HAL
open science

Learning radiance fields : from global illumination to generative models

Stavros Diolatzis

To cite this version:

Stavros Diolatzis. Learning radiance fields : from global illumination to generative models. Graphics [cs.GR]. Université Côte d'Azur, 2022. English. NNT : 2022COAZ4041 . tel-03814937

HAL Id: tel-03814937

https://theses.hal.science/tel-03814937v1

Submitted on 14 Oct 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITÉ CÔTE D'AZUR

ÉCOLE DOCTORALE
SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

THÈSE DE DOCTORAT

Apprentissage de champs de radiance: de l'éclairage global aux modèles génératifs

Stavros Diolatzis

Inria Sophia Antipolis-Méditerannée

Présentée en vue de l'obtention du grade de docteur en Informatique d'Université Côte d'Azur
Dirigée par : George Drettakis
Soutenue le : 24/06/2022

Devant le jury composé de :
Frederic Precioso, Professor, Inria & Université Côte d'Azur
George Drettakis, Research Director, Inria & Université Côte d'Azur
Matthias Zwicker, Professor, University of Maryland
Hendrik Lensch, Professor, Eberhard Karls University Tübingen
Tobias Ritschel, Professor, University College London

Inria

Apprentissage de champs de radiance: de l'éclairage global aux modèles génératifs

---

Learning Radiance Fields: From Global Illumination to Generative Models

Jury:

Président du jury / President of the jury

Frederic Precioso, Professor, Inria & Université Côte d'Azur

Rapporteurs / Reviewers

Matthias Zwicker, Professor, University of Maryland

Hendrik Lensch, Professor, Eberhard Karls University Tübingen

Examinateur / Examiner

Tobias Ritschel, Professor, University College London

Directeur de thèse / Thesis supervisor

George Drettakis, Research Director, Inria & Université Côte d'Azur

Acknowledgements

I would like to thank all the people who helped and supported me for the duration of this thesis. First I would like to thank my supervisor George Drettakis who always believed in me and was there to support me with ideas and insight for each one of the projects I worked on. Even though, like in every thesis, there were times where things didn't work out, he always was on my side and helped me overcome these difficulties.

I would also like to thank my partner Marina for her love and support during this PhD. While spending the Covid-19 lockdowns with me during the different deadlines was probably not ideal she was always there to hear me complain about my results and always make me see the bright side. Thank you Marina.

I thank my mother for her constant support through every stage of my life and for encouraging me to pursue my dreams even if it meant spending the majority of my time very far away.

Finally I would like to thank all the members of the GraphDeco group past and current. All of you are bright people and great researchers. We got to share the burden of doing a PhD and help each other and I thank you for being there.

Résumé

La création d'images réalistes de scènes virtuelles est un processus qui implique de simuler les interactions de la lumière, ce qui est fait traditionnellement par des méthodes de tracer de chemins et de Monte Carlo, la lumière étant transmise et réfléchie avant d'atteindre la caméra virtuelle. Ce processus est largement utilisé dans différents secteurs tels que le cinéma, les jeux vidéo, les simulations physiques et la conception architecturale. Les méthodes de Monte Carlo, dans un contexte de tracer de chemins, peuvent gérer des effets d'éclairage complexes, mais les images résultantes sont bruitées ; pour réduire le bruit il faut simuler des chemins supplémentaires. Ce calcul peut être coûteux ; de nombreuses recherches ont été menées pour le rendre plus efficace et plus précis. Si, par le passé, les méthodes se sont concentrées sur l'amélioration de la qualité d'échantillonnage du tracer de chemins, les réseaux neuronaux ont récemment gagné en popularité comme moyen de rendre des scènes synthétiques ou capturées. Cette évolution vers un pipeline de rendu augmenté par les réseaux de neurones se reflète dans les méthodes proposées dans cette thèse où de plus en plus d'aspects du rendu sont traités par les réseaux de neurones. Un élément clé de cette tâche est le choix de la représentation de la scène, avec de nombreuses alternatives proposées. Nous démontrons que les champs de radiance sont une bonne solution car ils peuvent être utilisés pour réduire le bruit dans le tracer de chemins traditionnel et être appris efficacement par les réseaux neuronaux. Tout d'abord, nous proposons une méthode permettant d'injecter notre connaissance des matériaux de la scène dans une approximation des champs de radiance afin d'améliorer l'échantillonnage, en particulier dans les scènes comportant des matériaux brillants. Ensuite, nous montrons que lors de l'apprentissage d'un réseau pour représenter les champs de radiance pour des scènes variables, un échantillonnage uniforme des configurations de la scène conduit à de mauvais résultats. Au lieu de cela, nous explorons activement l'espace des configurations de scènes possibles et utilisons le réseau pour rendre de manière interactive des scènes variables avec des effets complexes, tels que les caustiques. Même si nous utilisons un réseau pour le rendu final, notre vecteur explicite de représentation de la scène préserve le contrôle artistique sur les objets, les matériaux et les émetteurs de la scène. Enfin, nous développons un modèle génératif pour les matériaux à méso-échelle avec une structure

et une apparence complexes. Ici, nous utilisons des champs de radiance volumétriques et nous conditionnons notre réseau à des paramètres de géométrie et d'apparence pour un contrôle artistique des matériaux représentés, ce qui est crucial dans notre contexte.

Mots-clés: Tracer de chemins, Monte Carlo, rendu neuronal, champs de radiance

Abstract

Creating realistic images of virtual scenes is a process that involves simulating light interactions, traditionally through path tracing and Monte Carlo methods, as light gets transmitted and reflected before reaching the virtual camera. This process is being used extensively in different industries such as movies, video games, physical simulations and architectural design. Monte Carlo methods, in a path tracing context, can handle complex lighting effects but the resulting images are plagued with noise which is reduced by simulating additional paths. This can be computationally expensive and a lot of research has gone into making it more efficient and accurate. While methods in the past have focused on improving the sampling quality of path tracing, recently neural networks have gained popularity as a way to render synthetic or captured scenes. This shift towards a neural augmented rendering pipeline is reflected in the methods proposed in this thesis where increasingly more aspects of rendering are handled by neural networks. A key component in this task is the scene representation of choice with many alternatives being proposed. We demonstrate how radiance fields are a good fit as they can be used to reduce noise in traditional path tracing and also be learned efficiently by neural networks. First we propose a method to inject our knowledge of the scene materials in an approximation of radiance fields to improve sampling, especially in scenes with glossy materials. Next we show that when training a network to represent radiance fields for variable scenes, uniform sampling of the scene configurations leads to poor results. Instead we actively explore the space of possible scene configurations and use the network to interactively render variable scenes with hard effects, such as caustics. Even though we use a network for the final rendering, our explicit scene representation vector preserves artistic control over the scene's objects, materials and emitters. Finally we develop a generative model for mesoscale materials with complex structure and appearance. Here we use volumetric radiance fields and we condition our network on geometry and appearance parameters for artistic control of the materials represented, which is crucial in our context.

Keywords: Path Tracing, Monte Carlo, Neural Rendering, Radiance Fields

Contents

Contents vii

1 Introduction 1
- 1.1 Representing & Rendering Synthetic Scenes . . . 6
- 1.2 Contributions . . . 7
- 1.3 Funding and Publications . . . 9

2 Background 11
- 2.1 Radiometry . . . 11
- 2.2 Radiance Fields & Rendering Equations . . . 12

3 Previous Work 15
- 3.1 Traditional Path Tracing . . . 15
- 3.2 Neural Rendering & Radiance Fields . . . 18
- 3.3 Generative Neural Radiance Fields . . . 21
- 3.4 Summary . . . 22

4 Practical Product Path Guiding Using Linearly Transformed Cosines 23
- 4.1 Practical Product Path Guiding . . . 24
- 4.2 Results and Evaluation . . . 32
- 4.3 Limitations and Future Work . . . 39
- 4.4 Conclusions . . . 40

5 Active Exploration for Neural Global Illumination of Variable Scenes 43
- 5.1 Related Work . . . 45
- 5.2 Overview . . . 47
- 5.3 Explicit Encoding and On-the-fly Data Generation . . . 48
- 5.4 Active Data Space Exploration . . . 50
- 5.5 Training and Self-Tuning Sample Reuse . . . 54
- 5.6 Results, Analysis and Comparisons . . . 57
- 5.7 Future Work, Limitations and Conclusion . . . 70

6 MesoGAN: A Generative Model for Mesoscale Materials 73
- 6.1 Related Work . . . 75
- 6.2 Method . . . 76
- 6.3 Results . . . 86

6.4 Comparisons 87
6.5 Conclusions 88

7 Conclusions **89**
7.1 Lessons learned 90
7.2 Future directions 91

A Chapter 5 appendix **95**
A.1 Selected Views 95
A.2 Comparison to CNSR 96
A.3 Comparison to ANF 97
A.4 Comparison to GT 97
A.5 Network Architecture 99
A.6 MCMC States Lifespan 100
A.7 Sample Reuse Derivation 100

Chapter 1

Introduction

Computer graphics is an integral part of any application that requires visualization of synthetic or captured content. From architectural design to movies, video games and physical simulations, creating images of synthetic scenes is part of many industries with different restrictions and requirements. Computer graphics research develops the theory behind light transport simulation, so that the results are accurate, and proposes efficient algorithms to minimize computation. This thesis is focused on physically based rendering where the generated images must look as realistic as possible. Many challenges arise in this task, from creating physically based material models to modeling geometry and efficiently computing the final image. An example of the complex interactions that light can go through before reaching the camera are visible in the generated image of Figure 1.1. For complicated scenes, such as this one, generating a single image can take up to tens of hours of computation.

The well-established way to solve light transport problems has been through the use of Monte Carlo methods [54]. These methods, while being very general in terms of effects that they can handle, suffer from noise in the generated images. This noise needs more computation to be reduced, sometimes requiring hours to become imperceptible. It is also very hard to filter out since it can vary from pixel to pixel depending on the underlying effect. This means that in many cases the only option for getting a noiseless image is more computation.

This was the state of rendering for many years, where methods were trying to improve performance by generating images with less noise for the same time budget. Even small improvements in the generated noise were important since applications such as movies render hundreds of thousands of images in total. Two major events changed this over the 3 years of this PhD. First, the resurgence of neural networks in the form of deep learning and neural representations. Second, the development of dedicated ray-tracing hardware that made real-time path tracing possible. Neural networks have been used more and more in the rendering pipeline, augmenting some of its parts and replacing others. They

Figure 1.1: An image generated by a modern path tracer. To render this image the interactions of light with the scene must be simulated, including multiple transmissions and reflections before it reaches the sensor. Image from [95] .

also play a big role in real-time path tracing, solving issues that arise due to the limited computing budget. This work is testament to this shift to a neural augmented rendering pipeline and the methods presented reflect this.

Neural Networks in Rendering. Some of the earliest applications of shallow neural networks in computer graphics included denoising [47] and shading/rendering [94]. Deeper neural networks were introduced a few years later, thanks to improvements in GPU hardware and machine learning frameworks, again for denoising [3][12] and shading [79] but with much better performance (Figure 1.2). The introduction of hardware based path tracing in combination with this improved denoising made real time path tracing a reality albeit for simpler effects.

At the beginning of this thesis the first neural rendering methods were introduced, using neural networks directly to generate the final image, in both inverse [34] and forward tasks [25]. Neural rendering very quickly became an active and vast research field with every part of the traditional rendering pipeline being tested against the performance of neural networks. Early attempts at neural rendering for forward tasks were limited to low resolutions and simple scenes but improvements introduced in follow-up work and

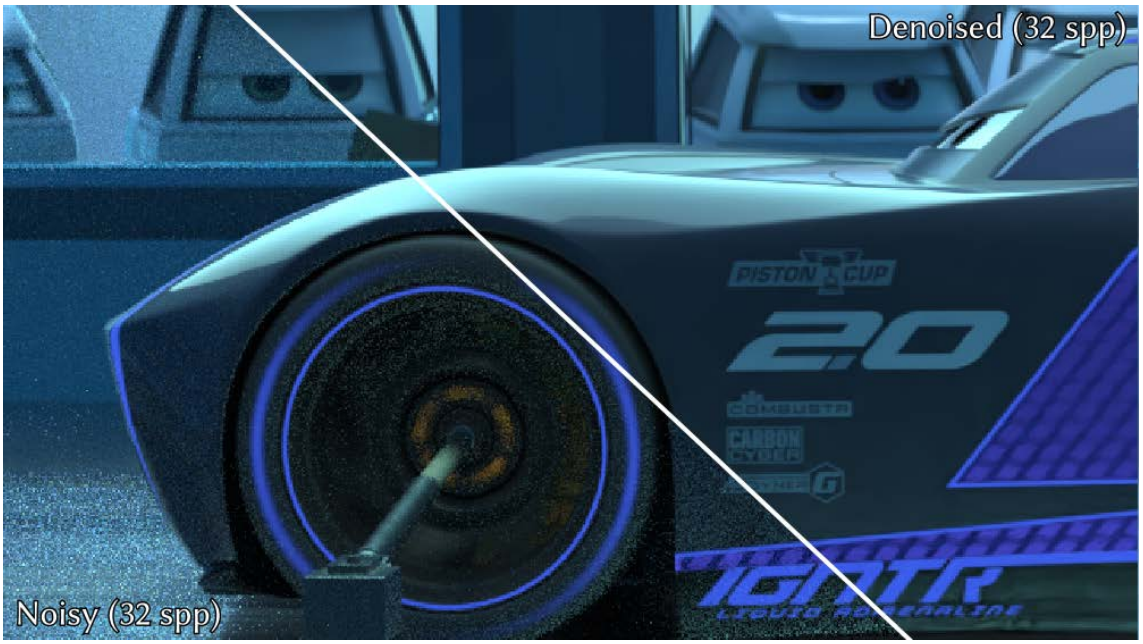


Figure 1.2: Deep neural networks can be trained to remove noise inherent in images generated using Monte Carlo methods. Image from [3].

in this thesis show that neural rendering offers some clear advantages over path tracing with denoising. Looking into the future, a neural augmented rendering pipeline could be the way forward to inject more complex effects in real-time path tracing.

One important advantage of neural rendering methods is that in many cases it doesn't need explicit modeling of geometry. This is a big advantage especially for objects where creating realistic geometry is challenging. Faces for example are extremely hard to model since they are very important in our interactions as humans and even the smallest detail can make a synthetic face look fake or unsettling. Another class of objects that is hard to model are mesoscale materials such as fur, grass or vegetation. In nature these materials have very complex geometry and their arrangement is stochastic which means that modeling one exemplar and repeating it is not an option. An important body of work and the next frontier for rendering such objects is Generative Adversarial Networks (GANs). These models involve a training scheme that is centered around a zero-sum game between a generator and a discriminator. GANs were introduced in 2014 [26] and in the following years the quality and resolution of their results improved rapidly [49]. The introduction of StyleGAN [50], close to the start of this PhD, established GANs as a powerful rendering tool for faces by demonstrating unprecedented quality and,

Figure 1.3: Images of faces generated by StyleGAN [50]. Styles from each column and row can be combined to create new realistic images that inherit features from both.

inspired by style transfer literature, the use of styles to control different aspects of the final appearance. StyleGAN was improved in multiple following iterations [51][52], each one introducing higher quality and less artifacts. The missing component for GANs to be introduced in traditional pipelines was explicit camera control. Many methods tackled this issue by trying to disentangle pose from the latent space [82][61] but a more appealing approach was using generative models to learn volumetric representations that are rendered into images through an explicit volumetric path tracing step [99][15].

Finally, it is interesting to look at the evolution of neural network architectures in computer graphics throughout the years of this PhD. In earlier years and for tasks such as denoising, convolutional neural networks and their variants were the main choice of architecture. At the time, networks were mainly seen as learnable image operations, where convolutions can be very efficient. This changed when new methods used neural networks as representations of occupancy grids [70], SDFs [85] and others. In such cases Multilayer Perceptrons (MLPs) became popular as now the input was no longer a 2D image but a tensor of positions, directions etc. The concept of using an MLP as an implicit representation by overfitting to a single scene/object was introduced under many different

contexts [70; 85; 104]. NeRF [71] showed the potential of such an implicit representation in computer graphics to render novel views of a captured scene. It demonstrated that an MLP can be trained to represent color and density for each point in the scene, and this very compact representation can be combined with volume path tracing for impressive results. This work also brought attention to the importance of encoding the input [107] to assist the network in creating high frequency details.

Context & Research Questions. All these breakthroughs impacted and inspired different aspects of the methods contained in this thesis; but our view is different compared to a lot of work in the field of neural rendering. We focus only on the forward physically based rendering scenario. In this scenario we have distinct advantages but also different expectations from neural rendering. One of the biggest advantages is the explicit control over the synthetic data generation. We are not restricted by capturing the real world and all the challenges this entails. Instead we are aware of the exact geometry, materials and details of our scenes. In addition, since the forward problem has been studied for decades in order to simulate it, we have extensive knowledge of its workings and a plethora of previous methods that have attempted to solve it. This provided inspiration and insight about how to solve the light transport problem but with the extra capabilities of neural networks. On the other hand, neural rendering needs to either compete with traditional, and now real-time, path tracing or it has to be able to become part of it to improve it. Another important requirement, unique to our scenario, is that our methods are targeted towards creating synthetic content and providing explicit control over the outcome to the creators is crucial. This is especially challenging with networks since many aspects of their workings involve intermediate latent spaces where extra effort is necessary to find semantic meaning. For us, artistic control is non negotiable and it needs to be explicit to meet the requirements of the target applications. These insights led to the research questions we addressed in this thesis, which we present next:

- How can we use the explicit nature of a radiance field representation to improve sampling by combining it with our knowledge of the scene materials?
- How can we *efficiently* train a network to learn an accurate radiance field representation in scenes modifiable by artists?
- How can GANs be used to learn a *distribution* of volumetric radiance fields repre-

senting 3D neural textures of mesoscale materials in the context of traditional path tracing while retaining artistic control?

1.1 Representing & Rendering Synthetic Scenes

A synthetic scene is typically defined by geometry, materials, emitters and cameras (Figure 1.4). Geometry is traditionally defined through triangular meshes which are modeled by artists or reconstructed from captured content. Materials are defined through different models that try to approximate how matter reacts to light. These models are represented by a Bidirectional Scattering Distribution Function (BSDF) for surfaces or a Bidirectional Transmittance Distribution Function for volumes. These functions give the amount of light reflected depending on the directions of the ray before and after the interaction with the material. Each surface point is associated with a material which defines its final appearance, varying from diffuse to more glossy and specular. Different types of emitters are placed in the scene which can be point lights, area lights or environment maps simulating distant incoming light from the broader environment but also complex light fixtures. Finally we place a camera to define the viewpoint from which the scene will be rendered.

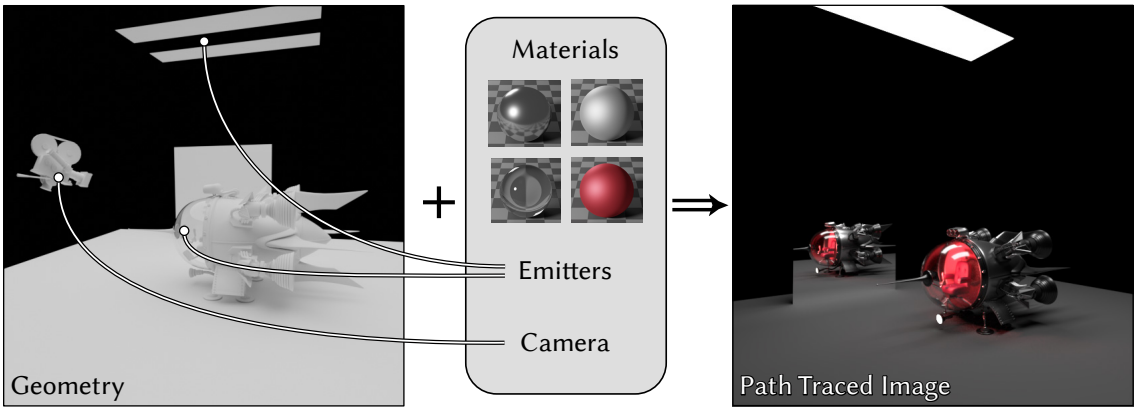


Figure 1.4: A traditional representation of a scene includes geometry, materials, emitter and cameras. The components of this representation define the look of the rendered image, which we produce by tracing thousands of paths from the camera into the virtual scene.

In order to create the image for a given camera in a synthetic scene we need to calculate how much energy is reflected towards the sensor from each visible point in the scene. This quantity, known as radiance, is estimated through path tracing, by shooting virtual rays from the camera into the scene. The scattering events during this process

are sampled at random using Monte Carlo methods, involving choices for the reflected direction, chosen material component, point on a light source to connect to and others. While these choices are random they can be *importance sampled* using our knowledge of the scene's materials, geometry, light sources etc. For example, the most common way to do importance sampling is using the scene materials which are available at the points of intersection. Light coming from the directions around the glossy lobe of a material will contribute more to the final image so shooting rays in those directions is more efficient. But this form of importance sampling is missing the global state of the scene where a lot of light might be coming from less important directions, resulting in significant contribution.

In the scope of this thesis we assume that, for a given scene, the light distribution has reached an equilibrium and is constant over time. In this case each point in the scene is receiving a constant amount of radiance from each direction. This can be represented by what we will later define as a radiance field in Section 2.2. By building representations of radiance fields we can improve rendering in multiple ways. Coarse approximations can be used to effectively importance sample the directions of paths and guide them towards light sources, thus reducing noise. More accurate representations allow us to compute how much light is reflected into the camera, creating the image directly. Radiance fields in volumes can also describe geometry implicitly, making them a very flexible and powerful representation.

1.2 Contributions

The goal of this thesis is to propose three different ways in which radiance fields can improve rendering performance. First we utilize more traditional data structures such as octrees and quatrees for our radiance field approximation. We show how these structures can be combined with our knowledge of the scene materials for improved sampling. Next we move to neural networks which we train in a precomputation step to build our approximations. These approximations are of much higher quality than when using traditional data structures. This allows us to use the trained neural network to render the image, replacing a traditional path tracer. In this context we demonstrate how efficient data generation has a big impact on the quality of the learned representation. Finally we turn our attention to materials with complex geometry and demonstrate that they can be represented and rendered through a generative model approximating volumetric

radiance fields.

The contributions of this thesis are:

- In Chapter 4 we present Practical Product Path Guiding using Linearly Transformed Cosines. In this method we build an approximation of the incoming radiance field, similar to previous work, but we combine it with information about the surface material during sampling, thus improving performance and reducing noise. To achieve this we use Linearly Transformed Cosines as a way to compute the material's response over a spherical polygon. This along with optimizations in the form of precomputation and vectorization allow us to perform product sampling on-the-fly with minimal overhead.

- In Chapter 5 we present Active Exploration for Neural Global Illumination of Variable Scenes. We demonstrate that neural networks can be used to turn rasterized geometry buffers into global illumination images even for scenes with variability. To describe the current state of the scene we create an explicit representation vector that describes the state of variables at inference. This leads us to define the space of all possible configurations in which some parts are more difficult to render than others. As a result we propose to explore this space by using Markov Chain Monte Carlo and rendering images that better help the network converge. At test time we can handle very complex effects such as caustics, at interactive rates, which cannot be captured by real time path tracing with denoising. Our scene representation vector also ensures explicit control of objects, materials and emitters which is crucial in a production environment.

- In Chapter 6 we present Meso-GAN: A Generative Model for Mesoscale Materials. Mesoscale materials such as fur, grass etc. can be quite expensive to render due to their very complex geometry. We propose the use of a Generative Adversarial Network trained on synthetic data for these types of materials to generate 3D neural textures on an infinite plane. For a given object our method allows the creation of a high resolution neural texture that is lifted into 3D and is ray marched to generate the final image. These neural textures include stochastic detail, retain explicit artistic control over their appearance and can be integrated into a path tracer through the use of shell mapping.

1.3 Funding and Publications

The work in this thesis was funded by the ERC Advanced Grant No.788065 FUNGRAPH[1]. The Meso-GAN project was partially conducted when the author was interning at Nvidia.

The work in this thesis has led to two first author publication in international venues and a first author project still under review:

- Practical Product Path Guiding using Linearly Transformed Cosines [23]
  Eurographics Symposium on Rendering 2020
- Active Exploration for Neural Global Illumination of Variable Scenes [24]
  ACM Transactions on Graphics (TOG)
- Meso-GAN: A Generative Model for Mesoscale Materials
  In preparation

[1] `https://project.inria.fr/fungraph`

Chapter 2

Background

In this chapter we will describe the basic metrics of light transport, set up some important conventions to avoid ambiguities, review the rendering equations and finally define what we consider a *radiance field*. We will also distinguish different types of radiance fields before we explore how we use them in the following methods.

2.1 Radiometry

The main quantity emitted from light sources is energy (E) [J]. How this energy is distributed depends on the scene configuration and changes from scene to scene. The energy carried by a photon can be computed from its wavelength λ by:

$$E = \frac{hc}{\lambda} \quad , \tag{2.1}$$

where h is Planck's constant and c is the speed of light in a vacuum.

A related quantity is radiant flux (Φ) [$W = \frac{J}{s}$] (also known as power), which measures the amount of energy per unit time:

$$\Phi = \frac{\mathrm{d}E}{\mathrm{d}t} \tag{2.2}$$

In this work we consider that our scenes are in an equilibrium and as such the energy distribution over time is constant.

Since the incident radiant flux changes from surface to surface we are interested in the metric of irradiance (I) [$\frac{W}{m^2}$], defined as the amount of radiant flux for a unit of area:

$$I = \frac{\mathrm{d}\Phi}{\mathrm{d}A} \tag{2.3}$$

Irradiance measures the total flux arriving on an infinitesimal surface from all directions.

Since we are also interested in how radiant flux is changing for different directions we arrive at the main metric that we will be attempting to estimate, radiance (L) [$\frac{W}{m^2 sr}$]:

$$L = \frac{\mathrm{d}\Phi}{\mathrm{d}A^{\perp}\mathrm{d}\omega} \tag{2.4}$$

For computing radiance we consider incoming radiant flux for a unit of solid angle (steradian) and a unit of area perpendicular to that direction. This means that we need to project the infinitesimal area $\mathrm{d}A$ onto the perpendicular plane of direction $\mathrm{d}\omega$.

2.2 Radiance Fields & Rendering Equations

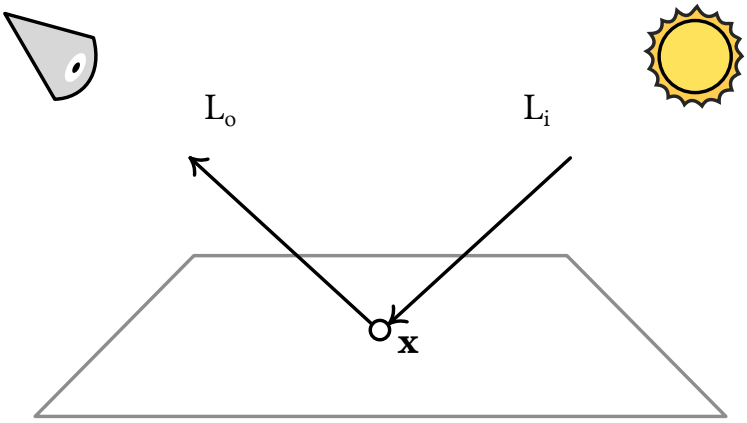


Figure 2.1: For an intersection point **x** we consider the radiance that is reflected towards the viewpoint as outgoing L_o. We name the radiance that arrives from a light source or other surfaces incoming L_i.

It is important to establish a convention (Figure 2.1) when we refer to radiance, distinguishing between it arriving on a surface and leaving from it, to avoid confusion when we examine surface interactions. Since in path tracing we are interested in radiance that is leaving the surface towards our viewpoint, we will refer to this direction as outgoing radiance L_o. For the radiance arriving onto a surface from a direction ω_i we will use the term incoming radiance L_i.

A *radiance field*[1] describes the distribution of outgoing or incoming radiance for a given scene. We define it as a scalar field $L(\mathbf{x}, \omega)$ that associates each point in a 5 dimensional space, $(\mathcal{V} \times \mathcal{S}^2)$ where $\mathcal{V}$ represents positions and $\mathcal{S}^2$ the sphere of directions,

[1]Similar to our definition of radiance fields are the terms light fields and plenoptic function [62]. The main difference between a light field and radiance field is that the former assumes that there are no occluders in the space around a main object, while a radiance field has no such restriction. The plenoptic function describes the same as what we call a volumetric radiance field but it is also associated with a light stage capturing setup that allows for some simplifications.

with a scalar representation of radiance. In the case of volumes it is common to add an extra scalar to the field which describes the medium's characteristics at each point $\mathbf{x}$. If a scene contains only surfaces and we assume there is no medium in between these surfaces then the scalar field is defined on the subset space that includes all positions on the surfaces $\mathcal{C} \subset \mathcal{V}$. Also in the case of surfaces we only consider the hemisphere of directions where the interaction happens, so the radiance field is defined in $\mathcal{C} \times \mathcal{H}^2$.

Depending on whether we are considering radiance leaving a surface or arriving on a surface we define the incoming and outgoing radiance fields $L_i(\mathbf{x}, \omega)$ and $L_o(\mathbf{x}, \omega)$. The main difference between the two fields is that the outgoing radiance field $L_o(\mathbf{x}, \omega)$ includes the interaction of all arriving light with the material of the point $\mathbf{x}$ as it gets reflected. So it implicitly includes a description of the material at the point of intersection. The radiance field $L_i(\mathbf{x}, \omega)$ only considers light arriving on the surface before it has interacted with it. A representation of an outgoing radiance field is more powerful since once computed we can create any novel view of the current scene configuration by sampling it on an image plane.

The outgoing and incoming radiance fields are associated through the *rendering equation* [46] when we consider only surfaces:

$$L_o(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + \int_{\mathcal{H}^2} L_i(\mathbf{x}, \omega_i)\, \rho(\mathbf{x}, \omega_o, \omega_i)\, \cos\theta_i \, \mathrm{d}\omega_i, \tag{2.5}$$

where L_o is outgoing radiance in direction ω_o towards an observer and L_e is the emitted radiance at the point $\mathbf{x}$ towards w_o. The BSDF ρ describes how much light is reflected towards direction ω_o from direction ω_i depending on the material at point $\mathbf{x}$. The integral is over the hemisphere of directions $\mathcal{H}^2$ around the point $\mathbf{x}$, and θ_i is the angle between the normal n at that point and direction ω_i.

Moving from surfaces to volumes we need to characterize the mediums that light interacts with. A medium can interact with photons by absorbing or scattering them. One coefficient is assigned to each of these interactions, the absorption σ_a $[m^{-1}]$ and the scattering coefficient σ_s $[m^{-1}]$ representing the probability density of each event per unit of distance. In our work we will not consider emissive media and as such will use only the scattering coefficient. The scattering coefficient σ_s can be used to compute the transmittance function $T(x_1, x_2)$ by integrating the loss of energy due to scattering between two points $\mathbf{x}_1, \mathbf{x}_2$:

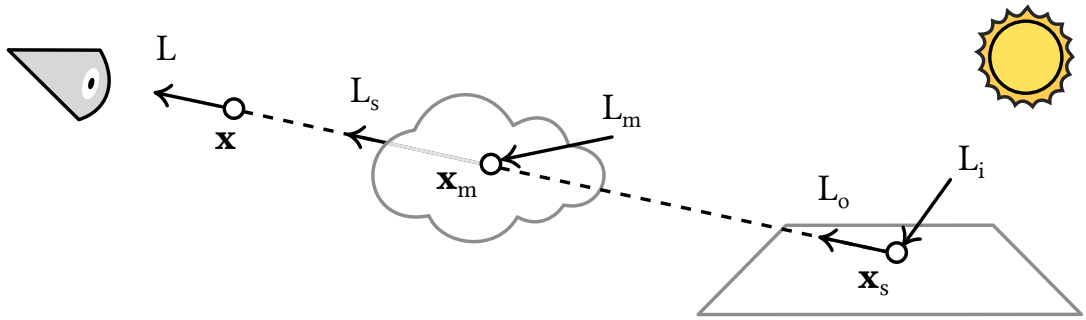


Figure 2.2: For a query point $\mathbf{x}$ and an intersection point $\mathbf{x}_s$ we consider the incoming L_i and outgoing radiance L_o as before. Now we also need to consider the radiance L_s that scattered towards the observer by the medium for any point $\mathbf{x}_m$ between the query point $\mathbf{x}$ and the intersection point $\mathbf{x}_s$.

$$T(x_1, x_2) = \exp^{-\tau(\mathbf{x}_1, \mathbf{x}_2)}, \quad \tau(\mathbf{x}_1, \mathbf{x}_2) = \int_0^{\|\mathbf{x}_2 - \mathbf{x}_1\|} \sigma_s(\mathbf{x}) \mathrm{d}x \tag{2.6}$$

Since now we take mediums into account, we need also examine the interactions at any point $\mathbf{x}_m$ between the query point $\mathbf{x}$ and the intersection point with the surface $\mathbf{x}_s$, as shown in Figure 2.2. These interactions are described by the *volume rendering equation* [16]

$$L(\mathbf{x}, \omega_o) = T(\mathbf{x}, \mathbf{x}_s) L_o(\mathbf{x}_s, \omega_o) + \int_0^s T(\mathbf{x}, \mathbf{x}_m) \sigma_s(\mathbf{x}_m) L_s(\mathbf{x}_m, \omega_o) \mathrm{d}x, \tag{2.7}$$

where $T(\mathbf{x}, \mathbf{x}_s) L_o(\mathbf{x}_s, \omega_o)$ describes radiance arriving from the surface and the integral over distance between $\mathbf{x}$ and $\mathbf{x}_s$ determines the total radiance scattered from the medium towards the viewpoint. Since σ_s describes the medium's interaction with light at each point $\mathbf{x}$ it can be considered as part of an extended volumetric radiance field $L(\mathbf{x}, \omega_o)^+$. This extended radiance field can be used to render an image by using quadrature methods, such as ray marching, or it can be used for importance sampling in Monte Carlo methods.

