Neural representations for object capture and rendering

Meghna Asthana

PhD

University of York

Computer Science

April 2023

Abstract

Photometric stereo is a classical computer vision problem with applications ranging from gaming, VR/AR avatars to movie visual effects which requires a faithful reconstruction of an object in a new space, and thus, there is a need to thoroughly understand the object's visual properties. With the advent of Neural Radiance Fields (NeRFs) in the early 2020s, we witnessed the incredible photorealism provided by the method and its potential beyond. However, original NeRFs do not provide any information about the material and lighting of the objects in focus. Therefore, we propose to tackle the multiview photometric stereo problem using an extension of NeRFs. We provide three novel contributions through this work. First, the Relightable NeRF model, an extension of the original NeRF, where appearance is conditioned on a point light source direction. It provides two use cases - it is able to learn from varying lighting and relight under arbitrary conditions. Second, the Neural BRDF Fields which extends the relightable NeRF by introducing explicit models for surface reflectance and shadowing. The parameters of the BRDF are learnable as a neural field, enabling spatially varying reflectance. The local surface normal direction as another neural field is learned as well. We experiment with both a fixed BRDF (Lambertian) and a learnable (i.e. neural) reflectance model which guarantees a realistic BRDF by tieing the neural network to BRDF physical properties. In addition, it learns local shadowing as a function of light source direction enabling the reconstruction of cast shadows. Finally, the Neural Implicit Fields for Merging Monocular Photometric Stereo switches from NeRF's volume density function to a signed distance function representation. This provides a straightforward means to compute the surface normal direction and, thus, ties normal-based losses directly to the geometry. We use this representation to address the problem of merging the output of monocular photometric stereo methods into a single unified model: a neural SDF and a neural field capturing diffuse albedo from which we can extract a textured mesh.

Acknowledgement

I wish to express my deepest gratitude to my supervisor, Dr William Smith for his excellent guidance, patience and for providing me with a brilliant atmosphere for doing research. I would like to especially thank my Thesis Advisory Panel member Dr Giuseppe Claudio Guarnera for his excellent input and support during my PhD. I would like to thank Dr Paulina Lewinska for always providing me with the best advice, Dr James Walker for linking me to the best contacts in industry and academia and everyone at the Vision, Graphics and Learning for their support during my journey.

I would also like to extend my gratitude to the academics at the Department of Computer Science, University of York who provided me with endless opportunities to enhance my PhD experience by supervising, teaching and outreach programs. I would like to thank my lifelines, Anvita and Bimod whose support and willingness to provide me with the best suggestions have helped me complete my PhD. I am thankful for my baby bunny, Lily for bringing light to my life during the darkest of times. Finally, I would like to thank my parents whose constant support and encouragement helped me through difficult times.

"Carrying on the academic legacy, eternally grateful..."

Declaration of Authorship

I declare that this thesis is a presentation of original work and I am the sole author. This work has not previously been presented for an award at this, or any other, University. All sources are acknowledged as References. The research in this thesis has resulted in the following paper (in collaboration with my supervisor Dr William Smith), corresponding to Chapter 5:

- Asthana M, Smith W, Huber P. Neural apparent BRDF fields for multiview photometric stereo. In *Proceedings of the 19th ACM SIGGRAPH European Conference on Visual Media Production*, 2022, Dec 1 (pp. 1-10).

Signed, April 24, 2023 Meghna Asthana

Contents

List of Figures

List of Symbols

In the order of appearance.

Chapter 2: Prerequisites	
Symbol	**Description**
E	Measured Intensity from Lambertian Reflectance
$\mathbf{n}$	surface normal
$\mathbf{s}$	incident illumination
I	intensity of light
$\mathbf{a}$	albedo
$\mathbf{a}_d$	diffused albedo
$\mathbf{a}_s$	specular albedo
f_r	BRDF
dL	target radiance
dE_i	incident irradiance
(θ_i, ϕ_i)	incident direction
(θ_o, ϕ_o)	reflected direction
ω_i	incident ray
ω_o	reflected ray
L_i	Intensity
$\text{NCC}(u, v)$	photometric similarity
X	matrix of co-ordinates in 3D space
$\mathbf{C}$	Camera center
D	number of depth planes

Chapter 3: Literature Review	
Symbol	**Description**
Δ_u	maximum camera sampling interval
K_x	highest spatial frequency
B_x	continuous light field
Δ_x	camera spatial resolution
W	image width
H	image height
$\mathbf{x}$	3D location
(θ, ϕ)	2D viewing direction
$\mathbf{c}$	emitted colour
F_Θ	Multilayer Perceptron
Θ	network weights
$\sigma(\mathbf{x})$	volume density
$\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$	ray from camera with origin $\mathbf{o}$ and direction $\mathbf{d}$
$C(\mathbf{r})$	expected colour
t_n	near bound
t_f	far bound
$T(t)$	amount of colour accumulated along the ray
N	number of bins for stratified sampling
δ	distance between adjacent samples
α	alpha compositing
γ	sinusoidal mapping $\mathbb{R} \mapsto \mathbb{R}^{2L}$
$\mathcal{L}$	Loss Function
$\mathcal{R}$	set of rays in each batch
$\hat{C}(\mathbf{r})$	ground truth RGB colours
$\hat{C}_c(\mathbf{r})$	predicted RGB colours from the coarse network
$\hat{C}_f(\mathbf{r})$	predicted RGB colours from the fine network

