Universidad Zaragoza
1542

Master Thesis

Neural Rendering of Complex Luminaires

Author

Jorge Condor Lacambra

Supervisor

Adrián Jarabo Torrijos

Escuela de Ingeniería y Arquitectura
2022

Repositorio de la Universidad de Zaragoza – Zaguan http://zaguan.unizar.es

Abstract

In this Master Thesis, we propose an efficient method for rendering complex luminaires based on neural networks. We reduce the geometric complexity of the luminaires by using a simple proxy geometry, and encode the visually-complex emitted light field by using a neural radiance field (NeRF). We tackle the multiple challenges of using NeRFs for representing luminaires, including their extreme dynamic range, their high-frequency content on the spatio temporal domain, and the spherical coverage, as well as the required modifications for seamlessly integrating our NeRF in synthetic enviroments. For that, we use a combination of non-exponential transmittance functions, and a novel loss that accounts for the HDR content as well as alpha blending for integration. We implement our model into a modern deep learning framework, and demonstrate high-quality neural rendering of such luminaires. Then, we integrate our model into the rendering software Mitsuba, and demonstrate renders with much less variance with a given sample count, simultaneously achieving a high visual quality. Finally, we propose several avenues for future work where our neural implicit luminaires could be used for importance sampling and drastically reduce rendering times.

Acknowledgements

I would like to thank my supervisor, Adrián, for his thoughtful insights, patience and his impressive ability for thinking out of the box. To the whole GiLab, for accepting me, helping when needed and being an awesome group of creative people all around. And to my family and girlfriend, for supporting me and always being there.

Contents

List of Tables

List of Figures

1. Introduction

Rendering techniques based on Monte Carlo methods have replaced rasterized graphics in the majority of applications where generating realistic computer images is required [1]. This is due to the simplicity of its implementation and its capacity to simulate most light interactions within a scene. These techniques are based on generating a random set of light paths from the different light sources in the scene to statistically estimate the total incident radiance on the virtual camera. However, these algorithms carry a high computational cost, which significantly increases as the light transport effects in the scene we are simulating become more complex [2]: as the complexity of the simulated light paths increases, so does the stochastic error of the Monte Carlo estimate (its variance), requiring larger sample counts for obtaining a tolerable error.

A particularly challenging scenario results when scenes are illuminated with a relatively complex artificial light source. The illumination from simple area lights or distant illumination from e.g. the sky can be solved by using advanced sampling routines [3, 4, 5] or even computed analytically using approximations [6]. However, as the light source increases in complexity these approaches become unfeasible: the complex light paths connecting the (potentially multiple) individual light sources in the luminaire, and the surfaces being illuminated by simply random chance is unlikely in the best of cases (impossible in a worst-case scenario of perfect reflectors and refractors). Thus, computing the illumination from a complex luminaire would require many thousands of samples to converge, even using complex Markov-Chain Monte Carlo methods [7] that are not suited for most production renderers due to its complexity and unpredictable convergence.

It is worth noting that complex luminaires are ubiquitous in real-world scenes, from simple light bulbs with tiny coils as light sources, to complex chandeliers made by a myriad of small glass pieces scattering the light of the (potentially many) emitters within. Thus, improving over the brute-force approach would enable increasing the complexity of the luminaires used in films or videogames, and in the market of architectural visualization and interior design [8]. Precomputing the radiance field of the luminaires is an effective approach for avoiding the sampling of difficult paths in run time [9, 10]. It allows to encode all light paths exiting the luminaire as a five-dimensional proxy function, which is fast to access during rendering, and avoid complex paths: to determine the illumination at a point, it is only needed to integrate over the proxy 5D function. Unfortunately, accurately precomputing a luminaire requires a very dense storage of the five-dimensional function, which might become unfeasible due to memory constraints, or to combine the precomputed proxy with the real complex geometry of the scene.

In this work we tackle this problem by leveraging the potential of recent advances on deep

learning [11]. In particular, deep learning has already demonstrated to be very effective for improving traditional rendering techniques in several applications including image supersampling [12]; state-of-the-art Monte Carlo denoising [13]; neural networks-guided importance sampling [14, 15]; accurate physically-based materials [16, 17, 18, 19]; or faster rendering of scattering media [20]. Inspired by the success of these approaches, we transform our complex luminaire into a *neural radiance field* (NeRF) [21]. NeRFs have emerged as a suitable representation of very complex three-dimensional scenes reconstructed from a (relatively) sparse set of views, by encoding the scene as a lightweight volumetric function, modeled using a neural network. It allows to compactly encode highly complex spatio-directional representations of 3D scenes, which can be efficiently rendered by standard ray-marching through the volumetric proxy (see Chapter 3 for details).

Unfortunately, off-the-shelf NeRFs are not adequate for our particular task, and modeling a luminaire using a NeRF, and integrating it into a Monte Carlo rendering engine, introduces several challenges, including: 1) luminaires have high dynamic range (HDR), as opposed to the usual low dynamic range (LDR) targeted by NeRFs, and 2) exhibit high angular frequency; 3) they require handling both opaque and transparent elements for integration in synthetic scenes; and 4) as opposed to most image-based NeRF-based rendering, luminaires must be represented over the full sphere of directions, increasing the amount of required samples to reconstruct the NeRF. We solve these challenges using a combination of a linear volumetric representation of the NeRF inspired in correlated media [22, 23], a training loss designed for handling HDR content and the introduction of an orthographic camera model for the training phase.

We integrate our model in the academic renderer Mitsuba, and demonstrate high quality rendering of complex luminaires using our method. In particular, the **contributions** of this work are:

- We pose the problem of high-quality representation of the emission of complex luminaries using NeRFs.
- We develop a NeRF adapted to representing the emission of luminaires, tackling problems such as their high dynamic range, and their combination of opaque and transparent surfaces.
- We integrate NeRF rendering into the physically-based renderer Mitsuba as an `Emitter` plugin with an special `BSDF`, that can be used for rendering any scene with NERF-based luminaires.

2. Related Work

2.1 Volumetric representations of appearance

Lately, volumetric approaches have been successfully deployed to approximate the appearance of complex geometries in a wide array of different applications, from trees [24, 25], cloth and hair [26, 27, 28, 29], or even participating media, as an aggregate of particles (sugar, salt, etc) [30, 31, 32]. More recently, based on previous work on non-exponential media [23, 33], Vicini et al. [22] improved the accuracy and quality of these volumetric representations of surfaces by introducing a non-exponential, parametric transmittance model, which extends beyond the traditional exponential transmittance model used by previous methods and is able to model a higher amount of different media (correlated or otherwise) with a higher level of detail. Our work builds upon these ideas, using a volumetric representation for approximating the appearance of complex luminaries.

2.2 Neural Radiance Fields (NeRFs)

Light fields are vector functions representing the amount of light flowing in every direction through every point in space. There's been an increasing amount of interest in using neural networks in the last few years for learning light fields of scenes from a set of sparse views [34, 35, 36], in order to be able to generate novel view points.

Neural Radiance Fields (NeRF) [21] is one of these methods. From a set of sparse views of a scene and known camera poses, it can learn the complete light field of the scene, enabling the generation of views that were not contained in the original dataset. Their combination of volume rendering techniques and positional encoding allowed them to largely outperform previous methods (see Chapter 3 for more information).

In the last 2 years, NeRF has seen an explosion in popularity, opening many possibilities in a wide range of applications spanning the fields of computer vision, robotics and graphics, such as novel SLAM algorithms that jointly optimize pose and scene reconstruction [37], handling of transparent materials by robotic arms [38], rendering of large scenes from unconstrained sets of images [39], better multi-view dense depth estimations [40], novel pose estimation methods [41, 42], better generative models from audio cues [43]; furthermore, NeRFs themselves have been extended to work in the temporal domain and in deformable scenarios [44, 45, 46], render and train faster [47, 48, 49, 50, 51] and even generalize to other geometries [52, 53, 54, 55, 56, 57],

among many, many other.

We adapt the NeRF pipeline to work with complex luminaries, which poses several challenges due to the need of blending them with a synthetic scene, their large dynamic range, their spherical coverage and the presence of diffractive materials.

2.3 Complex Luminaries

The previous body of work focusing on approximating the light distributed by a luminaire to accelerate its rendering mainly draws from the concept of light fields. The earliest works using light fields in the context of luminaire rendering attempted to measure their light field by enclosing them in virtual surfaces, a proxy, decoupling their emission from the underlying geometry [58, 59]. While our approach is similar to theirs in essence, the quality of their results heavily depended on the number of samples used to approximate the light field, which made it impractical for achieving high quality results. In a similar line, canned light sources [60] not only measured the light field of the luminaire, but stored it in a format that could be used by rendering engines. This approach, however, suffered from low quality results and immense data storage needs.

A different approach also involving some kind of light field precomputation is the usage of near-field raysets, which in essence are a set of light field samples parameterized in a suitable manner [61, 62]. The raysets approach can record the energy leaving the luminaire in the form of rays captured on the virtual proxy. These methods, and other derivations of them to support importance sampling and data compression [63] work well in the context of small and localized emmitters such as light bulbs, but scale badly to more complex luminaires.

More recent approaches include those by Lu et al. [64] and Velázquez et al. [9] also precompute the light field of the luminaire in order to accelerate its rendering. The former is able to importance sample a discretized light field, but it requires an uncompressed light field of the luminaire to do so, suffering from storage issues. The latter uses less storage, reducing the luminaire to a set of anisotropic point lights (APLs), capable of approximating the light coming from the luminaire and encoding it in a set of tabulated directional distributions; however, it hasn't been possible to date to integrate it into a multiple importance sampling strategy, which makes it very inconvenient for its usage in regular path tracing-based rendering engines.

The closest method to ours is that of Zhu et al. [10]. Instead of attempting to reduce the luminaire to a set of APLs, it uses a set of 3 multilayer perceptrons (MLPs) to not only learn the complete light field of the luminaire, but to perform importance sampling and blending with the rest of the scene. The models receive ray queries with the origin and direction as input, and output color (evaluation), transparency or a small image of the luminaire as seen from the ray origin, which gets transformed into a PDF to guide an importance sampling scheme. It does, however, suffer from several issues: 1) it requires hundreds of thousands of images of every single luminaire in order to be able to train each of their 3 networks (200k-350k for the evaluation and blending/transparency network, 1M for the sampling one), which account for thousands or millions of core compute hours; 2) the appearance of the luminaires is overly smoothed, and 3) it still needs to query the networks several times per ray, which is slow and requires a dedicated GPU for rendering. We feel we can improve on top of this brute-force learning approach by

making use of the recent advances in neural implicit representations, in particular NeRF.

Our method extends NeRF, enabling it to work on higher dynamic ranges, and requires only a 100 images to learn a high quality representation of the full light field of the luminaire. Unlike Zhu's method, which directly predicts final color for each input ray, we use a volume-rendering approach, which is effectively able to implicitly learn a correct 5D representation of the luminarie's radiance field and produce higher quality results, with finer detail. We also make use of spherical functions, so that our ray predictions are view-independent. This allows us to distill our model into an octree data structure, reducing the volume-rendering to a simple querying of stored values. This is much faster than using MLPs for inference, and can be perfectly integrated into a rendering engine without the need of GPUs. Alternatively, if data storage is critical, we can perform inference of our model during rendering; this is much slower, but we achieve a 4x storage reduction when compared with Zhu. Finally, our method does not require neither the original geometry nor a different network for transparency, as we can obtain alpha masks directly from our neural volume representation in order to blend the luminaire with the rest of the scene.

3. Background

3.1 Monte Carlo Direct Illumination

The rendering equation [65] is the basis of all current realistic rendering techniques; it defines the total amount of energy emitted from a point in the scene along a particular viewing direction L_o as a function of the incoming light L_i and a bidirectional reflectance distribution function f_r (BRDF) modeling the angular distribution of scattered light as

$$L_o(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + \int_\Omega f_r(\mathbf{x}, \omega_i, \omega_o)\, L_i(\mathbf{x}, \omega_i)(\omega_i \cdot \mathbf{n_x}) d\omega_i, \tag{3.1}$$

with $L_e(\mathbf{x}, \omega_o)$ the emission at $\mathbf{x}$, and $\mathbf{n_x}$ the normal at $\mathbf{x}$. Note that the rendering equation is recursive, since L_i is the result of the outgoing radiance at other point in the surface.

Monte Carlo (MC) integration has dominated computer graphics ever since their introduction in ray tracing algorithms [66]. Generally, MC methods stochastically estimate the expected value F of a function $f(x)$ so that $F = \int f(x)\, dx$ by averaging a number N of random samples, as

$$F \approx \langle F \rangle = \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{p(x_i)}, \tag{3.2}$$

with $\langle F \rangle$ the MC estimate of F, x_i a random sample, and $p(x_i)$ the probability density function (pdf) of generating sample x_i. Monte Carlo methods are unbiased, and their error is solely characterized by the variance of the estimate. This variance vanishes as a rate of $O(N^{-\frac{1}{2}})$, and it is proportional to how well the probability function $p(x)$ fits $f(x)$.