Chapter 3

Previous Work

The methods presented in this thesis are related to multiple different research fields and in this chapter we will discuss the most relevant to our work. We will first review related work from traditional path tracing and the use of radiance fields in path guiding and precomputation methods. Then we move to neural networks in rendering for inverse and forward problems and finally to neural networks as generative models.

3.1 Traditional Path Tracing

Unidirectional or 'simple' path tracing [46] is the algorithm in which paths are traced from the camera towards the scene and connected to emitters. Throughout the years many other variants of this basic method have been developed to improve sampling in cases of hard paths. These include bidirectional path tracing (BDPT) [111], metropolis light transport (MLT) [113], primary sample space MLT [53], manifold exploration [42] and others. All these variants improve drastically upon path tracing in some situations but at the same time introduce artifacts (splotchiness in MLT) and/or complexity (connecting paths in BDPT), as seen in Figure 3.1, which makes their use situational. The robustness and simplicity of unidirectional path tracing is the reason why it still is the standard method for rendering realistic physically based images in the film and visual effects industry [54]. An alternative option to improve the efficiency of path tracing is importance sampling. In the path tracing context, Monte Carlo integration involves randomly sampling and evaluating the integrand in the rendering equation (Equation 2.5). The sampling distribution or 'strategy' in this process affects the variance of the estimator, i.e., the noise in the resulting image. Importance sampling is the process of creating and sampling a distribution that approximates the integrand. The closer the sampling distribution is to the actual integrand the less variance and fewer samples needed to generate a noiseless image. Finding a sampling distribution that is both a good approximation and that can be sampled efficiently is challenging. In the rendering equation the factors that can be importance sampled are the incoming radiance L_i and the cosine weighted BSDF.

Although sampling based on the local cosine-weighted BSDF has been used since the conception of Monte Carlo rendering, doing so proportionally to the incoming radiance L_i is intricate since it requires solving the light transport problem itself.

Figure 3.1: a) The images on the left are the result of connecting different vertices of the light and camera path in BDPT. All these images must be carefully weighted using multiple importance sampling to form the image on the right. Image from [111]. b) The splochiness inherent in MLT is visible on the diffuse walls of the living room and artifacts that MLT can produce are pointed with arrows on the right. Image from [10].

Radiance field approximations have been used to introduce importance sampling according to L_i, i.e., to guide camera or light paths in a series of methods called Path Guiding. Vorba et al.'s method [115] iteratively built approximations of the radiance and importance fields by using Gaussian Mixture Models (GMMs) in a training step. However, the training step was computationally expensive and it was hard to evaluate the amount of training required, i.e., whether the current distributions were sufficiently converged. Reibold et al. [91] proposed the use of an outlier rejection algorithm to determine paths in a scene with high variance and apply guiding using GMMs only to those paths. In

this way the expensive guiding was used only where necessary. Another approach to overcome the computational expense was proposed by Müller et al. [74], who used a Spatial Directional tree (SD-tree) to efficiently approximate the incoming radiance field. The SD-tree consisted of a binary tree for the 3D spatial subdivision and a quadtree for 2D directional variation. This representation was refined by repeatedly rendering the scene with exponentially more samples. Dahm and Keller [20] used a similar data-structure but with a different update policy based on Q-learning. Path guiding has also been expressed in the primary sample space [31].

Path guiding made sampling according to L_i possible and added an extra sampling strategy to the options for importance sampling. When multiple sampling strategies are available they can be combined through multiple importance sampling (MIS), introduced by Veach and Guibas [112]. This combination of strategies works well when each strategy is a good approximation of the integrand for some part of the integral. However in some cases none of the strategies are ideal, for example BSDF sampling and path guiding in cases of glossy surfaces with complex visibility. In those cases creating and using a single strategy that approximates the total integrand, i.e., the *product* of incoming radiance and the BSDF can significantly improve sampling quality. Product sampling was originally used for direct illumination [18][17]. The product was also used to select cosine lobes used to represent the incident radiance field for indirect illumination computation [6], or with spherical harmonics to importance sample environment maps [43]. The GMM approach of Vorba et al. [115] was extended to compute product GMM importance sampling [36].

Radiance fields have also been used as a way to precompute and store global illumination that can be efficiently queried at runtime. Ward et al. [116] were among the first to suggest this to speedup rendering. In this work they considered only diffuse surfaces allowing them to ignore the directional component and store the total incoming irradiance in a spatial cache. This cache was used to infer the outgoing radiance by interpolating the values from neighboring points whenever possible. Photon mapping [44] can also be seen as an attempt at building a radiance field approximation to generate realistic images efficiently. In this method, Jensen [44] populated the scene with photons in a preprocessing stage and used density estimation [103] to approximate incoming radiance. In radiance caching, Krivánek et al. [58] expanded the approach from Ward et al. [116] by introducing the directional component using spherical harmonics and by proposing an efficient interpolation scheme using illumination gradients. This enabled

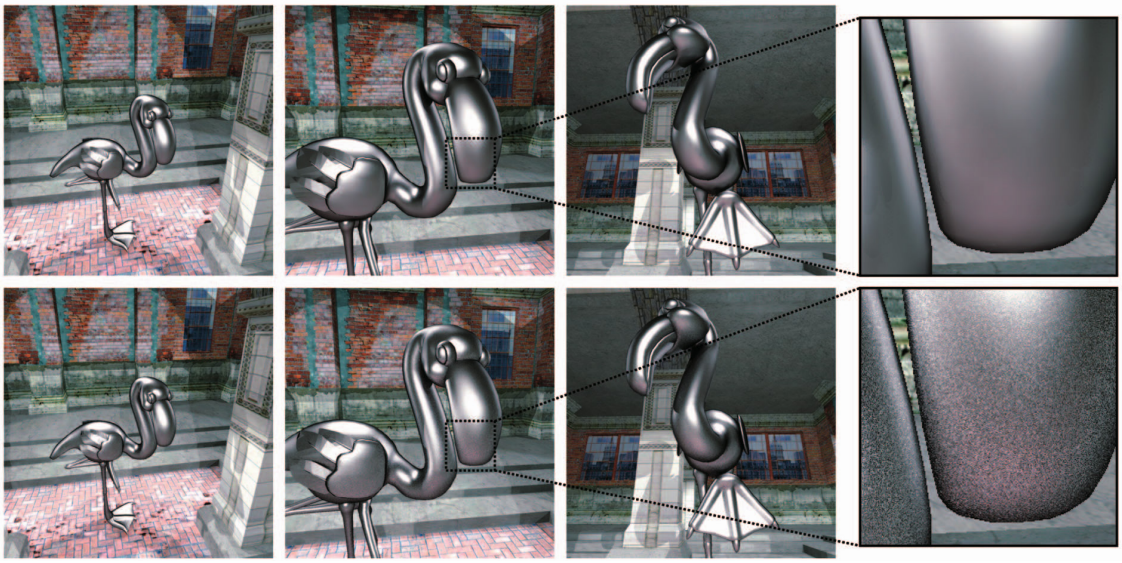


Figure 3.2: Caching radiance (top row) can be used to render novel viewpoints without path tracing and noise (bottom row) that is inherent in Monte Carlo methods. Image from [58]

interpolation for glossy materials such as the ones shown in Figure 3.2. More recently light probes [68; 96] have been used in a similar context by placing them in a scene and precomputing the incoming radiance for each one. At rendering time, information from the probes is used to interactively render a scene with global illumination. In all these cases dynamic scenes (moving objects or lights) are problematic since changes in the scene geometry affects the scene's radiance field. When such scenes are handled they are restricted to diffuse materials [66].

3.2 Neural Rendering & Radiance Fields

Radiance fields received a lot of attention recently as a scene representation in neural rendering, especially in the context of inverse problems, i.e., capturing and rendering real scenes. Sitzmann et al. [104] used neural networks to implicitly represent the scene by learning to associate any world space coordinate with a high dimensional feature vector. Here the networks are not forced to represent radiance but can encode any information necessary into the high dimensional vector. This vector was turned into an image through a learned ray marcher implemented as a LSTM network. NeRF [71] took a different approach by replacing the learned ray marching with explicit volumetric ray marching and the high dimensional feature vector with radiance and density, as

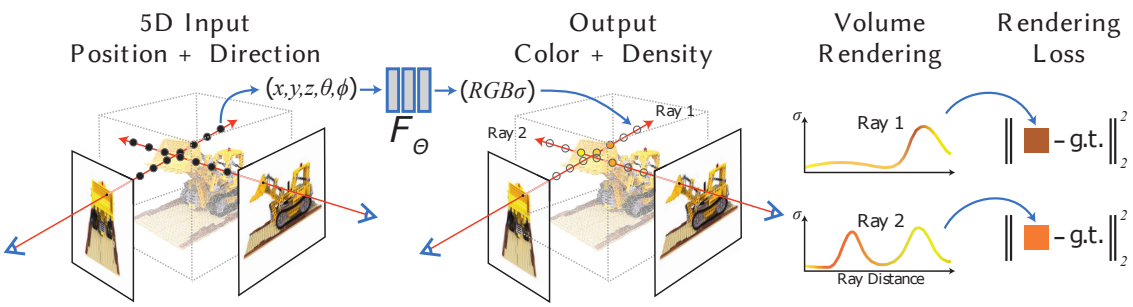


Figure 3.3: The training pipeline proposed by NeRF [71] where a network is trained to represent the 5 dimensional outgoing volume radiance field. The learned volumetric radiance field is rendered by using quadrature in the form of ray marching.

shown in Figure 3.3. These learned volumetric radiance fields proved to be a powerful representation enabling high quality novel view renderings after training on images of a real scene. The explicit rendering aspect and the robustness of the method led to many follow ups. One important follow up was the work by Barron et al. [5] which proposed a multi-scale representation for NeRF that supports filtered queries and results in anti-aliased renderings at all scales. A more detailed review of related neural rendering techniques is provided by the survey from Tewari et al. [108].

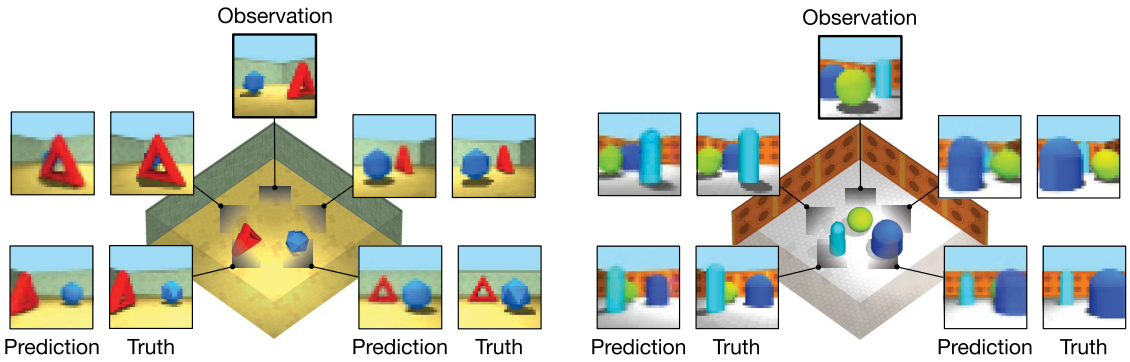


Figure 3.4: Scenes used by Eslami et al. [25] where a neural encoder compresses observations into a scene representation vector. This vector is used by the decoder to generate novel views of the scene. Image from [25].

In the context of rendering synthetic scenes, neural networks and radiance fields have been used for neural shading/rendering and neural importance sampling.

Ren et al. [93] used neural networks to inject global illumination into rasterized G-buffers. These networks can be seen as outgoing radiance field representations since they are trained to approximate it at any point in a scene, given information about

the geometry and materials. A later approach used neural networks in screen-space, rather than in world space, by learning shading effects including indirect lighting [78]. More recently, Eslami et al. [25] trained an encoder decoder network on variable scenes. Observations are encoded into a scene representation vector, which is then used to render the scene. The encoder, decoder and scene representation vector implicitly represent the scene including the radiance distribution, materials and geometry. This is a much more challenging task than learning only the radiance field and as a result the scenes used were simplistic and the training resolution low (Figure 3.4). Granskog et al. [28] expanded on this idea by using G-buffers to help the networks with explicit geometry and by enforcing structure on the neural scene representation. Baatz et al. [1] learned neural radiance fields to represent meso-scale structures such as fur or grass, a method that we will expand on.

The Neural Radiance Cache by Mueller et al. [77] is another case of a learned outgoing radiance field representation. In their method they use a small fully fused neural network which enables fast training and inference. This network is trained in an online fashion in a real-time path tracing context. Training neural networks to approximate incoming radiance can also be achieved by enforcing them to be consistent according to the rendering equation (Equation 2.5). This approach is demonstrated by Hadadan et al. [32] who minimize the rendering equation residual using a neural network.

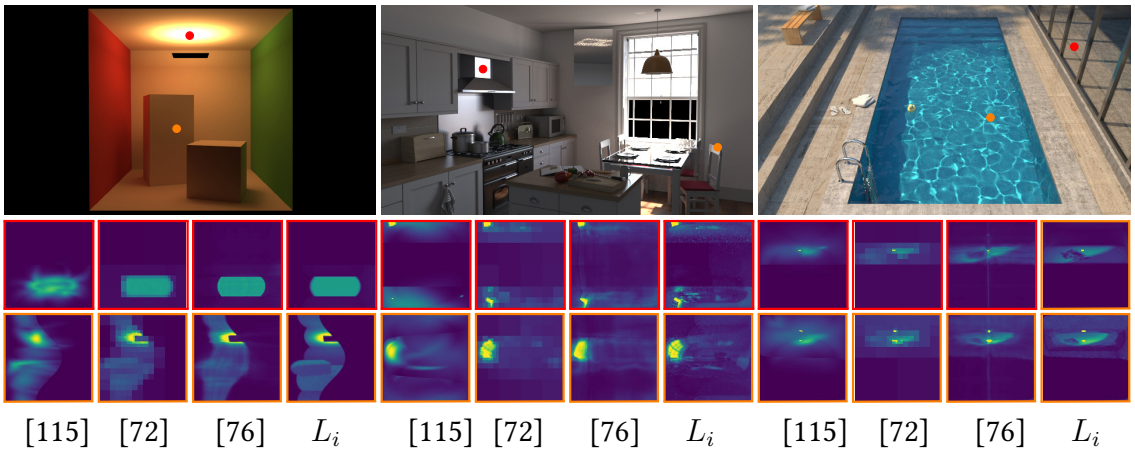


Figure 3.5: Different approximations of the incoming radiance field, from explicit approaches [115; 72] to learned ones [76]. The images on the bottom row visualize the directional component of the incoming radiance field for the red and orange points in each scene. Figure from [76].

The use of neural networks for importance sampling has been proposed in multiple

different contexts from unidirectional path tracing [76; 4] to primary sample space [122]. These networks are trained to approximate the incoming radiance field or even the full product with the BSDF for product importance sampling. A visualization of the representation learned through traditional data structures and neural methods is shown in Figure 3.5. The main requirement, and a challenge, when designing a network for importance sampling is that it needs to be easily invertible which is not the case with traditional architectures. The computational overhead of the forward pass and the inversion can diminish the benefits of the improved sampling. For that reason neural importance sampling is best suited for scenes with complex illumination where traditional importance sampling fails.

3.3 Generative Neural Radiance Fields

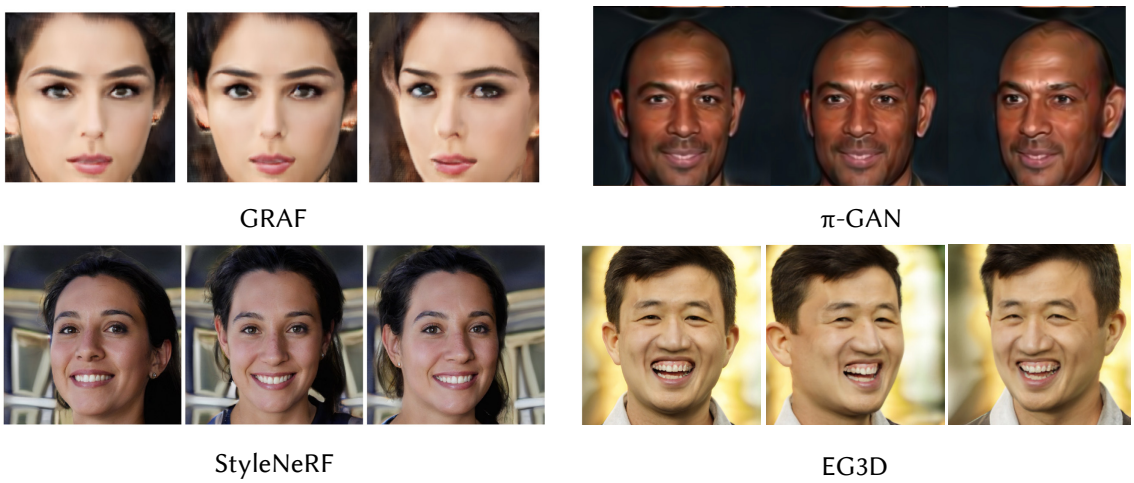


Figure 3.6: The evolution of the results in generative radiance field methods from GRAF [99] to π-GAN [15], StyleNeRF [30] and EG3D [14].

Generative models and more specifically GANs [27] are capable of learning how to generate a distribution of images for a class of objects. StyleGAN [50] and its follow ups [51; 52] demonstrated that GANs can be trained to reproduce faces with remarkable quality and details. The introduction of volumetric radiance fields following the methodology of NeRF into the GAN training scheme led to results with lower quality but explicit camera control, an important step towards incorporating generative models in rendering. GRAF [99] and π-GAN [15] were the first methods to propose the introduction of radiance fields in generative models with differences in the proposed architectures.

More recently StyleNeRF [30] improved on the quality of the results by using components from StyleGAN 2 which were proven to be effective at creating detailed imagery due to its convolutional nature. While previous methods used ray marching of the radiance field to compute the final image StyleNeRF proposed to ray march in feature space creating a feature map that is then translated into an image by a StyleGAN generator. While this improved the results it introduced view inconsistencies due to the operations in image space. Another approach at generative radiance fields is the one by Chan et al. [14] which also uses a StyleGAN2 generator before and after the raymarching. The StyleGAN 2 generator is tasked with creating a triplane representation which is used to query features at each ray marching step. This gets translated into a low resolution image and some extra features by an MLP decoder. The extra features are used to generate a high resolution image through a superresolution module that follows the architecture of StyleGAN 2. The low and high resolution images are fed to the discriminator with the goal of ensuring that they look realistic and consistent with each other. While this approach brought the quality of the generated results on par with StyleGAN 2 and 3 it again came at the cost of view consistency. A collection of results from the mentioned methods is shown in Figure 3.6.

3.4 Summary

We have reviewed the use of radiance fields in related work and demonstrated how they have been used in many different fields and with different goals. We draw inspiration from the extensive history of radiance fields from traditional path tracing to generative models and now we will present the different ways in which we apply them in our methods.

Chapter 4

Practical Product Path Guiding Using Linearly Transformed Cosines

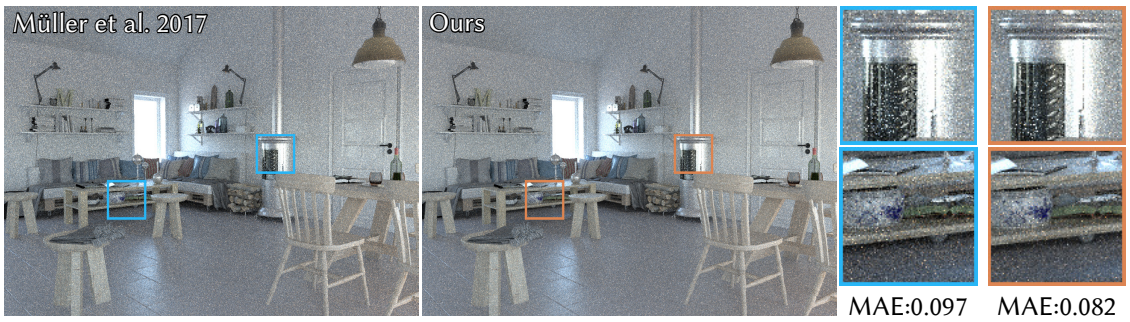


Figure 4.1: Equal-time comparisons (2 minutes). Our method reduces noise compared to Müller et al. [74], by efficiently guiding paths based on the product of the BSDF and incoming radiance at each path vertex. We show Mean Absolute Error (MAE).

In this chapter we will present how to augment radiance field approximations on-the-fly and efficiently, by using material information to improve sampling. More specifically in our method we combine the benefits of the *practical path guiding* technique of Müller et al. [72] with those of *product* importance sampling. We already discussed how path guiding can use approximations of incoming radiance to provide efficient importance sampling in path tracing. Usually path guiding is combined with BSDF sampling through multiple importance sampling which provides the best sampling quality from the two strategies. This is not sufficient when neither sampling strategy is a good approximation of the integrand. In such cases product-based sampling can provide significant gains. Product-based sampling has been introduced in path guiding by Herholz et al. [36] as an extension of the method by Vorba et al. [115]. This increases sampling efficiency, but also involves expensive precomputation inherited from the method it builds on. Our approach combines the computational benefits of practical path guiding and the sampling efficiency of product importance sampling.

Similar to Müller et al. [74], our technique relies on a spatial-directional tree (SD-tree),

to represent the continuously updated incoming radiance field. In prior work, this data structure provided estimates of radiance integrals over spherical sets that were required to drive a hierarchical sampling scheme. However, it is unclear how this could generalize to the product case: direct tabulation of the product of BSDF and incident radiance is clearly infeasible due to the prohibitive increase in dimensionality from 5D ($\mathcal{C} \times \mathcal{H}^2$) to 7D ($\mathcal{C} \times \mathcal{H}^2 \times \mathcal{H}^2$), where $\mathcal{C}$ represents positions on surfaces and H^2 the hemisphere of directions. Hierarchical sampling techniques that approximate the product at each level of the data structure [18] seem more promising but require estimates of BSDF integrals over spherical sets, which are not generally available in closed form. While these integrals could be estimated numerically, the resulting costs would likely diminish the benefits of product guiding. Our technique addresses this problem by providing an efficient approximation of the necessary integrals using Linearly Transformed Cosines (LTCs). Our choice is motivated by the observation that LTCs have been shown to be efficient for similar integrals required in the context of shading for polygonal lights [35].

Directly using LTCs for practical product sampling requires many integral evaluations during the hierarchical sample warping process, which unfortunately tends to negate the benefits of product guiding. To overcome this problem we introduce two main optimizations, one based on parallelisation and the other on precomputation. In addition, we show how to further improve results by carefully using multiple importance sampling and Russian roulette. We achieve on average 15-20% increase in computation speed for the same quality compared to previous work for our set of test scenes, both with practical guiding [74] and learning-based product sampling [36].

4.1 Practical Product Path Guiding

We use the SD-tree structure of practical path guiding [74] as our incoming radiance field representation.

This 5D spatial-directional tree is partitioned as shown in Fig. 4.2. Each node of the spatial subdivision tree (a) contains a quadtree which is stored in 2D directional space (b), parameterized by $\cos\theta$ and ϕ. Each node of the quadtree can be thought of as a spherical polygon in the global sphere of directions with surface normal $\mathbf{n}$. In the original method [74] these nodes record the total incident radiance $L_i(x, \omega_i)$ at each iteration of the guiding process. This incident radiance is then used to sample directions in the next iteration. Sampling relies on a hierarchical warping scheme that requires recursive

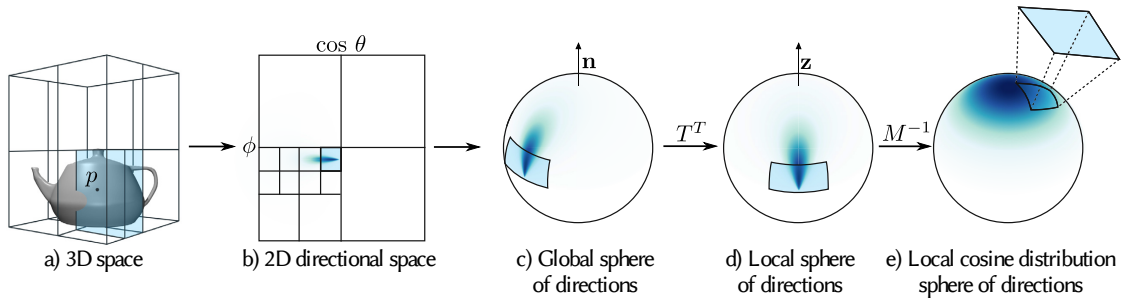


Figure 4.2: Given an intersection point p we find the voxel in the a) spatial subdivision that includes it and the corresponding b) directional quadtree. This node spans a region of directions in the c) global sphere of directions with surface normal **n**. Using the transformation T^T we transform this spherical polygon to the d) local sphere of directions with normal the axis **z**. Finally the LTC linear transformation M^{-1} takes the polygon to the e) local cosine distribution sphere of directions.

estimation of probabilities while descending in the quadtree of directions [67].

The main challenge when sampling the product $L_i(x, \omega_i)\, \rho(x, \omega_o, \omega_i)\, \cos\theta_i$, taking into account both the cosine-weighted BSDF and discretization of L_i, is that the BSDF varies with respect to position and outgoing direction ω_o. This implies that the sampling distribution must be re-generated at every shading point. Tabulation and normalization of the product distribution further require costly evaluation of the product at the finest level of the SD-tree.

We introduce a separability approximation to make this process more practical. In particular, we assume that

$$\int_{\Omega_i} L_i(x, \omega_i)\, \rho(x, \omega_o, \omega_i)\, \cos\theta_i \mathrm{d}\omega_i \approx \left[\int_{\Omega_i} L_i(x, \omega_i) \mathrm{d}\omega_i\right] \cdot \left[\int_{\Omega_i} \rho(x, \omega_o, \omega_i)\, \cos\theta_i \mathrm{d}\omega_i\right]$$

within spherical polygons Ω_i. This expression is approximate in particular when Ω_i covers a large solid angle, and it becomes more accurate under refinement. Our method samples this expression hierarchically in a coarse-to-fine manner, requiring many evaluations of spherical integrals over the BSDF, of the form

$$D \;=\; \int_{\Omega} \rho(x, \omega_o, \omega_i)\, \cos\theta_i \, \mathrm{d}\omega_i, \tag{4.1}$$

hence we seek an efficient approximation. Naturally, *too* approximate of an estimate may even increase variance, thus a suitable tradeoff between performance and accuracy is key.

Three possible options to accomplish this could be analytic integration, a numerical solution or a conservative estimation of the integral. Analytic solutions exist only for cosine-like distributions, which would limit us to diffuse materials. Numerical approaches, such as Monte Carlo integration, would be too slow to yield accurate results since they require many samples for each bounce of the path. A conservative estimate such as the one used by Estevez and Lecocq [19] would be problematic for the size of the spherical polygons, up to half a hemisphere, in the first levels of the quadtree. Instead, we use Linearly Transformed Cosines [35] which enable an analytical solution for more complex distributions, and have been demonstrated to be efficient for the integrals of the form of Eq. (4.1). We achieve this by transforming from a local sphere of directions Fig. 4.2(d) to a local cosine distribution sphere of directions (e).

Although LTCs admit a cheap integration scheme over spherical sets, the recursive nature of our method increases overhead and diminishes gains from product sampling in practice. We first discuss how we perform the product path guiding, and then present optimization strategies that exploit the parallelizable nature of the computations and the precomputation of frequently used factors. Finally we discuss how multiple importance sampling and Russian roulette can be used to further improve results.

4.1.1 LTC-based Product Sampling

We next discuss LTC fitting for the BSDF, discuss our product sampling approach and the technical specifics required for LTC-based product sampling.

LTC fitting for the BSDF at a shading point. Most realistic materials can be represented with a mixture of diffuse and glossy components, with varying roughness. For the diffuse components we use an LTC with an identity transformation $M = I$. For the glossy components we precompute a table of fitted LTCs over varying roughness α and incoming elevation angle θ, with 128 bins for each. Given a ray intersection point p, the BSDF at that intersection point with roughness α and the outgoing direction elevation angle θ we fetch the corresponding LTC.

The LTC is stored in local space; a linear transformation M that defines the LTC takes points from the local cosine distribution to the local current BSDF distribution (Fig. 4.2(e) to (d)). Since the quadtrees store incoming radiance in global coordinates we need to apply the transformation T, a rotation of the axis, to take the LTC from local to

global coordinates (Fig. 4.2(d) to (c)). As a result, in our representation, the LTC has a new linear transformation $M' = TM$.

To integrate the BSDF over a spherical polygon using the LTC, we apply the inverse linear transformation $(M')^{-1} = M^{-1}T^T$ to the vertices of the spherical polygon and analytically integrate the cosine distribution over it.

The integration of the BSDF using a fitted LTC uses the closed form expression described by Baum et al. [7]:

$$E(p_1, ..., p_n) = \frac{1}{2\pi} \sum_{i=1}^{n} \cos^{-1}(\langle p_i, p_j \rangle) \langle \frac{p_i \times p_j}{p_i \cdot p_j}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \rangle \tag{4.2}$$

where p_i are the polygon vertices. This expression assumes that the integration domain lies within the upper hemisphere. Otherwise, it must be clipped over the horizon.

Product sampling. During path generation, given a path vertex p we descend in the binary spatial tree and find the corresponding directional quadtree (Fig. 4.2(a) to (b)). Similar to Müller et al. [74], we use a hierarchical sample warping scheme proposed by McCool and Harwood [67] to generate samples that follow the desired distribution. During traversal, we estimate the probability to move to a child node k of a given internal node, and repeat this recursively until we reach a leaf node. Within the leaf node we sample uniformly.

In Müller et al.'s method each node stores estimated incoming radiance $\hat{L}_i$. At a given level of the quadtree, their sampling algorithm chooses between the four sibling nodes $v_j, j = 1..4$. Each such node k stores the incoming radiance $\hat{L}_i^k$, and the relative value $\frac{\hat{L}_i^k}{\sum_{j=1}^{4} \hat{L}_i^j}$ determines the probability to move to it next. Sampled directions thus follow the distribution of the incoming radiance field $L_i(x, \omega_i)$, one of the components of the integrand (Eq. 2.5). For the other component of the integrand, BSDF sampling generates directions following $\rho(x, \omega_o, \omega_i) \cos\theta_i$ and the two strategies are combined using MIS.

Our algorithm is summarized in Algorithm 1. Our path guiding method directly takes the product of the cosine-weighted BSDF into account. We thus need to compute the probability to descend into part of the quadtree based on the product of the BSDF integral D and the incoming radiance, using the LTCs for fast integration of D.

Technical specifics for LTC-based product sampling. The LTC representation described above allows us to integrate $\rho(x, \omega_o, \omega_i) \cdot \cos\theta_i$ over the four spherical polygons at each quadtree level.

During the hierarchical sampling process, the probability to move to a given child in the quadtree is given by the product of these values $D_j, j = 1..4$ and the stored values $\hat{L}_i^j, j = 1..4$. For child k, the probability is:

$$P_k = \frac{D_k \, \hat{L}_i^k(x, \omega_i)}{\sum_{j=1}^{4} D_j \, \hat{L}_i^j(x, \omega_i)}. \tag{4.3}$$

where $\hat{L}_i^k$ is an estimate of the corresponding spherical integral of incoming radiance. This probability is used in the traversal of the sample warping scheme, thus generating samples that follow the full (product) integrand.

Algorithm 1: Quadtree product sampling

```
if isNodeLeaf() then
    uniformlySampleWithinNode()
end
Π ← 0;
for i ← 1 to 4 do
    v ← getQuadVertices();
    for j ← 1 to 4 do
        v_j ← canonicalToCartesian(v_j);
        v_j ← (M')^-1 * v_j;
        normalize(v_j);
    end
    D_i ← ltcIntegrateQuad(v);
    L_i ← getStoredRadianceSum(i);
    Π_i ← D_i * L_i;
    Π ← Π + Π_i;
end
P_k ← Π_k / Π, k = 1, ..., 4;
chooseNodeWithProbabilities(P)
```

Discussion. Since the SD-tree stores the incoming radiance in global spherical coordinates over a spatial subdivision, the normals are averaged over this space. As a result sampled directions can be in the wrong hemisphere, terminating path generation. With

product sampling this only happens when the leaf node lies on the horizon and the uniform sample generated within falls below the horizon, which is quite rare. In addition there is a chance that the product of the BSDF and the incoming radiance is 0 over the entire sphere of directions. In such cases we sample according to the BSDF since this means that no incoming radiance has been recorded in the local hemisphere for this surface normal.