Chapter 3: Literature Review	
Symbol	**Description**
$L(\mathbf{c}, \omega_o)$	simulated light transport
$R(x, \omega_i, \omega_o)$	reflectance function
$\hat{V}_\theta$	environment light visibility at input location
$\hat{D}_\theta$	expected termination depth of input ray
L_i	incident illumination
L_e	radiance due to the light source
L_r	reflected radiance
E	environment map attenuation
τ	tone-mapping operator
$\mathbf{z}_{BRDF}$	BRDF parameters
Γ	Spherical Gaussian mixtures
$\bigtriangledown_x \sigma$	gradient of density field σ
x_{surf}	termination location of ray
S_θ	zero level set of a neural networks
$\phi_s(f(\mathbf{x}))$	probability density function, S-density
$f(\mathbf{x})$	Signed Distance Function
Φ_s	Sigmoid function
$\rho(t)$	opaque density function
BCE	Binary Cross Entropy function

Chapter 4: Relightable NeRF	
Symbol	**Description**
$f_{geo}^{\Theta_{geo}}$	Geometry Network for NeRF
Θ_{geo}	learnable parameters for Geometry Network
$\mathbf{z}$	latent code from Geometry Network
$f_{col}^{\Theta_{col}}$	Colour Network for NeRF
Θ_{col}	learnable parameters for Colour Network
$\gamma(\mathbf{x})$	input location
$\gamma(\mathbf{d})$	input viewing direction

Chapter 5: Neural Apparent BRDF Fields	
Symbol	**Description**
$f_{geo}^{\Theta_{geo}}$	Geometry Network
Θ_{geo}	learnable parameters for Geometry Network
$f_{BRDF}^{\Theta_{BRDF}}$	BRDF Network
Θ_{BRDF}	latent code for BRDF Network
$f_{shad}^{\Theta_{shad}}$	Shadow Network
Θ_{shad}	latent code for Shadow Network
$\mathbf{x}$	3D location
$\mathbf{v}$	view direction
$\mathbf{s}$	lighting direction
$\mathbf{z}$	latent code
$\mathbf{c}$	colour
σ	volume density
$\mathbf{n}$	normals
$\mathbf{a}$	albedo
L^{Θ}	reflected radiance
$\hat{C}(\mathbf{r})$	estimated colour
$C(\mathbf{r})$	ground truth colour
N	number of bins for stratified sampling
$\mathcal{R}$	batch of rays
$\mathcal{F}$	object foreground
$\mathcal{B}$	object background
$\mathcal{L}_{sil}$	silhouette loss
$\mathcal{L}_{app}$	mask appearance loss
$\mathbf{h}$	half way vector
θ_h	half angle
$\hat{\omega}_i$	direction of incident ray
$\hat{\omega}_o$	direction of outgoing ray

Chapter 6: NeUS for Monocular PS	
Symbol	**Description**
$f_{SDF}^{\Theta_{SDF}}$	SDF Network
Θ_{SDF}	learnable parameters for SDF Network
$f_{col}^{\Theta_{col}}$	Colour Network
Θ_{col}	learnable parameters for Colour Network
$f_{NeRF}^{\Theta_{NeRF}}$	NeRF Network
Θ_{NeRF}	learnable parameters for NeRF Network
$f_{Var}^{\Theta_{Var}}$	Variance Network
$\mathbf{x}$	3D location
$\mathbf{z}$	latent code
d	signed distance
$\mathcal{S}$	set of all points with zero SDF
$\mathbf{v}$	view direction
$\mathbf{c}$	colour
ρ	volume density
σ^2	variance
$\mathbf{n}$	normal
$\mathbf{a}$	albedo
$\mathbf{u}$	discrete pixel coordinate in an image
$w(t)$	weight function for NeUS
$\mathcal{L}_{normal}$	normal map loss
$\mathcal{L}_{albedo}$	albedo map loss
$\mathcal{L}_{Eikonal}$	Eikonal loss
$\mathcal{L}_{Mask}$	Mask Loss

Chapter 1

Introduction

Reconstructing 3D geometry from photographs is a classic Computer Vision problem that has occupied researchers for more than 30 years. Its applications range from 3D mapping and navigation to online shopping, 3D printing, computational photography, computer video games, or cultural heritage archival. Only recently, however, have these techniques matured enough to exit the laboratory-controlled environment into the wild, and provide industrial-scale robustness, accuracy and scalability.