Thus, MC integration can be used to estimate radiance at every point of a scene. To compute $L_o(\mathbf{x}, \omega_o)$ we use the MC estimate defined as

$$L_o(\mathbf{x}, \omega_o) \approx \langle L_o(\mathbf{x}, \omega_o) \rangle = L_e(\mathbf{x}, \omega_o) + \frac{1}{N} \sum_{i=1}^{N} \frac{f_r(\mathbf{x}, \omega_i, \omega_o)\, L_i(\mathbf{x}, \omega_i)(\omega_i \cdot \mathbf{n_x})}{p(\omega_i)}, \tag{3.3}$$

where $p(\omega_i)$ defines the probability of sampling the direction ω_i. Thus, in order to maximize the variance reduction of the MC estimate, we would need to find a pdf such that $p(\omega_i) \propto f_r(\mathbf{x}, \omega_i, \omega_o)\, L_i(\mathbf{x}, \omega_i)(\omega_i \cdot \mathbf{n_x})$. However, sampling proportionally to the product of the BSDF and the incoming radiance is almost impossible for most scenarios. A more feasible approach would be to use two or more pdfs, proportional to each of the terms (e.g. $p(\omega_i) \propto$ Ł$_i(\omega_i)$), and combine them using some combination strategy (e.g. multiple importance sampling [2]).

However, when light sources are occluded in most directions, or are hidden behind several layers of refractive materials, it is very hard to estimate where the actual light is coming from, and it is almost impossible to find a good $p(\omega_i)$. The reason is that the light incoming at $\mathbf{x}$ from direction ω_i is the result of potentially many bounces of high-frequency, difficult-to-sample, signals. Therefore, it is extremely unlikely to find a good sampling direction ω_i, and the scene requires many samples to be rendered with low amounts of noise. This is exactly why rendering complex luminaires is so challenging: these luminaires contain many layers and small pieces of crystal, and even ornamental pieces of plastic, brass or cloth, that further scatter or block light in many directions. Thus, if we were to know in advance where the final radiance from the luminaire comes from we could implement efficient importance sampling techniques, leading to immense reductions of variance when rendering with low samples.

3.2 Neural Radiance Fields (NeRFs)

Neural Radiance Fields (NeRF) [21] is a method for learning light fields of arbitrarily complex scenes from a set of sparse views and known camera poses, enabling the generation of views that were not contained in the original dataset. The essence of their approach can be summed in Figure 3.1

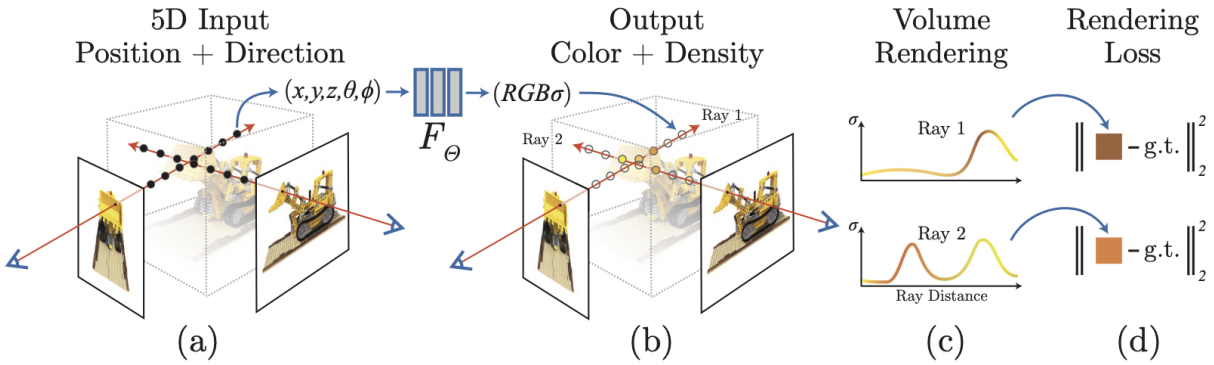

Figure 3.1: NeRF pipeline overview. First, (a) it feeds positional and directional ray data to the model, which (b) uses an MLP to estimate color and density at a given location along the ray, and (c) volume rendering techniques to output a final color prediction. The MLP is then supervised by computing a (d) MSE rendering loss between the estimated and ground truth color. Image courtesy of [21]

The key behind why NeRF performs so well when compared with previous methods lies in three main innovations:

Volume Rendering. NeRF represents the scene's 5D radiance (light) field as the volume density and directional emitted radiance at any point in space. Consequentially, it uses traditional volume rendering techniques to estimate the color $\mathcal{C}(\mathrm{r})$ of every ray $\mathrm{r}(t) = \mathbf{x} + t\,\omega$ traced across its volume as

$$\mathcal{C}(\mathrm{r}) = \int_{t_n}^{t_f} \mathcal{T}(t)\,\sigma\left(\mathrm{r}(s)\right)\,c\left(\mathrm{r}(s), -\omega\right)\,dt, \text{ where } \mathcal{T}(t) = \exp\left(-\int_{t_n}^{t} \sigma\left(\mathrm{r}(s)\right)\,ds\right), \tag{3.4}$$

with near and far bounds of the volume t_n and t_f, respectively, and $\sigma\left(\mathbf{x}\right)$ and $c\left(\mathbf{x}, \omega\right)$ are the density and color at point $\mathbf{x}$ towards direction ω, respectively. Finally, $\mathcal{T}(t)$ is the accumulated

transmittance along the ray; once this value saturates, it means it has reached a surface in the scene, and its accumulated color along the ray is the resulting pixel value. Since volume rendering is differentiable, this allows them to learn the full 5D light field of the scene from only a set of views.

Positional Encoding. NeRF is able to learn high frequency functions thanks to its mapping of input coordinates (position and viewing direction) into a higher dimensional space using a set of periodic functions (Fourier features [67]). Given that networks are naturally biased towards learning lower frequency functions first [68], by reformulating the input as a higher dimensional function they enable the network to learn higher frequency functions more easily.

In essence, they map the three coordinate values in $\mathbf{x}$ and the three components of the Cartesian viewing direction unit vector ω into a higher dimensional space, from $\mathbb{R}$ to $\mathbb{R}^2$ L, using a function γ defined as

$$\gamma(p) = (sin(2^0\pi p), cos(2^0\pi p), ..., sin(2^{L-1}\pi p), cos(2^{L-1}\pi p)). \tag{3.5}$$

The parameter L is empirically set.

Coarse-to-fine Optimization. NeRFs employ 2 different MLPs: a coarse model, and a fine model. This is done in order to avoid wasting a high amount of samples on empty space and occluded regions. The coarse model uses stratified sampling to sample a first set of N_c locations along the ray, while the fine model uses this to make a more informed sampling, sampling a second set of N_f locations on the most relevant parts of the volume, using a form of hierarchical sampling.

More specifically, in order to bias the sample placement of the fine network towards the more relevant parts of the volume as detected by the coarse network, they rewrite the alpha composited color from the coarse network, as a weighted sum of all sampled colors c_i along the ray

$$\hat{\mathcal{C}}_c(r) = \sum_{i=1}^{N_c} w_i c_i \,, \text{ where } w_i = T_i(1 - exp(-\sigma_i\delta_i)) \tag{3.6}$$

with T_i being the transmittance until that point in the ray; essentially the cumulative sum of densities. If these weights are normalized, we obtain a piece-wise constant PDF along the ray. Now, sampling the second set of locations from this distribution can be performed by using inverse transform sampling, finally evaluating the fine network at the union between both sets of samples to compute the final rendered color of the ray.

4. Modelling Luminaire Appearance

Our goal is to evaluate a complex luminaire as efficiently as possible within any given scene. In order to achieve that, we start by following previous works on luminaire modeling [9, 10]: these works partially (or totally) remove the geometric complexity, precomputing the complex light-matter interactions within the luminaire.

In order to model our luminaire, we first make the assumption that it will always be rendered from the outside; this is a reasonable assumption for most applications, even for close-up shots. This allows us to contain our luminaire inside a simple bounding box, acting as a geometric proxy. Then, the radiance field from the luminaire can be encoded in a per-ray basis as a 5D function f relating a position $\mathbf{x}_l$ in the proxy and a direction ω_o with the accumulated radiance exiting the proxy at $\mathbf{x}_l$ in direction ω_o, as

$$f : \mathbb{R}^3 \times \mathbb{R}^2 \to \mathbb{R}; (\mathbf{x}_l, \omega_o) \mapsto f(\mathbf{x}_l, \omega_o). \tag{4.1}$$

This approach has been used by previous works [60, 10], and it is illustrated in Figure 4.1. Thus, the key problem is how to model the function f, such that a) it can be efficiently evaluated in rendering time, b) it is compact from a storage perspective, c) it preserves the spatio-directional high frequency of f, and d) can deal with the potentially large dynamic range of the luminaires.

This mapping is exactly the same problem present in image-based rendering [69], where neural radiance fields (NeRFs, see Section 3.2) have demonstrated to excel at representing complex scenes. Thus, it is natural to pose our problem by using a neural representation of the luminaire radiance field. However, directly applying a state-of-the-art NeRF to model synthetic complex luminaires for rendering does not provide good results, and there are several challenges that need to be addressed. We will first discuss these challenges; then, describe our model and how it overcomes such challenges.

4.1 Main challenges

There are many challenges associated with attempting to fit the radiance field of a complex luminaire with a neural network. We have identified the following as the most important problems we will need to tackle throughout this project:

High Dynamic Range Implicit representations of neural radiance fields have only focused on capturing LDR content, with outputs bounded between 0 and 1, and colors mapped to the sRGB

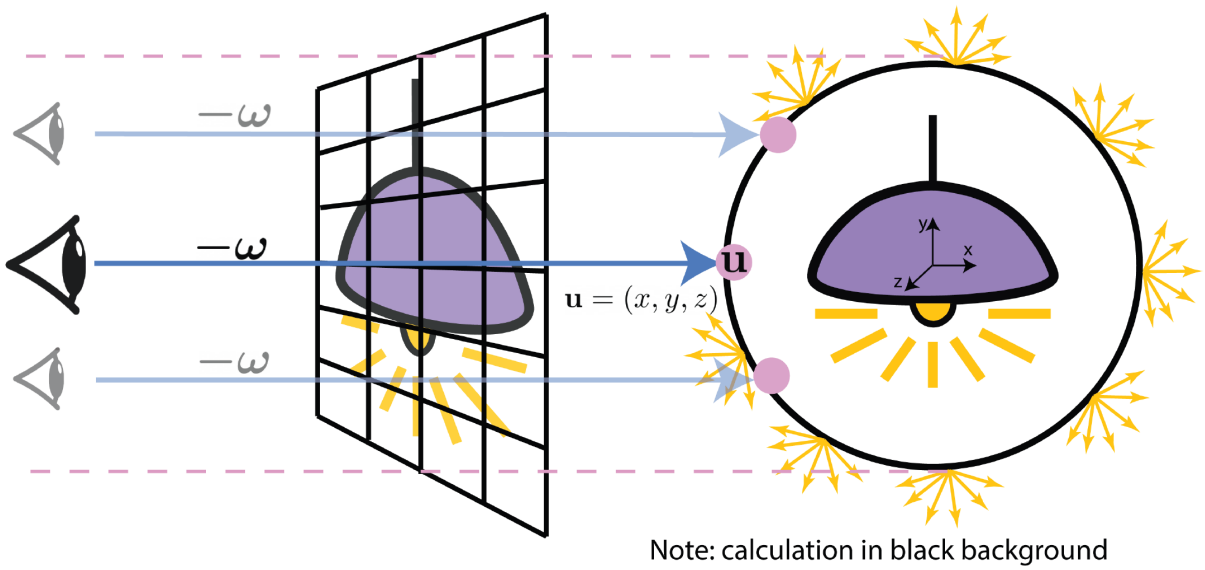

Figure 4.1: The light field model relates positions in the bounding box and ray directions with the accumulated radiance exiting the bounding box at that specific direction and direction. Figure courtesy of Zhu et al. [10]

non-linear curve. However, in a physically-based rendering engine, it is detrimental to sample from the full dynamic range of the luminaire, as the magnitude of the radiance coming from every point of it is lost when using a compressed sRGB representation. It is also problematic to optimize non-linear color curves, as the render is again tone-mapped at the end and this could lead to incorrect color reproduction; we need to optimize linear RGB. Extending these methods to work on HDR content is very challenging, as due to the nature of neural networks, the areas with the highest radiance will dominate the gradient and thus the training, at the expense of fitting the parts with lower radiance.

Null Emission Since we are modeling isolated objects that will be integrated into synthetic scenes, we need to account for rays that carry no radiance. In natural scenes, these absolute-black areas are uncommon as it would indicate the image has not been exposed correctly. Noise usually accompanies these areas as well, making it difficult for the network to actually be forced to optimize for 0 values. In a general machine learning context, dark pixels generate very small gradients during training, which makes them a difficult training target as the error is dominated by the more radiant parts.

Matting In order to blend our luminaire with the rest of the scene, we will need to let through the proxy those rays that actually hit the convex geometry but not the underlying light source. This requires us to incorporate not only the emission from a ray, but also the amount of light that passes through the luminaire (i.e. the *transparency*).