4.1.2 Optimization Strategies

In practice, using LTC fitting during recursive sampling adds computational overhead that tends to negate the benefits of product path guiding. The specific computation required for our new method enables two optimizations: First, we can evaluate per-node integrals in parallel using vectorization. Second, common factors can be precomputed. These two optimizations improve the performance of our approach compared to previous methods, as we show in the results (Sec. 4.2).

4.1.2.1 Parallel Processing

Clipping the polygon against the horizon results in a variable number of vertices (3 to 5) and requires branching code which impacts performance. Hill and Heitz [37] propose an approximation of this process by using the vector form factor of the unclipped polygon, i.e., Eq. 4.2 without the z axis dot product:

$$\mathcal{F} \;=\; E(p_1, ..., p_n) = \frac{1}{2\pi} \sum_{i=1}^{n} \cos^{-1}(\langle p_i, p_j \rangle) \tag{4.4}$$

From $\mathcal{F}$ we can compute the angular extent and elevation angle of a sphere that has the same form factor as the unclipped polygon. We use the precomputed ratio of the clipped sphere's form factor to the unclipped one to scale the polygon's form factor accordingly. With this approximation, the BSDF integration requires Eq. 4.1 to be evaluated four times for each node (once for each pair of vertices), for four child nodes at each level (Algorithm 1). This computation represents the additional overhead compared to the sampling in Müller et al. [74], and is a good candidate for vectorization. We perform these 16 computations at once on an AVX512 enabled CPU. The rest of the process involves fetching the stored $\hat{L}_i^j$ values so no further vectorization was possible there. Note that any optimization to other parts of the method from Müller et al. [74] would also benefit our solution.

4.1.2.2 Precomputation

Another way to reduce the overhead of the product sampling is by precomputing the diffuse vector form factors $\mathcal{F}$ (see Eq. 4.4 above). When a material is diffuse the corresponding LTC has $M = I$ so the total inverse linear transformation applied to the quad vertices is given by: $(M')^{-1} = T^T$. T is an orthogonal transformation and as such we can apply it to the resulting vector form factor instead of applying it to the vertices and then doing the computation. Given this observation, we precompute and store the vector form factors $\mathcal{F}$ for the five first levels of a quadtree with a total memory footprint of 256KB. For these levels we can avoid the arc cosine, dot and cross product in Equation 4.1. This increases performance for all diffuse and multiple component materials with a diffuse component with minimal storage cost.

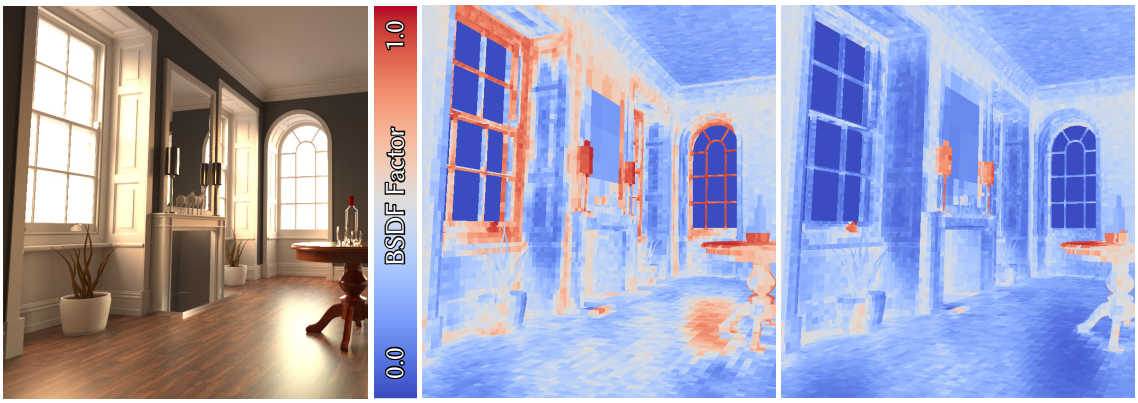
4.1.2.3 Discussion

Heitz et al. [35] mention that LTCs do not approximate the target distribution well in cases of incident grazing angles and high roughness materials. To avoid fireflies due to error in such cases we switch to path guiding without the BSDF LTC integration. Specifically we do this for cases of outgoing directions with local elevation angle $85^\circ \geq \theta \geq 90^\circ$. Moreover, if the material roughness is above a 0.5 threshold, we conservatively treat the material as diffuse when performing the product computation.

4.1.3 Optimization of MIS

Our product path guiding depends on the accuracy of the incoming radiance representation and of the LTC integration. In some cases the LTC representation (e.g., very shiny materials and/or grazing angles), these representations may not provide the best result. On the other hand the SD-tree approximation is coarse and even after many refining iterations it fails to capture all the details of the incoming radiance field. To overcome this issue, we combine our method with BSDF sampling using multiple importance sampling (MIS) as a defensive sampling strategy [36; 75].

We use the approach of Müller et al. [72], that learns the probability to either sample the BSDF or path guide based on the performance of each sampling technique. Specifically, we run an optimization step to learn the α value (see below), for the sampling probability



Müller et al. 2017 Ours

Figure 4.3: The proportion of BSDF sampling learned with ADAM for our product sampling and for Müller et al. [74]. Red regions show part of the scene where the guiding is used less than BSDF sampling. Our approach is more robust because it considers the product, thus lowering usage of BSDF sampling.

$p(\omega_o|\mathbf{x}, \omega_i)$ for the outgoing direction ω_o from position x and incoming direction ω_i

$$p(\omega_o|\mathbf{x}, \omega_i) = (1 - \alpha)p_g(\omega_o|\mathbf{x}, \omega_i) + \alpha p_{bsdf}(\omega|\mathbf{x}, \omega_i) \tag{4.5}$$

where p_g is the probability defined by our product guiding, and p_{bsdf} is the BSDF sampling probability. Note that for Müller et al.'s approach, p_g does not consider the incoming direction ω_i during sampling. We use the methodology of Müller et al. [75] to find α, i.e., we use the ADAM optimizer with the same parameters and optimize the Kullback-Leibler (KL) divergence as it is more robust to outliers [75].

It is interesting to visualize the learned α value for our product method compared to the original practical path guiding [72] (Figure 4.3). Observe that our α values are lower compared to Müller et al.'s, which indicates that guiding reduces the need for BSDF sampling as a defensive strategy. In particular, we rely more on our path guiding strategy on glossy surfaces (i.e., wooden floor) or in regions with normal variations (i.e., thin objects like the windows' frames). This is visible as more blue in the figure, indicating that BSDF sampling is used less in these regions with our approach.

4.1.4 Russian Roulette

As observed in previous work [36], product sampling increases average path length. This is because with product sampling we do not generate paths towards the light source if the BSDF value (and thus the path contribution) are low, in contrast to previous methods

that only take incoming radiance into account. In these cases, our technique introduces a tradeoff between path length and higher contribution. In practice, we perform Russian roulette for all paths on length two and higher, using Adjoint-driven Russian roulette [114] (without splitting) where pixel estimates are directly stored in the spatial binary tree nodes. This approach was originally proposed by Müller et al. [74].

4.2 Results and Evaluation

Figure 4.4: The six scenes used in our tests. From left to right: Bathroom, Living Room, Glossy Kitchen, Pink Kitchen, Attic and Necklace.

We implemented our method in the Mitsuba [40] renderer, and used the Enoki library [41] for parallelization with AVX-512 acceleration. We have published the source code of our method[1], including the Enoki optimization, for open research use.

We use L_1 difference for the metrics shown in the main paper. For other metrics, please refer to the additional material. We choose this specific metric as it is less prone to overweight fireflies compared to square error metrics such as L_2. Guiding methods may increase variance in important but undiscovered areas. To remove these few remaining fireflies we could use an outlier removal method [125].

We ran evaluations on a set of six test scenes shown in Fig. 4.4: Bathroom, Living Room, Glossy Kitchen, Pink Kitchen, Attic and Necklace. Some of these are variants of scenes used in previous work [74; 36]. We were unable to compare with GMM [36] on Pink Kitchen and Bathroom due to specific issues[2] with materials in these scenes. All reference images are computed with several hours of computation by averaging several independent runs of practical path guiding or by high sample count path tracing. We set the maximum path length to 10. To generate the results for comparisons, the authors have kindly provided their own implementations of the corresponding methods. All results are generated with implementations in Mitsuba [40], with 40 threads on a dual Intel Gold 6148 Skylake at 2.4Ghz, with dual AVX-512 units.

We disabled next-event estimation for all the techniques. Next-event estimation can be an ineffective sampling strategy in scenes with a highly occluded light source if no importance cache is used; this is the case for most of our test scenes. More generally, guiding techniques store the direct illumination directly inside the cache. Moreover, storing the direct and indirect illumination when doing the product guiding has the advantage of taking the BSDF at the shading point into account.

As our technique uses online learning, we combine all the iterations using the inverse variance scheme [72]. We also use a box filter and stochastic filtering when splatting a contribution on the directional and spatial data structure respectively.

We first present statistics illustrating the contribution of each of our optimizations to the efficiency of our technique. We discuss our experiments with a Monte Carlo alternative to LTCs. We then compare to the product GMM [36] approach and to practical path guiding [74]. We also compare to unguided path tracing to illustrate which part of

[1] https://gitlab.inria.fr/sdiolatz/practical-product-path-guiding

[2] Some of the materials generated from our in-house 3DS Max to Mitsuba exporter (of type Phong) resulted in crashes during the GMM fitting phase.

Table 4.1: Sampling cost of generating 64 sample per pixel for different scenes. These timings include all costs, which includes ray intersection and our guiding procedure. Our optimized version reduces the sampling cost by around $30 - 35$ % making our technique practical.

Scene	Naive	Optimized	Optimized AVX
Living-room	26.30	20.92 (79 %)	16.54 (62 %)
Pink Kitchen	53.60	44.22 (82 %)	32.94 (61 %)
Attic	64.04	54.47 (85 %)	41.93 (65 %)
Bathroom	58.28	48.13 (82 %)	36.91 (76 %)
Glossy Kitchen	20.12	17.90 (88 %)	13.69 (68 %)
Necklace	12.79	11.84 (92 %)	8.95 (69 %)

the light transport is difficult to sample.

Product optimization. We summarize the performance results of our technique with our different optimization strategies in Table 4.1 for various scenes. The naive version uses accurate polygon clipping and no diffuse precomputation (Section 4.1.2.2). The optimized version uses all optimizations listed in Section 4.1.2, except for parallel processing. We used the Enoki library [41] to achieve parallel processing using AVX-512 to compute the integrals at each level of the quad-tree in parallel. The scenes are ordered from the most diffuse to the most glossy one. For these results, no Russian roulette was used. We did not observe any noise increase when using the sphere form factor approximation to make our code branchless. Note that using AVX-512 instruction is a crucial optimization to making the technique even more computationally effective.

LTC vs Monte Carlo integration. An alternative to LTCs for the BSDF integration is a Monte Carlo approach, which we experimented with. To avoid repeating the integral estimation process for each node during traversal we created a temporary quadtree which we filled with N equal energy samples using BSDF sampling. We added a constant 5% of the total energy to all the nodes to ensure we explored nodes that, due to the low number of samples, haven't received any energy. The two quadtrees, temporary BSDF quadtree and incoming radiance quadtree, shared the same structure and the product of their values determined the traversal. For $N = 64$ and for same quality results the Monte Carlo integration approach was 9 times slower than the LTC approach.

Comparison with product GMM [36]. To achieve a fair comparison, we do not use Russian roulette since it was not available in the reference implementation of the product GMM method. We also set the BSDF sampling probability to 0.25 for both the techniques. As also noticed by Müller et al. [74] the GMM training implementation does not scale up with a high number of threads, while the other two methods do. To provide a fair comparison we trained radiance GMM with 8 threads and assumed perfect linear scaling of speedup to 40 threads to take training time into account. We do 30 training passes with 2M photons or importons emitted per pass. We use default parameters for the rest of the algorithm.

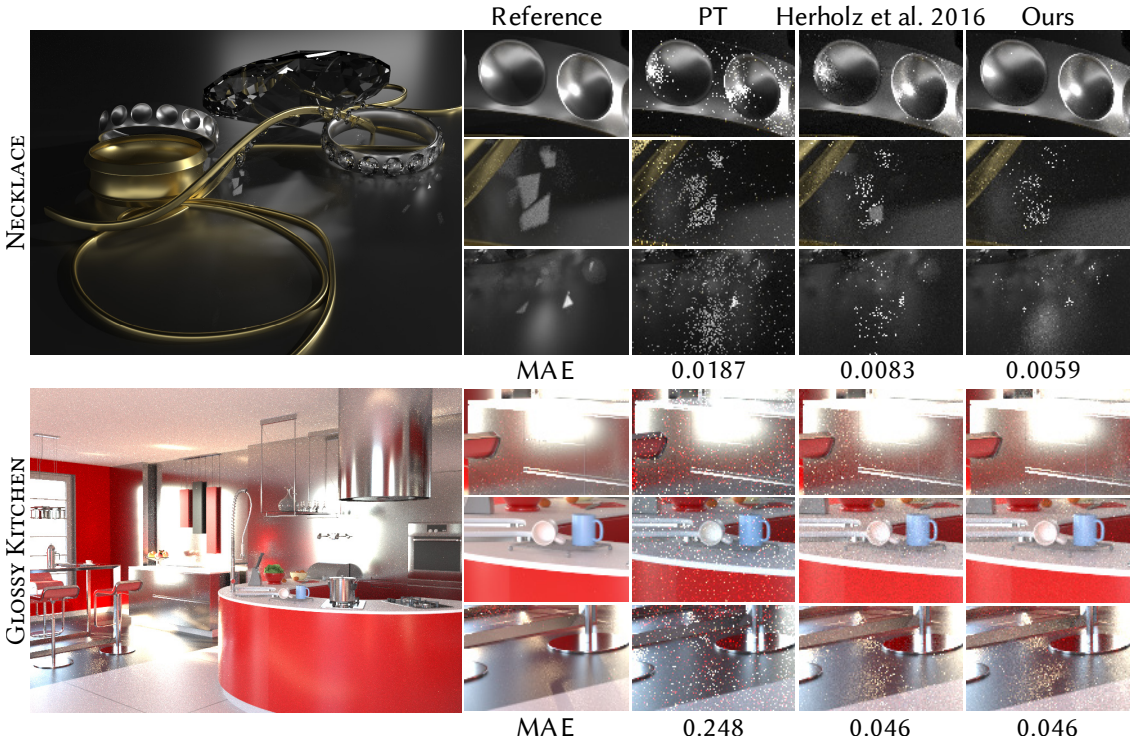


Figure 4.5: Equal-time comparison between standard uni-directional path tracing (PT), product GMM [36] and our product for the Necklace (5 minutes) and Glossy Kitchen (10 minutes). Due to the online nature and more robust radiance representation, our technique can generate images with lower noise. However, the GMM product can be better at capturing fine lighting details due to its high directional resolution. The training time for GMM is 61 and 194 seconds respectively.

Figure 4.5 shows the comparison of Product GMM [36] and our technique. Due to the online nature which uses all the samples combining different iterations using inverse variance and a more robust irradiance representation, our algorithm can perform more samples per pixel and achieve lower error. Still GMMs are usually better at capturing

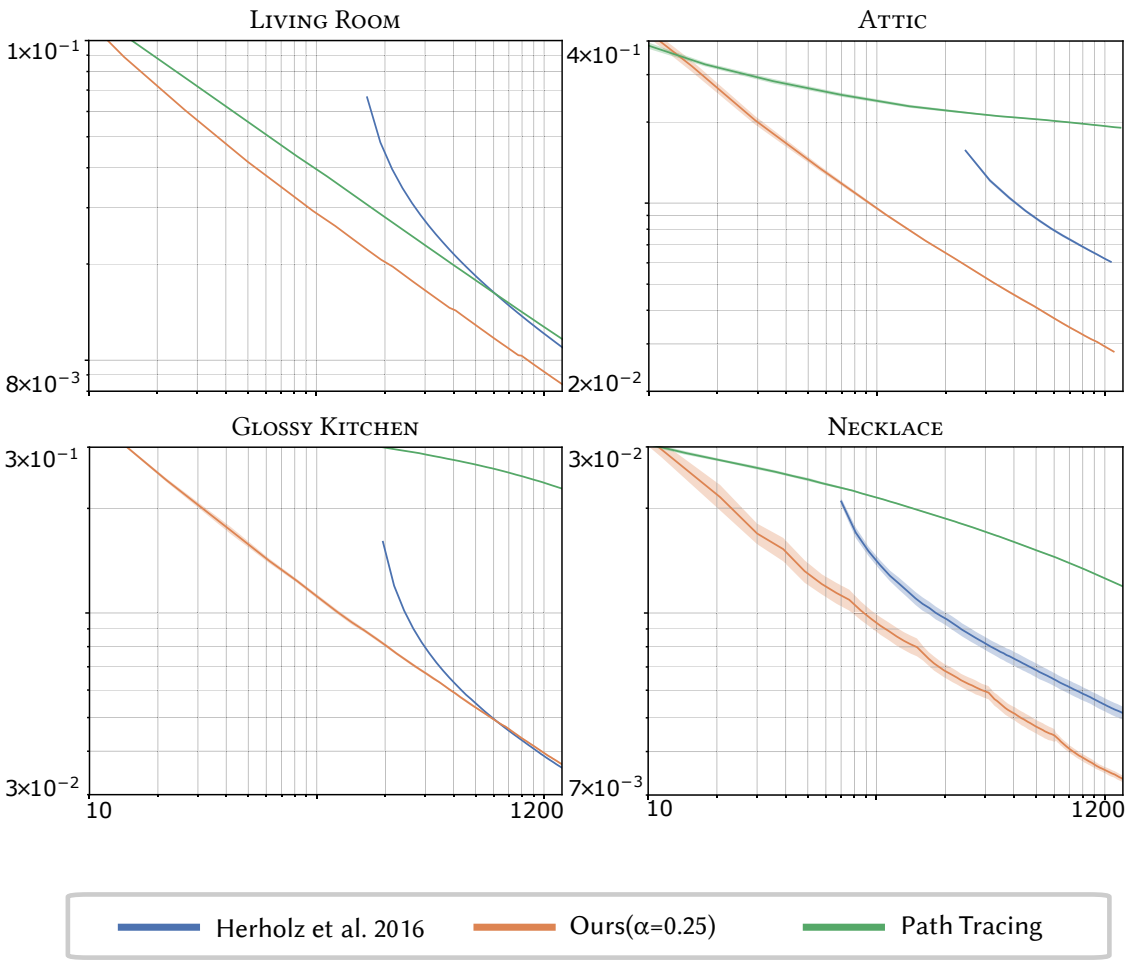


Figure 4.6: L_1 convergence graph for 4 scenes between product GMM [36] and our product without Russian roulette and fixed BSDF sampling probability. We show the average and variance over 5 runs of each technique.

fine highlights like caustics, due to a denser cache. However, due to GMM Expectation Maximization instability, GMM techniques can generate artifacts in some regions, like on the silver ring in Necklace scene. L_1 convergence graphs are shown in Figure 4.6. Overall our method is more efficient, however for the Glossy Kitchen scene performance is similar or better at later iterations.

Comparison with Practical Path Guiding [74]. We compared our product guiding with Müller et al. [74] by enabling all the improvements presented in [72]. Here all the BSDF improvements (Section 4.1.3) and ADRRS (Section 4.1.4) are enabled.

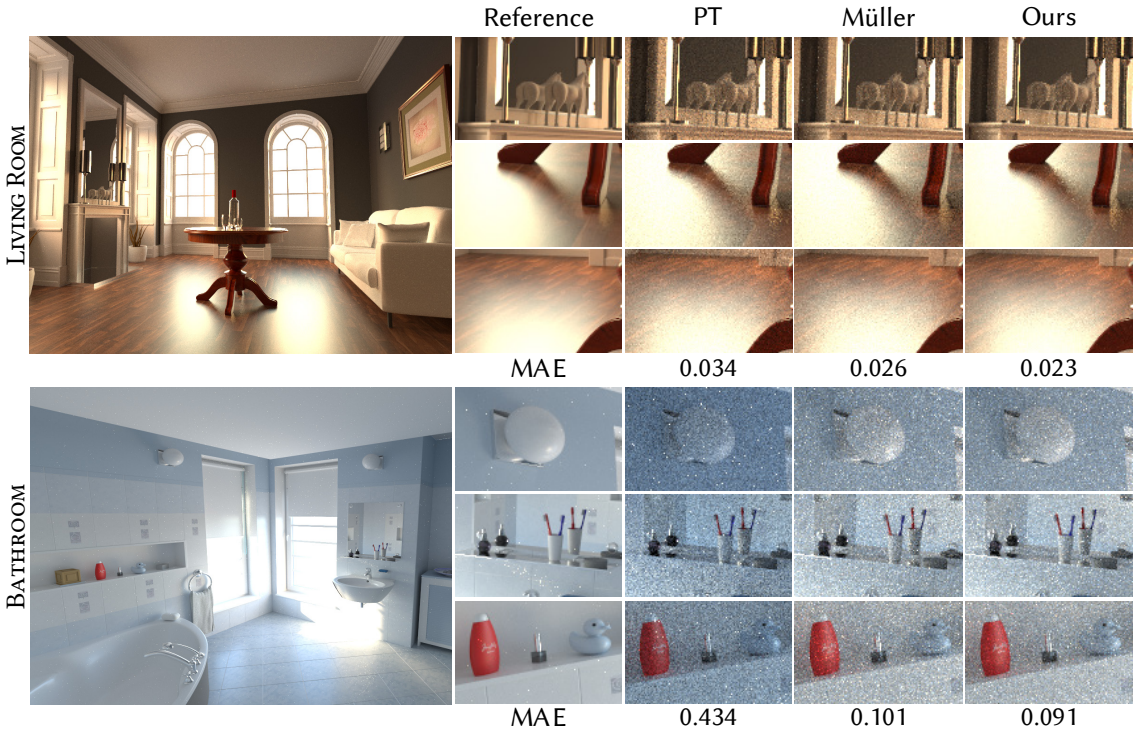


Figure 4.7: Equal-time comparison between standard uni-directional path tracing (PT), Müller et al. [74] and our product for the Living Room (2 minutes) and Bathroom (3 minutes) scenes.

We also present graphs of error convergence for the three methods shown in Figure 4.8 for the six test scenes. We see that our method almost always has lower error, converging faster than the previous solutions. On average, we are 15% faster for the same quality. Figure 4.7 shows equal-time comparison for two scenes.

Overall, we observed that our method is particularly helpful in two cases: (1) on glossy surfaces where our guiding reduces noise since the effect of the product is more pronounced than elsewhere (2) on diffuse surfaces where we can clamp the irrelevant directions. Overall, even if our computation is more expensive (due to the multiple LTC integrations), we almost always see improvement in the level of noise, compared to previous work.

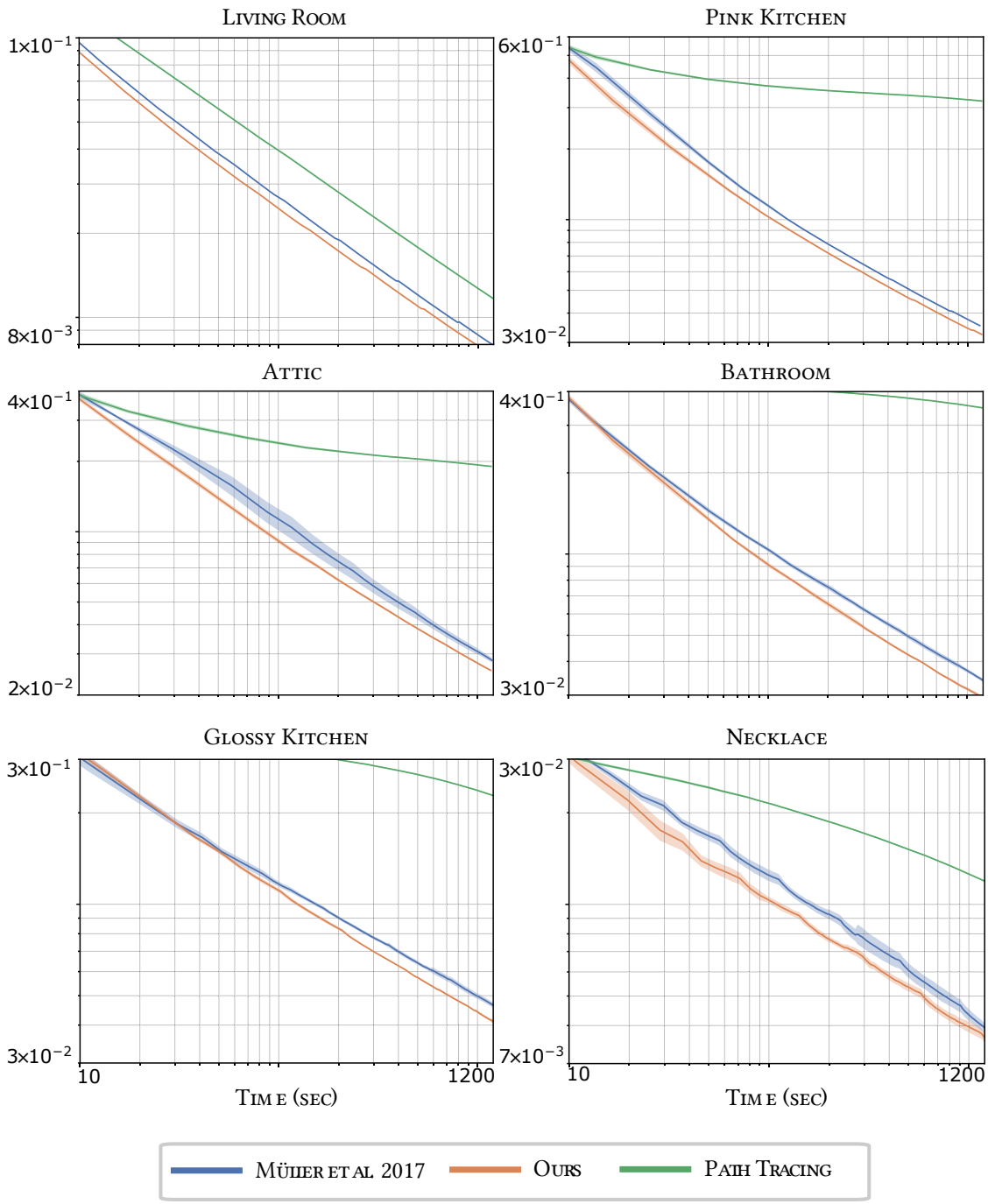


Figure 4.8: L_1 convergence graph for our six scenes, compared to [74] and simple path tracing. We show the average and variance over 5 runs of each technique.

4.3 Limitations and Future Work

Our method shows improvement in most of the scenes we tested, with more significant gains for scenes with many glossy/rough materials. It is however not without limitations that we discuss next, followed by directions for future work.

4.3.1 Limitations

For scenes with moderate to high complexity in geometry and materials, our product guiding is generally advantageous. However, for some simple scenes the overhead of path guiding may not be worthwhile. This is especially true for product guiding that involves a significantly higher overhead than simple strategies such as BSDF sampling, even though the tradeoff needs to be considered for all guiding methods. Despite recent work (e.g., [91; 9]), there is currently no easy way to identify "difficult paths" for which product guiding is guaranteed to be cost effective.

In our current approach, guiding is not used in some specific cases, e.g., for deterministic sampling techniques such as glass material. However, the treatment of such light interactions, e.g., the decision to reflect or refract could result in paths with high contribution where guiding could be beneficial. It is unclear how to adapt our data structures to effectively guide such sampling decisions, without storing the complete path [91].

Our separability approximation introduced in Sec. 4.1 performs adequately in our test scenes but in theory it still has failure cases. We could construct such a failure case with two checkerboard functions illustrated in Fig. 4.9. A similar scenario could arise with complex materials with multiple glossy lobes that don't overlap with the incoming radiance in some directions. In that case our method would overestimate the product value and allocate samples in regions of low importance.

Finally, one key element for the efficiency of our approach is the use of LTC to integrate the BSDF contribution over a node of the quadtree. This works well for some materials such as the GGX model we used in our tests, but the current LTC fitting procedure may need to be adapted for other models. In addition, the expense of our product approach is proportional to the number of LTCs needed to integrate for a given BSDF, making the treatment of complex materials more challenging.

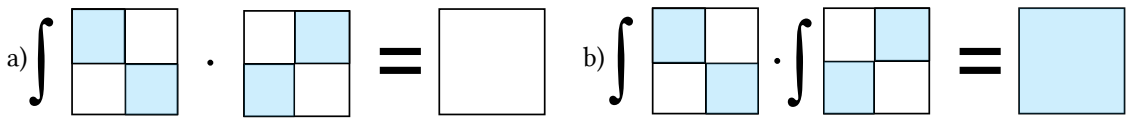


Figure 4.9: An artificial failure case for our separability approximation. a) The integral of the product of these two functions is zero but our approximation by a product of integrals b) gives a non zero value.

4.3.2 Future Work

In future work, we would like to further investigate the interaction between MIS and path guiding. For now we are using ADAM to optimize the BSDF selection probability. However, this BSDF selection probability is given for a spatial cell and does not take into account the incoming direction. It will be interesting to investigate if a more elaborate approach providing a finer BSDF selection probability can give better results. Finally, it is not clear how to incorporate such an optimization procedure with recent MIS techniques [29; 48].

Recent techniques restrict guiding to regions where necessary by storing complete paths [91], in contrast to a cache of all paths such as the SD-tree we use. Developing a method that combines the ability of the former to treat very hard paths and the full path expressivity of the latter is an exciting direction for future work.

A possible future research direction would be to build a data structure based in primary sample space for sampling a point on the emitter given a position in space. However, to apply our product approach, it would be necessary to know the light source geometry in terms of polygonal shapes to apply LTC integration.

Finally, both practical and product path guiding could be used in the context of volume rendering, e.g., by adapting LTC integration to support phase functions.

4.4 Conclusions

We have presented a new product based path guiding technique, that makes the incoming radiance field approximation material-aware in an efficient manner. The key element of our approach is the use of Linearly Transformed Cosines allowing on-the-fly integration of the BSDF during hierarchical importance sampling, i.e., when recursively traversing the quadtree representation of the directional component of the subdivision. To make the approach cost effective, we introduce two main optimizations, using parallelization and precomputation, and also exploit the benefits of MIS and Russian roulette to

further improve performance. We have demonstrated how our new approach is beneficial on a set of six test scenes, and we have also presented an analysis of the benefits of each of our optimizations.

While working on this project we witnessed the importance of representing radiance fields to speed up rendering. We also saw how traditional data structures cannot represent complex radiance fields accurately even after hours of refining. Due to this limitation ours and previous work [36] always use BSDF sampling as a defensive sampling strategy. Finishing this project we started exploring other alternative representations for radiance fields that would allow us to use radiance fields for inference instead of sampling.

Chapter 5

Active Exploration for Neural Global Illumination of Variable Scenes

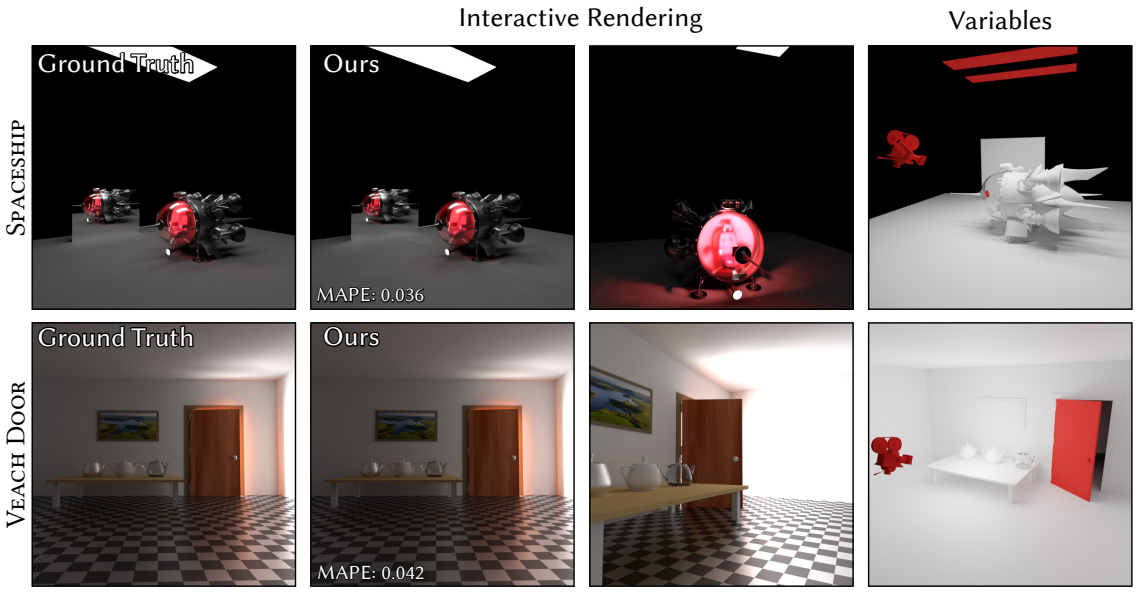


Figure 5.1: We introduce a neural rendering method that allows interactive navigation in a scene with dynamically changing properties, i.e., viewpoint, materials and geometry position and full global illumination effects. With our *Active Exploration* we can train a neural network efficiently to learn global illumination for all the configurations of these variable properties, allowing interactive rendering at runtime. Left to right: ground truth path traced images; our prototype interactive neural renderer, running at 4-6 fps with a variation of each scene and the variable parts of the scene depicted in red; each variable property (light intensity, camera position, object rotation, etc.) is controlled by an interactive slider (please see video).

