\left(\mathbf{x}_{t-1}\mid\mathbf{x}_t\right),$$
$$p_\theta\left(\mathbf{x}_{t-1}\mid\mathbf{x}_t\right):=\mathcal{N}\left(\mathbf{x}_{t-1};\boldsymbol{\mu}_\theta\left(\mathbf{x}_t,t\right),\boldsymbol{\Sigma}_\theta\left(\mathbf{x}_t,t\right)\right) \tag{11}$$

What distinguishes diffusion models from other types of latent variable models is that the approximate posterior $q(x_{1:T}|x_0)$, called the forward process or diffusion process, is fixed to a Markov chain that gradually adds Gaussian noise to the data according to a variance schedule $\beta_1,...,\beta_T$:

$$q\left(\mathbf{x}_{1:T}\mid\mathbf{x}_0\right):=\prod_{t=1}^{T}q\left(\mathbf{x}_t\mid\mathbf{x}_{t-1}\right),$$
$$q\left(\mathbf{x}_t\mid\mathbf{x}_{t-1}\right):=\mathcal{N}\left(\mathbf{x}_t;\sqrt{1-\beta_t}\mathbf{x}_{t-1},\beta_t\mathbf{I}\right) \tag{12}$$

D. PyTorch Lightning

Pytorch Lightning offers a standardized training loop, ensuring reproducible results across experiments and machines, as well as scaling training across multiple GPUs or machines. Additionally, it provides debugging and testing tools, such as automated testing and validation checks, and a lightweight model checkpoint format for easier model deployment. Overall, PyTorch Lightning offers a more efficient and scalable way to develop and train neural networks.