Noisy training data Rendering noise is inherent to path tracing. Due to the complex nature of our luminaires, this noise will be present unless we dedicate hours of compute to each of the

views rendered, which is unfeasible. Furthermore, we cannot filter over our renders, as bias would be introduced and view consistency would be lost. The most problematic kind of noise, however, are the so-called *fireflies*: bright pixels resulting from randomly finding low-probability high-contributing paths, which result into very large variance, aggravating the issue of learning a high dynamic range radiance field.

Full 360-degrees viewing freedom Most neural implicit representations require dozens of images of a scene in order to correctly learn it. As an intelligent interpolation scheme, these neural networks still require cameras placed relatively close to each other; otherwise, its ability to generate novel views is significantly degraded [54]. In our case, however, we need to model the full sphere around the luminaire, as radiance can come from any section of it. Without assuming any symmetries, we would need a larger amount of training data to model such a wide viewing range. This poses several other issues, from training times (more time to fit the data is required when the pool is larger), to data generation times (more time rendering training data); and most importantly, fitting more data, in essence a more complex and constrained function, may require bigger networks or more complex approaches.

4.2 Our Approach

Following a NeRF-like approach, we model our 5D function $f(\mathbf{x}_l, \omega_o)$ (4.1) by integrating the emitted radiance along a ray $\mathrm{r}(t)$ defined with origin at $\mathbf{x}_l$ and direction ω_o, following Equation (3.4). Furthermore, to integrate our luminaire in a scene, we need to also account for the light that passes through from behind our proxy. Thus, our luminaires are modeled following

$$\boxed{f(\mathbf{x}_l, \omega_o) = \mathcal{C}(\mathrm{r}) + \alpha(\mathrm{r})\, L\left(\mathrm{r}(t_f), -\omega_o\right),} \tag{4.2}$$

where $alpha(\mathrm{r}) = (1 - \mathcal{T}(t_f))$ models the transparency along the ray, and $\mathcal{L}\left(\mathrm{r}(t_f), -\omega_o\right)$ is the radiance from the synthetic scene arriving at the intersection of the ray at the far bound of the proxy, which is computed by the renderer.

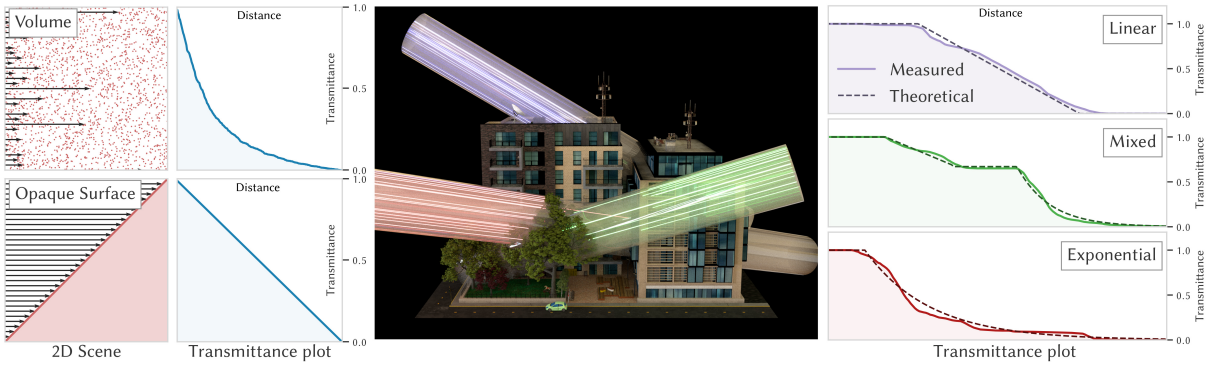

Figure 4.2: The function that models transmittance varies depending on the surface rays intersect with: tracing a batch of rays as in the picture above against a scene, we can see that solid surfaces have a distinct linearly decaying transmittance function, while foliage shows an exponentially decaying behavior. Our luminaires are mostly opaque and are formed by many solid pieces of metal and diffuse crystal: in theory, a linear model should fit them better than an exponential one. Image courtesy of Vicini et al. [22]

In order to compute $\mathcal{C}(\mathrm{r})$ and $\mathcal{T}(t_f)$ we need to integrate the transmittance: most works in NeRF use an exponential form of the attenuation following the Beer-Lambert law of classic radiative transfer [70]. However, this builds on the assumption that the volumetric media is formed by uncorrelated scatterers, which is not true for surfaces where a negative correlation of the differential points in the surfaces exists (i.e. they block all light, and do not shadow each other) [23] (see Figure 4.2), resulting into a linear, and faster-than-exponential, decay. Since we are interested on solid surfaces, we follow Vicini et al. [22] and implement such linear transmittance model in the volumetric definition of our NeRF, as

$$\boxed{\mathcal{T}(\mathrm{r}) = \max\left(0, 1 - \int_{t_n}^{t_f} \sigma\left(\mathrm{r}(s)\right)\, ds\right)} \tag{4.3}$$

Computing $\sigma(\mathbf{x})$ and $c(\mathbf{x}, \omega)$. While we could use any encoding for computing $\sigma(\mathbf{x})$ and $c(\mathbf{x}, \omega)$, a key characteristic of NeRFs is that they encode them using a neural network, which are queried in run-time for each sample in Equations (4.2) and (4.3). We query the directionally-invariant density $\sigma(\mathbf{x})$ directly from the neural network.

For the estimation of the directional color $c(\mathbf{x}, \omega)$, however, we slightly diverge from traditional NeRF. Following recent work on accelerating NeRF's rendering [47], we do not directly obtain the color at a given direction, but a set of spherical harmonic (SH) coefficients instead. SH are a set of orthogonal basis defined on the sphere, that allow representing any spherical function as an infinite sum of terms (for more information on spherical harmonics, see the appendix of [47]). This gives us a low-dimensional, view-independent representation of the color function $c(\mathbf{x}, \omega)$.

An even more important difference of color $c(\mathbf{x}, \omega)$ in our work is the range: While NeRFs are normalized in the interval $[0, 1]$ (low dynamic range), in our case we need to retain the full dynamic range of the light source. This introduces several practical problems in terms of encoding and specially training, where we need to adapt our training loss by introducing a regularizer to cope with extreme contrast ratios in HDR content. Finally, unlike NeRF, we also directly supervise the opacity $1 - \mathcal{T}(t_f)$ (the alpha mask) of our rays. In the following, we describe the architecture of our neural network (Section 4.3), our custom loss function (Section 4.4), our choice of camera model during training (Section 4.5) and some details regarding the training procedure itself (Section 4.6).

4.3 Architecture

In Figure 4.3 we present our architecture. It is a slightly modified version of the one used by Mildenhall et al. [21] and Yu et al. [47]. Overall, we use the same twin-network structure that NeRF uses (coarse and fine models, see Section 3.2). Each network is a densely connected, 8-layered 256-neurons-wide multilayer perceptron (MLP), with an extra output layer of 128 neurons. We use ReLU as the activation function between the inner layers. It is, all in all, the same architecture used in DeepSDF [71], with the inclusion of positional encoding for the inputs (see Section 3.2); we use $L = 10$ for the positions, and $L = 4$ for the directions.

Following Yu et al. [47], our network inputs a position $\mathbf{x}$ and a ω and returns the density

$\sigma(\mathbf{x})$ and the SH expansion of the color $c(\mathbf{x}, \omega)$ (the SH coefficients for each RGB channel) . In practice, it means that we have $(l+1)^2$ output neurons for each color channel, with l being the spherical harmonics degree used (in our case, we use a 4th degree SH expansion). Note that we also apply a non-linear transformation over the SH expansion to obtain the final color. The final color is then estimated as

$$c(\mathbf{x}, \omega) = S\left(\sum_{l=0}^{l_{\max}} \sum_{m=-l}^{l} k_l^m(\mathbf{x})\, Y_l^m(\omega)\right), \tag{4.4}$$

with $k_l^m(\mathbf{x})$ the SH coefficients returned by the neural network for each channel, $Y_l^m(\omega)$ the SH basis projected in direction ω, and $S(\cdot)$ a non-linear activation function. Our differences with Yu et al.'s model, at the architectural level, reside in the output activations used: Instead of using ReLU for the density (σ) estimation, we use a Softplus function [72], whereas for the final RGB estimation, we used an extended range sigmoid (that is, a sigmoid multiplied by the maximum scene radiance, which is known to us as we are only working with synthetic luminaires).

Unlike both NeRF and NeRF-SH, we do not learn a low dynamic range, sRGB tonemapped color function; instead we learn the full dynamic range of the linear color function. The model is thus optimized to learn linear color divided by its maximum scene radiance, and is then expanded back in the output. This way, we can compute losses in a bounded [0,1] range and still recover the full extent of the dynamic range. This proved much more stable than using unbounded activation functions such as the exponential, with the error committed by the loss of information by the division being negligible when training with full precision (`float32`) linear values. We tested various different approaches to unbound the output; we refer to the Appendix A for details on other explored activation functions.

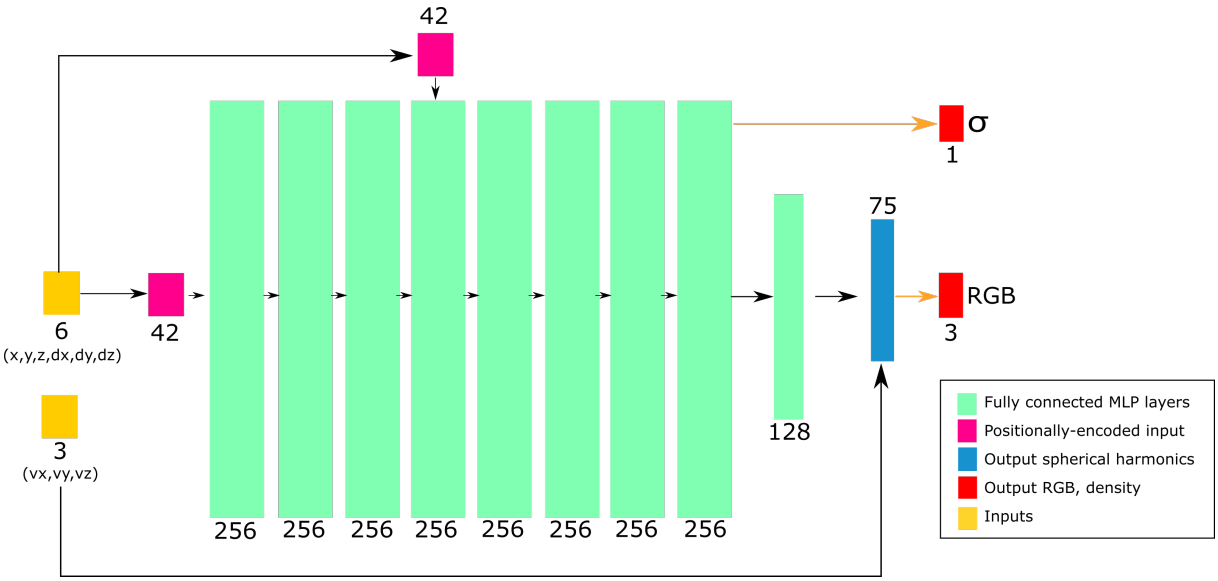

Figure 4.3: Our architecture

4.4 Loss function

As in NeRF, we supervise color in both the coarse and fine models with a mean squared error loss (MSE)

$$\mathcal{L} = \frac{1}{|\mathcal{R}|} \sum_{\mathrm{r} \in \mathcal{R}} \left[\left\| \hat{\mathcal{C}}_c(\mathrm{r}) - \mathcal{C}(\mathrm{r}) \right\|_2^2 + \left\| \hat{\mathcal{C}}_f(\mathrm{r}) - \mathcal{C}(\mathrm{r}) \right\|_2^2 \right], \tag{4.5}$$

where $\hat{\mathcal{C}}_\rfloor(\mathrm{r})$,$\hat{\mathcal{C}}_\{(\mathrm{r})$ are the color predictions by the coarse and fine networks respectively, $\mathcal{C}(\mathrm{r})$ the color ground truth, and $\mathcal{R}$ is the set of $|\mathcal{R}|$ sampled rays. However, using just MSE with the kind of wide dynamic range present in our luminaires produces wrong color reproductions, darkening low radiance areas or outright not converging to a result, as the error is dominated by the exposed light sources, strong specular reflections and other highly radiant areas.