A central problem in computer vision is that of inferring the physical geometry and material properties that together explain observed images. In addition to its importance for recognition and robotics, a solution to this open problem would have significant value for computer graphics — the ability to create realistic 3D models from standard photos could democratize 3D content creation and allow anyone to use real-world objects in photography, filmmaking, and game development.

Recovering an object's geometry and material properties from captured images, such that it can be rendered from arbitrary viewpoints under novel lighting conditions, is a longstanding problem within computer vision and graphics. The difficulty of this problem stems from its fundamentally under-constrained nature, and prior work has typically addressed this either by using additional observations such as scanned geometry, known lighting conditions, or images of the object under multiple different lighting conditions or by making restrictive assumptions such as assuming a single material for the entire object or ignoring self-shadowing. Furthermore, a mesh with a fixed texture map like from structure-from-motion will look completely unrealistic when composited into a new scene with lighting different from what the

object was captured in. Thus, relightable is a requirement for photorealism.

Photometric stereo has been an integral part of the computer vision and graphics field of study due to its usefulness in determining the surface orientation at each point in a particular image as well as for determining object points with specific surface orientation. Photometric stereo, first introduced by Woodham [Woodham(1980)], can be defined as a concept where directions of incident illumination between successive images can be varied while maintaining a constant viewing direction. The technique is photometric as it uses the radiance value recorded at a single image location, in successive views, rather than the relative position of displaced features. The most beneficial feature is that it works best for smooth objects without distinct texture details needed for feature-matching based methods. These include structure-from-motion method which recovers both geometry and material properties enabling relighting. Though the problem has attracted sustained attention for the four decades following. However, only very recently attempts have been made to tackle the multiview version of the problem.

Multiview stereo is a popular and well-studied passive 3D reconstruction technique. It is based on the principle that dense pixels which correspond to multiple calibrated images captured from different viewpoints make it possible to estimate the 3D shape of the scene via triangulation. This involves intersecting rays being back-projected from the corresponding pixels. The benefit of this technique is that it provides full coverage of the object as well as applies more photometric constraints like BRDF properties and geometry details. Until very recently, the challenge of multiview photometric stereo was addressed by finding a geometry representation that allowed smooth and efficient optimisation to fit multiview data. However, these classical representations (like meshes) are ill-suited since self-occlusion is discontinuous and hence not differentiable thus, making a volumetric differential technique like NeRF a very lucrative option to address this problem.

The vision and graphics research communities have recently made substantial progress towards the novel view synthesis portion of this goal. The NeRF approach has shown that it is possible to synthesize photorealistic images of scenes by training a simple neural network to map 3D locations in the scene to a continuous field of volume density and colour. Volume rendering is trivially differentiable, so the parameters of a NeRF can be optimized for a single scene by using gradient descent to minimise the difference between renderings of the

NeRF and a set of observed images. The use of NeRF to produce a high-quality geometry estimation for initialization helps break the inherent ambiguities among shape, reflectance, and lighting, thereby allowing us to recover a full 3D model for convincing view synthesis and relighting using just a re-rendering loss, simple spatial smoothness priors for each of these components, and a novel data-driven Bidirectional Reflectance Distribution Function (BRDF) prior.

Though NeRF produces compelling results for view synthesis, it does not provide a solution for relighting. This is because NeRF only models the amount of outgoing light from a location — the fact that this outgoing light is the result of interactions between the incoming light and the material properties of an underlying surface is entirely ignored. NeRF does not decompose appearance at all and doesn't explain the reconstruction in terms of materials and lighting. If we consider the relighting problem as an extension to NeRF, by including a simulation of transport of the scene's light source, the task would become extremely challenging and expensive. This is due to the nature of volume density representation used by NeRF i.e. content could exist at any point within the scene and would require constant querying of the neural network for determining the amount of light reflected by particles at each location. Furthermore, if there are multiple light sources, for instance, global illumination, the querying would increase multi-fold making the training and rendering very slow. Thus, there would be a need for either moving toward classical computer graphics techniques for global illumination which could pair well with NeRF or reduce the complexity of illumination in some way.

Since 2020 there has been an explosion of methods building on top of NeRF, some of which overlap with this thesis and were done concurrently. During our research on the big gaps of knowledge in NeRF's realm, we realised that multiple models have been developed which provide faster inferencing and training, handling deformation and relighting, overall object generalisation and video rendering. However, there were no methods which would specifically tackle the problem of photometric multiview stereo reconstruction with the NeRF technique as a baseline model. There are multiple advantages of incorporating photometric stereo into the NeRF model, for instance, performing photometric stereo data collection provides visuals for a single point under different light which could possibly reveal information about the surface that non-photometric stereo data cannot thereby enriching the dataset. This will, in turn, provide us with far better renderings from the NeRF model. Furthermore,

the ability to modify the lighting condition in the renderings provides us with the unique opportunity to faithfully place the object in any artificial or real scenario by mimicking the ambient light condition which has multiple use cases in high-end gaming and VR/AR applications as explained earlier. Thus, we make distinct, unique contributions in building upon NeRF and neural scene representations more generally in various ways to specifically tackle the multiview photometric stereo problem.