In our search for a more powerful representation we focused our attention on neural networks, with the goal of learning a high quality outgoing radiance field representation which can be queried at runtime to form the image. We demonstrate how a network can be trained *efficiently* to represent the outgoing radiance field of a *variable* scene with moving objects, emitters, viewpoint and changing materials. Altering the scene composition affects the distribution of radiance and for some configurations complex

lighting effects can appear. For example placing an emitter over a glass bottle will create a caustic meaning that the radiance field will vary rapidly for that spatial region. These effects are harder to learn for a network because of their high frequency content and due to the low probability to be observed when the scene configuration is chosen at random. We propose an alternative to this uniform sampling of the scene configurations and show how sampling of training data in this case is very important.

We explored how recent methods [25; 28] are trained on large numbers of rendered images for variable scenes. Uniformly sampling the space of these path-traced images is expensive; in the case of a high-dimensional space $\mathcal{D}$ containing all the possible configurations of a variable scene each with a different outgoing radiance field distribution, it quickly becomes unmanageable. To address this limitation, we introduce an *Active Exploration* strategy, that guides sampling to parts of the space $\mathcal{D}$ where the radiance field is more complex and harder to represent with a neural network.

We demonstrate the efficiency of our Active Exploration approach on a neural renderer, by training a generator network that can interactively render global illumination with dynamic modifications (moving viewpoint, lights, objects etc.). Training time varies from minutes to hours, depending on the scene variability, complexity and available hardware. To represent a variable scene (see Fig. 5.1), we use an explicit representation with a vector v of variable parameters that precisely define an instance of the possible scene configurations in $\mathcal{D}$ enabling fine control of each parameter and interactive rendering.

We *interleave* training with on-the-fly generation of the data it needs. Uniformly sampling the space of parameters to generate the data for training does not allow the network to achieve satisfactory visual quality, especially when increasing the dimensions of $\mathcal{D}$, because in many cases light transport has hard, localized effects that have low probability of being observed.

Our Active Exploration method finds samples best suited for training but also locally explores these regions of $\mathcal{D}$ which is crucial in our context especially for enabling high resolution training (5.5.2), compared to Active Learning (see Sec. 5.1). For this we use a Markov Chain Monte Carlo (MCMC) approach, with small and large steps and a custom acceptance policy.

Despite our focused Active Exploration, training data generation – i.e., ground truth path-tracing – is still expensive; it is thus beneficial to *reuse* such rendered samples during training. For best results, we introduce a *self-tuning sample reuse* strategy that optimizes

the probability for training sample reuse, further reducing training time.

Active Exploration, together with sample reuse and resolution enhancement allow us to train our neural renderer network very efficiently. In contrast, uniform sampling converges to a low quality solution, while our guided exploration of the training space allows us to significantly improve visual quality, especially for reflections and hard light paths. Therefore our renderer is well suited for interactive rendering of effects such as complex caustics or specular-diffuse-specular paths, that are not handled by other real time methods.

In summary our contributions are:

- A novel Active Exploration approach, interleaving training with on-the-fly generation of training data, together with an adaptively increasing resolution method.
- A self-tuning sample reuse approach, further optimizing training time and storage.
- A neural renderer that allows direct control of parameters for global illumination and interactive inference, based on an explicit scene parameterization.

We demonstrate our system that allows interactive modifications of lighting, geometry, materials and viewpoint (at 4-6 fps in our prototype Python implementation, see Fig. 5.1 and video). We have released all data and our reference implementation[1].

5.1 Related Work

We review and discuss some aspects of deep learning that inspired our Active Exploration and training sample reuse methods.

5.1.1 Machine Learning

Our on-the-fly data generation, Active Exploration and training sample reuse approach do not have obvious equivalents in related work to our knowledge. However, several sub-fields of machine learning explore ideas with some similarities; we review these briefly.

Active Learning. Parallels can be drawn between our on-the-fly training data generation and Active Learning, where the data generation (labeling) is done procedurally to decrease cost. As reviewed by Settles [101] in Active Learning an algorithm chooses

[1] `https://gitlab.inria.fr/fungraph/active-exploration`

when a data sample needs to be labeled, i.e., to be given the ground truth. Active learning has been applied to convolutional neural networks [100] and generative adversarial networks [124]. Different metrics can be used to define the importance of each sample; some are related to our metrics to identify the most important samples during Active Exploration (Sec. 5.4.1). Our context of working with synthetic scenes allows us to expand on Active Learning and introduce Active Exploration. We not only identify hard samples for training but we also use mutations to propose new hard samples which helps with catastrophic forgetting [55] and overfitting.

Curriculum Learning. Importance sampling methods with Stochastic Gradient descent have been developed under the general curriculum learning framework [11]. They learn the probability distribution of choosing a training sample and use it for importance sampling. Similarly Hazan et al. [33] learn a distribution for picking training data. In contrast to such methods, we know the exact dimensions of our data space and can sample them at will, making the task easier.

Recent work investigates issues with adaptive sampling, and the cost of using the ideal target function [106], and provide guarantees about the quality of sampling given limited information on the gradients. There have been some techniques that use self-augmentation with synthetic rendering to overcome the lack of labeled or real-world ground truth data [56; 64]; the goal is to match the synthetic and real distributions, which implies different design choices from our context.

Compared to all these methods the major difference is that we have a forward problem, and thus have full knowledge of the parameters that define the space of training data and their dimensions. We can thus sample any part of this space on-the-fly. This aspect of the space of training data makes it amenable to an MCMC exploration method, which is not the case of static, pre-captured training datasets.

Learning and MCMC. MCMC methods have been used in Bayesian learning from the early days of neural networks [80]. More recently, Stochastic-Gradient MCMC has been proposed [117; 120] with various applications [63]. We also use MCMC for deep learning, but in a different context: since we solve a *forward* problem and can generate training samples on-the-fly, we use an MCMC approach inspired by Metropolis-Hastings to guide the sampling process.

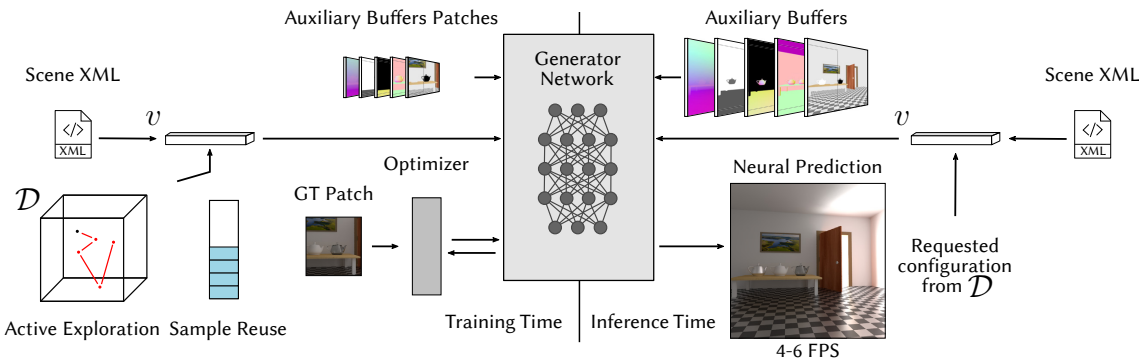


Figure 5.2: Overview of our approach. Left: During training we define a scene and the set of variable parameters via an *xml* file resulting in an explicit scene representation vector v. Using Active Exploration we guide the configurations of the *variable scene* towards more difficult instances that are important for the PixelGenerator network. Right: After 5-18 hours of training – depending on the complexity of the variable parameters and quality required – we can *interactively* request any variation of the scene with visual dynamic changes in illumination, move objects, the viewpoint and modify materials.

5.2 Overview

Our goal is to significantly improve the efficiency of the training process in neural renderers that are trained on synthetic rendered data by introducing Active Exploration of the high-dimensional sample space and re-using these samples. With this scheme we are able to efficiently train our neural renderer which provides explicit control of the scene parameters and has constant rendering performance regardless of the difficulty in the underlying lighting effect.

We represent the scene variability by an explicit scene parameter vector v (Fig. 5.2), which defines the space $\mathcal{D}$ of all possible configurations of the scene; thus any $v \in \mathcal{D}$ corresponds to an individual scene configuration. Our goal is to train a network to take a specific v and the set of corresponding G-buffer images (normals, albedo, etc.) as input, and generate full global illumination images (Inference Time in Fig. 5.2).

When training the network some visually significant effects are very localized in the high dimensional space $\mathcal{D}$. Finding sufficiently useful samples in $\mathcal{D}$ to train our network for those effects is very unlikely using uniform sampling and our limited budget. It becomes more unlikely as the dimensionality of $\mathcal{D}$ grows.

Since we are solving a *forward* problem, we can generate ground truth training samples on-the-fly using a fast path-tracer. A *training batch* will be 16 *samples* each consisting of a 32x32 patch of ground truth, path-traced image, each patch in the batch

corresponding to a different configuration of the vector v. Each patch is sampled by a different Markov Chain and rendered in parallel. Using patches allows more efficient exploration of $\mathcal{D}$. In terms of pixels generated, this is equivalent to an image of resolution 128x128.

Even though our path tracer is fast, the cost of generating a training batch is still high. We thus *reuse* training samples as much as possible. We introduce a sample reuse strategy that further improves the speed of training. Training times vary from 5-18 hours depending on the complexity of the variable parameters and the quality required.

Once trained, the generator network allows interactive rendering of dynamic global illumination effects (Fig. 5.2, right) for the variable scene, e.g., interactively navigating in the scene, opening the door, change lighting etc. (please see video).

5.3 Explicit Encoding and On-the-fly Data Generation

We explain the explicit scene representation, the generator network and the on-the-fly data generation process, used in our neural rendering algorithm, before presenting the actual Active Exploration approach in Sec. 5.4.

5.3.1 Explicit Scene Representation

Previous methods [25][28] use an encoder network to create a neural scene representation vector of a scene configuration.

However, this representation lacks interpretability and editability. In addition, rendering a new scene configuration requires new observations (i.e., ground truth renderings) to be generated, since the parameters of the scene representation have no explicit interpretation or "meaning", and thus the renderings are needed to generate the new neural scene representation vector.

We focus on variable scenes, commonly used in production [102]. Given that we know explicitly which parts of a scene are variable and how much they can vary, we avoid training an encoder network to represent this variability and instead create the scene representation vector from the scene definition. As a result all fixed properties are stored in the generator and associated with a set of rasterized G-buffers, while scene variability is compactly represented in the explicit vector. This vector contains the normalized values of the variable scene parameters for a given scene instance.

Consider a variable Cornell box scene (Fig. 5.3). Here we vary the two vertical wall

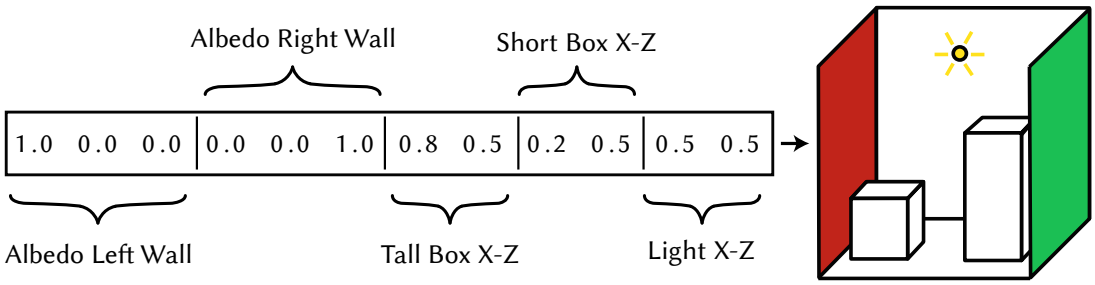


Figure 5.3: The explicit scene representation vector v that defines this instance of the variable Cornell box scene.

albedos, positions of the boxes and light source position, and define the ranges of these parameters.

The normalized parameter values make up the explicit scene representation vector v (Fig. 5.3). The scene representation vector along with the camera position and lookat vector are repeated along the width and height dimension to be the same size as the G-buffers, and also passed to the neural network. Since our generator operates on a per pixel basis, this repeated vector injects the global scene information to all pixels.

5.3.2 Network Architecture, Buffers and Training Data

We optimize a modified PixelGenerator architecture [28; 104] (a Multilayer Perceptron network with skip connections) to map the inputs for each pixel to the final pixel color value. We choose this over a convolutional neural network such as a UNet since [28] has demonstrated the PixelGenerator architecture to perform better at upscaling. Unless stated otherwise, we use 512 hidden features and 8 hidden layers. For the optimization we use the ADAM [57] optimizer with learning rate 1×10^{-4}.

For the G-buffers, we provide all the information a traditional path tracer would require to evaluate the rendering equation of path tracing (Equation 2.5). We create first-intersection G-buffers with the world position of the intersection x, normal of the surface n, reflectance and roughness of the BSDF ρ and outgoing direction ω_o. The normal and material information help the network understand the existing correlations between these signals and outgoing radiance L_o.

We optimize the neural generator to map this input to the value of the integration over the hemisphere. Emission is also computed as a first-intersection buffer and is passed through to the output directly.

The world position x conditions all the other inputs since it is where the integration happens. For this reason we precondition the PixelGenerator to the position G-buffer by passing it alone through the first network layer. In subsequent layers all the buffers are concatenated with the global information of the scene representation vector and passed to the network; this is similar in spirit to NeRF [71] that inputs only position to the first layers. We experimented with Fourier features [107], but this resulted in artifacts due to the noise in the training data. We show the effect of this choice in Sec. 5.6.3.

5.4 Active Data Space Exploration

For a given variable scene, we will optimize a neural generator using on-the-fly synthetic training data; we describe the training process and loss in Sec. 5.5. This training data is generated within the space $\mathcal{D}$ of all possible configurations of the scene.

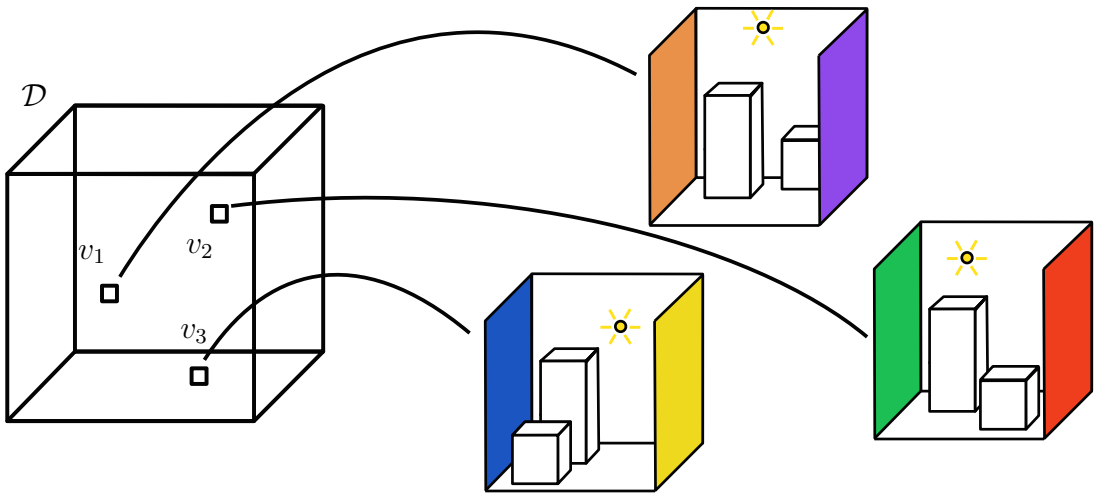


Figure 5.4: A point v_i in the data space $\mathcal{D}$ defines a scene instance out of all the possible configurations of the *variable* scene.

Each point in this space is defined by the values of the scene variables of the explicit scene representation vector v. Since the scene variables are normalized, the data space $\mathcal{D}$ can be seen as a hypercube, see Fig. 5.4.

A uniform random sampling of this high-dimensional space converges to a local minimum with low quality (see Sec. 5.6.3).

To overcome this difficulty, we propose *Active Exploration* of the space $\mathcal{D}$ of training samples. The ability to generate on-the-fly training samples defined by the explicit vector v offers great flexibility, allowing us to *interleave* sample generation and training.

The high-level goal is to find a sampling strategy that will find samples in $\mathcal{D}$ that maximize the progress of training and locally explore these pockets of importance. We introduce a MCMC exploration strategy that guides sampling of $\mathcal{D}$, towards scene configurations where the network struggles to recreate global illumination. MCMC is well suited to searching such high-dimensional spaces, and has proven its utility both in learning [117] and illumination [113].

The Markov Chain is initialized with a state picked uniformly from the hypercube of the data space $\mathbf{u} = \mathbf{u}_0 \in \mathcal{D}$, i.e., a random configuration of the variable scene. The next proposed state is sampled from the proposal distribution $\mathbf{v} \in T(\mathbf{u}_i \rightarrow \mathbf{v})$. Similar to the Primary Sample Space exploration [53] we balance global and local exploration of the space with large and small steps, by choosing large steps with probability p_{LS}. Specifically:

$$T(\mathbf{u}_i \rightarrow \mathbf{v}) = \begin{cases} \mathcal{U}() & \text{with probability } p_{\mathrm{LS}} = 0.3 \\ \text{Perturb}(\mathbf{u}_i) & \text{else} \end{cases} \tag{5.1}$$

5.4.1 Markov Chain Exploration

The data space $\mathcal{D}$ can have arbitrarily high dimension, depending on how much variability exists in the scene. Our goal is to generate training samples that follow the distribution of sample importance, i.e., the impact of the sample on training. In MCMC terminology, our target function f and corresponding target distribution p should be defined so that the sampling process produces samples that maximize benefit for training.

The high dimensional space of $\mathcal{D}$, with pockets of importance, is ideal for a MCMC random walk exploration.

The hypercube of our data space $\mathcal{D}$ has a very similar structure to the primary sample space [53] and we take inspiration from the exploration choices of that method. The Metropolis-Hastings algorithm defines a proposal distribution $T(\mathbf{u}_i \rightarrow \mathbf{v})$ from a given state $\mathbf{u}_i$ to a proposed state $\mathbf{v}$. The target distribution p is defined such that new states should be proposed and accepted for the Markov Chain to have a stationary distribution (i.e., the distribution at convergence) proportional to the target function.

Our goal is to define a target distribution that will guide the training process to samples that accelerate training. Previous work has suggested different metrics of sample importance [121]. Two common such metrics are the training loss or the norm of the

gradients after a backward pass, which we combine in our target function.

Our experiments showed that if only the loss is used, MCMC doesn't take into account where the network can improve the most.

However, the *product* of the loss and the norm works well, see Fig. 5.17. Since we use ADAM [57], instead of the norm of the gradients we use the norm of the total step to take into account the momentum and RMSprop [110].

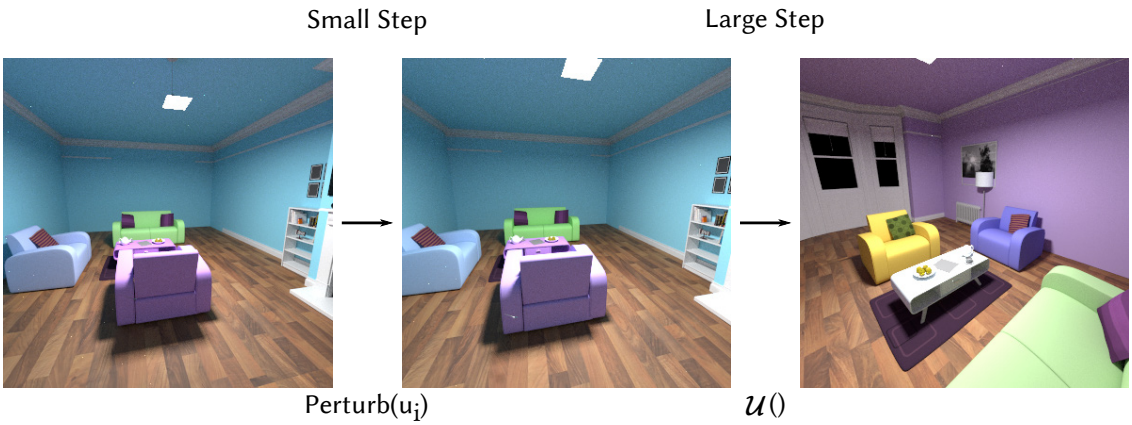


Figure 5.5: Visualization of the impact of a small and large step on a variable scene. In small steps (left) minor perturbations are applied – here the light source, furniture positions and materials have been altered slightly. In large steps (right), major changes have been applied to the scene (position of furniture, albedo of objects etc.).

The small step involves applying normally distributed perturbations to each component of $\mathbf{u}_i$. A visualization of the impact of these steps on the final rendering is shown in Fig. 5.5. Since the proposal distribution is symmetric, meaning $T(\mathbf{u} \rightarrow \mathbf{v}) = T(\mathbf{v} \rightarrow \mathbf{u})$, the acceptance probability of the proposed state similar to the Metropolis-Hastings algorithm is:

$$\alpha(\mathbf{u}_i \rightarrow \mathbf{v}) \;=\; min\left(1, \frac{p(\mathbf{v})}{p(\mathbf{u}_i)}\right) \tag{5.2}$$

The acceptance probability transforms the Markov Chain's stationary distribution to the target distribution. In our case we have a) an evolving target distribution that b) changes based on the samples we provide.

In our experiments, the acceptance probability of Eq. 5.2 does not converge to the target distribution fast enough, i.e., before it has changed. For this special case we propose instead a more aggressive acceptance policy:

$$\alpha(\mathbf{u}_i \rightarrow \mathbf{v}) \;=\; \begin{cases} 1 & \text{if } \; p(\mathbf{v}) > p(\mathbf{u}_i) \\ 0 & \text{else} \end{cases} \tag{5.3}$$

This acceptance probability has the desirable property that the more we remain in a state the more the target function – which is related to the error – decreases for this state. If we assume that the network can represent this state, then it will learn from it, meaning that the gradients and error will decrease, allowing a new proposed state to be accepted. If there are states that cannot be represented (e.g., pixel perfect reflections) the gradients will guide the MCMC towards states that still have room for improvement avoiding the issue of getting stuck.

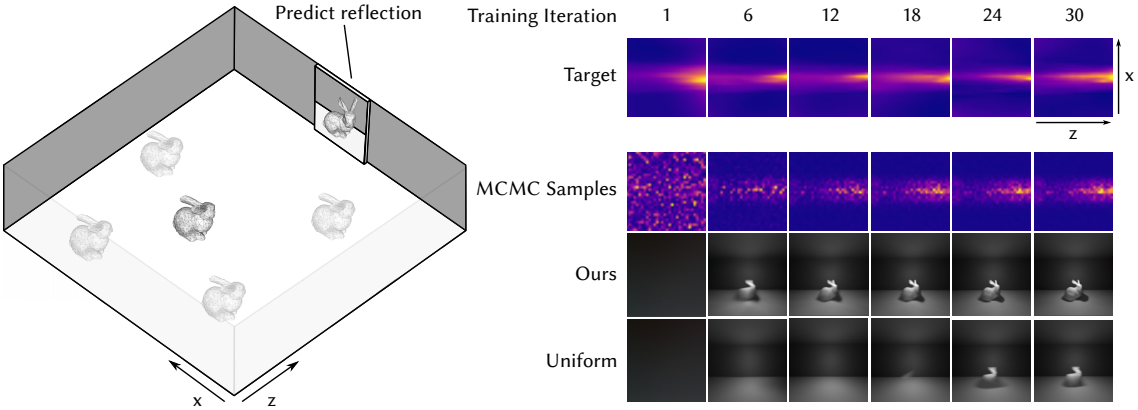


Figure 5.6: We test our method in this simple example to verify the convergence of the samples generated by Active Exploration. In this scene the variable parameters are the X-Z placement of a Bunny figure in an empty room (left). We fix the viewpoint to always look into the mirror and ask our generator to predict the reflection. The Bunny appears in the reflection for only a specific range of X values. We plot the target function (loss times gradients) for different X-Z values of the Bunny position (the heatmap can be seen as a top down view of the room) through training iterations, on the right. We show that the 2D histogram of Bunny placement from our Active Exploration, after a burn in period, starts following the distribution of the target function. As a result the reflection of the Bunny, when it is placed at the center of the room, starts appearing much sooner compared to uniform sample generation.

In the initial phase of data generation, known in the literature as the burn-in phase, the Markov Chain the samples do not follow the target distribution. To alleviate this issue we use 16 Markov Chains in parallel, one for each patch rendered, leading to a shorter burn-in phase. This can be seen in Fig. 5.6 'MCMC Samples".

We evaluate our proposed acceptance policy and the sample distribution in a simple scenario shown in Figure 5.6 and by disabling sample reuse. Here the 2D variable

parameter is the position of a Bunny figure in a room with a mirror at the center of its wall. We task the generator with predicting the reflection (viewpoint is fixed to always look into the mirror). For this case the static reflection of the walls is learned quite easily but the variable reflection needs the Bunny to be placed in view of the mirror. Our method correctly does so and leads to its reflection (when the bunny is placed at the center of the room) appearing much faster compared to Uniform sampling.

5.5 Training and Self-Tuning Sample Reuse

For training, we use the combination of L_1 and structural dissimilarity loss, as in Granskog et al. [28]. Since rendering is still slow it is beneficial to reuse samples as much as possible. We next discuss our self-tuning sample reuse and resolution enhancement methods.

5.5.1 Self-tuning Sample Reuse

Traditional supervised deep learning typically uses a fixed sized pre-computed dataset and runs optimization steps many times on batches, running through the entire dataset several times. Each such run is referred to as an epoch, resulting in the reuse of each data point many times.

In our case, we are generating training samples on-the-fly, and thus we do not have the notion of epochs. However, sample generation is costly (typically 2.5 sec for the 16 32x32 patches), and it is thus important to reuse training samples as much as possible, to speed up training, and also prevent the network from forgetting over the course of training. We do this by introducing a new self-tuning sample reuse strategy based on the divergence between the loss of newly seen data points and those already seen, to achieve a balance between overfitting and training speed.

Inspired by these observations, we achieve this balance by tracking two different losses $Loss_{\text{new}}$ and $Loss_{\text{exist}}$, i.e., the loss of newly generated, unseen samples and the loss of the previously generated samples that were already used to train the network. Both are tracked using an exponential moving average to lessen the effect of the stochasticity of the optimization process. When $Loss_{\text{exist}}$ starts decreasing faster than $Loss_{\text{new}}$, thus diverging from it, our model is starting to over-fit (as new data is performing worse than previously generated data). In this case we need new samples to augment the size of our dataset. This can be done by decreasing p_s, i.e., reusing with a lower probability.

We start training on-the-fly, generating and storing a new sample for the first 100 samples. After this short initialization, for each new step we randomly decide to reuse a previously generated sample, or generate and store a new one. The decision is made based on a Bernoulli distribution with a self-tuning probability p_s over steps s, representing the probability of reusing a training sample.

We build probability p_s to satisfy two goals. First we would like p_s to be as high as possible, so that we save as much computation as possible. But if it is too high, or even equal to 1, we would over-fit to the already generated samples and stop exploring the space of parameters. Thus p_s should also be sufficiently low to avoid over-fitting. Over-fitting is usually measured by the difference of performance of a model between a training and validation dataset.

We propose a mechanism with a single parameter to control p_s:

$$p_s = \sigma(Loss_{\text{exist}} - Loss_{\text{new}} + \beta) \tag{5.4}$$

where σ represents the sigmoid function and β is the parameter controlling the reuse probability when both losses are equal. This formulation decreases the probability of reusing a sample when $Loss_{\text{exist}}$ is lower than $Loss_{\text{new}}$. Intuitively the above equation is derived by associating the losses to negative log-likelihood of probability distributions parameterized by the ground truth. More details can be found in Appendix A.7. Since one component of the MCMC target function maximizes the loss (see Sec. 5.4.1) we use only large step samples to keep track of both $Loss_{\text{exist}}$ and $Loss_{\text{new}}$. In all our experiments β is set to 4.6, to have $p_s = 0.99$ when $Loss_{\text{new}} = Loss_{\text{exist}}$. When a sample is reused we build a batch of training images from the stored patches. We use the previous loss of the sample as a weight, i.e., setting the probability of selecting a patch proportional to its last recorded loss. We update the weight of a sample whenever it is reused, using the network loss on that sample in the current iteration. This allows hard samples to be reused more often and discards those for which the network performs well, leading to better adequacy between reuse and MCMC.

5.5.2 Resolution of Training Images

One of the main advantages of using a PixelGenerator for the generator architecture, as demonstrated by [28], is its performance during inference on much higher resolutions than that used for training. Shading effects that depend heavily on G-buffers such as textured diffuse materials benefit from buffer upscaling, providing improved quality.

That is less true for high frequency view dependent effects, such as reflections, that are typically small, and band-limited by the resolution of training images. Our goal is to progressively reduce the area each training sample covers, allowing the model to gradually focus on such effects.

We adopt a multi-resolution approach to address this. We start training with 32x32 patches extracted from 128 by 128 pixel images with a $90°$ field of view. Note that the MCMC controls the respective patch position on the image plane and that we only render the patch pixels. We then progressively increase the resolution of the images used to select the 32x32 patches; we found that doing so by 4 pixels every 2000 iterations worked well, all the way up to 600 by 600 which is closer to our target resolution. This shrinks the area of the patch on the sensor and allows the network to observe finer details in hard regions, such as reflections, during training. This process is made possible by the MCMC exploration, due to its ability to locally explore the scene configuration through the small steps, and to adapt to this progressive change in resolution. On the other hand, adopting such a multi-resolution approach with uniform sampling of $\mathcal{D}$ results in worse overall results as it decreases the probability of observing a given point in the scene. This makes sampling even less efficient (see Fig. 5.7) resulting in lower perceived image quality.

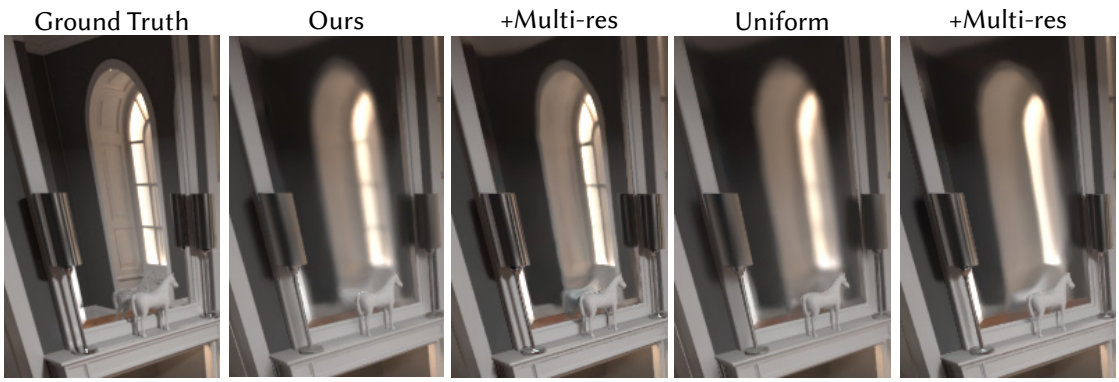


Figure 5.7: Ablation study for adaptive resolution MCMC vs Uniform training. Ours and Uniform: the resolution is always 128x128. Multi-res: we progressively increase training resolution up to 600x600.

We show results for several variable scenes. We also present comparisons to previous work and analyze the various design decisions of our solution through ablation studies and quantitative evaluation.

We have implemented our system in Python interfaced to Mitsuba 2 [84] which we use to render global illumination and G-buffers. We use between 200 and 24,000 samples

per pixel for ground truth renderings, depending on the scene see Table 5.1.

Scene	Sphere Caustic	Living Room	Bedroom	Veach Door	Bathroom	Spaceship	Veach Egg
spp	200	400	400	600	800	1200	24000

Table 5.1: Samples per pixel used for each scene during training.

We allow transformation of geometry and lights, material editing and viewpoint changes, including discrete events (e.g., changing between different materials, objects appearing/disappearing).

Our prototype implementation runs at 4-6 fps at inference/rendering time (900x900 resolution on a NVIDIA 3090 GPU), allowing interactive exploration of dynamic global illumination in variable scenes with potential applications in architecture, design, games, etc.

Currently we only show results with a forward path tracer (the only integrator available in Mitsuba 2). However, our method is agnostic to the type of integrator and if we used a different renderer, we could train with bi-directional path tracing, Metropolis or any other method.