In order to achieve an even learning across the full dynamic range, we needed to differently weight low and high radiant areas. We achieved this by regularizing MSE by the square of the approximate luminance of the pixel, in the spirit of Lehtinen and colleagues [73]. In essence, Lehtinen et al. proposed to use a mapping similar to Reinhard's global tonemapping operator [74][1]. However, unlike their approach, we also need to multiply the estimated pixel luminance by the maximum scene radiance $L_{\max}$ to compensate the fact that we are estimating color in the (0,1) range; this results in the following HDR-aware loss

$$\mathcal{L} = \frac{1}{|\mathcal{R}|} \sum_{r \in \mathcal{R}} \left[\left\| \frac{\hat{\mathcal{C}}_c(r) - \mathcal{C}(r)}{\lambda \hat{\mathcal{C}}_c(r) + \epsilon} \right\|_2^2 + \left\| \frac{\hat{\mathcal{C}}_f(r) - \mathcal{C}(r)}{\lambda \hat{\mathcal{C}}_f(r) + \epsilon} \right\|_2^2 \right], \tag{4.6}$$

where ϵ is a regularizing empirical term; we found that $\epsilon = 0.01$ works well for all our tested luminaires. In practice, we found that any decaying function can be used as a regularizer to the MSE loss, balancing the supervision of low and high radiance areas. For example, we found success with exponentially decaying functions of the type $e^{-\beta x}$, with β being a tunable hyperparameter. However, these required manually setting the β for each scene, depending on the contrast ratio and maximum radiances, making the previous regularization more convenient.

Finally, when we analyzed the original NeRF model in detail, we discovered that when using NeRF with individual objects in empty space (empty backgrounds), training convergence heavily depended on opacity supervision. This is not explicitly done through a direct alpha mask supervision, but rather by ingeniously using white backgrounds. By changing the color output of the model to

$$\hat{\mathcal{C}'}(\mathrm{r}) = \hat{\mathcal{C}'}(\mathrm{r}) + (1 - \hat{\alpha}(\mathrm{r})), \tag{4.7}$$

NeRF performs an alpha mask supervision under the hood, as incorrectly predicting the opacity of a pixel makes its target color unreachable (the sigmoid cannot output negative values to compensate incorrect alpha predictions), producing high errors. This works for the original NeRF due to the nature of their selected examples. In purely absorbing scenes white only corresponds to the background and very uncommon high energy highlights. In contrast, in our luminaires white can correspond to exposed light sources, and it can amount to a significant number of pixels in the scene. Thus, using this kind of alpha supervision either destabilizes

[1] Reinhard's global tonemapping has the form $M(y) = \frac{\log(\lambda\, y + \epsilon)}{L_{\max}}$ and derivative $M'(y) = \frac{\lambda}{(\lambda\, y + \epsilon)^{-1}}$.

training or produces incorrect results in most luminaires. We alleviated this issue by combining white backgrounds with a direct supervision of alpha masks through an MSE loss, as

$$\boxed{\mathcal{L} = \frac{1}{|\mathcal{R}|} \sum_{r \in \mathcal{R}} \left[\left\| \frac{\hat{\mathcal{C}}_c(r) - \mathcal{C}(r)}{\hat{\mathcal{C}}_c(r) + \epsilon} \right\|_2^2 + \left\| \frac{\hat{\mathcal{C}}_f(r) - \mathcal{C}(r)}{\hat{\mathcal{C}}_f(r) + \epsilon} \right\|_2^2 + \|\hat{\alpha}_f(r) - \alpha(r)\|_2^2 \right]}. \tag{4.8}$$

Note that we only need to supervise opacity in the fine model, as it is the one that will define the final opacity of a ray. As all our data is synthetic, we have ground truth alpha masks available for supervision.

4.5 Camera Model

NeRF uses a perspective camera model; this essentially means rays of each camera pose have a common origin, and their direction is defined by the focal length of the virtual camera and the resolution of the image. However, during our testing, we discovered that NeRF struggles to learn correct volumes when using wide angles (i.e. small focal lengths; we tested with focal lengths from 18 to 70mm). Every working example of NeRF already uses abnormally high focal lengths (in the order of thousands of mm) or forward-facing scenes; both cases have in common a much smaller variability in ray directions and almost parallel rays. Inspired by computational tomography, which also attempts to learn volumetric representations of a mix of participating media and opaque surfaces (bones, fatty tissue, muscle, etc) and where rays are traced completely parallel to each other, we decided to implement an orthographic camera model in NeRF.

In an orthographic camera model, rays are perpendicular to the camera plane and parallel to each other, with the origin being placed at the center of every pixel in the image. The lack of foreshortening (i.e. objects do not become smaller when placed further from the camera) only impacts the learning phase, and we can use our resulting neural radiance field with perspective models during inference without any issues. In Figure 4.4 we can see a comparison between using NeRF's perspective camera model with a focal length of 50mm and our orthographic model. While when using the perspective model the neural network generates artifacts close to the camera plane to justify many details in the training views, rapidly deteriorating its capacity to generalize to novel views, the orthographic model is able to learn smooth surfaces and obtain much better quality results. We also tested deactivating color supervision, supervising depth only; the model was still not able to produce smooth surfaces with small focal lengths.

4.6 Training Details

We used JAX [75] and its machine learning API, FLAX [76] for training. We trained our models with a batch size of 1024 rays, each of them sampled at $N_c = 64$ positions in the coarse model, and $N_f = 128$ additional coordinates in the fine one. We used $d = 4$ degrees of spherical harmonics, and an $\epsilon = 0.01$ for the color loss regularizer. Max scene radiances varied from the $\lambda = 1.5$ of the *dallas* dataset, to the 12.0 of the *portica* dataset. All three components of the loss (coarse color supervision, fine color supervision, and alpha supervision) were given the same

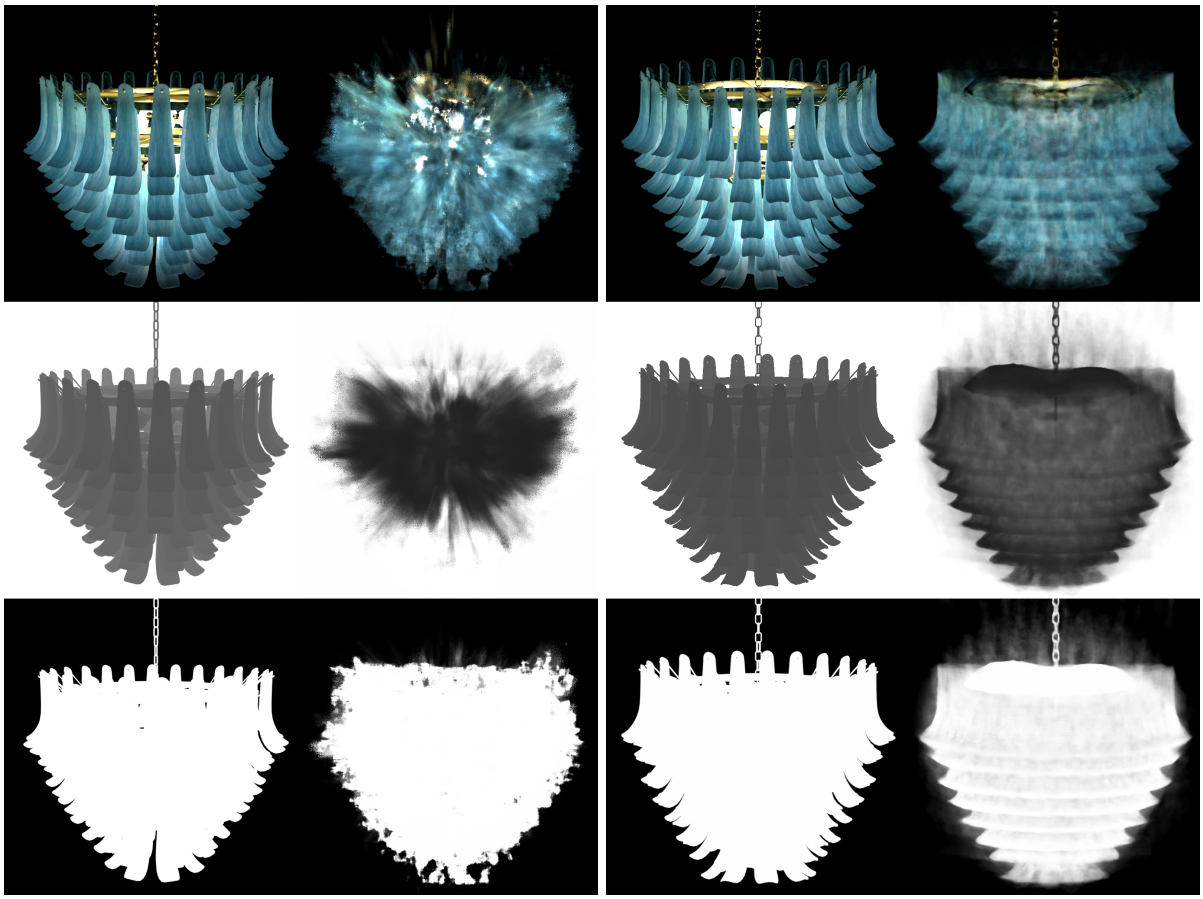

Figure 4.4: Comparison between using a perspective or an orthographic camera model during training, on the *Portica* dataset. On the left we use a 50mm focal length perspective camera, while on the right an orthographic camera model was used; in both cases, we show the ground truth on the right and the estimate on the left, with the first row being the color prediction, the second its depth estimation and last the predicted alpha mask. Notice the high frequency noise present in the left depth map, in contrast to the smooth surfaces learned by the orthographic model.

weight. We used Adam[77] combined with an exponentially decaying learning rate, starting at $lr = 5 \times 10^{-4}$ and decaying to $lr = 5 \times 10^{-6}$ by the end of training. Our models were trained for 700k-1.1M iterations, which took between 1 and 2 days on an RTX 2080Ti.

5. Integration into a Rendering Engine

We integrate our NERF-based complex luminaires in the physically-based open-source renderer Mitsuba [78]. For that, we define a new `Emitter` plugin implementing our model (`complexlum`), which is attached to the proxy geometry (in our case, a `Sphere`). This plugin has two main functions: 1) an evaluation function `eval` that returns the radiance emitted in a given position $\mathbf{x}_l$ in direction ω_o; and 2) a sample function `sample` that samples the light source. In addition, to account for the non-occluding areas of the light source (e.g. semi-transparent areas, or empty space within the proxy), we assign to the proxy a null BSDF that allows us to handle such transparency (`nerf_mask`).

Emitter evaluation Once a ray hits the proxy geometry, we evaluate the contribution of the ray by evaluating the NeRF encoding the luminaire. We first put the ray in the local coordinates of the light source. Then, we simply evaluate Equation (4.2) numerically. While there are a number of unbiased approaches for volume rendering [79], we use the same procedure as in other NeRF methods, based on ray marching[1].

Emitter sampling We currently only support a uniform sampling procedure; that is, our complex luminaire is sampled uniformly over the surface of the proxy that contains it. However, we acknowledge that directing samples towards the areas where energy is higher in the luminaire would provide massive speedups and variance reductions with low sample renderings. Since having the complete light field of our luminaire provides us with the final sources of radiance in the luminaire, we can avoid the issue of having to connect light source and shading point, which usually requires massive extra computational resources for some of these complex luminaires. Sampling procedures based on these implicit light fields remain a very interesting avenue of future work.

Transparency BSDF Blending is a fundamental component of our implementation. Our luminaires don't fill the entire bounding box that contains them in the scene, meaning that it is necessary to let through transparent rays across the proxy. In Figure 5.1, we can see an example of our *dallas* luminaire rendered without the blending component. As you can see, our entire spherical proxy is opaque, and the luminaire is incorrectly blended with the scene.

For this reason, we create a custom bidirectional scattering distribution function (BSDF),

[1] Ray marching estimates the integral along the ray by using simple piece-wise constant quadrature, i.e. by taking small steps along the path and sample on each step.

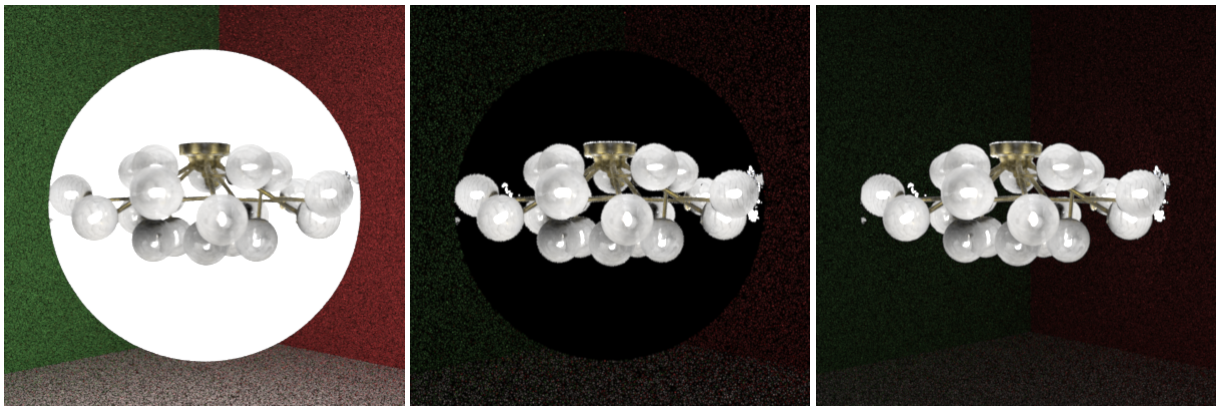

Figure 5.1: Results of rendering our *dallas* luminaire with our blending mechanic deactivated (left), just computing the product of opacity by estimated color (centre) and our full blending scheme (right)

which allows us to perform BSDF sampling of our luminaire and combine it with multiple importance sampling (MIS) methods. The new BSDF exploits the predicted alpha channel of our models for each ray to let through rays across the bounding sphere of the luminaire that intersect empty areas, where no radiance is being emitted, allowing us to blend the NERF with the rest of the scene. MITSUBA is extended accordingly to enable emitters to have indexed values, which it naturally wouldn't allow. Blending the radiance field with the rest of the scene in this way enables us to still compute bounces from the luminaire in areas where no radiance is being emitted (some brass parts, for example). While having null radiance, rays still saturate the density in these areas, as they are recognized as surfaces by the model.