Neural Radiance Fields (NeRFs) [Mildenhall et al.(2020)] use a coordinate-based multi-layer perceptron (MLP) to represent volumetric density and view-dependent radiance for a single scene. A NeRF can be rendered in a differentiable manner, meaning they can be trained from image data alone and they have proven to be an ideal representation for integrating information from multiple views. A trained NeRF is capable of state-of-the-art photorealistic novel view synthesis. Although a much less explored avenue, fitting such neural representations to images also provides a potential route to shape and material estimation.

In practice, a NeRF comprises two MLPs. The first is conditioned only on position and computes density. This can be thought of as capturing the geometry of the scene. The second is additionally conditioned on the view direction. This network effectively learns the convolution of the bidirectional reflectance distribution function (BRDF) at each point in the scene with the lighting environment and non-local effects such as shadowing. In the specific case where lighting is provided by a single point source, it effectively learns a slice of the *apparent BRDF* (i.e. the BRDF combined with non-local effects) for each point in the scene in which lighting is fixed and viewpoint is allowed to vary over the full hemisphere.

1.1 Novel Contributions

In this thesis, we aim to develop this idea further. We focus on reconstructing objects captured from multiple viewpoints and with varying, controlled lighting. We present a variety of techniques, ranging from black box relightable models to models in which surface reflectance, shadowing and lighting are explicitly modelled using neural fields. The work does not aim to explicitly model the physics of reflection but applies some of the restrictions to the neural models which have been derived from physics-based limitations. Specifically, we make the following **novel contributions**:

- **Relightable NeRF:** We begin by proposing an extension to the original NeRF in which appearance is conditioned on a point light source direction. This has two benefits: 1. we are able to learn from and exploit the additional information in images with varying lighting (NeRF makes the assumption of fixed lighting over the dataset), 2. the trained model can be used for relighting under arbitrary conditions, including those not seen in the training set. Our model is entirely black box - we expect the neural field to learn lighting and view dependence from scratch with no guarantee of physical plausibility. To the best of our knowledge, we are the first to learn a point lighting conditioned NeRF on multiview photometric stereo style data.

- **Neural BRDF Fields:** We extend the black box relightable NeRF by introducing explicit models for surface reflectance and shadowing. We make the parameters of the BRDF learnable as a neural field, enabling spatially varying reflectance. We also learn the local surface normal direction as another neural field. We experiment with both a fixed BRDF (Lambertian) and a learnable (i.e. neural) reflectance model which guarantees the satisfaction of two of the three requirements for a physically-valid BRDF. We learn local shadowing as a function of light source direction enabling the reconstruction of cast shadows. Relative to our relightable NeRF model, this improved model enables extrapolation to lighting conditions far from those observed during training. To the best of our knowledge, we are the first to combine learnable neural BRDFs with learnable point light-conditioned shadow fields and regressed normal fields for solving multiview photometric stereo.

- **Neural Implicit Fields for Merging Monocular Photometric Stereo:** The density model used by NeRF and our first two contributions is ill-suited to representing surface-based models. There is also no meaningful way to define a surface normal for non-binary density fields and use the gradient of the density, thus, leading to extremely noisy surface normals. We, therefore, switch to a neural implicit surface representation (signed distance function). This provides a straightforward means to compute the surface normal direction and therefore to tie normal-based losses directly to the geometry. We use this representation to tackle the problem of merging the output of monocular photometric stereo methods (i.e. a normal map and diffuse albedo map per viewpoint) into a single unified model: a neural SDF and a neural field

capturing diffuse albedo from which we can extract a textured mesh. To the best of our knowledge, we are the first to use neural SDF representations to merge monocular photometric stereo output across views into a unified model.

1.2 Thesis Structure

The thesis is divided into seven chapters. The first chapter, *Introduction* gives us a brief understanding and reasons for choosing this topic of research, the novel contributions provided by the research and the structure summary of the thesis. In the second chapter, *Preliminaries* we will first introduce the theoretical concepts of the Lambertian model and the Bidirectional Reflectance Distribution function which are the core components of our work. We also provide detailed information about two datasets used for training all our models - synthetic, Stanford Bunny and multiview photometric stereo, DiLiGeNT-MV.

The third chapter, *Literature Review*, will provide an in-depth explanation of the predecessors of NeRF which include neural 3D representations, view synthesis through image-based rendering and Local Light Field Fusion as well as inner workings of NeRF. In order to fully understand the gap in literature we will provide an exhaustive library of NeRF variants, developed to achieve multiple shortcomings exhibited by original NeRF, as well as a critical analysis of each type of variant.