5.6 Results, Analysis and Comparisons

In this section we will demonstrate results in multiple variable scenes, analyze our main design choices in the method and compare to state-of-the-art denoising and neural shading methods.

5.6.1 Results

We present results of our method on several modified scenes from the Bitterli dataset [8]; the viewpoint is variable in all 7 scenes except Sphere Caustic, Figures 5.1 and 5.8.

For the Bathroom scene, we added a showerdoor with variable roughness; additional variables are the intensity and position of the light source (total 8 dimensions). For the Living Room scene, we added blinds on the windows that can open and close; additional variables include the light intensity (7 dimensions). For the Bedroom scene, we have simulated variable sunlight with a distant source coming in through the window (6 dimensions). We present a modified version of the Veach Door scene, where the variable

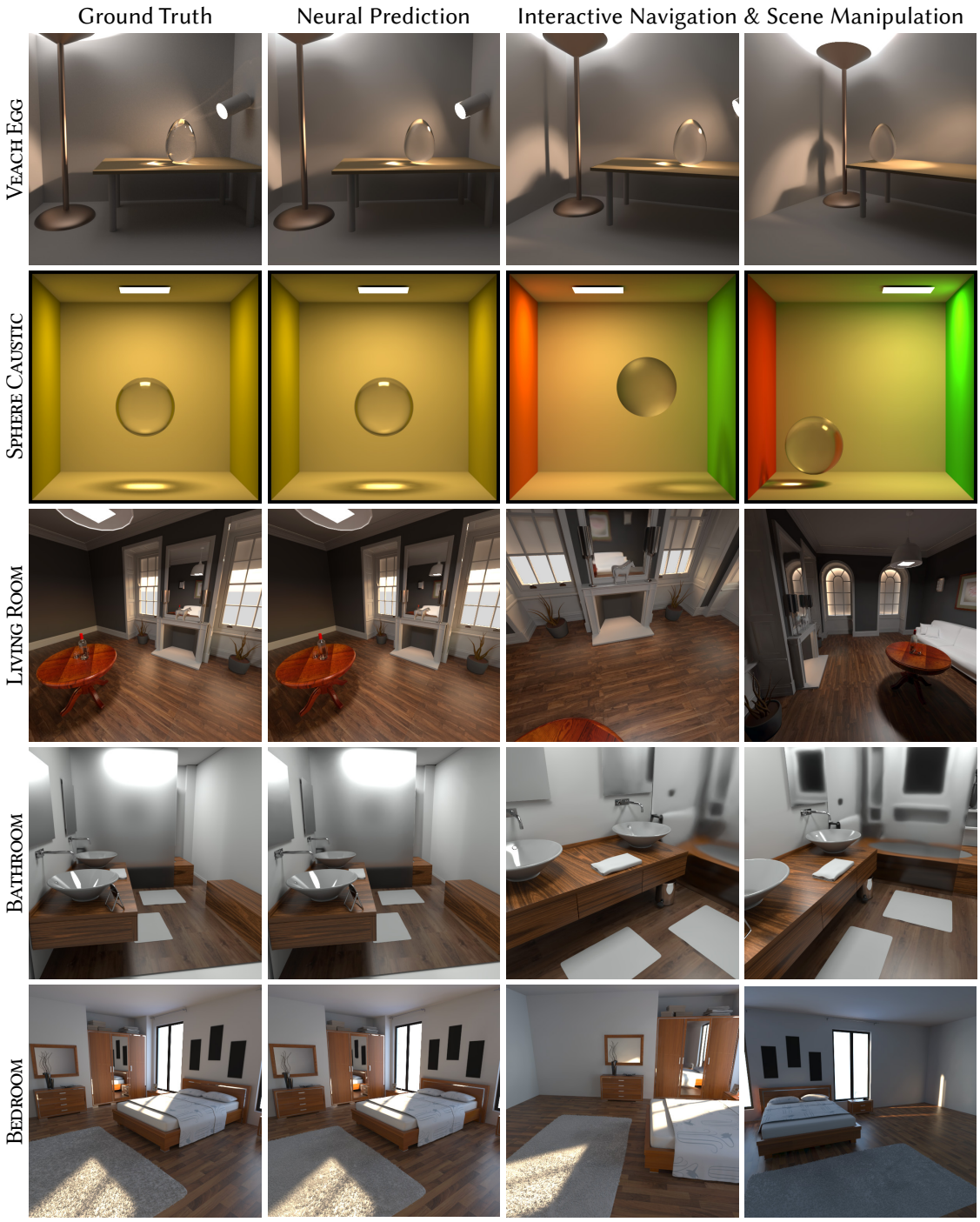


Figure 5.8: Results of our method for 5 different scenes each with different variations. Note how we can capture view changes (all but row 2) reflections, approximate caustics, global illumination etc. all at interactive rates. The scene variables include: material albedos and roughness (Sphere Caustic, Bathroom), moving and rotating objects (Veach Door, Living Room) and time of day Bedroom.

is the opening door (6 dimensions). We also have a modified Cornell Box, Sphere Caustic with variable wall colors and a large sphere that can move in the scene and vary in roughness, for a fixed viewpoint (11 dimensions). The Spaceship scene contains 3 variable emitters, 2 on the ceiling and one in the cockpit and variable viewpoint (8 dimensions). Finally in the Veach Egg scene we can vary the position of the glass egg and the spotlight emitter (9 dimensions). We show several configurations of each scene in Fig. 5.8 and Fig. 5.1.

Scene		DSSIM	MAPE	MAE	LPIPS
Veach Egg	Ours	0.0141	0.079	0.20	0.0245
Sphere Caustic		0.0012	0.031	0.01	0.0023
Living Room		0.0068	0.048	0.04	0.0220
Bathroom		0.0029	0.033	0.03	0.0089
Bedroom		0.0149	0.074	0.05	0.0512

Table 5.2: Quantitative results using 4 metrics for the configuration shown in Figure 5.8.

We train the scenes for 5-18 hours on a single NVIDIA RTX 6000. If training speed is important, we obtain a reasonable first approximation after a few hours, but longer training is required if we want to be very close to ground truth (see Fig. 5.9).

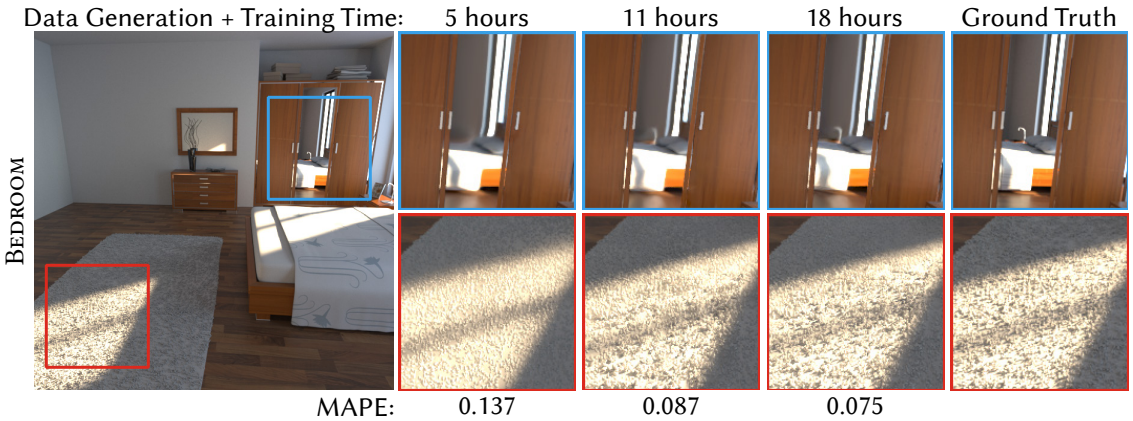


Figure 5.9: Results of our method after increasing hours of training. Depending on the application if speed is valued over quality, our method yields plausible results after a few hours of training and data generation. For the best quality possible our method requires around 18 hours in the Bedroom scene.

Different application scenarios have different hardware resources. If the target platform is a modern desktop computer with a ray tracing GPU, specular interactions can be traced and not inferred. In this case our method needs 30 minutes for acceptable results; more training is required to achieve the highest possible quality as seen in Fig. 5.10.

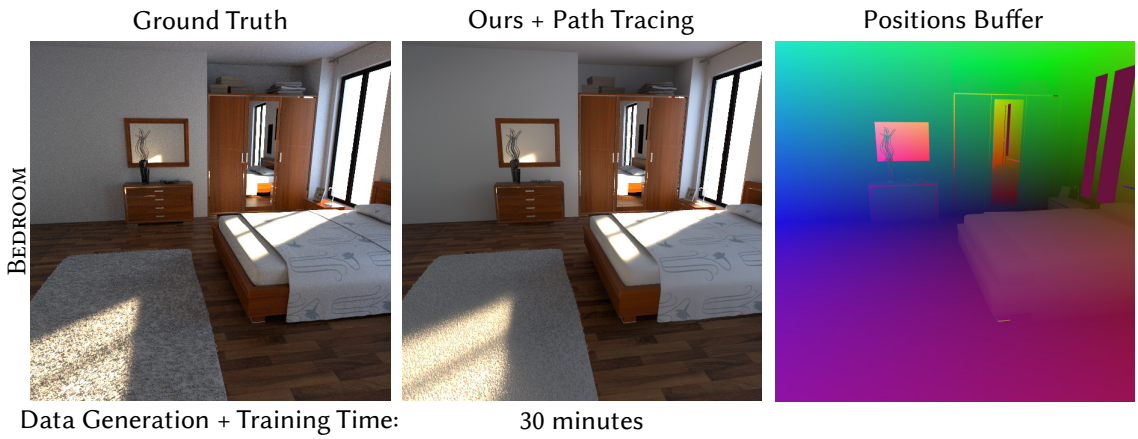


Figure 5.10: When ray tracing hardware is available our method benefits by tracing all specular bounces during the buffer generation, as in the positions buffer shown. This means that with only 30 minutes of data generation and training our method learns the non specular shading for the Bedroom scene. The harder high frequency details on the carpet still need full training to appear.

Our solution shows good temporal stability. The results show that we can capture a wide variety of light transport effects: global illumination (Living Room with different blind positions; Fig. 5.8), soft shadows, glossy (or even partially specular) reflections (Living Room, transmission (Bathroom), caustics, (Sphere Caustic, Spaceship, Veach Egg) etc. A major strength of our approach is that we can render very hard light paths with good quality at interactive rates, e.g., the caustic in Spaceship (Fig. 5.1, or the shadow from the caustic in Veach Egg, Fig. 5.14, last row). The quantitative results in Tab. 5.2 show that we achieve low error rates in all scenes.

5.6.2 Comparisons

The most significant comparison we will present is to Uniform sampling, since this clearly reveals the advantage of our active exploration approach. We also compare to Compositional Neural Scene Representations (CNSR) [28], since we share similar inputs and some goals. The comparison mainly shows the benefits of our Active Exploration, explicit scene representation, and sample reuse in terms of training and inference speed.

Finally a compelling alternative to our method for real time rendering of dynamic scenes is Real Time Path Tracing plus denoising. We compare with the state of the art denoising method of [38], further illustrating that our method is one of most efficient solutions for interactive rendering of hard light transport configurations, that require a very high sampling rate to be captured by path-tracing.

5.6.2.1 Comparison to Uniform sampling.

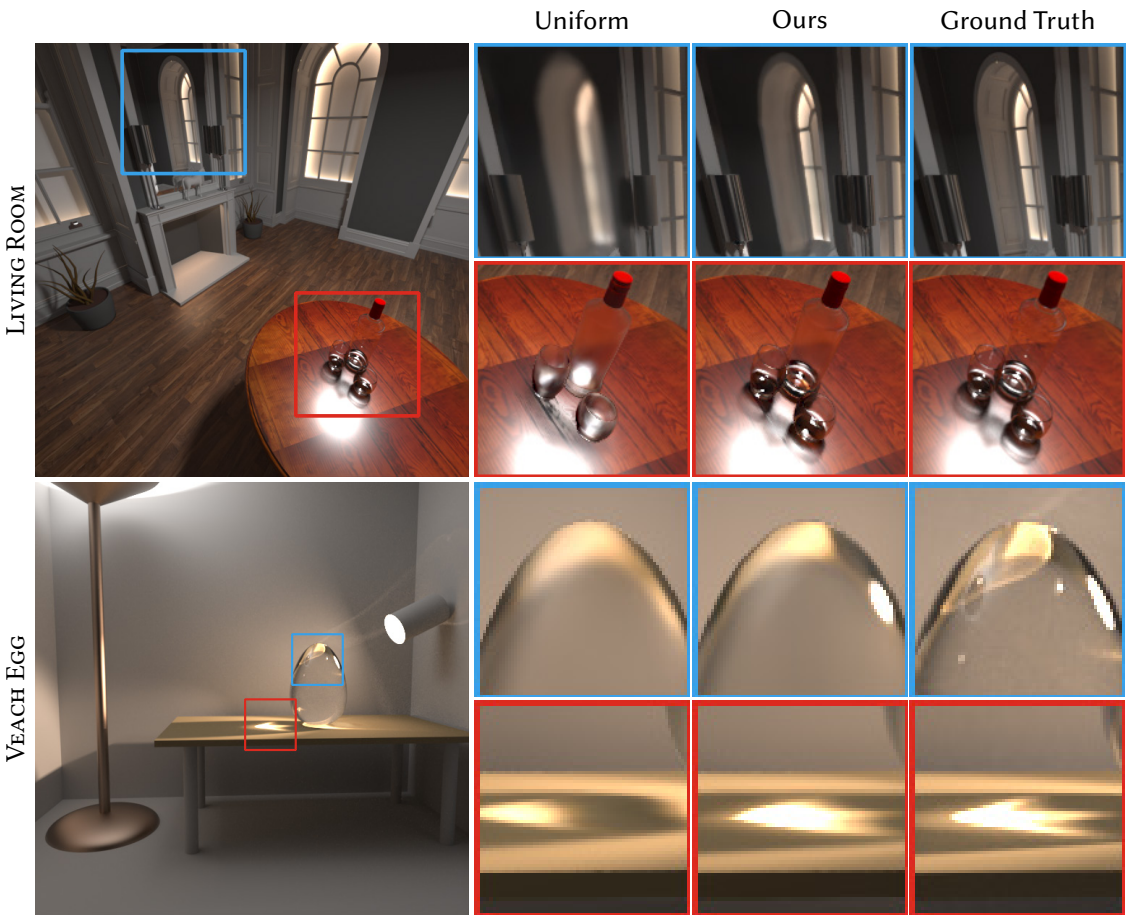


Figure 5.11: We compare our active exploration MCMC method vs. Uniform sampling of the space $\mathcal{D}$ trained for the same time. We see that Uniform search running for the same time cannot produce sharp shadows, reflections and caustics.

To evaluate the effect of MCMC active exploration our first comparison is to a simple uniform sampling baseline (in Fig. 5.11). To simulate uniform sampling, we replace our MCMC method with large steps only, that are always accepted; note that this baseline

includes our sample reuse method, but not resolution adaptation since it gives worse results (Sec. 5.6.3). As we can see, for the same computation time, active exploration achieves sharper reflections, caustics and shadows, thanks to the guiding sampling it affords. We tried to obtain equal quality with the uniform sampling, however this naive approach converges to a low quality local minimum. The results shown in Fig. 5.11 were generated with the best quality this approach could achieve; after this point in training the loss does not decrease. In all cases, our method provides sharper results, generally much closer to the ground truth. This is confirmed with quantitative analysis in Tab. 5.3.

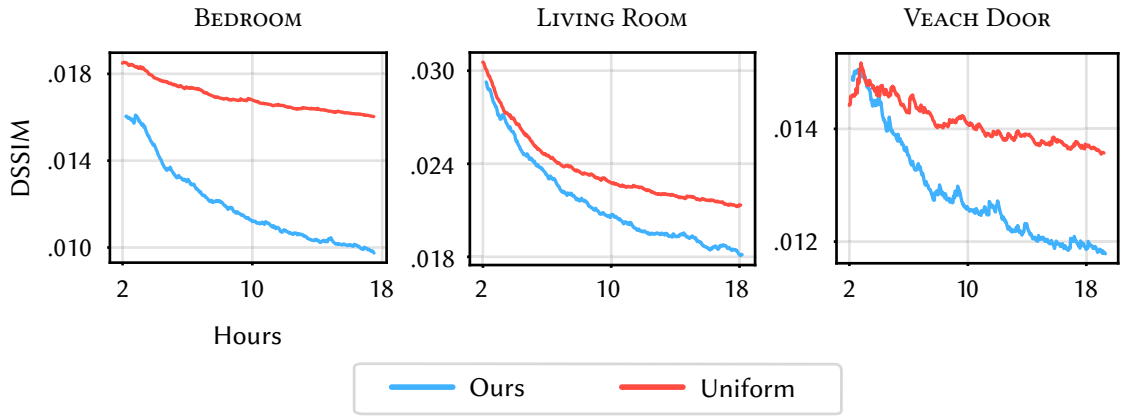
Scene		DSSIM	MAPE	MAE	LPIPS
Living Room	Ours	**0.0141**	**0.079**	**0.20**	**0.0245**
	Uniform	0.0241	0.162	0.28	0.0652
Veach Egg	Ours	**0.0116**	**0.065**	**0.22**	**0.0477**
	Uniform	0.0147	0.076	0.25	0.0738

Table 5.3: Quantitative results using 4 metrics for the configuration shown in Figure 5.11.

We also show quantitative results in Fig. 5.12 using the Mean Absolute Percentage Error (MAPE), DSSIM [65], Mean Absolute Error (MAE) and LPIPS [119] error metrics and a graph with the evolution of error over time. Since our main goal is to handle difficult lighting configurations, we select 10 frames from each path of each scene which correspond to such cases, and evaluate our method against ground truth; we show the selected frames for each scene in Appendix A.1.

5.6.2.2 Comparison to CNSR

We compare our method to Compositional Neural Scene Representations [28] (CNSR) using the variable ArchViz scene, the more complex of the two datasets used in CNSR; Our implementation of this scene has 71 dimensions. For best-effort same quality comparison we use a pretrained model provided by the authors. Note that the ArchViz dataset ‘consists of variations of a living room with a dining area” [28]. Both the pretrained model of Granskog et al. [28] and ours are trained on identical data involving variations of this scene. We recreated the ArchViz variable scene in our framework as closely as possible, using publicly available resources [28]. The CNSR pretrained model is trained



	DSSIM	MAPE	DSSIM	MAPE	DSSIM	MAPE
Ours	**0.0097**	**0.0458**	**0.0180**	**0.0892**	**0.0117**	**0.0712**
Uniform	0.0160	0.0791	0.0212	0.1058	0.0135	0.0752

Figure 5.12: Quantitative evaluation of our method and ablations compared to ground truth (graphs start at 2h of training.)

on a dataset of 9000 sample points. Each point includes 16 batches of 3 observations at 64x64 resolution and a query image at the same resolution, trained for 1M iterations.

The high complexity of this scene's variations (constrained, specific configurations, e.g., the teapot appears at a specific position on the table etc.) challenges our base method; we show results using 256 features/layer and without resolution enhancement. This gives blurrier results (on a par with Granskog et al. in terms of quality) but avoids high frequency artifacts. Our method achieves the same qualitative results with *36 hours* of *both training and rendering*. In comparison Granskog et al. [28] needs *11 days of only training* (accounting for hardware differences), and an unspecified amount of rendering time to generate the data.

We also retrain CNSR on three of our scenes, providing same time comparisons on Living Room, Bedroom and Veach Door. For this we used the publicly available code provided by the authors to train on data generated by our framework. As in the case of the ArchViz scene we use 16 batches of 3 observations at 64x64 resolution and a query image at the same resolution to train their model.

The results of the same quality ArchViz and same time Living Room comparisons with Granskog et al. [28] are shown in Fig. 5.13. Additional examples are shown in the Appendix A.2. Our method achieves much sharper results that are significantly closer to

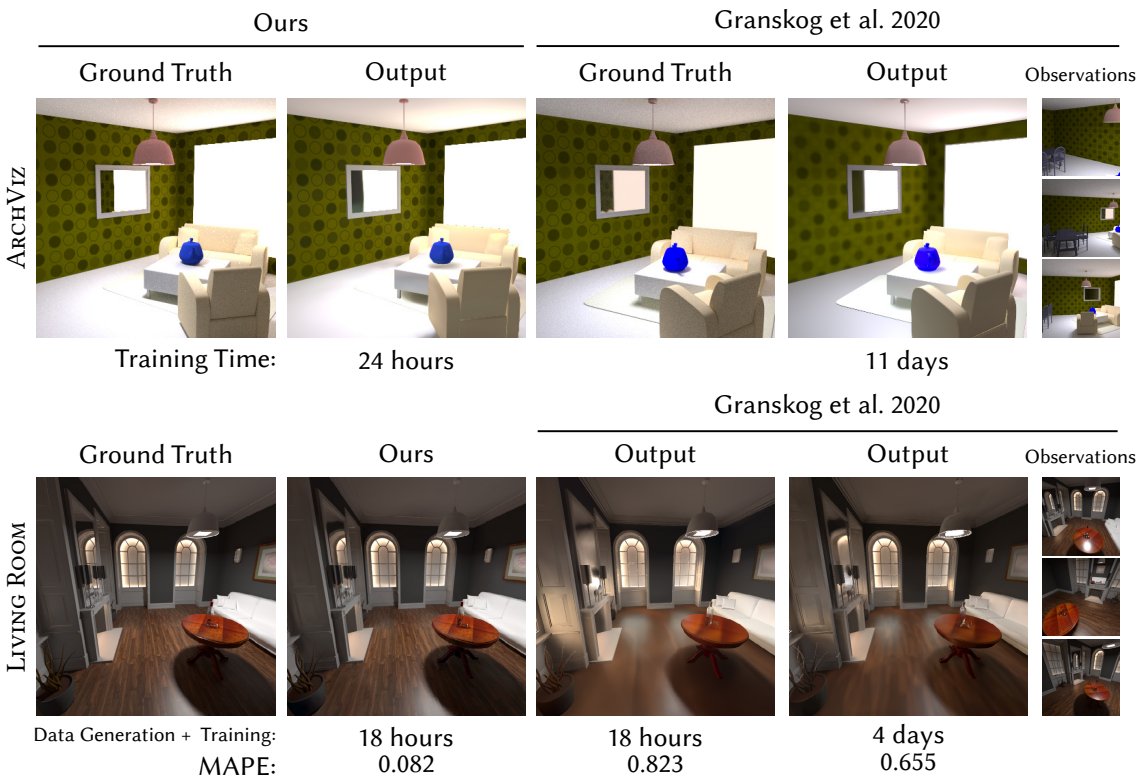


Figure 5.13: Same quality (top) and same time (bottom) comparison with Granskog et al. [28]. We show result of ArchViz for same quality as ours using the pretrained network provided by the authors in their rendering framework. We also show Living Room for same time as ours by training their method on our variable scene in our rendering framework. The 3 path traced observations required by Granskog et al. [28] are shown on the right in both cases.

the ground truth.

We want to note that this comparison is provided only as an indication of the efficiency of our approach, since the goals of the two methods differ in several ways.

5.6.2.3 Comparison to ANF

We compare with the recent Affinity of Neural Features (ANF) denoising method [38] in Figure 5.14. For a fair comparison we take the pretrained model provided by the authors and fine tune it in each specific scene, using the authors original implementation. Since our method uses a different renderer than ANF (Mitsuba 2 vs PBRT v3), we give the same budget in terms of pixels generated during fine tuning. Also we fine tune the pretrained ANF model using sequences of 8 frames in random paths as in the original method. Fine-tuning improves temporal stability (please see videos), and sometimes improves

visual quality (e.g., sharper results for Spaceship). Finally during inference we provide an 8 samples-per-pixel (spp) input image along with all the buffers required (albedo, depth, etc.).

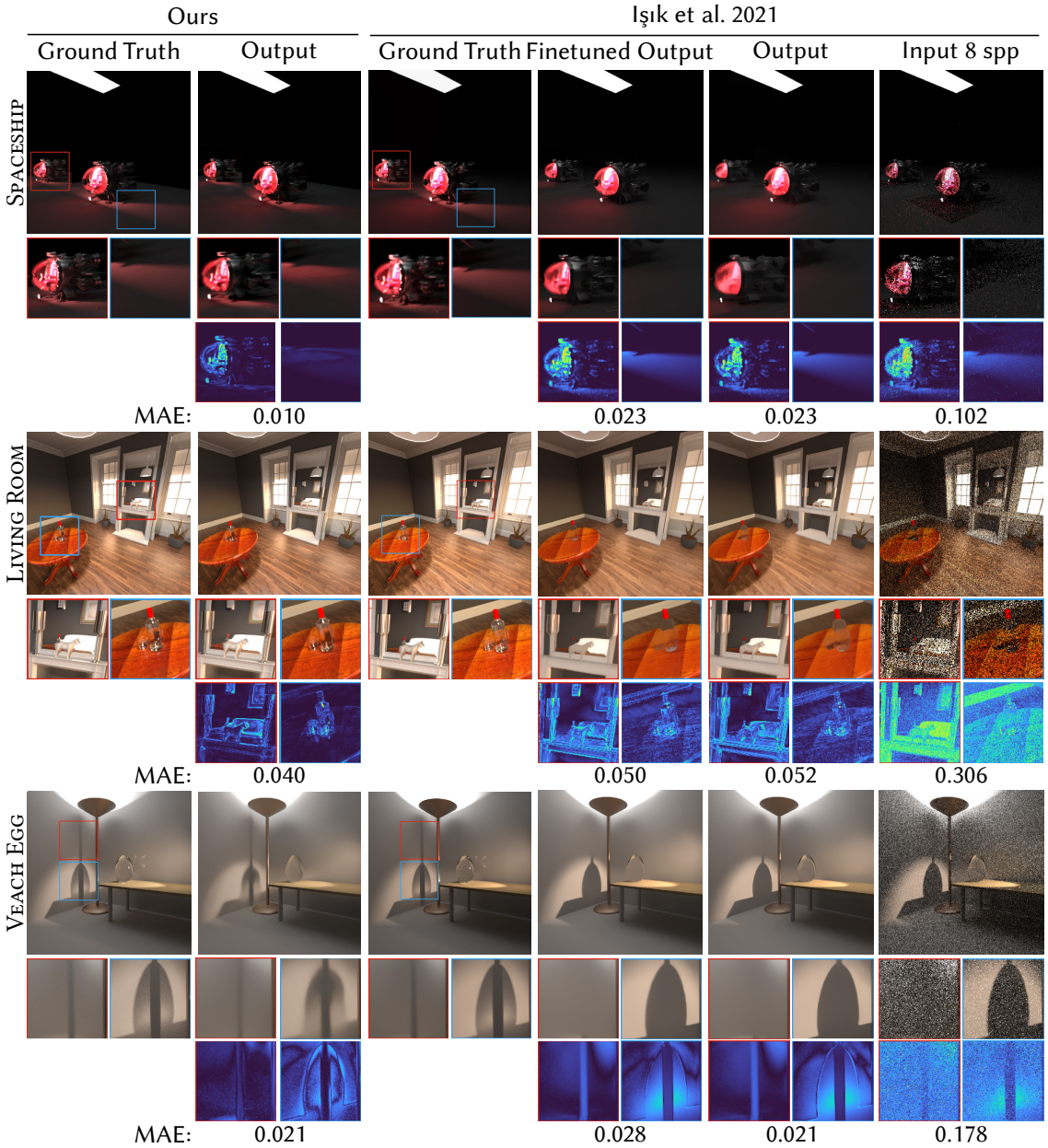


Figure 5.14: Same time comparison with Işık et al. [38], fine-tuned on our scenes. Note how our neural renderer captures hard light paths, e.g., caustics (Spaceship) or even shadows from caustics (Veach Egg) that are almost completely missing from the path-tracing + denoising solution.

Our method demonstrates better temporal stability especially in parts of the scene where the noise in the input is higher such as the reflections in the Spaceship and Veach Egg scenes. ANF manages to successfully reconstruct parts of the scene where there is a big correlation between the input buffers and the final color, such as diffuse walls in Living Room, and parts where the light effect exists in the noisy input, highlights on the floor in Living Room. The limitation of ANF is clear in cases of complex light effects that do not appear in the noisy input due to the low spp and where the input buffers do not help, such as the red caustic in Spaceship, the bottle caustic in Living Room and the complex shadow of the glass egg Veach Egg.

Scene		DSSIM	MAPE	MAE	LPIPS
Spaceship	Ours	**0.0155**	**0.047**	**0.001**	**0.0176**
	Işık et al. 2021	0.0461	0.068	0.023	0.0693
	+ Finetuned	0.0483	0.067	0.023	0.0659
Living Room	Ours	**0.0074**	**0.051**	**0.040**	**0.0287**
	Işık et al. 2021	0.0164	0.096	0.052	0.0691
	+ Finetuned	0.0205	0.109	0.050	0.0722
Veach Egg	Ours	**0.0170**	0.082	**0.021**	0.0844
	Işık et al. 2021	0.0182	**0.071**	**0.021**	0.0765
	+ Finetuned	0.0194	0.081	0.027	**0.0758**

Table 5.4: Quantitative results using 4 metrics for the configuration shown in Figure 5.14.

These effects have a significant impact on the observed realism of the scene. However, they are completely missing from the path-traced+denoising solution, despite these effects being present in the ground truth images used for fine-tuning (provided in Appendix A.3). Quantitative results are shown in Tab. 5.4; our method outperforms Işık et. al. [38] in Spaceship and Living Room. For Veach Egg, two metrics give our method a lower score, even though we clearly capture indirect effects that are completely missing in Işık et. al.

This illustrates one of the major strengths of our approach: the only way to render such hard light transport in a path-tracing context is to dramatically increase the number of samples per pixel. In contrast, our method encodes light transport in the neural network and uses the explicit scene representation vector to get information about such

effects, such as the position of the glass egg or the cockpit light. As a result, we achieve interactive rendering with all effects present for the same training time.

5.6.3 Evaluation

We first study the effect of the number of variable dimensions, then present other ablations concerning different design choices of our method.

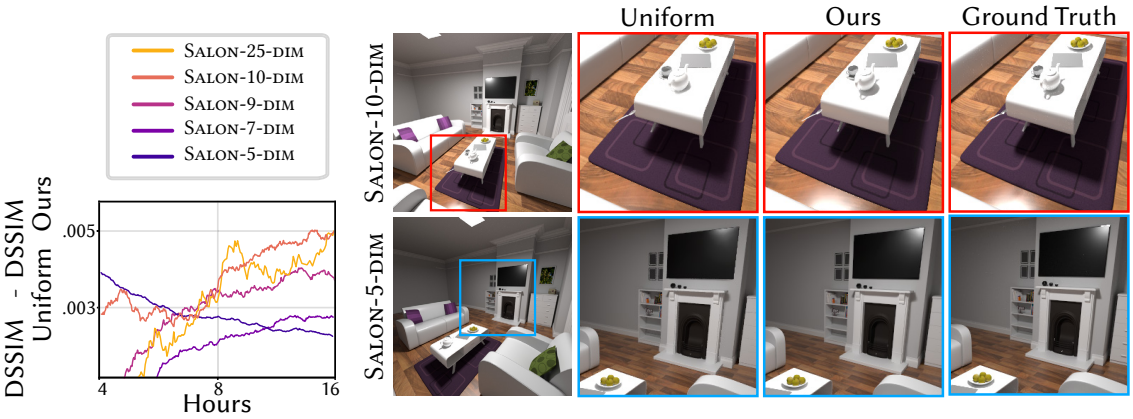


Figure 5.15: Study of the number of dimensions on the effectiveness of our approach on the Salon scene. Left: we show the difference between the loss using the uniform approach and our method; the graph starts at 4h of training. In general, the benefit of our method increases with the number and complexity of the variable elements in the scene, but some elements are more important: despite going from 10 to 25 dimensions the difference of gain between Salon-10-dim and Salon-25-dim – which only involves albedo changes – is smaller than adding a single important dimension such as light position (difference from Salon-7-dim to Salon-9-dim. Right, for low dimensions (bottom row, Salon-5-dim) our method slightly improves the glossy highlight on the TV compared to uniform sampling; however, once the dimensions increase (top row, Salon-10-dim), we capture a many effects completely missed by the uniform, namely the TV and floor glossy highlight as well as the detailed shadows of the teapot on the table.

Study of number of variable dimensions. We investigate the impact of the number of scene variables on our results. We trained our method and the uniform approach described above on 5 increasingly variable variants of the Salon scene. The first variant – Salon-5-dim – only varies viewpoint (5 dimensions), Salon-7-dim adds a movable set of furniture on the floor (7 dimensions). In Salon-9-dim the light source moves on the ceiling (9 dimensions). In Salon-10-dim the roughness of the wooden floor is also variable (10 dimensions). To demonstrate that some variables have a bigger impact on training time than others, e.g., light source position compared to changing albedo, we

introduce Salon-25-dim which also varies the albedos of the furniture and walls (25 dimensions). In Fig. 5.15 we see that while for Salon-5-dim the difference in validation loss between our method and the uniform sampling is small, Salon-10-dim demonstrates that the benefit of our method increases with higher numbers of variable dimensions. For this case, uniform search almost completely misses the important highlight on the glossy wooden floor due to the very specific configuration of parameters that create it. As a result Active Exploration is crucial for scenes with many variable elements, such as the ones used in production.