5.1 Implementation details

Our model is trained with JAX, which allows for faster training and inference than other frameworks. However, it comes with a major caveat for our particular application: JAX does not have a C++ API. In order to avoid re-implementing NeRF in C++, we exported our JAX functions into a TensorFlow frozen graph using JAX2TF [75]. This disables the posibility of computing gradients (which we do not need as we are not re-training our models outside of JAX) and is not very memory efficient (as the graph explicitely describes every basic function performed in the order it is used), but allows us to rapidly obtain an end-to-end model that inputs ray queries and outputs final RGB and density values. Once we obtained the exported model, we used TensorFlow's C API to perform inference of our frozen graph within Mitsuba.

Note that, as some authors [16] have previously suggested, launching inference queries to TensorFlow when using purely fully connected MLP models results in a heavy overhead. Given that we are essentially performing matrix multiplications with pre-trained weights, we could obtain huge performance gains by implementing a vectorized version of our neural network using e.g. Eigen [80]. This could significantly improve the performance of our implementation.

Another avenue for further performance gains is to perform volume rendering natively in C++, instead of freezing the volume rendering equations along with the model graph. Finally, exploiting the fact that we are optimizing spherical harmonics instead of direct RGB values

allows us to extract a representation of our neural radiance field into an octree, a data structure that would enable much faster rendering, as no matrix multiplication would be needed, just accessing stored data values [47]. We demonstrate the potential of these methods for improving the performance of our approach in Section 7, though we could not implement them in due time and remain lines of future work.

6. Evaluation and Results

Finally, we will evaluate the performance of our appearance model, its integration in Mitsuba and show some renders in realistic scenarios. But first, we will talk about the different datasets we generated and our intentions with them. Extra renders, videos and other results are stored in a public Google Drive folder[1].

6.1 Data Generation and Dataset Properties

In order to generate our datasets, we used Blender for the modelling part and Mitsuba for rendering. We obtained 3D models from a variety of sources [81, 82, 83] and also created some of our own. We sample cameras in a 360-degree sphere surrounding our luminaires using a 2D Halton low-discrepancy sequence warped to the surface of the sphere. Using a pseudo-random camera placement scheme allows us to ensure cameras are not placed either too close or too far from each other, evenly sampling the full area surrounding the target geometry, while allowing us to easily increase or decrease the amount of views used during training while ensuring this property is still maintained at any moment. For extensive details into the generation of poses, rendering and the adaptation of the datasets to NeRF/NSVF data conventions, we refer to Appendix A.

In order to test our method, we rendered and used 4 different datasets, which can be seen in Figures 6.1. We include the raw color histogram of each view, to better visualize the extreme contrast ratios our method has to model. We sum our intentions with each dataset in Table 6.1. The datasets were rendered with a resolution of 800x800 using an orthographic camera model.

[1] https://drive.google.com/drive/folders/1S4oS3fkwXUjSakcxgoigJt5pbFdfNt6W?usp=sharing.

Geometry	**Description and Intention**
Dallas	Classic chandelier with many different light sources, crystal components and view-dependent metallic reflections
Shard	Test the limits of the model to learn highly refractive luminaires
Portica	Very Complex geometry and exposed light sources. Test a combination of extreme dynamic range curves and intricate geometry
Neoclassical	Test our capacity to model luminaires with a significant amount of non-emissive areas

Table 6.1: Purposes of our different datasets

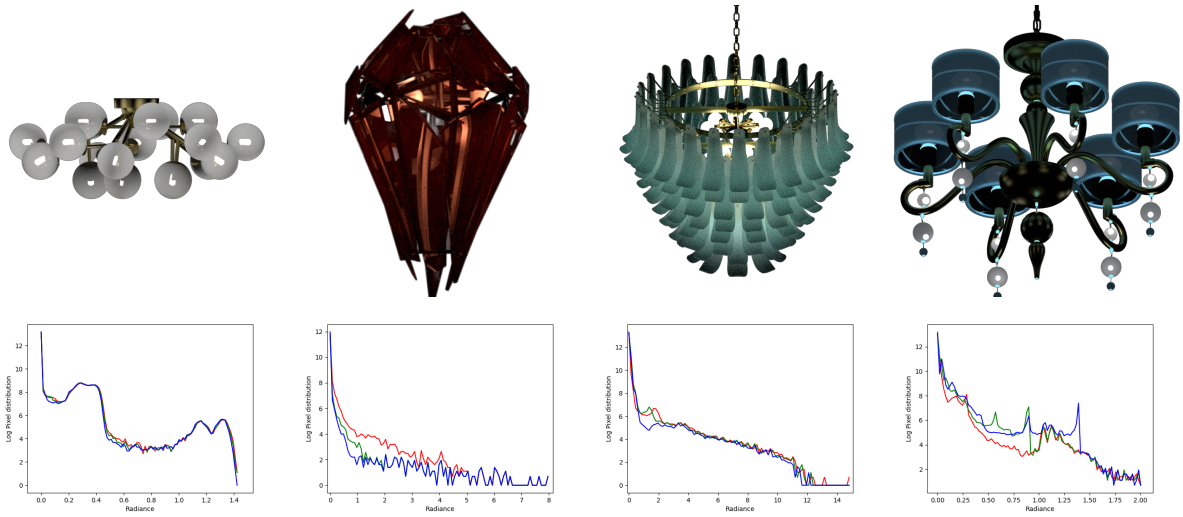

Figure 6.1: Random views of the different datasets we rendered, and their respective raw color histograms for reference. From left to right: *dallas*, *shard*, *portica*, *neoclassical*

6.2 Evaluation of our appearance model

We rendered some test poses using models trained on our datasets in Figure 6.2. In Table 6.2, we evaluate the performance of these luminaires on their respective test dataset. It must be said, however, that due to how we place cameras and select test poses, these are placed at the maximum parallax point between 2 train poses; effectively making them the worst case possible. We encourage the reader to see our videos [2] to visualize real-world performance. Training time oscillated between 1 and 2 days for each scene, and in all cases we used a 100 images as training input.

	PSNR	SSIM
dallas	32.1973	0.9541
portica	20.9595	0.7859
shard	17.8199	0.7412
neoclassic	25.6657	0.8974

Table 6.2: Measured results for the different datasets used

Loss ablation Additionally, in Figure 6.3, we present an ablation study comparing the results obtained when using the different components of our loss function in the *shard* dataset. This is a very challenging dataset, due to the numerous refractive shields, complex light paths and extreme contrast ratio. We compare them against each other and against the results obtained by training a normal, LDR sRGB NeRF on our luminaire dataset, which we can be used as a baseline, providing further proof of the difficulty of the dataset even when trained with a compressed low dynamic range. We also evaluate the whole test dataset on different metrics (PSNR, SSIM) and compute their averages in Table 6.3. We include the result for training the

[2] https://drive.google.com/drive/folders/1S4oS3fkwXUjSakcxgoigJt5pbFdfNt6W?usp=sharing.

Figure 6.2: Random test views rendered with our approach

same luminaire on with a traditional NeRF (that is, with a low dynamic range, MSE and sRGB colors) for reference.

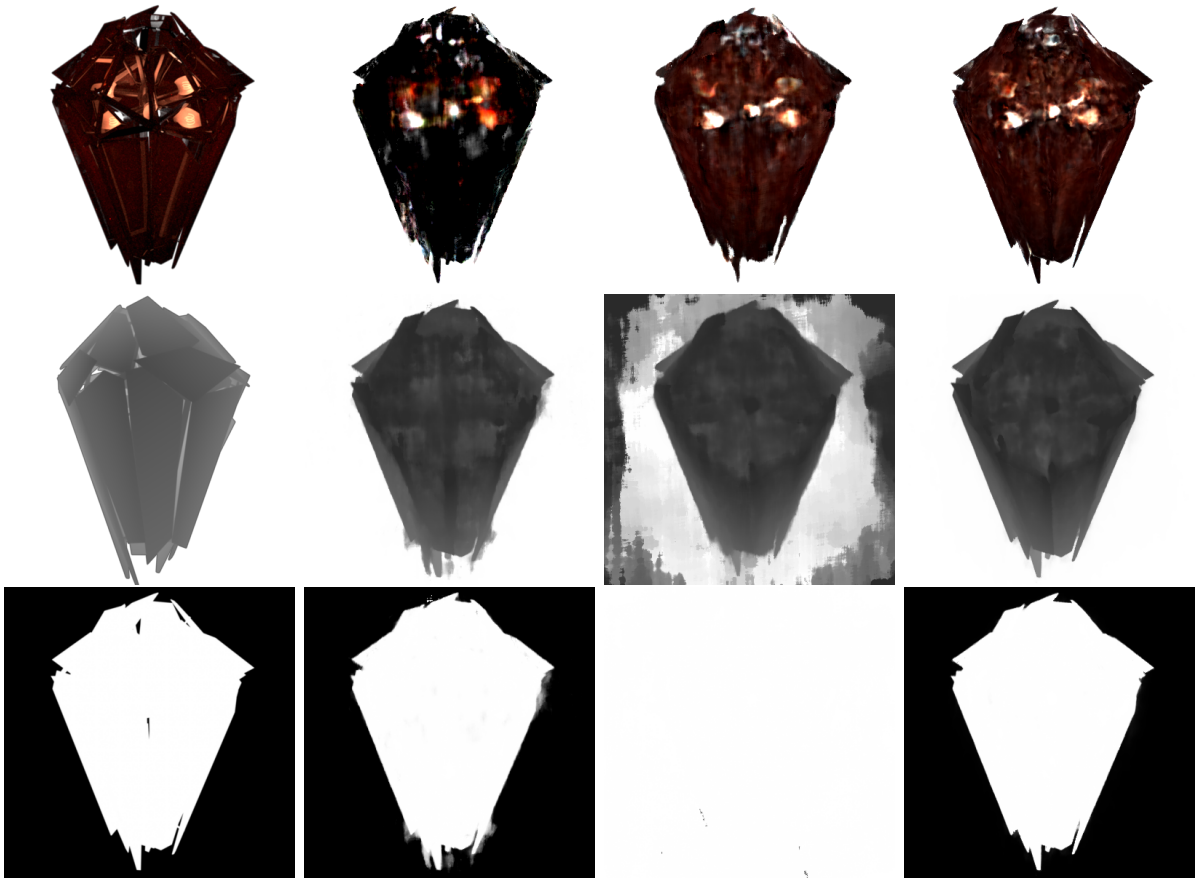

Figure 6.3: Test renders for the ablation study. First row corresponds to color renders, second to predicted depth, and third to predicted alpha. From left to right: Groundtruth test view from the *shard* dataset; MSE+whiteBKG; MSE+WhiteBKG+regularization; MSE+WhiteBKG+regularization+Opacity loss

	PSNR	SSIM
sRGB LDR MSE	19.1936	0.7659

Linear HDR MSE	(collapses)	(collapses)
Linear HDR MSE + White BKG	16.1787	0.7095
Linear HDR MSE+WhiteBKG+regularization	16.6968	0.7183
Linear HDR MSE+WhiteBKG+regularization+Opacity loss	**17.8199**	**0.7412**

Table 6.3: Ablation study for the different components of our loss function

Transmittance model ablation We also compared the impact of using our linear transmission model vs the exponential model traditionally used in NeRF (Table 6.4). As you can see, our linear model outperforms the exponential one on the test dataset of our *dallas* luminaire.

Transmittance model	PSNR	SSIM
Exponential	31.0389	0.9432
Linear	**32.1973**	**0.9541**

Table 6.4: Comparison between the two different transmittance models implemented, training on the *dallas* dataset

Performance We did not finish the Mitsuba integration of the luminaire octree rendering method in time, but we can analyze the performance of the different methods for generating a view of a luminaire outside of the path tracer. In Figure 6.4 we can compare the ground truth render with the NeRF and Octree renders, together with their respective rendering times. We used 4 spherical harmonics degrees to train all our models, and achieve a speedup of x10000 when compared with NeRF. However, on difficult scenes such as *dallas*, with a full sphere of possible viewing angles, this extraction process requires tweaking several hyperparameters in order to work best, such as the density threshold for considering a voxel empty, the number of cells to create, etc. Otherwise, it can lead to results with a clearer, whiter appearance, due to considering some voxels in front of the camera solid instead of empty. This can be slightly noticed in Figure 6.4; a more careful choice of hyperparameters is required to correctly extract this scene.