The fourth chapter, *Relightable NeRF* will provide a detailed step-by-step process of developing our first novel contribution. We will start by introducing the process of preparing a dataset to make it compatible with NeRF and our training setup to check the feasibility of the baseline framework on any data other than ones already provided with the model Following this, we will explain the pipeline enhancement performed in order to achieve Relightable NeRF. We will finally provide multiple comparative and ablation studies to assess the performance of the upgraded model.

The fifth chapter, *Neural Apparent BRDF Fields for Photometric MV-Stereo*, and sixth chapter, *Neural Implicit Fields for Merging Monocular Photometric Stereo*, we will follow a similar storyline as chapter four where we will explain the enhancement performed on the pipeline, a step-by-step implementation process and an ablation study to assess the performance enhancement provided by the upgrades.

In the final chapter, *Conclusion and Future Work* we will summarise our findings and provide an exhaustive list of options which could be used to extend this piece of work.

1.3 Reproducibility

The implementation of many of the methods in this thesis was highly complex, building on top of large code bases. Even reproducing results from already-published papers proved very challenging, with many practical obstacles to overcome. For this reason, to help assist researchers in future and to make our work more reproducible, in each chapter, besides the theory describing our methods, we also present detailed implementation specifics for exactly how we constructed and trained the models we proposed and processed the datasets we work with. We hope that this will provide value to the thesis besides purely theoretical contributions.

1.4 Publications

- Asthana M, Smith W, Huber P. Neural apparent BRDF fields for multiview photometric stereo. In *Proceedings of the 19th ACM SIGGRAPH European Conference on Visual Media Production*, 2022, Dec 1 (pp. 1-10).

Chapter 2

Preliminaries

The chapter will be introducing the core theories essential to understand our work and novel contributions. We will be examining the Bidirectional Reflectance Distribution Function (BRDF) - which would be central for modeling diffuse surfaces - starting with Lambertian reflectance and developing towards more general data-driven BRDFs. We will dive deep into the mathematics behind the phenomena and constraints to be satisfied. Second, we will discuss the process of 3D reconstruction under varying light conditions (Photometric Stereo) and using calibrated overlapping images from different viewpoints (Multiview Stereo). We will also be explaining the mathematical theory which makes image-based modelling and image-based rendering possible. Finally, we will make the reader acquainted with the datasets - Stanford Bunny and DiLiGeNT-MV used for training all our models.

2.1 Lambertian Model

Lambertian reflectance is defined as a material property which distributes the incident energy from an illumination into all view directions equally. It is a simple two-dimensional function with no dependence on the viewing direction which reduces its complexity. Due to such simplifications, it does not allow for the modelling of cast shadows or specularities [Ikeuchi(2021)].

Theory. A scene exhibits Lambertian reflectance [Sanjeev(2014)] when the measured intensity E and can be expressed as a function of surface normal $\mathbf{n}$ and incident illumination direction $\mathbf{s}$:

$$E = I\mathbf{a}\max(\mathbf{n}.\mathbf{s}, 0)$$

where I is the intensity of light and $\mathbf{a}$ is the scene point albedo i.e. the ratio of the reflected and incident energies. Furthermore, Lambertian reflectance is a clamped cosine function which allows the model to consider shadows. An extension of Lambertian reflection called the "diffuse + specular" model is described as:

$$E = I\mathbf{a}_d \max(\mathbf{n.s}, 0) + I\mathbf{a}_s\delta(v - \mathbf{s} - 2\left\|\mathbf{n.s}\right\|\mathbf{n})$$

where $\mathbf{a}_d$ and $\mathbf{a}_s$ are diffused and specular albedos of the scene and δ is the function which defines the viewing, incident and surface normal vectors aligned according to the Law of Reflection. Another extension is the visual appearance of rough diffused Lambertian surfaces, modelled as a Gaussian distribution of the orientations of microstructures within any scene point.

Applications. The variants of Lambertian models find their use in multiple Computer Graphics tasks due to their linearity. For example, if we consider a scene with surface points $i = 1, 2, ...k$ with surface normals $n_i \in \mathbf{N}$ and light-source direction $s_i \in \mathbf{S}$ we can represent the image as **E=S.N** which is linearly separable. Some of the applications include:

1. *Explanation of Ambiguities.* In the equation above, we can introduce any invertible Q as $E = S.Q.Q^{-1}.N$ where it can represent any orthographic viewing or distant illumination.

2. *Scene Reconstruction.* In classical photometric stereo, surface normals can be obtained by $N = (S^{-1}.E)$ with known lighting.

3. *Relighting Scenes.* For Lambertian models, relighting can be performed as a summation of two sets of known illuminations S_1 and S_2 and the resultant image can then be defined as $E = S_1.N + S_2.N = S_3.N$.