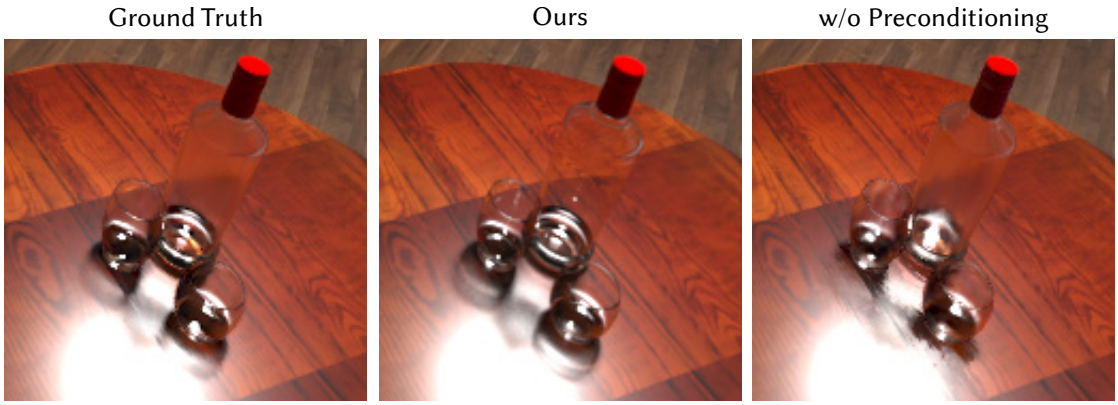


Figure 5.16: Position Preconditioning allows the generator to ignore the high frequencies of the wood texture when it forms the shadows and caustic, resulting in better quality.

Scene		DSSIM	MAPE	MAE	LPIPS
Living Room	Ours	**0.0141**	**0.079**	**0.20**	**0.0245**
	w/o Preconditioning	0.0184	0.098	0.25	0.0393

Table 5.5: Preconditioning improves the quantitative performance (see also Figure 5.16).

Ablation: preconditioning on position. In Fig. 5.16 we show the difference in results between our full method and an ablation where position is concatenated with all other dimensions and fed directly to the network. We see clearly that the position preconditioning greatly improves overall performance. In the Living Room scene the albedo buffer for the table has high frequency variations due to the wood texture. Without the preconditioning it is hard for the PixelGenerator to learn to ignore this information

when shading the caustic and the shadow of the bottle. Quantitative results in Tab. 5.5 confirm this choice.

Scene		DSSIM	MAPE	MAE	LPIPS
Living Room	Ours	0.0201	0.135	0.23	0.0590
	+Mutli-res	**0.0141**	**0.079**	**0.20**	**0.0245**
	Uniform	0.0241	0.162	0.28	0.0652
	+Multi-res	0.0250	0.117	0.38	0.0573

Table 5.6: Quantitative results illustrating the effect of resolution on error.

Ablation: increasing resolution. We next study the effect of progressively increasing resolution during training (Sec. 5.5.2). In Fig. 5.7, we compare to an ablation where we do not increase resolution during training. We can see that the increase in resolution allows our active exploration to resolve high frequency effects such as reflections and shadows (lamp on the left) much more effectively. The corresponding quantitative results in Tab. 5.6 confirm the improvement in quality.

Figure 5.17: Use of the loss alone for the target function results in blurrier results.

Scene		DSSIM	MAPE	MAE	LPIPS
Living Room	Ours	**0.0141**	**0.079**	**0.20**	**0.0245**
	Loss Based	0.0149	0.085	0.22	0.0306

Table 5.7: Quantitative results using 4 metrics illustration the benefit of our choice of target function (see in Figure 5.17).

Ablation: target function. In Fig. 5.17 and Tab. 5.7, we see that using only the loss for the target function degrades quality, since the training process gets stuck in local

minima. The MCMC finds configurations that cannot be improved anymore, such as the mirror reflection, and does not accept other states where the network could still improve, such as the bottle caustic. As a result the latter is lacking detail.

5.7 Future Work, Limitations and Conclusion

Despite providing interactive global illumination in dynamic scenes, our method is not without limitations; we discuss these below together with some avenues for future work before concluding.

Rendering using our unoptimized Python implementation currently runs at 4-6 fps, including a 15ms overhead for generating G-buffers in Mitsuba – which could be performed with hardware acceleration – and an unoptimized inference step. We are confident that significant speedup can be achieved with further optimization. We chose to learn *all* light paths, including mirror reflections. While we achieve acceptable results in many cases, high-frequency effects may not be reproduced exactly. However, our approach can be used in a hybrid setting, using real-time ray-tracing for specular interactions as seen in Fig. 5.10, overcoming this issue. If the use of path tracing is not an option, Neural Textures such as the ones used in [109] could improve reflections in cases where the G-Buffers do not provide any meaningful information.

The Active Learning literature has explored many different metrics for deciding the value of each sample. In this work we explored two functions that can be efficiently computed in a single GPU training scenario but there are alternatives. In a multi GPU training scenario one option is to use a query by committee. Different copies of the model could be trained in parallel in each GPU and whenever a large step is performed all the models could be evaluated on the proposed state. Using the prediction variance of all the models' answers can be a good fit for a target function as it shows there is uncertainty on what the result should be. Additionally Bayesian Neural Networks [80] with explicit access to uncertainty metrics could possibly be an option for our Active Exploration in the future.

One aspect we would like to explore in future work is how to take into account the importance/difficulty of each scene variable. From our tests different variables can have a different impact on the scene's global illumination and can be harder/easier to represent by the generator. In general, variables that create or control high frequencies, such as reflections and shadows, are much harder to learn than variables such as the

color of emitters or objects. Explicit injection of this knowledge using some form of Importance Sampling could help reduce training times and improve quality. Another property of the variables that we do not handle explicitly is the difference in their ranges. Since we normalize each variable the network needs to learn to scale the normalized values accordingly to match their impact on the final rendering. For instance for two rotatable objects that can be rotated 360° and 15° respectively the network in both cases will receive values between 0 and 1 even though the first object will create much higher frequency shadows in that range. Finding a way to adapt the MCMC mutations to such range differences per variable could increase the efficiency of Active Exploration.

The types of variables we demonstrated can have a big impact on the overall appearance of our scenes but they are simple to represent with a few floats (rotation, translation, roughness etc). In future work we would like to expand our method to variables that are difficult to represent such as the parametric deformations in [105]. We believe that finding inventive ways to represent such variability (such as using the keyframe as a parameter) is a promising avenue of future research.

We still can require up to 18 hours of training time for a given scene, depending on the required quality and the number of variable parameters. As discussed earlier, the network architecture used can play a significant role in the quality of the results; it is possible that different architectures will further improve quality and thus training speed. Another possible extension could be to train with a set of variable parameters and allow fine-tuning of the network, e.g., allowing fast addition of a new object etc. Evidently, use of a faster path-tracer could also accelerate training.

In future work, we believe our Active Exploration approach has significant promise for any neural rendering method (e.g., [2]) that trains on synthetic data, allowing potentially significant reduction in training time and improvements in quality.

One limitation by design for our method is that we cannot handle thousands of variables. The scene representation vector is repeated to match the size of the G-Buffers so that the generator, which operates on a per pixel basis, is aware of the global state of the scene. For example, given 5000 variables (such as a variable texture) we would need to create a tensor of size 128x128x5000 that would be unmanageable in terms of memory. In such cases there is a need to encode this information in a different way, possibly through an encoder neural network. Our method, as shown in Figure 5.18, can work with 128 variables with similar training times (18 hours) but the quality is lower than in simpler

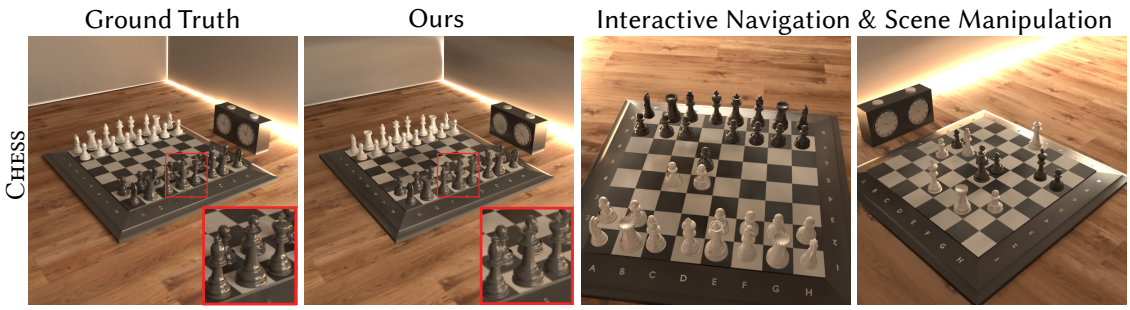


Figure 5.18: The Chess scene tests the limits of our method with 128 variables. Each chess piece can be moved on the board, lifted and captured. Our method still gives plausible results but is missing some shadows and highlights.

scenes (some missing highlights and shadows).

In conclusion, we introduced a resolution-aware Active Exploration method that guides the sampling of the training data space, and a self-tuning sample reuse method that enables interleaved on-the-fly data generation and training.

Our neural renderer, combined with our explicit scene instance parameterization vector, uses these contributions to capture hard light transport effects, allowing interactive exploration with full global illumination, including all light paths.

Using these elements we can render variable scenes after 5-18 hours of training, depending on scene and variation complexity and the quality required, including indirect lighting, shadows, transmission, glossy effects etc. Looking forward, we believe that our main contributions can be used beyond the precomputation scenario presented here: Active Exploration and self-tuning reuse could be used for future solutions that can provide data online, e.g., with real-time path tracing.

Chapter 6

MesoGAN: A Generative Model for Mesoscale Materials

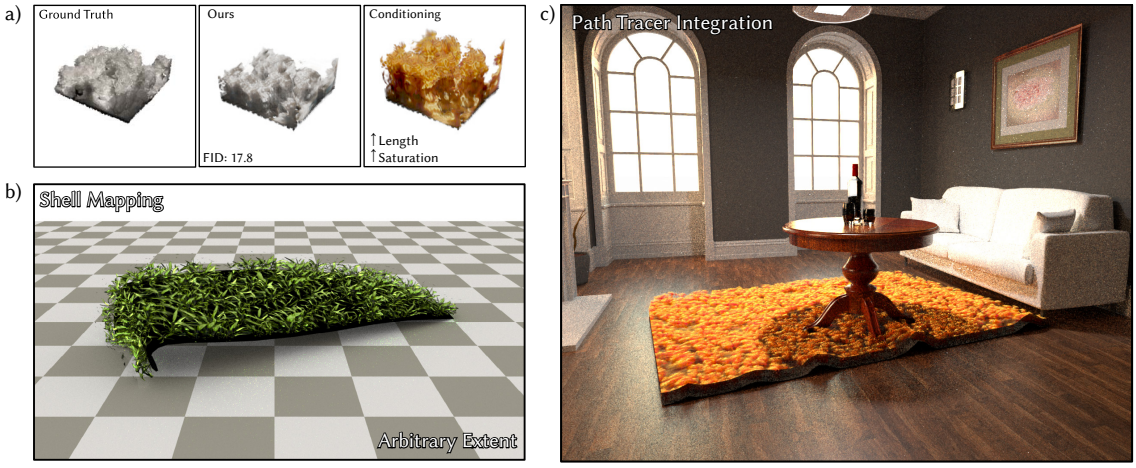


Figure 6.1: Our method combines the strengths of StyleGAN and volumetric neural field rendering to generate a 3D mesoscale texture that can be mapped to objects and used in a path tracer (c). We train on datasets of synthetic patches (a); our method can generate textures that have artistic parameters (such as fur saturation and length) which can be used to create shell maps of arbitrary extent (b)

In this chapter we expand on the concept of using neural networks to represent a single radiance field with variations. Our goal is to introduce a generative model for volumetric radiance fields which represents different classes of mesoscale materials such as fur, grass etc, that can be used directly in a path tracer. Our model, Meso-GAN, builds on top of previous 3D aware generative models to create infinite 3D neural textures with high quality, explicit artistic control and stochastic detail.

While micro- and macroscale structures can be well represented using statistical and geometrical models, representing the visuals at the transition—the mesoscale appearance—remains a challenge. The challenge stems from the need to sufficiently resolve visual details at a fraction of memory storage and authoring load of geometrical models. We propose to strike the right balance between visual accuracy and storage, and between

artistic control and authoring load, using a data-driven volumetric model based on generative adversarial networks.

Our work is inspired by the realism and visual diversity of image generators like StyleGAN 3 [52]. These generators reproduce the rich mesoscale details and irregularities that are key to photo-realism. Unfortunately, their reliance on screen-space processing precludes their use as material primitives in light transport simulators such as path tracing. Operating with independent rays means there is no access to local neighborhood information which is necessary for operations such as convolutions. For rendering purposes, one could instead operate directly in world space using models based on volumetric neural fields [98; 13]. However, these approaches do not yet deliver the necessary visual fidelity. We posit this stems from the fact that scene content is generated for independent point queries, which hinders exploitation of spatial correlations. Our goal is therefore to design a generator that can initially leverage information from the entire neighborhood, while still relying only on independent point queries at rendering time, while providing explicit artistic control.

We base our method on the recently proposed geometry-aware GANs [14] and NeRF textures [1]. Expanding on these methods we

We introduce a novel approach for generating 3D neural textures that can be easily incorporated into a modern path tracer with explicit control over their appearance and geometry. Our method builds on elements from the recently proposed geometry-aware GANs [14] and NeRF textures [1] which we adapt and expand to achieve our goal. First we discuss how to adapt the StyleGAN 3 generator into an infinite texture generator with no seams. This is necessary to create a single consistent high resolution texture instead of repeating a single low resolution exemplar. This infinite texture cannot be lifted into 3D in the same way as previous methods suggested so instead we propose an alternative novel representation.

With these components our approach works well but for low resolutions; our restriction of using independent point queries at inference do not allow us to use upsamplings and convolutions after the ray marching step to achieve higher resolutions. Such operations also harm the view consistency of the results which is crucial for our goal of integrating our method into path tracing. Treating this by increasing the ray marching resolution leads to memory issues. Previous methods have handled this by either discarding the gradients for some parts of the image [123] or by rendering patches [99].

While this enables training in higher resolutions it results in longer training times and lower quality. Instead we introduce a mipmapping approach and allow the network to observe the materials from close up, thus learning the details while keeping the training resolution and training times low.

Once trained, our material model represents an entire distribution of mesoscale structures. Given a 3D model we apply our neural textures using a shell mapping approach by extruding the surface of the model. Once rays from the path tracer hit the extruded surface we switch to ray marching, evaluating our model conditioned on the incoming light direction. As a result the rendered mesoscale materials have a realistic appearance with global illumination effects and stochastic details.

Similarly to other neural materials [1; 59], we do not address energy conservation, reciprocity, or importance sampling; these are left for future work. Our work is exploratory: we present a prototypical neural material that features generative modeling. This is a significant advancement as it aims at capturing an entire distribution of materials rather than a single instance, and at providing means to draw unique infinite support material samples from the distribution to combat repetitive artifacts.

6.1 Related Work

We already reviewed related work to the generative aspect of our method in Section 3.3. Here we will discuss how conventional and neural models have represented mesoscale materials in the past.

6.1.1 Conventional models of mesoscale appearance

Mesoscale materials, such as fur and granular media, are extremely complex in their appearances. In addition to this, they are often challenging to accurately capture and model, requiring different methods for each type. In graphics, fur is often modeled using geometric curves, where each curve represents a strand of fur. Alpha-masked triangle meshes are commonly used for modeling vegetation, such as plants and grass, whereas granular media is often represented using large numbers of instanced volumetric primitives [69; 73]. Volumetric primitives have also found some use in representing leaves and fur [45; 22; 81]. Each representation has its own weaknesses such as poor filtering (levels of detail) or high memory consumption. The lack of a single versatile mesoscale primitive to handle all these cases robustly is apparent.

6.1.2 Neural material models

Recent years have seen the emergence of neural bidirectional texture function (BTF) models [21] suitable for representing mesoscale structures [88; 89]. Such neural models can be extended to perform filtered queries for robust level-of-detail rendering [59]. BTFs however operate strictly in texture space, resulting in 'vanishing" mesoscale structures when seen from the side. Instead, we opt to model the mesoscale structure as a volume.

6.2 Method

We will first review the concept of neural radiance fields and their applications and then discuss the core components of our method.

6.2.1 Neural Radiance Fields

We have defined a volumetric outgoing radiance field as a scalar field $L_o(x, \omega)$ where $x \in \mathcal{V}^3$ and $\omega \in \mathcal{S}^2$. Mildenhall et al. [71] proposed to represent the extended volumetric radiance field $L(x, \omega_o)^+$, which includes the scattering coefficient $\sigma_s \forall (x, \omega)$[1], using an MLP. The parameters of the MLP are optimized to represent this field $f_\theta : (x, \omega) \rightarrow L_o^+$ given a number of ground truth images.

Once optimized, the learned representation L_o^+ can be used to compute the radiance arriving at the sensor following the volume rendering equation (Equation 2.7). This value is estimated using quadrature in the form of ray marching. A ray $r(t) = \mathbf{o} + t\mathbf{d}$ is shot through each pixel and the network is queried at regular intervals along each ray. In this way the network learns to represent the appearance but also the geometry of the given scene.

Tancik et al. [107] showed that if NeRF is trained by mapping (x, ω) directly to the outgoing radiance L_o^+ it converges to a blurry result. They proposed to encode the input position x using Fourier frequencies in the form of:

$$\gamma(x) = [\sin(x), \cos(x), \cdots, \sin(2^{R-1x}, \cos(2^{R-1x})], \tag{6.1}$$

and accordingly the direction ω, where R is the number of frequencies used for the encoding.

[1]The term used in NeRF is volume density.

6.2.2 Overview

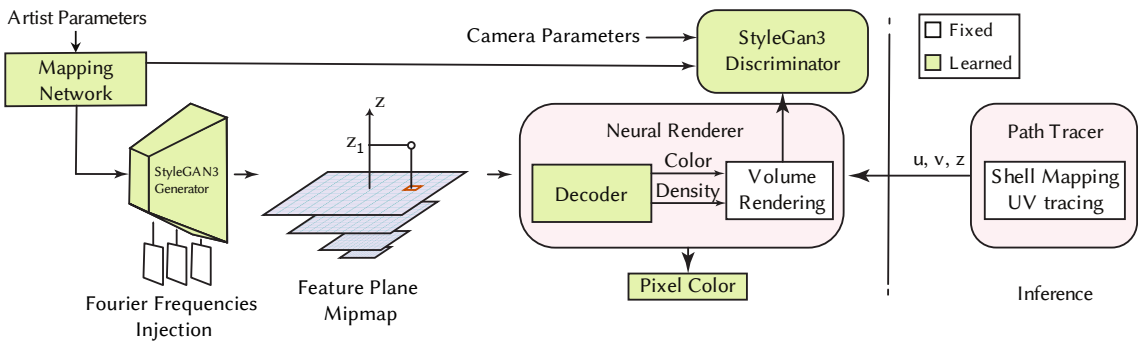


Figure 6.2: Overview of our method.

Our goal is to train a generative model on meso-scale content with the following restrictions:

- The model needs to be able to generate an infinite 3D neural texture of mesoscale materials without any seams, with high quality, view consistency (when the camera moves) and without aliasing.
- With the target application of incorporating our method into a path tracer, convolutions, upsamplings with interpolation and any other type of image space neighborhood operation cannot be used during the ray marching step.
- Explicit conditioning of the appearance and geometry of the generated neural texture is essential, including conditioning on the incoming light direction to enable global illumination rendering through path tracing.

All these requirements create a unique scenario in which no previous method can succeed. Our main contribution is our training pipeline which involves our infinite texture StyleGAN 3 [52] generator, a texture space single plane representation and a multi scale training procedure. We also demonstrate how our neural primitives can be conditioned and textured onto any given model through shell mapping. This model can then be placed in path traced scenes and support all global illumination effects. An overview of our method is shown in Figure 6.2.

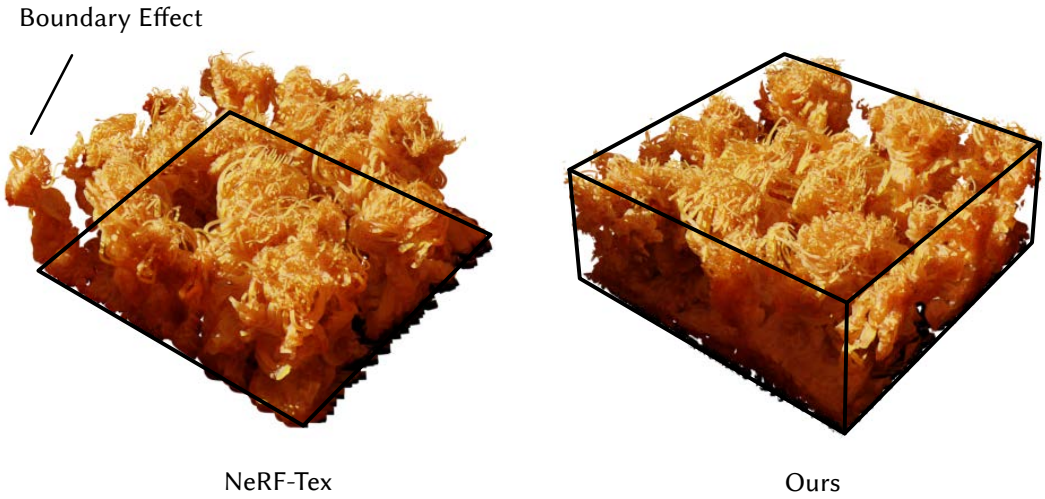


Figure 6.3: Our patch of mesoscale material generated by Blender compared to NeRF-Tex [1]. In NeRF-Tex the patch is defined at the base of the material allowing it to grow outside of it and have boundary effects. In our case the patch defines a bounding box where all the material is contained within.

6.2.3 Data Generation

We generate our synthetic datasets using Blender by placing a camera randomly on a hemisphere looking at the target mesoscale material. Similar to NeRF-Tex [1] we create a patch of the material which is rendered from different viewpoints. Additional patches are placed around the main patch to simulate the effect of surrounding geometry. There are three main differences in our data generation compared to NeRF-Tex. The first is that since we are training a generative model the structure of the material is randomized in each instance. In NeRF-Tex the dataset included multiple views of the same mesoscale material instance. While there were some parameters that could control the geometry of the patch, having a parameter that controls the structure cannot be handled as we show in Section 6.3.

The second is in the way that the patch of materials is created. In NeRF-Tex, given a patch of a specific size the material was created so that its base is within the bounds of the patch. This means that fur, for example, could extend outwards and over these bounds. In their method this was desirable as it helped with the seams when repeating an exemplar on a model. In our case we treat each instance as a 3D crop of an infinite volume in the tangent direction. As such we crop everything outside of a predefined

bounding box, see Figure 6.3.

Finally with the target application of integrating our model into a modern path tracer we generate high dynamic range images. Since training the network directly on the HDR images has been demonstrated to be problematic [60] we tonemap the images using the $\frac{x}{x+1}$ tonemapping function proposed by Reinhard et al. [92]. At inference we invert the tonemapping to generate our renderings with high dynamic range.

6.2.4 Infinite StyleGAN 3 Generator

The StyleGAN 3 generator uses a Fourier feature map as its building block, following previous work [118] that showed GANs benefit from explicit spatial bias to create patterns contained in natural images. This Fourier feature map defines, by definition, a naturally spatial infinite plane. In StyleGAN 3 the frequencies of this feature map are sampled randomly but kept constant throughout training. A learned affine transformation allows the generator to translate this map so that it can learn specific patterns such as eyes once, and then use it to create faces in different parts of the image.

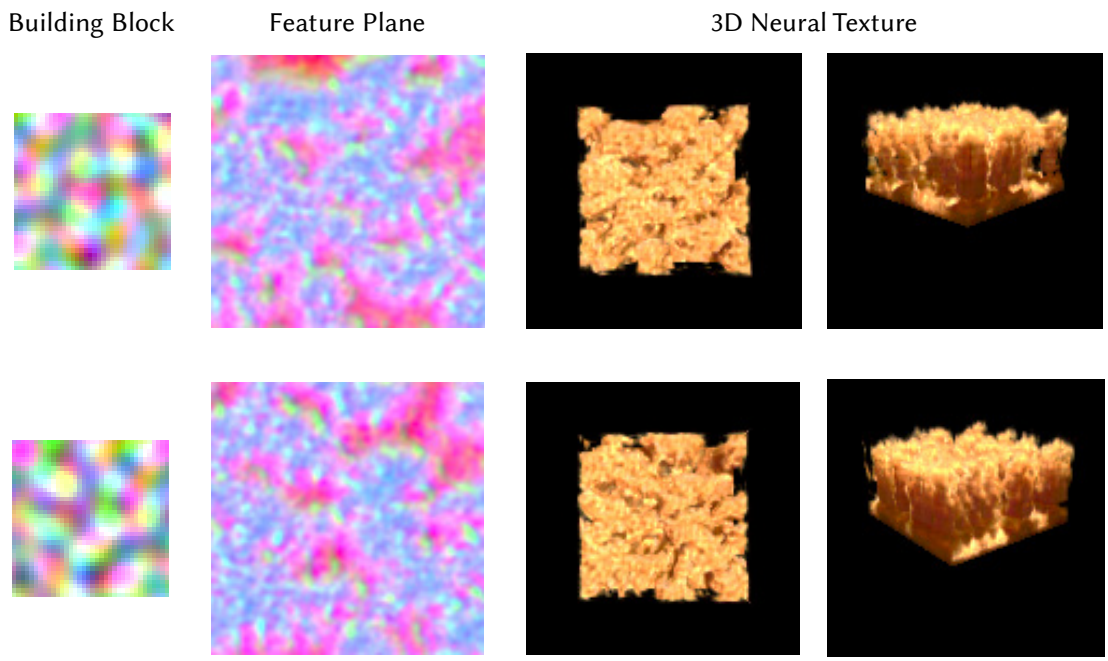


Figure 6.4: Once trained, the StyleGAN generator transforms patterns in the Fourier feature map bulding block into patterns of the mesoscale material in the feature plane. We demonstrate this for two random instances of a building block. Notice how the structure of fur changes.

In our scenario StyleGAN 3 is tasked with generating textures where there is no high level structure to the small scale patterns as in the case of faces. That means that our generator doesn't need explicit spatial bias to learn these high distance correlations. With this observation in mind we randomize the sampled frequencies of our building block in each training iteration. This forces the network to translate local patterns in the Fourier feature map into patterns that exist in the target material, as shown in Figure 6.4.

Once trained our generator can be queried with an arbitrarily high resolution building block and it will generate an accordingly large plane with no seams or repetition.

6.2.5 Fourier Frequencies Injection

In the original StyleGAN [50], noise injections were introduced to help with the stochastic elements when generating faces, such as hair, beards etc. The argument was that the network wouldn't need to learn these effects and instead could utilize inputs of noise to generate them. This was removed with StyleGAN 3 since they targeted a natural transformation hierarchy, i.e., each pixel generated being dependent only on the previous coarser features. While this change was reported as FID-neutral for faces where these stochastic effects are only part of the whole image, this is not the case for our mesoscale materials. For materials such as fur the generator is struggling to create random hair strands even after long training. To help with this we reintroduce this stochasticity in the form of random Fourier Features. At each layer the Fourier Feature map contains frequencies between the cutoff frequency of the previous layer and the current cutoff frequency.

6.2.6 Generative 3D Neural Textures

We take advantage of the generative capabilities of StyleGAN 3 and combine them with the explicit camera control of volumetric rendering, adapted to our scenario of 3D texture synthesis. EG3D proposed a triplane representation to lift the StyleGAN output into 3D. The StyleGAN generator was used to create three orthogonal feature planes which were queried during ray marching and the result decoded by a small decoder network.

In our infinite plane context the triplane approach of [14] is problematic. The three orthogonal planes are generated from the same building block and even though they are treated as separate planes they share a lot of information. Our 3D neural textures are

infinite only in the tangent directions to the surface. The vertical axis has a finite range and so the planes cannot be generated in the same way. The naive approach of splitting the StyleGAN generator into two components, one for the infinite plane and one for the two perpendicular, introduces memory and performance issues during training. Instead we use our infinite plane approach to generate only one tangent feature plane. For a queried position $\mathbf{p} = (x, y, z)$ we bilinearly interpolate features from the tangent plane using (x, y) and encode z using Fourier frequencies $\gamma(z)$. The plane features and encoded height are concatenated and passed through a small MLP decoder network which outputs radiance and density. The decoder uses the encoded height information to lift the tangent feature plane into 3D with high frequency details. We use volume rendering to compute the final image which is fed to the discriminator.

6.2.7 Mipmap Based Multi Scale Training

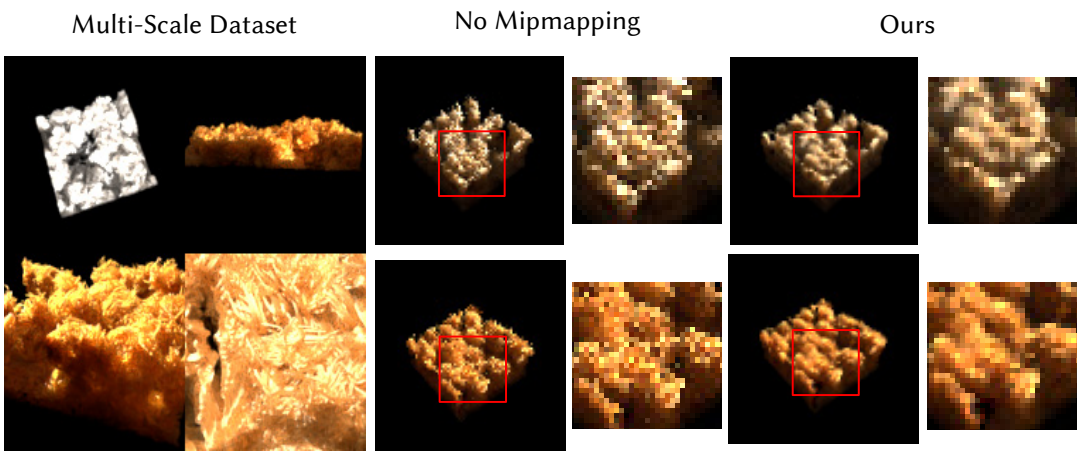


Figure 6.5: When we train on a multi-scale dataset (show on the left) the network is not aware of the scale when queried. This results in aliasing as shown on the right. Our mipmapping approach on the feature planes removes the aliasing issue.

As we mentioned in the restrictions of our target applications, during inference we do not have access to a neighborhood which is necessary for operations such as convolutions or upsamplings with interpolation. Our outputs also need to be view consistent without flickering artifacts. The maximum training resolution of the raymarched images, before GPU memory becomes an issue, is 128x128. EG3D overcomes this restriction by using a convolutional super resolution module to generate a high resolution image. While this step provides high resolution details it does so in a view inconsistent way.

In order to keep the same training resolution but learn the appearance of our materials in more detail we use a multi scale dataset by allowing the camera to move closer to the patch (see Figure 6.5). Training on this multiscale dataset with the ray marching formulation of NeRF leads to aliasing when the material is rendered from far away and blurriness when it is rendered close up.

Since we shoot only one ray through the center of each pixel we are not taking into account the physical size of the pixel. Normally this would introduce aliasing but since the network is trained on filtered images it incorporates this filtering in its outputs. This works well when the dataset is single scale, meaning the ground truth images are from a constant distance. As Barron et al. [5] showed when NeRF is trained on a multi scale dataset the network cannot adapt its filtering based on the camera distance. This results in aliasing and over blurring. We follow Barron et al.'s methodology to solve this issue by considering cones instead of rays, the footprint of which we approximate using 3D gaussians. These Gaussians, with mean γ and covariance matrix Σ, are used to adapt the Fourier frequencies used when encoding the input:

$$\gamma(\mu, \Sigma) = \begin{bmatrix} \sin(\mu) exp(-\frac{1}{2} diag(\Sigma)) \\ \cos(\mu) exp(-\frac{1}{2} diag(\Sigma)) \end{bmatrix} \tag{6.2}$$

This encoding scales down the magnitude of the frequencies based on the footprint of the cone segment. When the cone footprint is big the high frequencies are scaled down while when it is small the high frequencies remain unchanged.

We use this integrated encoding for the queried height z to filter frequencies based on the distance of the camera. For the feature plane we apply a mipmapping approach. We create a stack of mipmaps by applying different low pass filters. We then use the footprint of the projected 3D Gaussian to trilinearly interpolate between the two closest mipmap scales. The 3D Gaussian projected onto the plane is an ellipse $(x, y) = (a \cos t, b \sin t)$. We approximate with a circle of radius r where $r = \min(a, b)$. The size of the circle is used to choose the mipmap with the corresponding pixel size.