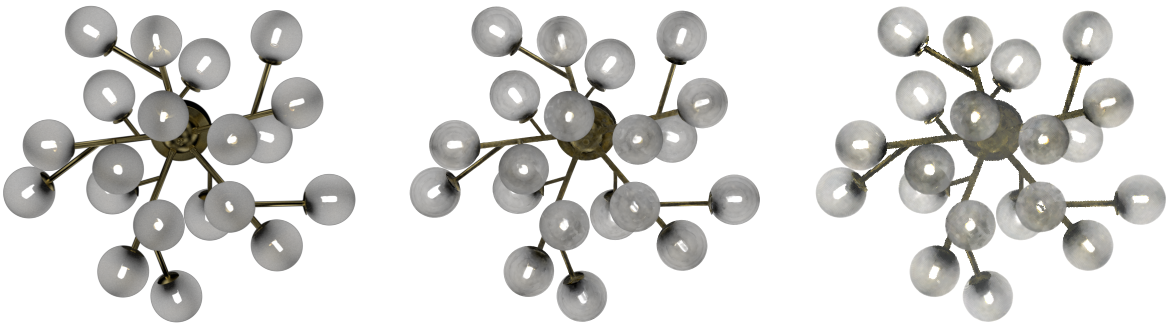

Figure 6.4: From left to right: path-traced ground truth (630 seconds), NeRF render (16 seconds) and Octree render (1.456ms) for a random test view

6.3 Evaluation of our Mitsuba integration

In Figure 6.5, we can see the result of rendering our *portica* luminaire within a Cornell Box. We compare it against the same scene, but using a traditional path tracer and the original geometry. As we can see, at the same number of samples, our method thoroughly outperforms the traditional path tracer. Even with a uniform sampling scheme, variance in the scene is much smaller. Rendering times, however, paint a different picture: our method takes 54.5m to render, while the normal path tracer takes just 7 seconds. This is mainly due to using single-ray batches, and our CPU-only test implementation. Batch inference on GPU, or using the proposed luminaire octree, could certainly bring performance up by several orders of magnitude; this remains a promising avenue for future work.

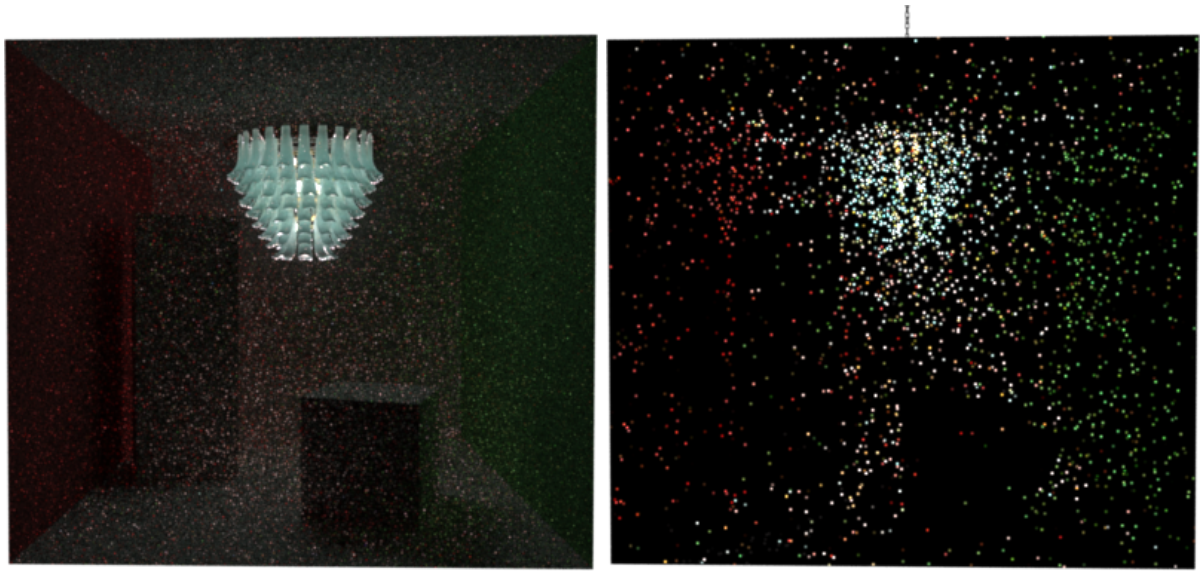

Figure 6.5: Mitsuba renders for the *portica* luminaire in a Cornell Box. Left is our method; right, the same scene rendered with a traditional path tracer and the original geometry. Both scenes rendered with 16 samples per pixel.

We can also render a close-up of our luminaires, as in Figure 6.6. As we can see, the quality of the rendered luminaires is generally quite high (with perhaps the exception of *shard*, our most challenging dataset); however, we can notice the presence of small light leaks in the alpha blending due to the slightly imperfect alpha estimation around the borders, as well as small artifacts on the edge of the bounding box for some of the luminaires. We believe these artifacts are caused due to the periodic nature of the functions used to map our inputs (Fourier features): when querying the edges of the bounding box, due to how we raymarch from the intersection point along the ray for a fixed distance, it is possible to access previously unseen spaces. Our theory is that the network propagates the luminaire in space periodically, generating these small portions of the opposing side of the luminaire when accessing these unseen areas. This is however uncommon and properly adjusting the spatial scaling of the luminaire fixes the issue. For this reason, octrees, which avoid the process of querying the MLPs, do not suffer from this issue. We highlight these limitations by rendering up-close; nevertheless, as we can see in Figure 6.5, in normal scenes these are hardly perceivable. Also, rendering at higher resolutions mitigates the issue as well.

Our implementation is not limited to HDR luminaires, but to any pre-trained neural volumetric representation, as long as they are exported to a graph and their weights frozen. Figure 6.7 shows the *Lego* dataset from [21] rendered inside a Cornell Box-style scene in Mitsuba. As you can see, we can also combine our neural luminaires with other traditional light sources.

In Figure 6.8, we can observe that our implementation performs well when tested at different levels of detail (LOD); rendering the volume at different camera distances provides consistent results, but it suffers a bit from aliasing. Recent work has successfully reduced NeRF's aliasing issue [72]; however their approach of rendering using conical frustums instead of rays would not be suitable for a path tracer.

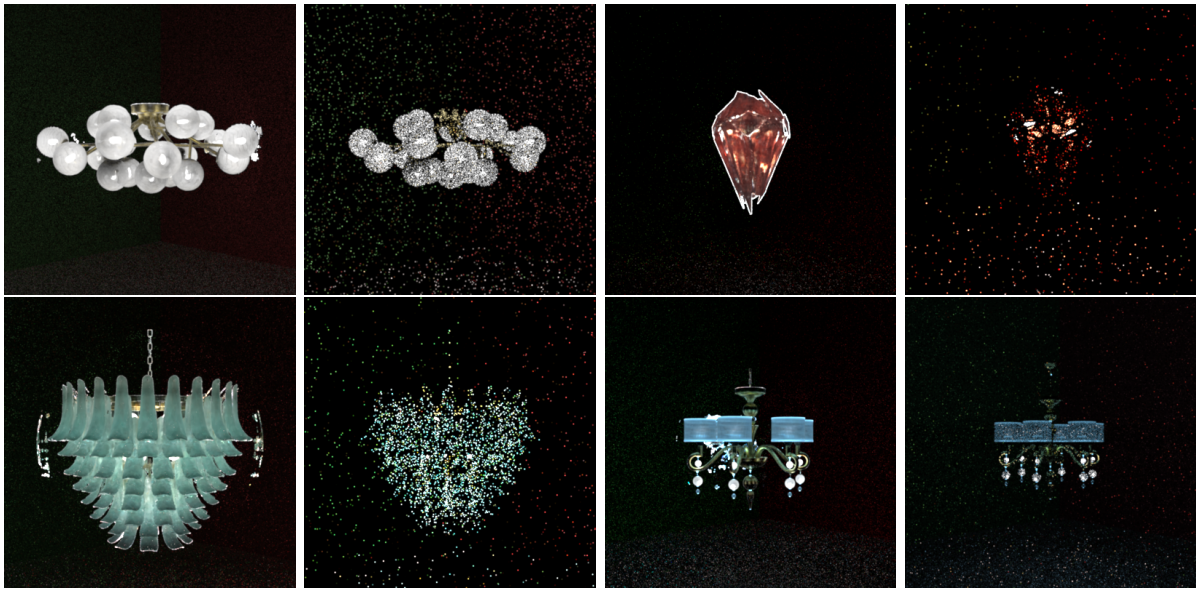

Figure 6.6: Mitsuba renders for all 4 of our datasets in a Cornell Box-style scene. For every pair, left is our method; right, the same scene rendered with a traditional path tracer and the original geometry. Both scenes rendered with 16 samples per pixel.

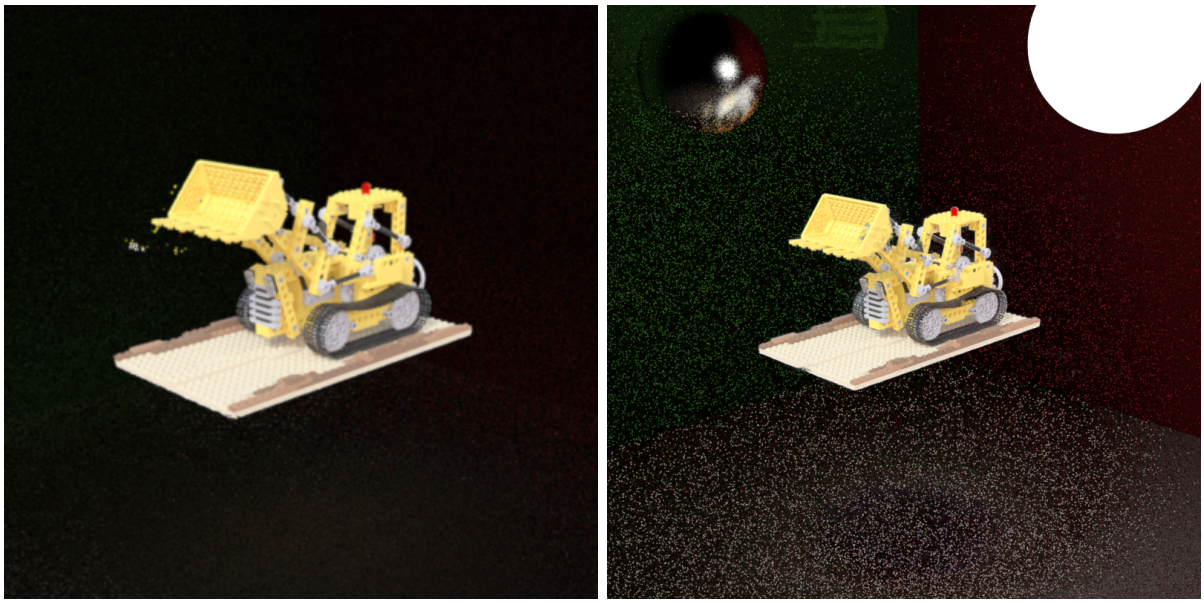

Figure 6.7: The Lego scene rendered in Mitsuba, within a Cornell Box-style scene. The model was originally trained in LDR. On the right, the same model has been rendered along with another light source. Of course, with the Lego being modelled as a light source itself, it does not interact with the other one.

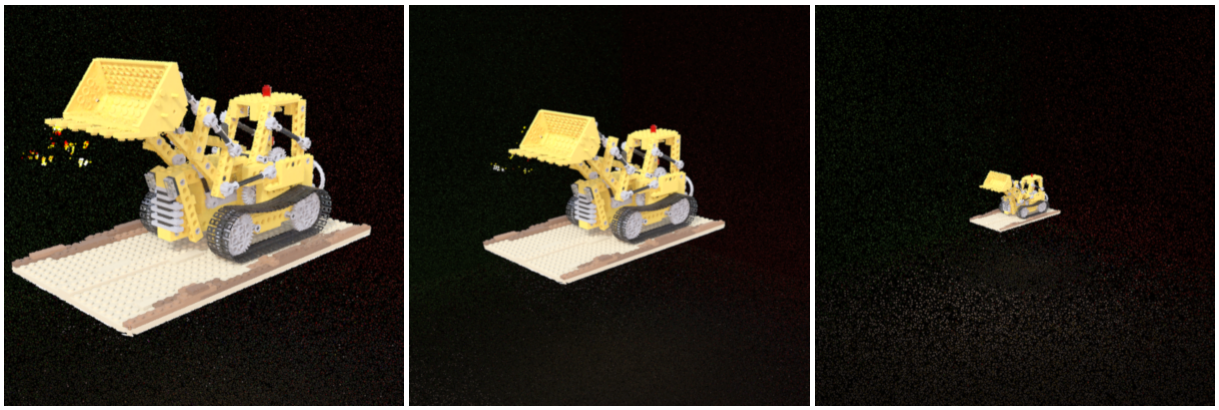

Figure 6.8: Lego model rendered at different distances.

Finally, we showcase the potential of our method with a realistic bedroom scene [**?**] in Figure 6.9. Artifacts on the sides of the luminaire can be easily eliminated by correctly adjusting the bounding box scaling; however, we lacked the time to repeat the renderings. In any case, we demonstrate high quality renderings of realistic scenes, with very low amounts of noise for the samples used. This demonstrates the enormous potential using neural light fields currently has in traditional rendering pipelines.