2.2 Bidirectional Reflectance Distribution Function BRDF

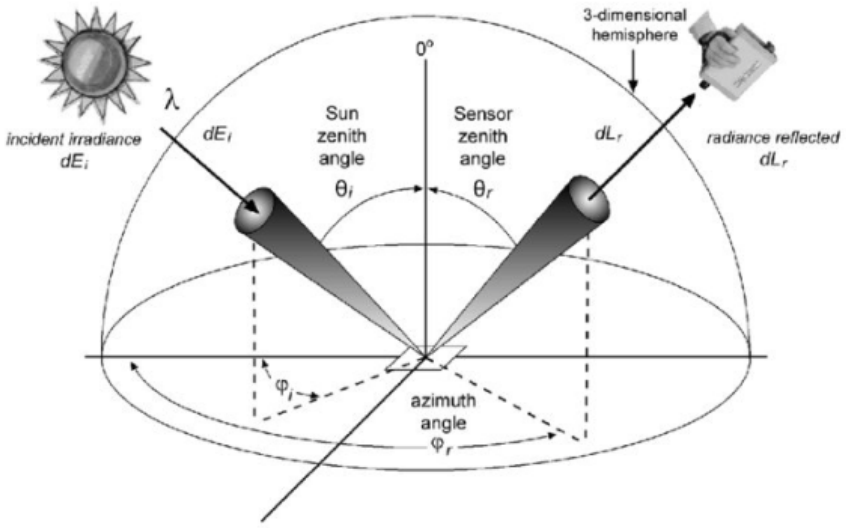


Figure 2.1: BRDF Overview (figure taken from [Westlund et al.(2002)]).

A Bidirectional Reflectance Distribution Function (BRDF) is a theoretical concept that describes how light is reflected at an opaque surface. Formally, BRDF, f_r is defined as the ratio of the target radiance dL reflected in one direction (θ_o, ϕ_o) to the incident irradiance dE_i from direction (θ_i, ϕ_i) as shown in figures 2.1 and 2.2:

$$f_r(\theta_i, \phi_i; \theta_o, \phi_o; \lambda) \approx \frac{dL_r(\theta_i, \phi_i; \theta_o, \phi_o; \lambda)}{dE_i(\theta_i, \phi_i; \lambda)}$$

The equation above explains the distribution of reflected light at a surface. For a given incoming light ray at a point surface, a BRDF approximates the bidirectional sub-surface scattering reflectance distribution function (BSSRDF) but ignores the sub-surface scattering and assumes that the light striking the surface at the point will be reflected by the same point.

Solid Angles. For a given point on a surface, solid angle can be used to refer to some small surface area on the hemisphere and is approximated as the area intercepted by a cone whose apex is at the sphere's centre.

Irradiance and Radiance.

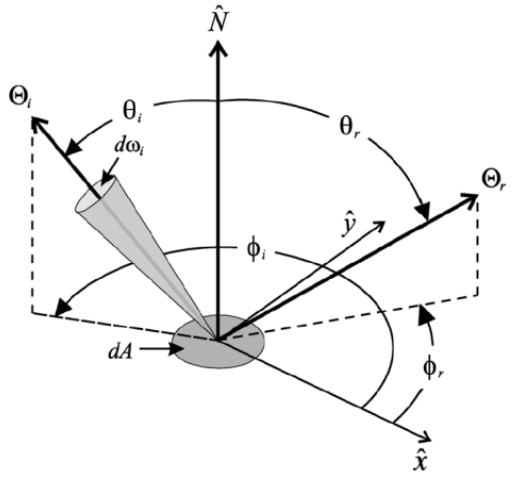


Figure 2.2: Angles and Vectors for BRDF (figure taken from [Westlund et al.(2002)]).

Irradiance, E_i is the rate of incoming energy at a surface point per unit surface area and describes the light arriving at a point from all directions. Whereas, radiance describes the amount of light arriving at a point from a specific direction. The irradiance at a point on the surface can be defined as:

$$\mathrm{E}_i = \int_{all\omega_i} L_i(-\omega_i)\cos\theta_i d\sigma$$

$$= \int_{\phi=0}^{2\pi}\int_{\theta=0}^{\frac{\pi}{2}} L_i(\theta_i,\phi_i)\cos\theta_i \sin\theta_i d\theta_i d\phi_i$$

BRDF Constraints. Every physically based BRDFs require to obey two constraints:

1. Helmholtz Reciprocity Rule states that the BRDF f_r is symmetric relative to ω_i and ω_o.

$$f_r(\omega_i,\omega_o) = f_r(\omega_o,\omega_i)$$

2. Positivity Principle which states that BRDF f_r should remain positive with respect to ω_i and ω_o.

$$f_r(\omega_i,\omega_o) \geq 0$$

3. Law of Conservation of Energy normalises the BRDF f_r and states that the amount

of reflected radiance must not be greater than the incidence radiance.

$$\forall \omega_i \int_{2\pi} f_r(\omega_i, \omega_o) \cos\theta_r d\omega_o \leq 1$$

It must be noted that only simple empirical models like Phong lighting model ignore these constraints.