This enables the feature plane to include high frequency detail, used when the camera is placed close to the object, and it suppresses aliasing for faraway shots (Figure 6.5). Without filtering the introduces high frequency into the feature plane; thus it is penalized due to aliasing in the image. With mipmapping the generator can add details for the up close shots which get filtered when the object is viewed from far away. In that way we are no longer restricted by the resolution of the ray marched image but by the resolution

of the StyleGAN generated plane which can be much higher.

6.2.8 Integration into a path tracer

The goal of our approach is rendering the material primitives within a path tracer. In this section, we first describe the interaction with path tracing and then how we condition the model on explicit parameters to enable artistic control. Afterwards, we explain how to instance the learned infinite generative neural field onto arbitrary 3D objects.

Path tracing interaction. To facilitate integration into a path tracer, we choose radiance as the radiometric quantity and relate the outgoing to incident radiance via a small amount of numerical integration (ray marching) and neural predictions.

The radiance $L_\mathrm{o}(x, \omega)$ outgoing from a point on the boundary is defined as the amount of incident flux that hits the boundary, propagates through the shell, and exits at (x, ω). L_o is a sum of two terms: scattered and uncollided radiance, where the scattered radiance reads:

$$L_\mathrm{s} = \int_0^b T(x, y)\sigma_t(y)\alpha(y) \int_A \int_{H^2} f(y, \omega, z, \omega') L_\mathrm{i}(z, \omega') d\omega' dz dt, \tag{6.3}$$

where b is the distance (through the shell) to the intersection of ray $(x, -\omega)$ with the boundary, $T(x, y)$ is transmittance, $\sigma(y)$ and $\alpha(y)$ are extinction coefficient and albedo at point $y = x - t\omega$, A is the boundary area, H^2 is the upper hemisphere at boundary point z, $f(z, \omega', y, \omega)$ is a transport function quantifying all light traveling from z to y in directions $-\omega'$ and ω, respectively, and $L_\mathrm{i}(z, \omega')$ is incident radiance at the boundary.

We introduce a simplifying assumption to facilitate pretraining the neural model without accounting for macroscale geometric variations. We assume that the shell volume, which is relevant for evaluating the triple integral, is small in comparison to macroscale variations (same assumption as in NeRF-Tex). This allows us to locally treat point sources of illumination as distant, i.e., producing parallel, spatially invariant incident radiance $L_\mathrm{i}(\omega')$ and assume the shell forms a slab receiving light through the top and bottom sides only.

Incorporating the assumption and reorganizing yields:

$$L_\mathrm{s} = \int_0^b T(x, y)\sigma_t(y)\alpha(y) \int_{H^2} L_\mathrm{i}(\omega') \int_A f(y, \omega, z, \omega') dz d\omega' dt, \tag{6.4}$$

We train our neural model to approximate the albedo-weighted transport function integrated over the boundary: $\alpha(y)f(y,\omega,z,\omega')dz$, denoted ρ, and extinction coefficient $\sigma(y)$. The remaining two integrals are evaluated numerically: $L_{\mathrm{i}}(\omega')$ is estimated by tracing a single path from x, the remaining terms are computed via ray marching.

The uncollided term equals to radiance incident at other side of the boundary attenuated by transmittance through the shell. In practice, we randomly estimate either the uncollided or scattered term proportional to transmittance.

Conditioning. As artistic control is crucial in our generative neural primitives, we include additional input parameters to enable modifying their appearance using intuitive parameters. Since we use synthetic data we have access to these parameters from our dataset in the form of labels. The parametric control, which is learned from the dataset, is split into 2 categories: geometry and appearance.

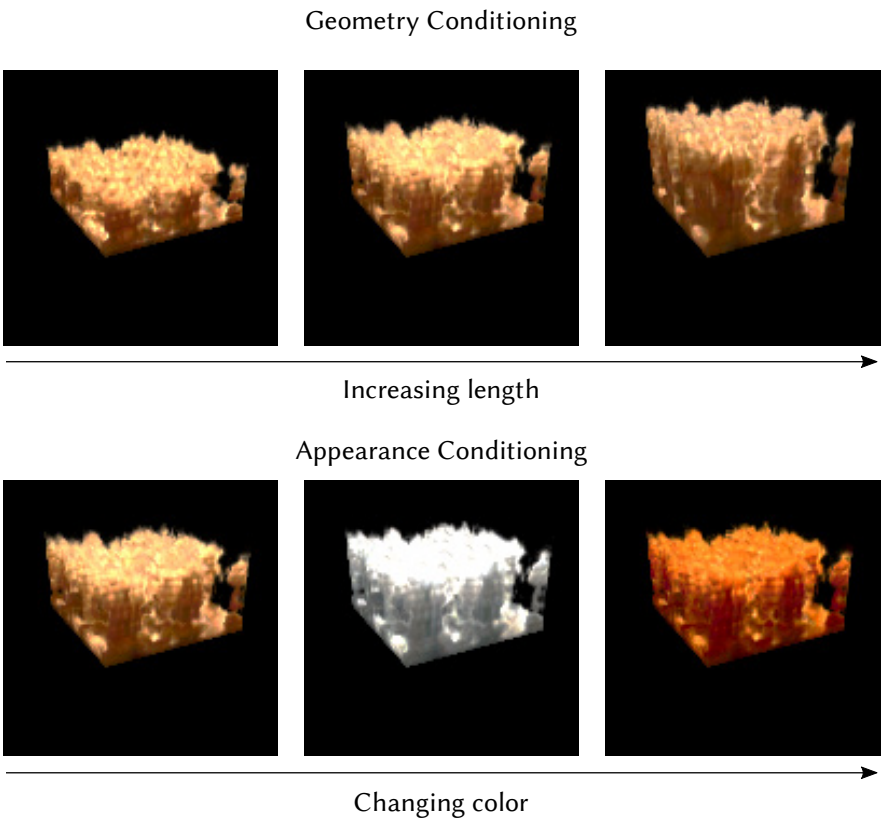


Figure 6.6: Our neural 3D textures can be controlled by intuitive parameters controlling the geometry and appearance of the material. Here we show an instance of fur with varying length and color.

StyleGAN's generator is conditioned on different styles $\mathbf{w}$ which get generated by a mapping network. The mapping network is an MLP which maps a noise vector $\mathbf{z}$ to the style latent space $\mathcal{W}$. The geometric conditioning parameters, such as fur length, should change the contents of the feature plane and as such they condition the StyleGAN generator. The geometric parameters are passed to the mapping network and affect the latent styles $\mathbf{w}$ which are responsible for the final structure of the tangent place. Appearance parameters should affect only the final color of the decoded features. For that reason they are concatenated along with the features and encoded height before being fed to the shallow MLP decoder. Incoming light direction is treated as appearance conditioning which is important to enable relighting. In this way our primitives can appear natural in different locations in a scene and various lighting conditions. All the conditioning labels are also fed to the discriminator which helps with disentanglement. Similar to previous work we also feed the camera position as an extra label to the discriminator which helps accelerate training.

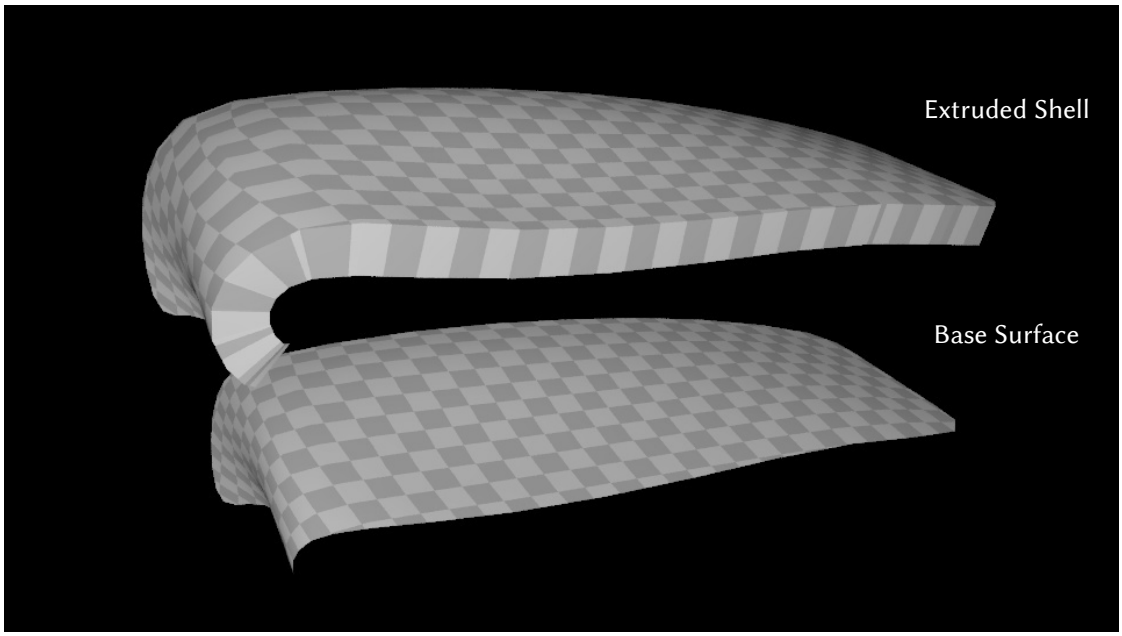


Figure 6.7: Given a base surface mesh and a target length we extrude the surface to create a shell. We visualize the texture coordinates to show how they are affected by the extrusion.

Shell mapping. Our infinite neural texture primitive can be applied onto arbitrary 3D meshes through a shell mapping procedure. First, we generate a feature map of a

desired resolution that is attached to the object. Then, we assign a *shell map* [86] to the mesh with a specific height, visualized in Figure 6.7. To construct the shell map, we extrude the triangle faces of the mesh along the normal direction and tell the renderer to perform intersections against these during rendering. When an intersection occurs, we look up the corresponding UV-coordinates and height of the intersection point and then ray marching is initiated. At each ray marching step within the shell, we compute the 2D UV coordinates and height to fetch generated features from the StyleGAN3-produced tangent feature planes (Figure 6.8). These are then passed to the MLP to output a density and a radiance value similar to NeRF that can be used for volumetric integration.

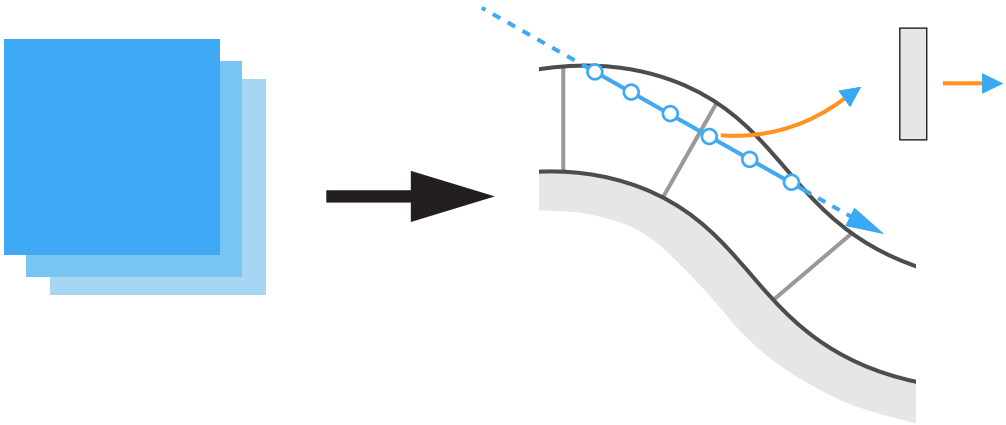


Figure 6.8: The feature map outputs from the StyleGAN3 generator are applied to a mesh through a shell map. The features from the maps are looked up based on the height of the query point in the shell map and the UV-coordinates.

6.3 Results

We show preliminary results of our shell mapping approach on a Shell model. In order to demonstarte the benefits of our generative model we render the same model with a single high resolution 3D neural texture with no seams and with repeating the same low resolution 3D neural texture. As is shown in Figure 6.9 our method can create a single 3D neural texture that spans the uv space of the model. As a result the resulting fur has stochastic effects such as clumps and gaps with no seams. In comparison repeating the same exemplar results in repetitive artifacts and lower quality.

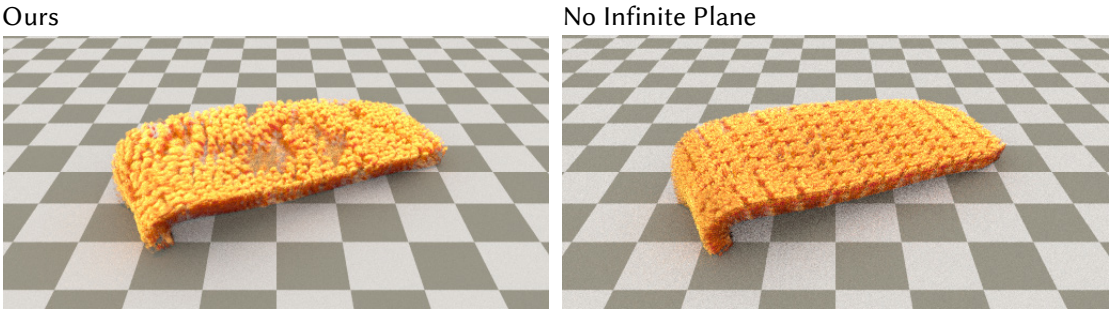


Figure 6.9: Comparison of our infinite texture approach to the naive solution of repeating an examplar over the surface of a model.

6.4 Comparisons

In this section we compare our approach to two alternative 3D generative methods, π-GAN [15] and Style-NeRF [30], and show how in our context they fail to satisfy the requirements we defined in Section 6.2.2. We train all methods on the same Carpet dataset to convergence and show the results in Figure 6.10. From the results it is clear that mesoscale material are harder to represent compared to faces due to the lack of landmarks or structured nature. As a result π-GAN performs poorly with wavy artifacts and blurry structure. Style-NeRF can reach high quality due to the image space operations which don't require an accurate representation of the material in the 3D volumetric radiance field. As stated before, these image space operations cannot be used in a path tracing context and they also come at the cost of view consistency. As is highlighted by the red circle in Figure 6.10 which tracks the same point between the images, in the case of Style-NeRF as the camera moves the structure of the material changes. Eventually the point highlighted disappears. Our method in comparison has better quality than π-GAN and the view consistency lacking from Style-NeRF.

We also compare to NeRF-Tex [1] to demonstrate the advantages of a generative method. For this comparison we train our method and NeRF-Tex with an increasing number of parameters for the Carpet dataset. The results shown in Figure 6.11 demonstrate the advantages of our generative model. As the number of parameters increases NeRF-Tex fails to learn all the different appearances that can appear and instead generates a very blurry result. In contrast our method can handle all these parameters without losing out on quality.

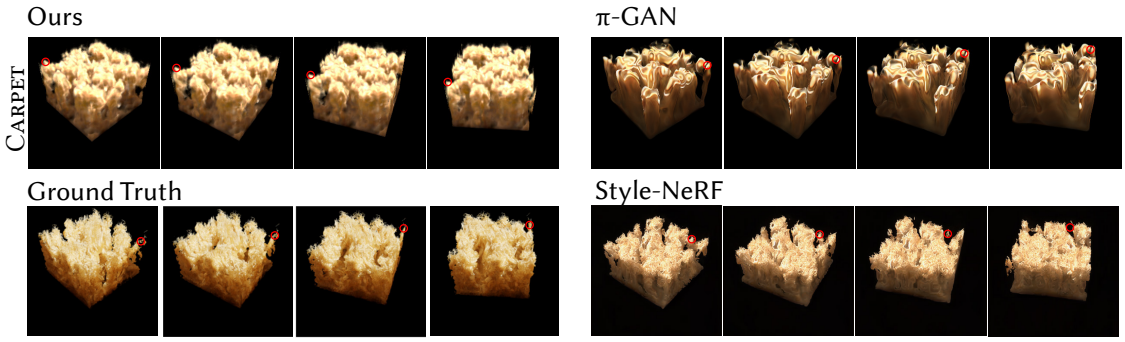


Figure 6.10: Comparison of our method against π-GAN [15] and Style-NeRF [30]. The red circle tracks the same 3D point between views to help with evaluating view consistency.

Figure 6.11: Comparison of our method against NeRF-Tex [1]

6.5 Conclusions

We have presented a method for generative neural 3D texture primitives which can simulate the appearance of mesoscale materials and be incorporated into a traditional rendering pipeline. Our infinite plane generator solves the issue of seams when texturing a model while the generative aspect of our method introduces realistic stochastic detail into the results. This takes the quality of neural radiance field primitives one step closer to production level. The incorporation of our model into a path tracer requires per pixel operations and view consistency which we achieve through our mipmapping multi scale training scheme. Our comparisons to NeRF-Tex show the advantages of learning neural radiance fields in a generative way without sacrificing artistic control. We believe that *generative*, neural 3D textures are a versatile tool for rendering the complex geometries of mesoscale materials while being efficient and easily editable.

Chapter 7

Conclusions

This thesis reflects the evolution of research in computer graphics in the past few years. While the main research goal in the context of physically based rendering has remained the same, i.e., render realistic images as efficiently as possible, the tools and methodology applied have changed drastically. In this thesis we explored, discussed and expanded upon some of the breakthroughs that took place in its span. Neural networks, real-time path tracing and generative models have created new and exciting opportunities for computer graphics and we hope our work inspires future methods in these directions.

In the first part of the thesis we worked with explicit representations of radiance fields to improve sampling. We set out to augment these representations with material information in an efficient way without increasing the already high dimensionality of the problem. We achieve this through on-the-fly integration of complex materials over spherical polygons while using the practical spatial directional tree to represent the incoming radiance field.

After working with explicit radiance field representations and seeing first hand their limitations, in the second part of the thesis, we switch to using neural networks to learn the radiance field. Compared to the explicit structures we do not build an approximation on-the-fly but precompute the radiance field. We trained them to represent outgoing radiance fields for variable scenes, and highlighted the issue of lengthy training in previous methods even for simple scenes.

To answer our research question of efficiently training these networks with artistic control we presented Active Exploration with an explicit scene representation. Active Exploration guides data generation towards scene configurations with complex radiance fields while our explicit scene representation approach allows us to control all the variable aspects of the scene. We believe Active Exploration is just the first step towards more efficient data generation and training which is crucial so that neural rendering research becomes more accessible and has a smaller impact on the environment.

Moving to generative models, we demonstrated that the representation power of

neural networks goes beyond representing a single radiance field. These models allow us to learn a *distribution* of radiance fields that represent a class of objects. Doing so with explicit camera control requires the use of volumetric radiance fields to implicitly learn geometry and appearance from generated images. We have explored how this can have multiple benefits in the case of hard to render mesoscale materials. Once trained our models can generate and render 3D neural textures while retaining artistic control of parameters such as color, fur length, curliness etc. We incorporate our models in a traditional path tracer combining the strengths of each component.

7.1 Lessons learned

In the three projects presented, there is a transition from traditional path tracing to neural rendering. The first project gave us a lot of insight about the process of creating a realistic image with path tracing and the underlying light transport problem. This experience was necessary to successfully tackle the challenges of infusing the rendering pipeline with neural networks. This is reflected in the inspiration we drew from Markov Chain Monte Carlo methods for our Active Exploration and in our design choices for Meso-GAN with the goal of integrating it into path tracing. We believe that a deep knowledge of computer graphics literature is crucial for the future of neural rendering as it can be used to inject strong priors into networks. For example in our Meso-GAN pipeline we use a volume rendering step to create explicit camera control in the neural rendering. Without this volume rendering step regaining control of the camera position after training is much harder and would result in view inconsistencies. Using such explicit steps to set constraints in the training procedure can unlock the true potential of neural networks and for that strong computer graphics background is key.

The unifying aspect of using radiance fields in all three projects is a good opportunity to observe the evolution of our representations. In the first project the SD-tree had enough representation power to be used for importance sampling but it was also very restrictive due to its memory footprint. The radiance fields used in the second project showed the true potential of neural representations. A single MLP was capable of representing all the possible radiance fields for a variable scene and with accuracy much higher than the SD-tree for a static scene. The neural volumetric radiance fields used in Meso-GAN are even more expressive as they include the complex geometry and materials of the scene. We will continue to explore the representation capabilities of neural networks as we are

persuaded we are not yet close at reaching their limitations.

A final aspect is our focus on to artistic control. When we transitioned to using neural networks for rendering we observed that in many cases previous work treated it as optional. As computer graphics is a tool used to create content and art, we must attempt to restrict and condition our models to control the appearance and feel of the final outcome. On the other hand implicit or latent representations have an important role as descriptions of ambiguous and high level concepts. The efficient combination of implicit and explicit representations is key to creating practical methods with the creators in mind.

7.2 Future directions

We already presented some future directions in the conclusion of each method but here we will discuss some high level future directions and goals.

A living area with a television and a table | A small kitchen with a low ceiling | A teddy bear on a skateboard in times square

Figure 7.1: Results from Dalle 2 [90]; generated images in the top row and the given text prompt at the bottom. Each of the examples have hard to render effects complete with specular and glossy reflections, global illumination and bokeh effects.

Versatile embeddings for rendering in computer graphics. A recent development that could have a huge impact on the future of computer graphics is rapid improvement of text to image neural generators. This improvement comes after the development of high quality embeddings between text and images [87]. CLIP, standing for Contrastive Language-Image Pre-training, is a model that is trained on 400 million pairs of images

and descriptive text with the goal of generating a description from a single image. Follow-up work demonstrated that the embeddings generated by CLIP are very powerful and versatile and they can be used to perform the inverse operation of generating images from text [90] with unprecedented quality, such as the examples in Figure 7.1. This is ground breaking and it opens the door for creating content through text prompts which is very intuitive. From the point of view of computer graphics the main questions that we would need to answer is how to inject explicit and in particular 3D control in this pipeline. This could be achieved by combining volumetric radiance fields with these embeddings, as already attempted [39] or by disentangling pose from the latent space of CLIP. Training for such a task is a big bottleneck with many methods using the pretrained CLIP model, but in order to adapt CLIP to our contexts, retraining might be unavoidable. In such cases optimizing the training procedure with an extension of Active Learning could bring sufficient gains in computation times and also reduce the environmental impact of such methods.

Scene-scale generative models. Until now generative models have been restricted to specific types of objects such as faces, cars or in our case mesoscale materials. Recent research has shown that GANs can be trained to represent broader and more varied collections of objects in 2D [97]. Expanding this to 3D by using volumetric radiance fields could be possible to create a generative model that can render realistically novel 3D scenes. In the context of computer graphics this could be used to handle complex lighting effects without the need to retrain for each scene as is done in our method in Chapter 5. Since the generative model could represent the radiance field of any (within a limit) scene conditioning it to an artist created scene would give us the outgoing radiance distribution without any extra training. The conditioning could be achieved either through an encoder or by using pretrained embeddings such as the ones from CLIP. Should such a latent embedding be used it could be necessary to introduce an intermediate step where the explicit artistic control is injected to ensure that the level of control is not reduced.

Differentiable rendering. A lot of work has gone recently into computing the gradients of path traced images relative to changes in the scene parameters. Such gradients can be used in many different scenarios, for example to optimize geometry to match a captured scene [83], making the geometry modeling process much more intuitive. This process, known as differentiable path tracing, is quite intricate since path tracing involves

computing integrals with many discontinuities. We believe that in the future neural renderers that are differentiable by construction could provide an avenue for computing global illumination gradients over scene parameters. This is possible, albeit in a very simple scenario, in the method presented in Chapter 5 where we can compute the gradients of the global illumination relative to the rotation of the door in the Veach Door scene. In the limit we could imagine our explicit representation vector including the positions of all geometry vertices in a scene. If we then were able to train this configuration for many different shapes, we could optimize a starting shape based on target multiview images. The main challenge in such a direction would be memory constraints and proposing efficient data generation algorithms.

Concluding, this is a fascinating period for computer graphics research with an auspicious future to which we are eager to continue contributing.

Appendix A

Chapter 5 appendix

A.1 Selected Views

We show the chosen views that we use in our quantitative evaluation in Figure A.1.

Figure A.1: The chosen views, which correspond to scene configurations with complex illumination effects, that were used for the ablations.

A.2 Comparison to CNSR

We show additional results for same quality and same time comparisons against CNSR [28] in Figure A.2.

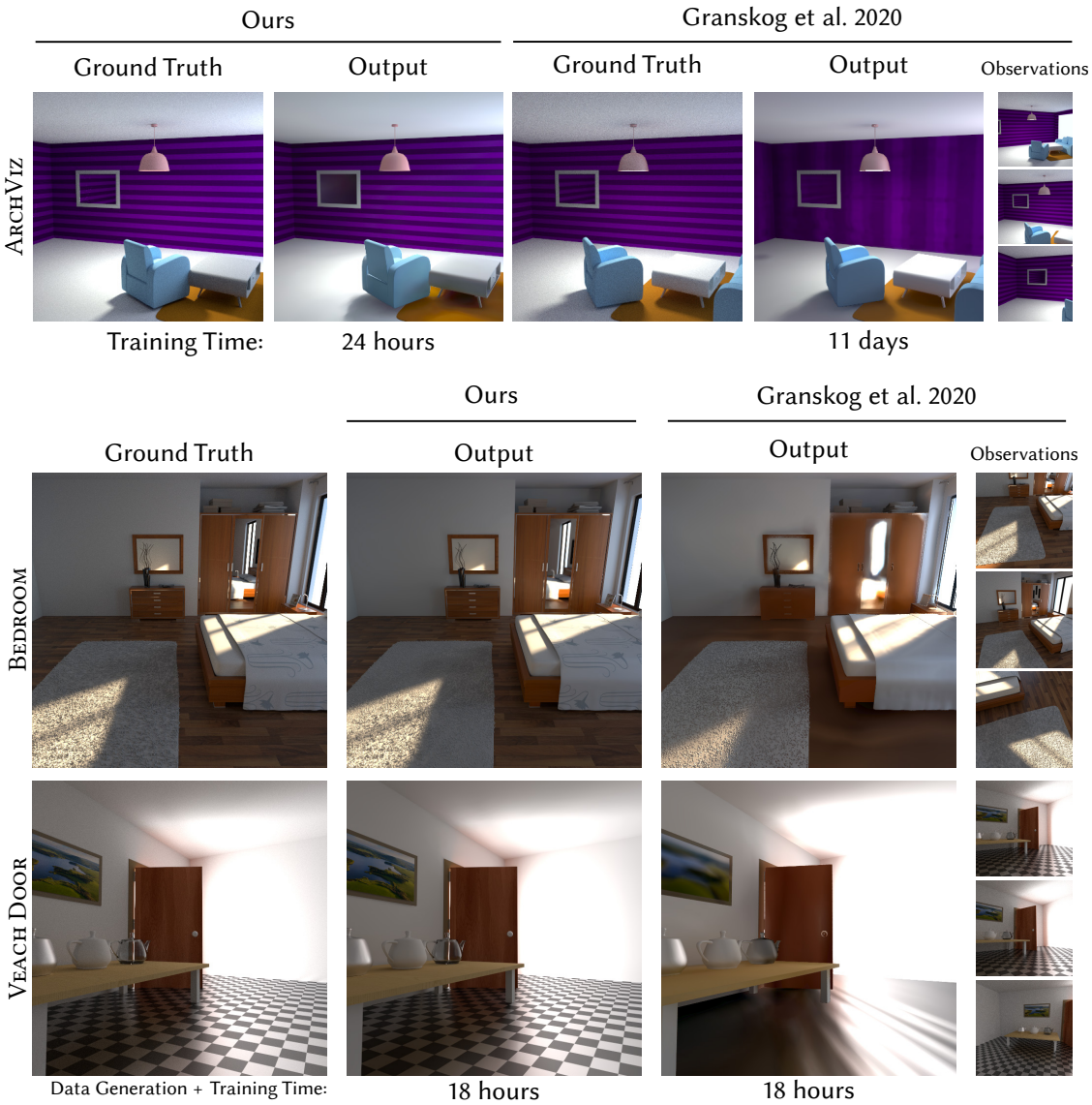


Figure A.2: Additional same quality and same time results with CNSR [28].

A.3 Comparison to ANF

In Figure A.3 we display a sample of ground truth images used during the finetuning of the ANF [38] pretrained model on our scenes. Please observe how the complex caustic effects that the models fail to reproduce, even after fine tuning, exist in the ground truths. The amount of noise in the ground is equivalent to that in ours but our model is able to both learn these effects and average out the noise in world space during training.

Figure A.3: Sample ground truth images used for fine tuning ANF on our scenes.

A.4 Comparison to GT

In Figure A.4 we provide difference images for the comparison of our method to ground truth, using the MAPE metric, to help with visual inspection.

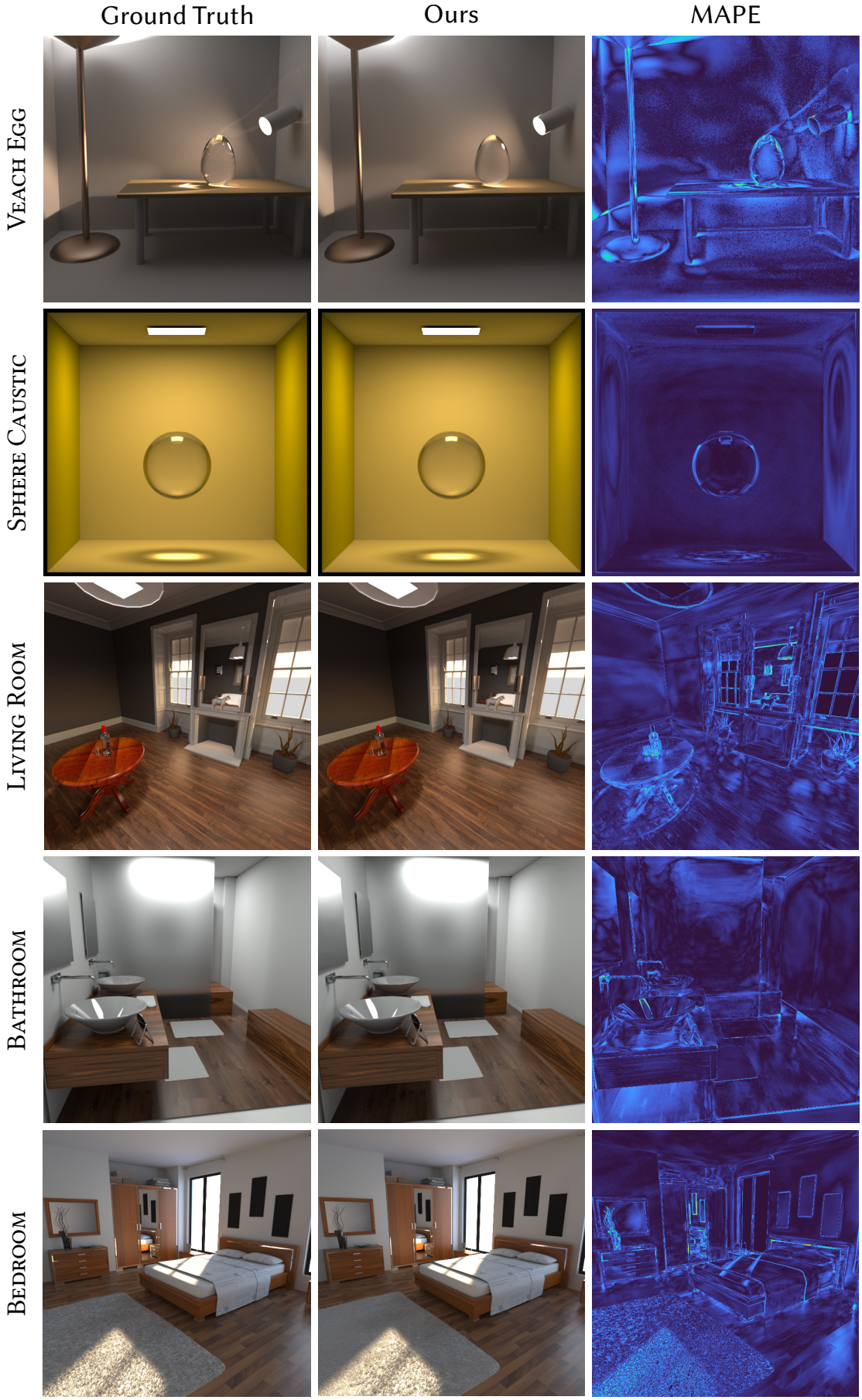


Figure A.4: Comparison of our method to ground truth with additional difference images, using the MAPE metric, for visual inspection.

A.5 Network Architecture

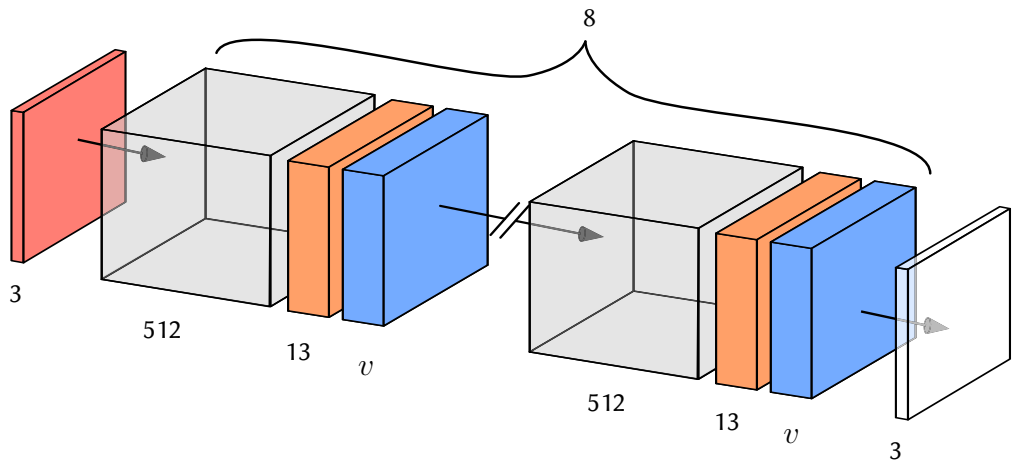


Figure A.5: The architecture of our generator. The positional buffer is shown in red, all the G-Buffers in orange, the explicit scene representation vector v in blue and the output in white.