Figure 6.9: We showcase the potential of our method rendering a realistic bedroom scene with one of our luminaires, (*dallas*). Top used 128 samples per pixel and extra light sources in the windows, while bottom used only 2 samples per pixel and no extra light sources. A third, cleaner image without light sources can be found in our Drive Folder.

7. Future Work and Conclusions

7.1 Future work

While it's only the beginning for neural radiance fields in traditional rendering engines, we have already demonstrated that not only it is possible to integrate them, but rather to obtain high levels of quality. Ultimately, what will make this a more appealing proposal is via accelerating the querying process of these luminaires. Throughout this work we have suggested several paths to accelerate the rendering process of these luminaires within Mitsuba: from Eigen-based MLPs, through TensorRT batch inference and finally the proposed octree-based data structure for the luminaire.

We also believe in the potential of developing novel sampling techniques that take advantage of implicit light field representations: as we do not need to path trace through complicated geometries, and we already know how light flows out of the luminaire, we can use this knowledge to direct samples accordingly.

Another avenue for future work is modelling the light transport within the luminaire. While our method is able to capture the radiance field of the luminaire, it does not, for example, take into account the effect other light sources in the scene have on the appearance and emitted radiance of our luminaire: that is, light coming from the scene reflecting or refracting from/through the luminaire. This is different from current relighting methods [84], where diffuse and absorbing-only surfaces are assumed, and no light transport within the object is explicitly modelled. It remains an interesting line of future work to obtain neural implicit representations that not only model the radiance field in a given scenario (i.e. void in our case), but rather be able to estimate properties of the learned luminaire, so that it is possible to blend them in a more physically accurate way, and even edit these properties without having to re-train the luminaire.

Regarding our transmittance model, we reckon that for geometrically more complex luminaires, composed of mixed correlated and uncorrelated media, this model could also be replaced with a parametric non-exponential model [23, 33, 22], capable of fitting both. We did not have a rendered dataset that could benefit from such a model so it remains as future work to test it.

7.2 Conclusions

In this work we have presented a method to successfully extend NeRF to work in the HDR domain, demonstrating its good performance in the context of rendering complex luminaires.

Furthermore, we have successfully integrated our neural luminaires into a traditional rendering engine, opening many avenues for future work to dramatically reduce rendering times and variance, developing novel rendering and sampling techniques that take advantage of implicit light field representations. We believe our solution shows great promise and should ignite interest in combining state-of-the-art computer vision techniques such as these neural light fields with traditional physically-based rendering.

Bibliography

[1] Per H. Christensen and Wojciech Jarosz. The path to path-traced movies. *Found. Trends. Comput. Graph. Vis.*, 10(2):103–175, oct 2016.

[2] Eric Veach. *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, Stanford, CA, USA, 1998. AAI9837162.

[3] Sameer Agarwal, Ravi Ramamoorthi, Serge Belongie, and Henrik Wann Jensen. Structured importance sampling of environment maps. In *ACM SIGGRAPH 2003 Papers*, pages 605–612. 2003.

[4] Ibón Guillén, Carlos Ureña, Alan King, Marcos Fajardo, Iliyan Georgiev, Jorge López-Moreno, and Adrian Jarabo. Area-preserving parameterizations for spherical ellipses. In *Computer Graphics Forum*, volume 36, pages 179–187. Wiley Online Library, 2017.

[5] Christoph Peters. Brdf importance sampling for polygonal lights. *ACM Transactions on Graphics (TOG)*, 40(4):1–14, 2021.

[6] 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):1–8, 2016.

[7] 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):1–13, 2012.

[8] Jaroslav Křivánek, Christophe Chevallier, Vladimir Koylazov, Ondřej Karlíik, Henrik Wann Jensen, and Thomas Ludwig. Realistic rendering in architecture and product visualization. In *ACM SIGGRAPH 2018 Courses*, SIGGRAPH '18, New York, NY, USA, 2018. Association for Computing Machinery.

[9] Edgar Velázquez-Armendáriz, Zhao Dong, Bruce Walter, and Donald P Greenberg. Complex luminaires: Illumination and appearance rendering. *ACM Transactions on Graphics (TOG)*, 34(3):1–15, 2015.

[10] Junqiu Zhu, Yaoyi Bai, Zilin Xu, Steve Bako, Edgar Velázquez-Armendáriz, Lu Wang, Pradeep Sen, Miloš Hašan, and Ling-Qi Yan. Neural complex luminaires: representation and rendering. *ACM Transactions on Graphics (TOG)*, 40(4):1–12, 2021.

[11] Ayush Tewari, Ohad Fried, Justus Thies, Vincent Sitzmann, Stephen Lombardi, Kalyan Sunkavalli, Ricardo Martin-Brualla, Tomas Simon, Jason Saragih, Matthias Nießner, Rohit Pandey, Sean Fanello, Gordon Wetzstein, Jun-Yan Zhu, Christian Theobalt, Maneesh Agrawala, Eli Shechtman, Dan B Goldman, and Michael Zollhöfer. State of the Art on Neural Rendering. *arXiv e-prints*, page arXiv:2004.03805, April 2020.

[12] NVIDIA. Nvidia dlss 2.0: A big leap in ai rendering, Mar 2020.

[13] Steven G. Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, and Martin Stich. Optix: A general purpose ray tracing engine. *ACM Trans. Graph.*, 29(4), jul 2010.

[14] Thomas Müller, Brian Mcwilliams, Fabrice Rousselle, Markus Gross, and Jan Novák. Neural importance sampling. *ACM Trans. Graph.*, 38(5), oct 2019.

[15] Thomas Müller, Fabrice Rousselle, Alexander Keller, and Jan Novák. Neural control variates. *ACM Trans. Graph.*, 39(6), nov 2020.

[16] Delio Vicini, Vladlen Koltun, and Wenzel Jakob. A learned shape-adaptive subsurface scattering model. *ACM Trans. Graph.*, 38(4), jul 2019.

[17] Gilles Rainer, Wenzel Jakob, Abhijeet Ghosh, and Tim Weyrich. Neural btf compression and interpolation. *Computer Graphics Forum (Proceedings of Eurographics)*, 38(2), March 2019.

[18] Yu Guo, Cameron Smith, Miloš Hašan, Kalyan Sunkavalli, and Shuang Zhao. MaterialGAN: Reflectance Capture using a Generative SVBRDF Model. *arXiv e-prints*, page arXiv:2010.00114, September 2020.

[19] Alejandro Sztrajman, Gilles Rainer, Tobias Ritschel, and Tim Weyrich. Neural BRDF Representation and Importance Sampling. *arXiv e-prints*, page arXiv:2102.05963, February 2021.

[20] Simon Kallweit, Thomas Müller, Brian Mcwilliams, Markus Gross, and Jan Novák. Deep scattering: Rendering atmospheric clouds with radiance-predicting neural networks. *ACM Trans. Graph.*, 36(6), nov 2017.

[21] 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.

[22] Delio Vicini, Wenzel Jakob, and Anton Kaplanyan. A non-exponential transmittance model for volumetric scene representations. *ACM Transactions on Graphics (TOG)*, 40(4):1–16, 2021.

[23] Adrian Jarabo, Carlos Aliaga, and Diego Gutierrez. A radiative transfer framework for spatially-correlated materials. *ACM Transactions on Graphics (TOG)*, 37(4):1–13, 2018.

[24] Guillaume Loubet and Fabrice Neyret. Hybrid mesh-volume lods for all-scale pre-filtering of complex 3d assets. *Computer Graphics Forum*, 36(2):431–442, 2017.

[25] Fabrice Neyret. Modeling, animating, and rendering complex scenes using volumetric textures. *IEEE Trans. Vis. Comput. Graph.*, 4(1):55–70, 1998.

[26] Carlos Aliaga, Carlos Castillo, Diego Gutiérrez, Miguel A. Otaduy, Jorge Lopez-Moreno, and Adrián Jarabo. An appearance model for textile fibers. *Computer Graphics Forum (Proc. of the Eurographics Symposium on Rendering)*, 36(4), 2017.

[27] Pramook Khungurn, Daniel Schroeder, Shuang Zhao, Kavita Bala, and Steve Marschner. Matching real fabrics with micro-appearance models. *ACM Trans. Graph.*, 35(1), dec 2016.

[28] Kai Schröder, Reinhard Klein, and Arno Zinke. A Volumetric Approach to Predictive Rendering of Fabrics. *Computer Graphics Forum*, 2011.

[29] Shuang Zhao, Wenzel Jakob, Steve Marschner, and Kavita Bala. Building volumetric appearance models of fabric using micro ct imaging. *ACM Trans. Graph.*, 30(4), jul 2011.

[30] Johannes Meng, Marios Papas, Ralf Habel, Carsten Dachsbacher, Steve Marschner, Markus Gross, and Wojciech Jarosz. Multi-scale modeling and rendering of granular materials. *ACM Trans. Graph.*, 34(4), jul 2015.

[31] Jonathan T. Moon, Bruce Walter, and Stephen R. Marschner. Rendering discrete random media using precomputed scattering solutions. In *Proceedings of the 18th Eurographics Conference on Rendering Techniques*, EGSR'07, page 231–242, Goslar, DEU, 2007. Eurographics Association.

[32] 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.

[33] Benedikt Bitterli, Srinath Ravichandran, Thomas Müller, Magnus Wrenninge, Jan Novák, Steve Marschner, and Wojciech Jarosz. A radiative transfer framework for non-exponential media. *ACM Transactions on Graphics*, 37(6):225, 2018.

[34] Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. Neural volumes: Learning dynamic renderable volumes from images. *ACM Trans. Graph.*, 38(4), jul 2019.

[35] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene Representation Networks: Continuous 3D-Structure-Aware Neural Scene Representations. *arXiv e-prints*, page arXiv:1906.01618, June 2019.

[36] Ben Mildenhall, Pratul P. Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. *ACM Transactions on Graphics (TOG)*, 2019.

[37] Edgar Sucar, Shikun Liu, Joseph Ortiz, and Andrew Davison. iMAP: Implicit mapping and positioning in real-time. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2021.

[38] Jeffrey Ichnowski, Yahav Avigal, Justin Kerr, and Ken Goldberg. Dex-NeRF: Using a Neural Radiance Field to Grasp Transparent Objects. *arXiv e-prints*, page arXiv:2110.14217, October 2021.

[39] Ricardo Martin-Brualla, Noha Radwan, Mehdi S. M. Sajjadi, Jonathan T. Barron, Alexey Dosovitskiy, and Daniel Duckworth. NeRF in the Wild: Neural Radiance Fields for Unconstrained Photo Collections. In *CVPR*, 2021.

[40] Yi Wei, Shaohui Liu, Yongming Rao, Wang Zhao, Jiwen Lu, and Jie Zhou. Nerfingmvs: Guided optimization of neural radiance fields for indoor multi-view stereo. In *ICCV*, 2021.

[41] Lin Yen-Chen, Pete Florence, Jonathan T. Barron, Alberto Rodriguez, Phillip Isola, and Tsung-Yi Lin. INeRF: Inverting Neural Radiance Fields for Pose Estimation. *arXiv e-prints*, page arXiv:2012.05877, December 2020.

[42] Chen-Hsuan Lin, Wei-Chiu Ma, Antonio Torralba, and Simon Lucey. BARF: Bundle-Adjusting Neural Radiance Fields. *arXiv e-prints*, page arXiv:2104.06405, April 2021.

[43] Yudong Guo, Keyu Chen, Sen Liang, Yongjin Liu, Hujun Bao, and Juyong Zhang. Ad-nerf: Audio driven neural radiance fields for talking head synthesis. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[44] Keunhong Park, Utkarsh Sinha, Peter Hedman, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Ricardo Martin-Brualla, and Steven M. Seitz. Hypernerf: A higher-dimensional representation for topologically varying neural radiance fields. *ACM Trans. Graph.*, 40(6), dec 2021.

[45] Tianye Li, Mira Slavcheva, Michael Zollhoefer, Simon Green, Christoph Lassner, Changil Kim, Tanner Schmidt, Steven Lovegrove, Michael Goesele, and Zhaoyang Lv. Neural 3D Video Synthesis. *arXiv e-prints*, page arXiv:2103.02597, March 2021.

[46] Keunhong Park, Utkarsh Sinha, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Steven M. Seitz, and Ricardo Martin-Brualla. Nerfies: Deformable neural radiance fields. *ICCV*, 2021.

[47] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. PlenOctrees for real-time rendering of neural radiance fields. In *ICCV*, 2021.

[48] Stephan J. Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin. FastNeRF: High-Fidelity Neural Rendering at 200FPS. *arXiv e-prints*, page arXiv:2103.10380, March 2021.

[49] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. KiloNeRF: Speeding up Neural Radiance Fields with Thousands of Tiny MLPs. *arXiv e-prints*, page arXiv:2103.13744, March 2021.

[50] Thomas Neff, Pascal Stadlbauer, Mathias Parger, Andreas Kurz, Joerg H. Mueller, Chakravarty R. Alla Chaitanya, Anton Kaplanyan, and Markus Steinberger. DONeRF: Towards Real-Time Rendering of Compact Neural Radiance Fields using Depth Oracle Networks. *arXiv e-prints*, page arXiv:2103.03231, March 2021.