2.3 Photometric Stereo

The process of reconstructing the three-dimensional shape of a stationary scene from a collection of images representing the scene under variable lighting conditions is known as Photometric stereo (PS). Contrary to shape from shading, a method of recovering the shape of an object from a single image, PS essentially leads to a more robust solution and could potentially be obtained in the absence of any prior knowledge of lighting conditions or material properties [Ikeuchi(2021)].

Theory. For a scene composed of a matte surface with surface reflectance defined by the Lambertian model and illuminated by a single directional source i.e, a point source at infinity, the light reflected by a point $(x, y, z(x, y))$ and normal $\mathbf{n}(x, y)$ and albedo $\mathbf{a}(x, y)$ is defined as:

$$I_i(x, y) = \mathbf{a}(x, y)\mathbf{I}_i'\mathbf{n}(x, y).$$

If we consider k images $I_1, ..., I_k$ with matrix $M = [I_1, ...I_k]^T$ of intensity measurements, $L = [\mathbf{I}_1, ..., \mathbf{I}_k]^T$ of light sources and matrix $S = [\mathbf{a}(x_1, y_1)\mathbf{n}(x_1, y_1), ..., \mathbf{a}(x_k, y_k)\mathbf{n}(x_k, y_k)]$ of surface normal, then the combined equation for all images:

$$M = LS \Rightarrow S = L^{-1}M.$$

Now the scene depth values can be recovered by normalising each column of S:

$$\mathbf{n}(x_i, y_i) = \frac{\mathbf{s}^i}{\|\mathbf{s}^i\|}$$

$$\mathbf{a}(x_i, y_i) = \left\|\mathbf{s}^i\right\|.$$

In order to simultaneously recover shape and light, a decomposition of a 3×3 non-singular

matrix A can be used to introduce a nine-parameter ambiguity component:

$$\hat{M} = (\hat{L}A^{-1})(A\hat{S}).$$

Application and Open Problems. The method, though reliable for faithful reconstruction, is mostly confined to controlled laboratory conditions for stationary scenes under variable lighting conditions. The existing photometric stereo methods aim to handle Lambertian objects by removing specular pixels. Other methods also try to model the effects of a cast shadow and interreflections.

2.4 Multiview Stereo

Multiview stereo is defined as the process of reconstructing a 3D shape from calibrated overlapping images which have been captured from different viewpoints. The existing methods can be broadly divided into two categories - active and passive methods. It is based on the principle that dense pixel correspondences in multiple calibrated images make it possible to estimate the 3D shape of the scene. Multiview stereo works best when surfaces are textured and Lambertian whereas glossy or specular surfaces are non-Lambertian and are more difficult to handle. For reconstructing a static or dynamic scene (which is far more complex due to the additional time parameter) with multiview stereo, a single camera can be used to capture images from multiple viewpoints over time. For 3D shape representation, many multiview stereo methods represent the 3D scene as a set of depth maps however, polygonal meshes and explicit surface-based representations are used as well [Sinha(2014)].

Photoconsistency. It is a measure for the photometric similarity of the 2D projections of any 3D scene point in a set of calibrated images. A pair of 2D image patches can be compared using normalized cross-correlation.

$$\mathrm{NCC}(u, v) = \frac{\sum_{j=0}^{n}(u_j - \bar{u}).(v_j - \bar{v})}{\sqrt{\sum_{j=0}^{n}(u_j - \bar{u})^2 \sum_{j=0}^{n}(v_j - \bar{v})^2}}$$

The NCC-based similarity measure lies in the range [-1,1] where a value closer to 1 indicates that the vectors are similar in appearance. Many multiview stereo methods require the photoconsistency function to be evaluated densely on a 3D voxel grid for the volume containing the scene which is then referred to as the cost volume. One simple way to construct this

cost volume is to evaluate the photoconsistency of all 3D points (or voxels) using a pairwise similarity measure. Another general approach to constructing the cost volume involves first estimating depth maps from each camera's viewpoint using a state-of-the-art multi-baseline stereo matching algorithm.

Optimization Methods. The majority of multiview stereo methods formulate the reconstruction task as a local or global optimization problem. A local method such as space carving starts with an overestimate of the 3D scene. Global methods for depth map estimation incorporate image-based smoothness constraints into the formulation. Such methods are often based on a 2D Markov Random Field (MRF) framework where the energy function is minimized using partial differential equations.

2.5 Image-Based Modeling

Image-based modelling is the process of creating a three-dimensional model using two-dimensional images. In this, the appearance of a shape is often modelled by specifying a colour for each point on the shape. In comparison, appearance modelling dedicates to modelling both the shape and precise surface reflectance. These models are useful for various applications such as simulation, robotics, virtual reality, and digital entertainment.