The architecture of our generator is the Pixel Generator proposed by Granskog et al. [28] with a preconditioning on position (Fig. A.5). The Pixel Generator is an MLP (we use leaky ReLU activations) with skip connections on every layer. We map the position buffer (red) from 3 to 512 channels and then we concatenate all the G-buffers (orange) and explicit vector v (blue) at each layer. The total hidden layers are 8 with 512 hidden features. In Figure A.6 we show that using a smaller network can provide acceptable results and lower inference speed, resulting in 13 FPS in our prototype Python implementation.

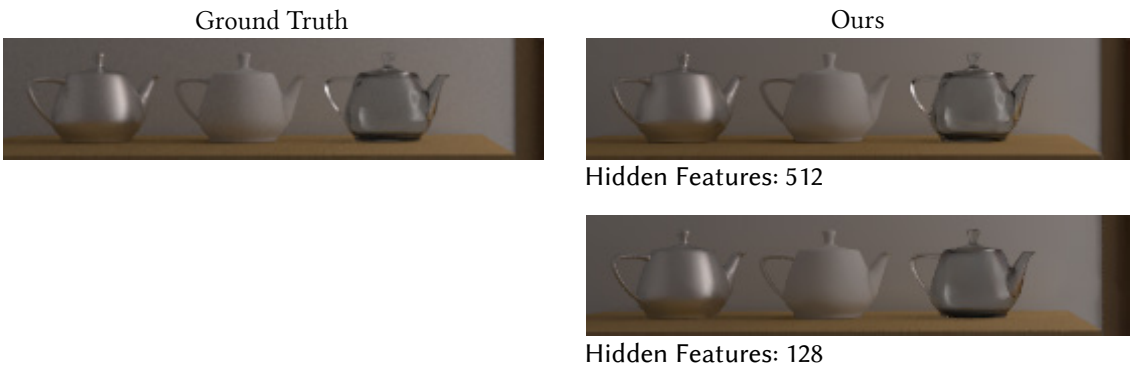


Figure A.6: Using 128 hidden features results in acceptable results and higher frame rates, but lower quality compared to using 512.

A.6 MCMC States Lifespan

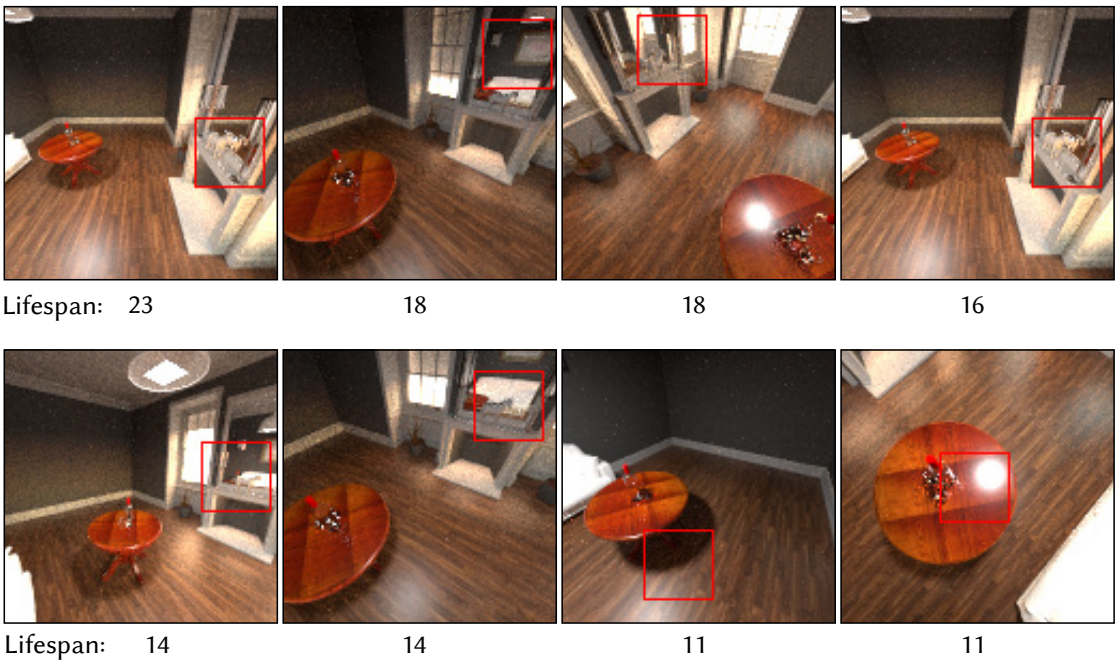


Figure A.7: The 8 longest lifespan MCMC states when training on the Living Room scene.

In order to evaluate what type of effects our Active Exploration focuses on, we visualize the MCMC states with the longest lifespan (consecutive times being the current state) for the Living Room scene in Figure A.7. We observe that our Active Exploration spends more time on effects that require more training to be represented such as reflections, glossy highlights and shadows. During training only the red patch would be rendered and used for training, here we render the whole image for visualization purposes.

A.7 Sample Reuse Derivation

Given the two options to either reuse or generate a new sample with respective likelihood l_{exist} and l_{new}, a simple Bernoulli distribution that respect the likelihood ratio has a probability p of reusing defined by:

$$p = \frac{l_{new}}{l_{exist} + l_{new}}$$

This Bernoulli distribution can further be skewed as to favor the reuse case by dividing the likelihood of the reuse case l_{exist} by α:

$$p = \frac{l_{new}}{\frac{l_{exist}}{\alpha} + l_{new}}$$

For instance setting alpha to 99 skews the probability distribution so that for equal likelihood $p = \frac{99}{100}$

We then assume that the losses $Loss_{\text{new}}$ and $Loss_{\text{exist}}$ represent the negative log-likelihood of the network output with respect to a probability distribution parameterized by the ground truth, which for the L2 loss case would be a Normal distribution centered around the ground truth value and for the L1 loss is a Laplace distribution also centered around the ground truth value. We thus have:

$$\sigma(Loss_{\text{exist}} - Loss_{\text{new}} + \beta) = \frac{e^{Loss_{\text{exist}} - Loss_{\text{new}} + \beta}}{1 + e^{Loss_{\text{exist}} - Loss_{\text{new}} + \beta}}$$

$$\sigma(Loss_{\text{exist}} - Loss_{\text{new}} + \beta) = \frac{e^{-Loss_{\text{new}}}}{e^{-Loss_{\text{exist}}} e^{-\beta} + e^{-Loss_{\text{new}}}}$$

$$\sigma(Loss_{\text{exist}} - Loss_{\text{new}} + \beta) = \frac{l_{new}}{\frac{l_{exist}}{e^{\beta}} + l_{new}}$$

Which inspired our reuse strategy.

Bibliography

[1] Hendrik Baatz, Jonathan Granskog, Marios Papas, Fabrice Rousselle, and Jan Novák. Nerf-tex: Neural reflectance field textures. In *Eurographics Symposium on Rendering*. The Eurographics Association, June 2021. 20, 74, 75, 78, 87, 88

[2] Hendrik Baatz, Jonathan Granskog, Marios Papas, Fabrice Rousselle, and Jan Novák. Nerf-tex: Neural reflectance field textures. In *Eurographics Symposium on Rendering (DL-only track)*. The Eurographics Association, June 2021. 71

[3] 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

[4] Steve Bako, Mark Meyer, Tony DeRose, and Pradeep Sen. Offline deep importance sampling for monte carlo path tracing. In *Computer Graphics Forum*, volume 38, pages 527–542. Wiley Online Library, 2019. 21

[5] Jonathan T. Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P. Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. *ICCV*, 2021. 19, 82

[6] Thomas Bashford-Rogers, Kurt Debattista, and Alan Chalmers. A significance cache for accelerating global illumination. *Comput. Graph. Forum*, 31(6):1837–1851, 2012. doi: 10.1111/j.1467-8659.2012.02099.x. 17

[7] Daniel R Baum, Holly E Rushmeier, and James M Winget. Improving radiosity solutions through the use of analytically determined form-factors. *ACM Siggraph Computer Graphics*, 23(3):325–334, 1989. doi: 10.1145/74334.74367. 27

[8] Benedikt Bitterli. Rendering resources, 2016. https://benedikt-bitterli.me/resources/. 57

[9] Benedikt Bitterli and Wojciech Jarosz. Selectively Metropolised Monte Carlo light transport simulation. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 38(6), November 2019. doi: 10.1145/3355089.3356578. 39

[10] Benedikt Bitterli, Wenzel Jakob, Jan Novák, and Wojciech Jarosz. Reversible jump metropolis light transport using inverse mappings. *ACM Transactions on Graphics (TOG)*, 37(1):1–12, 2017. 16

[11] Guillaume Bouchard, Théo Trouillon, Julien Perez, and Adrien Gaidon. Online learning to sample. *arXiv preprint arXiv:1506.09016*, 2015. 46

[12] 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

[13] Eric Chan, Marco Monteiro, Petr Kellnhofer, Jiajun Wu, and Gordon Wetzstein. pi-GAN: Periodic implicit generative adversarial networks for 3d-aware image synthesis. In *arXiv*, 2020. 74

[14] Eric R. Chan, Connor Z. Lin, Matthew A. Chan, Koki Nagano, Boxiao Pan, Shalini De Mello, Orazio Gallo, Leonidas J. Guibas, Jonathan Tremblay, Sameh Khamis, Tero Karras, and Gordon Wetzstein. Efficient geometry-aware 3d generative adversarial networks. volume abs/2112.07945, 2021. URL `https://arxiv.org/abs/2112.07945`. 21, 22, 74, 80

[15] Eric R Chan, Marco Monteiro, Petr Kellnhofer, Jiajun Wu, and Gordon Wetzstein. pi-gan: Periodic implicit generative adversarial networks for 3d-aware image synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5799–5809, 2021. 4, 21, 87, 88

[16] Subrahmanyan Chandrasekhar. *Radiative transfer*. Courier Corporation, 2013. 14

[17] Petrik Clarberg and Tomas Akenine-Möllery. Practical product importance sampling for direct illumination. In *Computer Graphics Forum*, volume 27, pages 681–690. Wiley Online Library, 2008. doi: 10.1111/j.1467-8659.2008.01166.x. 17

[18] Petrik Clarberg, Wojciech Jarosz, Tomas Akenine-Möller, and Henrik Wann Jensen. Wavelet importance sampling: efficiently evaluating products of complex functions. In *ACM SIGGRAPH 2005 Papers*, pages 1166–1175. 2005. doi: 10.1145/1073204.1073328. 17, 24

[19] Alejandro Conty Estevez and Pascal Lecocq. Fast product importance sampling of environment maps. In *ACM SIGGRAPH 2018 Talks*, pages 1–2. 2018. doi: 10.1145/3214745.3214760. 26

[20] Ken Dahm and Alexander Keller. Learning light transport the reinforced way, 2017. 17

[21] Kristin J. Dana, Bram van Ginneken, Shree K. Nayar, and Jan J. Koenderink. Reflectance and texture of real-world surfaces. *ACM Trans. Graph.*, 18(1):1–34, jan 1999. ISSN 0730-0301. doi: 10.1145/300776.300778. URL `https://doi.org/10.1145/300776.300778`. 76

[22] Philippe Decaudin and Fabrice Neyret. Volumetric billboards. *Computer Graphics Forum*, 28(8):2079–2089, 2009. doi: 10.1111/j.1467-8659.2009.01354.x. 75

[23] Stavros Diolatzis, Adrien Gruson, Wenzel Jakob, Derek Nowrouzezahrai, and George Drettakis. Practical product path guiding using linearly transformed cosines. In *Computer Graphics Forum*, volume 39, pages 23–33. Wiley Online Library, 2020. 9

[24] Stavros Diolatzis, Julien Philip, and George Drettakis. Active exploration for neural global illumination of variable scenes. *ACM Transactions on Graphics*, 2022. URL `http://www-sop.inria.fr/reves/Basilic/2022/DPD22`. 9

[25] SM Ali Eslami, Danilo Jimenez Rezende, Frederic Besse, Fabio Viola, Ari S Morcos, Marta Garnelo, Avraham Ruderman, Andrei A Rusu, Ivo Danihelka, Karol Gregor, et al. Neural scene representation and rendering. *Science*, 360(6394):1204–1210, 2018. 2, 19, 20, 44, 48

[26] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger,

editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf. 3

[27] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014. 21

[28] Jonathan Granskog, Fabrice Rousselle, Marios Papas, and Jan Novák. Compositional neural scene representations for shading inference. *ACM Transactions on Graphics (TOG)*, 39(4):135–1, 2020. 20, 44, 48, 49, 54, 55, 60, 62, 63, 64, 96, 99

[29] Pascal Grittmann, Iliyan Georgiev, Philipp Slusallek, and Jaroslav Křivánek. Variance-aware multiple importance sampling. *ACM Transactions on Graphics (TOG)*, 38(6):1–9, 2019. doi: 10.1145/3355089.3356515. 40

[30] Jiatao Gu, Lingjie Liu, Peng Wang, and Christian Theobalt. Stylenerf: A style-based 3d-aware generator for high-resolution image synthesis. *CoRR*, abs/2110.08985, 2021. URL https://arxiv.org/abs/2110.08985. 21, 22, 87, 88

[31] Jerry Guo, Pablo Bauszat, Jacco Bikker, and Elmar Eisemann. Primary sample space path guiding. In *Eurographics Symposium on Rendering*, volume 2018, pages 73–82. The Eurographics Association, 2018. doi: 10.2312/sre.20181174. 17

[32] Saeed Hadadan, Shuhong Chen, and Matthias Zwicker. Neural radiosity. *arXiv preprint arXiv:2105.12319*, 2021. 20

[33] Elad Hazan, Tomer Koren, and Nati Srebro. Beating sgd: Learning svms in sublinear time. In *Advances in Neural Information Processing Systems*, pages 1233–1241, 2011. 46

[34] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. Deep blending for free-viewpoint image-based rendering. 37(6): 257:1–257:15, 2018. 2

[35] Eric Heitz, Jonathan Dupuy, Stephen Hill, and David Neubelt. Real-time polygonal-light shading with linearly transformed cosines. *ACM Transactions on Graphics (TOG)*, 35(4):41, 2016. doi: 10.1145/2897824.2925895. 24, 26, 30

[36] Sebastian Herholz, Oskar Elek, Jiří Vorba, Hendrik Lensch, and Jaroslav Křivánek. Product importance sampling for light transport path guiding. In *Computer Graphics Forum*, volume 35, pages 67–77. Wiley Online Library, 2016. doi: 10.1111/cgf.12950. 17, 23, 24, 30, 31, 33, 35, 36, 41

[37] Steven Hill and Eric Heitz. Advances in real-time rendering. real-time area lighting: a journey from research to production. In *ACM SIGGRAPH 2016 Courses*, SIGGRAPH 16, 2016. 29

[38] Mustafa Işık, Krishna Mullia, Matthew Fisher, Jonathan Eisenmann, and Michaël Gharbi. Interactive monte carlo denoising using affinity of neural features. *ACM Transactions on Graphics (TOG)*, 40(4):1–13, 2021. doi: 10.1145/3450626.3459793. URL `https://doi.org/10.1145/3450626.3459793`. 61, 64, 65, 66, 97

[39] Ajay Jain, Ben Mildenhall, Jonathan T Barron, Pieter Abbeel, and Ben Poole. Zero-shot text-guided object generation with dream fields. *arXiv preprint arXiv:2112.01455*, 2021. 92

[40] Wenzel Jakob. Mitsuba renderer, 2010. URL `http://www.mitsuba-renderer.org`. 33

[41] Wenzel Jakob. Enoki: structured vectorization and differentiation on modern processor architectures, 2019. URL `https://github.com/mitsuba-renderer/enoki`. 33, 34

[42] Wenzel Jakob and Steve Marschner. Manifold exploration: a markov chain monte carlo technique for rendering scenes with difficult specular transport. *ACM Transactions on Graphics (TOG)*, 31(4):58, 2012. doi: 10.1145/2185520.2185554. 15

[43] Wojciech Jarosz, Nathan A. Carr, and Henrik Wann Jensen. Importance sampling spherical harmonics. *Computer Graphics Forum (Proceedings of Eurographics)*, 28 (2), April 2009. doi: 10.1111/j.1467-8659.2009.01398.x. 17

[44] Henrik Wann Jensen. Importance driven path tracing using the photon map. In *Rendering Techniques 95*, pages 326–335. Springer, 1995. doi: 10.1007/978-3-7091-9430-0_31. 17

[45] J. T. Kajiya and T. L. Kay. Rendering fur with three dimensional textures. *SIGGRAPH Comput. Graph.*, 23(3):271–280, July 1989. ISSN 0097-8930. doi: 10.1145/74334.74361. 75

[46] James T Kajiya. The rendering equation. In *ACM SIGGRAPH computer graphics*, volume 20, pages 143–150. ACM, 1986. doi: 10.1145/15922.15902. 13, 15

[47] Nima Khademi Kalantari, Steve Bako, and Pradeep Sen. A machine learning approach for filtering monte carlo noise. *ACM Trans. Graph.*, 34(4):122–1, 2015. 2

[48] Ondřej Karlík, Martin Šik, Petr Vévoda, Tomáš Skřivan, and Jaroslav Křivánek. Mis compensation: optimizing sampling techniques in multiple importance sampling. *ACM Transactions on Graphics (TOG)*, 38(6):1–12, 2019. doi: 10.1145/3355089. 3356565. 40

[49] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017. 3

[50] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4401–4410, 2019. 3, 4, 21, 80

[51] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of StyleGAN. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. 4, 21

[52] Tero Karras, Miika Aittala, Samuli Laine, Erik Härkönen, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Alias-free generative adversarial networks. In *Proc. NeurIPS*, 2021. 4, 21, 74, 77

[53] Csaba Kelemen, László Szirmay-Kalos, György Antal, and Ferenc Csonka. A simple and robust mutation strategy for the metropolis light transport algorithm. In *Computer Graphics Forum*, volume 21, pages 531–540. Wiley Online Library, 2002. 15, 51

[54] Alexander Keller, Luca Fascione, Marcos Fajardo, Iliyan Georgiev, Per Christensen, Johannes Hanika, Christian Eisenacher, and Gregory Nichols. The path tracing revolution in the movie industry. In *ACM SIGGRAPH 2015 Courses*, pages 1–7. 2015. 1, 15

[55] Ronald Kemker, Marc McClure, Angelina Abitino, Tyler Hayes, and Christopher Kanan. Measuring catastrophic forgetting in neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018. 46

[56] Hyeongwoo Kim, Michael Zollhöfer, Ayush Tewari, Justus Thies, Christian Richardt, and Christian Theobalt. Inversefacenet: Deep monocular inverse face rendering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4625–4634, 2018. 46

[57] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference for Learning Representations*, 2015. 49, 52

[58] Jaroslav Krivánek, Pascal Gautron, Sumanta Pattanaik, and Kadi Bouatouch. Radiance caching for efficient global illumination computation. *IEEE Transactions on Visualization and Computer Graphics*, 11(5):550–561, 2005. 17, 18

[59] Alexandr Kuznetsov, Krishna Mullia, Zexiang Xu, Miloš Hašan, and Ravi Ramamoorthi. NeuMIP: Multi-resolution neural materials. *ACM Transactions on Graphics (Proc. SIGGRAPH 2021)*, 40(4), 2021. 75, 76

[60] Jaakko Lehtinen, Jacob Munkberg, Jon Hasselgren, Samuli Laine, Tero Karras, Miika Aittala, and Timo Aila. Noise2noise: Learning image restoration without clean data. *arXiv preprint arXiv:1803.04189*, 2018. 79

[61] Thomas Leimkühler and George Drettakis. Freestylegan: Free-view editable portrait rendering with the camera manifold. *arXiv preprint arXiv:2109.09378*, 2021. 4

[62] Marc Levoy and Pat Hanrahan. Light field rendering. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, page 31–42, New York, NY, USA, 1996. Association for Computing Machinery. ISBN 0897917464. doi: 10.1145/237170.237199. 12

[63] Chunyuan Li, Andrew Stevens, Changyou Chen, Yunchen Pu, Zhe Gan, and Lawrence Carin. Learning weight uncertainty with stochastic gradient mcmc for shape classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5666–5675, 2016. 46

[64] Xiao Li, Yue Dong, Pieter Peers, and Xin Tong. Modeling surface appearance from a single photograph using self-augmented convolutional neural networks. *ACM Transactions on Graphics (ToG)*, 36(4):1–11, 2017. 46

[65] A. Loza, L. Mihaylova, N. Canagarajah, and D. Bull. Structural similarity-based object tracking in video sequences. In *2006 9th International Conference on Information Fusion*, pages 1–6, 2006. doi: 10.1109/ICIF.2006.301574. 62

[66] Zander Majercik, Jean-Philippe Guertin, Derek Nowrouzezahrai, and Morgan McGuire. Dynamic diffuse global illumination with ray-traced irradiance fields. *Journal of Computer Graphics Techniques Vol*, 8(2), 2019. 18

[67] Michael D McCool and Peter K Harwood. Probability trees. In *Graphics Interface*, volume 97, pages 37–46, 1997. 25, 27

[68] Morgan McGuire, Mike Mara, Derek Nowrouzezahrai, and David Luebke. Real-time global illumination using precomputed light field probes. In *Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 1–11, 2017. 18

[69] Johannes Meng, Marios Papas, Ralf Habel, Carsten Dachsbacher, Steve Marschner, Markus Gross, and Wojciech Jarosz. Multi-scale modeling and rendering of granular materials. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 34(4), July 2015. doi: 10/gfzndr. 75

[70] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4460–4470, 2019. 4, 5

[71] 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. In *ECCV*, 2020. 5, 18, 19, 50, 76

[72] Thomas Müller. "practical path guiding" in production. In *ACM SIGGRAPH Courses: Path Guiding in Production, Chapter 10*, pages 18:35–18:48, New York, NY, USA, 2019. ACM. doi: 10.1145/3305366.3328091. 20, 23, 30, 31, 33, 36

[73] Thomas Müller, Marios Papas, Markus Gross, Wojciech Jarosz, and Jan Novák. Efficient rendering of heterogeneous polydisperse granular media. *ACM Trans. Graph.*, 35(6):168:1–168:14, November 2016. ISSN 0730-0301. doi: 10.1145/2980179.2982429. 75

[74] Thomas Müller, Markus Gross, and Jan Novák. Practical path guiding for efficient light-transport simulation. In *Computer Graphics Forum*, volume 36, pages 91–100. Wiley Online Library, 2017. doi: 10.1111/cgf.13227. 17, 23, 24, 27, 29, 31, 32, 33, 35, 36, 37, 38

[75] Thomas Müller, Brian Mcwilliams, Fabrice Rousselle, Markus Gross, and Jan Novák. Neural importance sampling. *ACM Trans. Graph.*, 38(5), 2019. ISSN 0730-0301. doi: 10.1145/3341156. 30, 31

[76] Thomas Müller, Brian McWilliams, Fabrice Rousselle, Markus Gross, and Jan Novák. Neural importance sampling. *ACM Trans. Graph.*, 38(5):145:1–145:19, October 2019. ISSN 0730-0301. doi: 10.1145/3341156. 20, 21

[77] Thomas Müller, Fabrice Rousselle, Jan Novák, and Alexander Keller. Real-time neural radiance caching for path tracing. *arXiv preprint arXiv:2106.12372*, 2021. 20

[78] O. Nalbach, E. Arabadzhiyska, D. Mehta, Seidel. H-P., and T. Ritschel. Deep shading: Convolutional neural networks for screen space shading. *Computer Graphics Forum (Proc. EGSR)*, 36(4), 2017. 20

[79] Oliver Nalbach, Elena Arabadzhiyska, Dushyant Mehta, H-P Seidel, and Tobias Ritschel. Deep shading: convolutional neural networks for screen space shading. In *Computer graphics forum*, volume 36, pages 65–78. Wiley Online Library, 2017. 2

[80] Radford M Neal. *Bayesian learning for neural networks*. Springer Verlag, 1996. 46, 70

[81] F. Neyret. Modeling, animating, and rendering complex scenes using volumetric textures. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):55–70, 1998. doi: 10.1109/2945.675652. 75

[82] Thu Nguyen-Phuoc, Chuan Li, Lucas Theis, Christian Richardt, and Yong-Liang Yang. HoloGAN: Unsupervised learning of 3d representations from natural images. In *The IEEE International Conference on Computer Vision (ICCV)*, Nov 2019. 4

[83] Baptiste Nicolet, Alec Jacobson, and Wenzel Jakob. Large steps in inverse rendering of geometry. *ACM Transactions on Graphics (TOG)*, 40(6):1–13, 2021. 92

[84] 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. 56

[85] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 165–174, 2019. 4, 5

[86] Serban D. Porumbescu, Brian Budge, Louis Feng, and Kenneth I. Joy. Shell maps. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, page 626–633, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 9781450378253. doi: 10.1145/1186822.1073239. 86

[87] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021. 91

[88] Gilles Rainer, Wenzel Jakob, Abhijeet Ghosh, and Tim Weyrich. Neural BTF compression and interpolation. *Computer Graphics Forum (Proc. Eurographics)*, 38 (2):235–244, March 2019. doi: 10.1111/cgf.13633. 76

[89] Gilles Rainer, Abhijeet Ghosh, Wenzel Jakob, and Tim Weyrich. Unified neural encoding of BTFs. *Computer Graphics Forum (Proc. Eurographics)*, 39(2):167–178, July 2020. doi: 10.1111/cgf.13921. 76

[90] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022. 91, 92

[91] Florian Reibold, Johannes Hanika, Alisa Jung, and Carsten Dachsbacher. Selective guided sampling with complete light transport paths. *ACM Transactions on Graphics (TOG)*, 37(6):1–14, 2018. doi: 10.1145/3272127.3275030. 16, 39, 40

[92] Erik Reinhard, Michael Stark, Peter Shirley, and James Ferwerda. Photographic tone reproduction for digital images. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 267–276, 2002. 79

[93] Peiran Ren, Jiaping Wang, Minmin Gong, Stephen Lin, Xin Tong, and Baining Guo. Global illumination with radiance regression functions. *ACM Trans. Graph.*, 32(4), July 2013. ISSN 0730-0301. 19

[94] Peiran Ren, Jiaping Wang, Minmin Gong, Stephen Lin, Xin Tong, and Baining Guo. Global illumination with radiance regression functions. *ACM Transactions on Graphics (TOG)*, 32(4):1–12, 2013. 2

[95] Damien Rioux-Lavoie, Joey Litalien, Adrien Gruson, Toshiya Hachisuka, and Derek Nowrouzezahrai. Delayed rejection Metropolis light transport. *ACM Transactions on Graphics*, 39(3), April 2020. doi: 10.1145/3388538. 2

[96] Simon Rodriguez, Thomas Leimkühler, Siddhant Prakash, Chris Wyman, Peter Shirley, and George Drettakis. Glossy probe reprojection for interactive global illumination. *ACM Transactions on Graphics (SIGGRAPH Asia Conference Proceedings)*, 39(6), December 2020. URL `http://www-sop.inria.fr/reves/Basilic/2020/RLPWSD20`. 18

[97] Axel Sauer, Katja Schwarz, and Andreas Geiger. Stylegan-xl: Scaling stylegan to large diverse datasets. *arXiv preprint arXiv:2202.00273*, 2022. 92

[98] Katja Schwarz, Yiyi Liao, Michael Niemeyer, and Andreas Geiger. GRAF: Generative radiance fields for 3d-aware image synthesis. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. 74

[99] Katja Schwarz, Yiyi Liao, Michael Niemeyer, and Andreas Geiger. Graf: Generative radiance fields for 3d-aware image synthesis. *Advances in Neural Information Processing Systems*, 33:20154–20166, 2020. 4, 21, 74

[100] Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. *arXiv preprint arXiv:1708.00489*, 2017. 46

[101] Burr Settles. Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2009. 45

[102] Dario Seyb, Peter-Pike Sloan, Ari Silvennoinen, Michał Iwanicki, and Wojciech Jarosz. The design and evolution of the uberbake light baking system. *ACM Transactions on Graphics (TOG)*, 39(4):150–1, 2020. 48

[103] Peter Shirley, Bretton Wade, Philip M Hubbard, David Zareski, Bruce Walter, and Donald P Greenberg. Global illumination via density-estimation. In *Eurographics Workshop on Rendering Techniques*, pages 219–230. Springer, 1995. 17

[104] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. In *Advances in Neural Information Processing Systems*, 2019. 5, 18, 49

[105] Peter-Pike Sloan, Ben Luna, and John Snyder. Local, deformable precomputed radiance transfer. *ACM Transactions on Graphics (TOG)*, 24(3):1216–1224, 2005. 71

[106] Sebastian U Stich, Anant Raj, and Martin Jaggi. Safe adaptive importance sampling. In *Advances in Neural Information Processing Systems*, pages 4381–4391, 2017. 46

[107] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *NeurIPS*, 2020. 5, 50, 76

[108] A. Tewari, O. Fried, J. Thies, V. Sitzmann, S. Lombardi, K. Sunkavalli, R. Martin-Brualla, T. Simon, J. Saragih, M. Nießner, R. Pandey, S. Fanello, G. Wetzstein, J.-Y. Zhu, C. Theobalt, M. Agrawala, E. Shechtman, D. B Goldman, and M. Zollhöfer. State of the Art on Neural Rendering. *Computer Graphics Forum (EG STAR 2020)*, 2020. 19

[109] Justus Thies, Michael Zollhöfer, and Matthias Nießner. Deferred neural rendering: Image synthesis using neural textures. *ACM Transactions on Graphics (TOG)*, 38(4): 1–12, 2019. 70

[110] T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012. 52

[111] Eric Veach and Leonidas Guibas. Bidirectional estimators for light transport. In *Photorealistic Rendering Techniques*, pages 145–167. Springer, 1995. doi: 10.1007/978-3-642-87825-1_11. 15, 16

[112] Eric Veach and Leonidas J Guibas. Optimally combining sampling techniques for monte carlo rendering. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 419–428. ACM, 1995. doi: 10.1145/218380.218498. 17

[113] Eric Veach and Leonidas J Guibas. Metropolis light transport. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 65–76, 1997. 15, 51

[114] Jiří Vorba and Jaroslav Křivánek. Adjoint-driven russian roulette and splitting in light transport simulation. *ACM Transactions on Graphics (TOG)*, 35(4):1–11, 2016. doi: 10.1145/2897824.2925912. 32

[115] Jiří Vorba, Ondřej Karlík, Martin Šik, Tobias Ritschel, and Jaroslav Křivánek. Online learning of parametric mixture models for light transport simulation. *ACM Transactions on Graphics (TOG)*, 33(4):101, 2014. doi: 10.1145/2601097.2601203. 16, 17, 20, 23

[116] Gregory J Ward, Francis M Rubinstein, and Robert D Clear. A ray tracing solution for diffuse interreflection. In *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 85–92, 1988. 17

[117] Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688. Citeseer, 2011. 46, 51

[118] Rui Xu, Xintao Wang, Kai Chen, Bolei Zhou, and Chen Change Loy. Positional encoding as spatial inductive bias in gans. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13569–13578, 2021. 79

[119] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 586–595, 2018. 62

[120] Ruqi Zhang, Chunyuan Li, Jianyi Zhang, Changyou Chen, and Andrew Gordon Wilson. Cyclical stochastic gradient MCMC for bayesian deep learning. *arXiv preprint arXiv:1902.03932*, 2019. 46

[121] Peilin Zhao and Tong Zhang. Stochastic optimization with importance sampling for regularized loss minimization. In *international conference on machine learning*, pages 1–9, 2015. 51

[122] Quan Zheng and Matthias Zwicker. Learning to importance sample in primary sample space. In *Computer Graphics Forum*, volume 38, pages 169–179. Wiley Online Library, 2019. 21

[123] Peng Zhou, Lingxi Xie, Bingbing Ni, and Qi Tian. CIPS-3D: A 3D-Aware Generator of GANs Based on Conditionally-Independent Pixel Synthesis, 2021. 74

[124] Jia-Jie Zhu and José Bento. Generative adversarial active learning. *arXiv preprint arXiv:1702.07956*, 2017. 46

[125] Tobias Zirr, Johannes Hanika, and Carsten Dachsbacher. Re-weighting firefly samples for improved finite-sample monte carlo estimates. In *Computer Graphics Forum*, volume 37, pages 410–421. Wiley Online Library, 2018. doi: 10.1111/cgf.13335. 33