[51] Weiping Song, Chence Shi, Zhiping Xiao, Zhijian Duan, Yewen Xu, Ming Zhang, and Jian Tang. AutoInt: Automatic Feature Interaction Learning via Self-Attentive Neural Networks. *arXiv e-prints*, page arXiv:1810.11921, October 2018.

[52] Katja Schwarz, Yiyi Liao, Michael Niemeyer, and Andreas Geiger. GRAF: Generative Radiance Fields for 3D-Aware Image Synthesis. *arXiv e-prints*, page arXiv:2007.02442, July 2020.

[53] Alex Trevithick and Bo Yang. {GRF}: Learning a general radiance field for 3d scene representation and rendering, 2021.

[54] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelNeRF: Neural radiance fields from one or few images. In *CVPR*, 2021.

[55] Eric R. Chan, Marco Monteiro, Petr Kellnhofer, Jiajun Wu, and Gordon Wetzstein. pi-GAN: Periodic Implicit Generative Adversarial Networks for 3D-Aware Image Synthesis. *arXiv e-prints*, page arXiv:2012.00926, December 2020.

[56] Qianqian Wang, Zhicheng Wang, Kyle Genova, Pratul Srinivasan, Howard Zhou, Jonathan T. Barron, Ricardo Martin-Brualla, Noah Snavely, and Thomas Funkhouser. Ibrnet: Learning multi-view image-based rendering. In *CVPR*, 2021.

[57] Michael Niemeyer and Andreas Geiger. CAMPARI: Camera-Aware Decomposed Generative Neural Radiance Fields. *arXiv e-prints*, page arXiv:2103.17269, March 2021.

[58] Ian Ashdown. Near-field photometry: Measuring and modeling complex 3-d light sources. In *SIGGRAPH 1995*, 1995.

[59] P.Y. Ngai. On near-field photometry. *Journal of the Illuminating Engineering Society*, 16(2):129–136, 1987.

[60] Wolfgang Heidrich, Jan Kautz, P. Slusallek, and Hans-Peter Seidel. Canned lightsources. In *Rendering Techniques*, 1998.

[61] Julius Muschaweck. What's in a ray set: moving towards a unified ray set format. In Tina E. Kidger and Stuart David, editors, *Illumination Optics II*, volume 8170 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, page 81700N, October 2011.

[62] Ian Ashdown and Ron Rykowski. Making near-field photometry practical. *Journal of the Illuminating Engineering Society*, 27:67–79, 07 1998.

[63] Albert Mas, Ignacio Martín, and Gustavo Patow. Compression and importance sampling of near-field light sources. *Comput. Graph. Forum*, 27:2013–2027, 12 2008.

[64] Heqi Lu, Romain Pacanowski, and Xavier Granier. Position-dependent importance sampling of light field luminaires. *IEEE Transactions on Visualization and Computer Graphics*, 21:241–251, 02 2015.

[65] James T. Kajiya. The rendering equation. *SIGGRAPH Comput. Graph.*, 20(4):143–150, aug 1986.

[66] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. *SIGGRAPH Comput. Graph.*, 18(3):137–145, jan 1984.

[67] 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.

[68] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred A. Hamprecht, Yoshua Bengio, and Aaron Courville. On the Spectral Bias of Neural Networks. *arXiv e-prints*, page arXiv:1806.08734, June 2018.

[69] Gaochang Wu, Belen Masia, Adrian Jarabo, Yuchen Zhang, Liangyong Wang, Qionghai Dai, Tianyou Chai, and Yebin Liu. Light field image processing: An overview. *IEEE Journal of Selected Topics in Signal Processing*, 11(7):926–954, 2017.

[70] Subrahmanyan Chandrasekhar. *Radiative transfer*. Courier Corporation, 1960.

[71] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. *arXiv e-prints*, page arXiv:1901.05103, January 2019.

[72] 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, 2021.

[73] Jaakko Lehtinen, Jacob Munkberg, Jon Hasselgren, Samuli Laine, Tero Karras, Miika Aittala, and Timo Aila. Noise2Noise: Learning Image Restoration without Clean Data. *arXiv e-prints*, page arXiv:1803.04189, March 2018.

[74] 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*, SIGGRAPH '02, page 267–276, New York, NY, USA, 2002. Association for Computing Machinery.

[75] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[76] Jonathan Heek, Anselm Levskaya, Avital Oliver, Marvin Ritter, Bertrand Rondepierre, Andreas Steiner, and Marc van Zee. Flax: A neural network library and ecosystem for JAX, 2020.

[77] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv e-prints*, page arXiv:1412.6980, December 2014.

[78] Wenzel Jakob. Mitsuba renderer, 2010.

[79] Jan Novák, Iliyan Georgiev, Johannes Hanika, and Wojciech Jarosz. Monte carlo methods for volumetric light transport simulation. In *Computer Graphics Forum*, volume 37, pages 551–576. Wiley Online Library, 2018.

[80] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[81] CGTrader, 3d model store.

[82] Turbosquid, 3d models for professionals.

[83] 3d models 3ddd.

[84] Kaiwen Guo, Peter Lincoln, Philip Davidson, Jay Busch, Xueming Yu, Matt Whalen, Geoff Harvey, Sergio Orts-Escolano, Rohit Pandey, Jason Dourgarian, Danhang Tang, Anastasia Tkach, Adarsh Kowdle, Emily Cooper, Mingsong Dou, Sean Fanello, Graham Fyffe, Christoph Rhemann, Jonathan Taylor, Paul Debevec, and Shahram Izadi. The relightables: Volumetric performance capture of humans with realistic relighting. *ACM Trans. Graph.*, 38(6), nov 2019.

Appendix A. Project management, tools, other tests and further implementation details

A.1 Project management

Table A.1 shows a detailed breakdown of time spent in each section of the project.

Project Task	**Estimated Time Spent**
Literature review: neural implicit geometry, importance sampling, complex luminaires, neural rendering, neural radiance fields	85h
Datasets: modeling, generation and transformation code	50h
NeRF-SH: varied tests, HDR extension and significantly expanded evaluation suite	350h
Ablation studies, model performance evaluations	40h
NeRF-SH to Octree conversion	40h
Mitsuba NeRF expansion code	140h
Mitsuba tests	60h
Thesis Document	70h
TOTAL	**835h**

Table A.1: Project management

A.2 Tools

Throughout this project I used the following tools:

1. **Python**. General scripting, test development, and language used throughout most of the project.

2. **C++**. Used in Mitsuba in order to extend its functionality.

3. **Mitsuba**. Academic physically based renderer we extended for NeRF inference in our project.

4. **JAX**. GPU Numpy and ML framework.

5. **FLAX**. ML API for JAX.

6. **Tensorflow**. We used its C API for model inference within Mitsuba.

7. **Github**. Git tool used.

8. **Pytorch**. NeRF-SH to Octree conversion.

9. **WSL2**. Training and rendering in Linux within a Windows environment.

10. **SSH**. Deploying workloads on the Lab workstations.

11. **LaTeX**. Writing this document.

A.3 Data generation

In order to obtain an implicit representation of our luminaires, we first need some fine-quality renders. We modelled the different luminaires using Blender, using models and parts both original and taken from various sources [81, 82, 83]. For the rendering part, we used Mitsuba. Mitsuba is an open-source, research-oriented rendering engine with plenty of features and parallelization in CPU. The fully accessible source code makes it suitable to our application, where we look to alter the how light sources are defined and queried inside Mitsuba.

A.3.1 Blender and Scene Composition

We use Blender [**?**] to compose our luminaires. We designed a variety of different luminaires with different attributes to test our method. These attributes can be summed up in low-high frequency geometry, low-high frequency appearance, intensity of the specular highlights, dynamic range of the luminaire and low-high amounts of dark/non-emissive areas. In Section 6.1 we include an analysis of our selected scenes and their properties.

In order to facilitate camera placement for the bulk generation of renders later on, the geometrical center of the luminaires was placed in the center of the scene. We used Blender 2.79 in order to take advantage of the Mitsuba scene exporter plugin. This plugin allows us to model our scenes in Blender and conveniently export the result into a Blender-style format, which uses a combination of .obj elements for the geometry and a HTML-style format for the scene description, which includes camera definition, geometry placement, light sources definition, material properties, etc.

A.3.2 Generation of the set of cameras

Traditional NeRF, which is the base of our approach, fails when its cameras are placed at very different distances from the target geometry [72]. More recent works, such as Mip-NeRF [72], account for this fact and enable a correct, anti-aliased representation of the target geometry at different distances and levels of detail, but it is not suitable to our application due to how rendering is carried out, using conical frustums instead of individual rays, which we need to be able to integrate properly into an overarching rendering engine. Therefore, we place all our cameras in a sphere surrounding our luminaire, at a distance which varies depending on the size of the luminaire.

It has been thoroughly demonstrated that NeRF representations fail when cameras are placed too far from each other, as the ability of the neural network to interpolate rapidly deteriorates [54]. In order to avoid this issue, we generate camera positions using a Halton Low-Discrepancy Sequence, a pseudo-random number generation approach which is also used in path tracing samplers in rendering engines [**?**]. Once we generate a 2D sequence of the required size (squared number of cameras), we geometrically warp it to the surface of a sphere, which yields the final unitary camera positions in XYZ coordinates. If required, we can scale them by any float number to account for luminaires of different sizes.

A.3.3 Rendering

In order to extend our testing capabilities during training and explore possible improvements to the network based on easily accessible ground truth data (given that our work full utilizes synthesized luminaires) we rendered 3 different representations of every luminaire: its appearance, stored as an HDR linear RGB image; its normal map; and its depth map (distance to the camera of every pixel in the luminaire. Every render carries its own alpha channel as well, and everything is stored as a multi-channel EXR file that we will later decompose into its different elements to adapt our dataset to the NeRF/NSVF dataset style and simplify our code.

Rendering properties such as samples per pixel, integrator used, resolution, clipping planes distance and camera properties depended on the characteristics of each luminaire or the test objective of the dataset. In general, luminaires featuring volumetric components, such as tinted glasses, required a volumetric path tracer; while more difficult luminaires, featuring very small light sources, like single coils inside various layers of glass, required a bidirectional path tracer. Otherwise, we used a standard path tracer. Similarly, samples per pixel depended on luminaire complexity, while other camera characteristics were defined by luminaire size or other specific objectives of the dataset.

Camera transformation matrix was computed using a look-at matrix scheme, which is easily defined by the generated XYZ positions in space of our cameras, an up vector (a constant vector 0,0,1) and a target XYZ position (fixed to 0,0,0)

In order to launch our complete set of renders at once and change the HTML scene file for each one to account for the varying camera positions, we use a Python interface to the Mitsuba rendering API integrated in Mitsuba 0.6.

Rendering was performed on a workstation featuring 2x Intel Xeon Gold (72 total threads),

and rendering times varied drastically depending on the number of available threads at any given time, and luminaire complexity, but most datasets took around 2 or 3 days to generate.

A.3.4 Adapting to NeRF/NSVF dataset conventions

Finally, we have our renders completed and we only need to adapt the dataset to the NeRF/NSVF format style. This allows us to recycle a good amount of code from the original NeRF implementation, which we later on expanded to accomodate all our changes. We implemented a Python script to separate the individual channels of our multi-channel EXR into RGB, normal and depth maps, generate the camera transformation matrices and store them in txt files, split between train, validation and test, normalize normal maps and depth maps, the latter using the near-far clipping plane distances used during rendering, and store each map, including both an EXR linear RGB map and PNG sRGB one, normal PNG and depth PNG into their own folders using the NSVF naming convention (files starting with 0_, 1_, and 2_ correspond to training, validation and test splits, respectively).

A.4 Other explored activation functions

A.4.1 Exponential activation function

The most straightforward way of unbounding the RGB output was using an exponential activation function, going from (0,1) to (0,inf). The function can be seen in Equation A.1. However, we found that in scenes where geometry had a large amount of black pixels, or exposed light sources with radiance peaks at specific values, the activation function was largely unstable.

$$color = e^x \tag{A.1}$$

A.4.2 Softly-bounded log-sigmoid activation function

In order to overcome the shortcomings of exponential activations, we tested with functions that would slightly bound the output. We did this by extending the range of the sigmoid function with a $log(x + 1)$ function, which resulted in Equation A.2. One of the advantages of working with synthetic luminaires is that the maximum radiance that they output is known to us. By selecting the ϵ parameter, we could set the minimum and maximum range of our outputs: for example, for the *portica* dataset, with a maximum radiance of 12, we selected $= 10^{-6}$, which results in an output range of $(10^{-6}, 13.8155)$. This activation function proved much more stable; however, it does not solve by itself the problem of extending to HDR: many of our luminaires have extreme dynamic ranges, and the most radiant areas in the luminaire dominate the error gradients, making an even learning of the radiance field difficult, with balance completely skewed towards exposed light sources and susceptible to high energy rendering noise. We further develop this issue when talking about our loss functions in Section 4.4.

$$color = -log(1 - \frac{1}{1 + e^{-x}} + \epsilon) \tag{A.2}$$