Theory. In order to generate these shape models, a minimal surface can be obtained automatically from the input data. The minimal surface is a surface that minimizes the following

$$\int\int wds.$$

In the equation above, an infinitesimal element ds with its surface defined through the consistency exhibited in the input 3D point cloud or 2D images. The method could be applied to a simple Euclidean distance between the surface and 3D points and works best where the density of points is high. However, it is difficult for the method to model discontinuous surfaces like hair, clothes, tree branches, or buildings [Ikeuchi(2021)].

2.6 Image-Based Rendering

Image-based rendering refers to the representations of a 3D scene or object which allows its visualisation and manipulation without a full 3D model reconstruction. These represen-

tations can be obtained directly from RGB images and depending on how the images are being taken and auxiliary information, such as depths, etc., required, a number of image-based representations supporting different viewing freedom and functionalities are available. The rendering of novel views can be viewed as the reconstruction of the plenoptic function (which is a five-dimensional function representing the intensity of the light observed from every position and direction in the three-dimensional space) from its samples.

Theory. Rendering refers to the process of generating new views from the images. The geometry information can either be implicit that relies on positional correspondences or explicit in the form of depth along known lines-of-sight or 3D coordinates. The rendering methods can be classified into three types - (i) point-based, (ii) layer-based and (iii) monolithic rendering.

In a Point-Based Rendering system, the points are mapped through forward mapping techniques onto a target screen. This can be mathematically represented as:

$$X = \mathbf{C}_r + P_r x_r = \mathbf{C}_t + P_t x_t$$

In the equation above, x_t and x_r are homogeneous coordinates of the projection of X on target screen and reference images, respectively, $\mathbf{C}$ is the camera center and P is the projection matrix. Layer-based and monolithic rendering usually represents geometry as a collection of planar layers and continuous polygon meshes with textures, respectively [Ikeuchi(2021)].

2.7 3D Volume and Rendering

Volume visualisation is a method of extracting meaningful information from volumetric data using interactive graphics and imaging. Volumetric data has many forms which include 3D entities with or without relevant volumetric information, devoid of tangible surfaces and edges, or too voluminous to be represented geometrically. Though these varied forms exist, a volumetric data set can be represented by a standard set V of samples (x,y,z,v), also called voxels, representing the value v of some property of the data, at a 3D location (x,y,z).

The voxel could be used to represent a multitude of states and properties in and around the object. For instance, in binary data, the value 0 or 1 can simply represent whether the voxel represents the background or the object. Multivariate data could be used to represent colour,

density or normal direction. During rendering, the voxels can be replaced by geometric primitives like triangles to reduce the complexity of the task [Hansen and Johnson(2011)].

Volume rendering can be defined as the process of producing a two-dimensional image directly from three-dimensional volumetric data. It can be achieved using three major techniques - object-order rendering, image-order rendering and domain-order rendering. Image-order rendering uses a backward mapping scheme where rays are cast from each pixel in the image plane through the volume data to determine the final pixel value. For full volume rendering, interpolated samples are processed to simulate the light transport within a volumetric medium and can be represented using a low-albedo optical model as

$$I_\lambda(\mathbf{x}, \mathbf{r}) = \int_0^L C_\lambda(s)\mu(s) \exp\left(\int_0^S \mu(t)dt\right)ds \tag{2.1}$$

where $I_\lambda(\mathbf{x}, \mathbf{r})$ is the amount of light of wavelength λ coming from ray direction $\mathbf{r}$ that is received at point $\mathbf{x}$ on the image plane for each cast ray. On the other hand, object-order techniques decompose the volume into a set of basis elements or basis functions which are individually projected to the screen and assembled into an image. When solving for generalised volume rendering integral, voxels are represented as disjointed cubes, however, this provides inferior renderings. The quality has been proven to improve using kernel splatting which is achieved by (i) calculating the screen-space coordinate of the projected grid point, (ii) centering the footprint around that point and stretching it according to the image magnification factor and (iii) rasterising the footprint to the screen [Hansen and Johnson(2011)].

Hybrid techniques combine the advantages of both methods, i.e., they use object-centered storage for fast selection of relevant material (from object-order methods) and early ray termination for fast occlusion culling (from image-order methods). The shear-warp algorithm is one such hybrid method where the volume is rendered by a simultaneous traversal of RLE-encoded voxel and pixel runs (opaque pixels and transparent voxels are efficiently skipped during these traversals).

In domain rendering, the spatial 3D data is first transformed into another domain, such as compression, frequency, and wavelet domain, and then a projection is generated directly from that domain or with the help of information from that domain. The frequency domain rendering applies the Fourier slice projection theorem, which states that a projection of the 3D data volume from a certain view direction can be obtained by extracting a 2D slice per-

pendicular to that view direction out of the 3D Fourier spectrum and then inverse Fourier transforming it. This approach obtains the 3D volume projection directly from the 3D spectrum of the data, and therefore reduces the computational complexity for volume rendering from $O(N^3)$ to $O(N^2 log(N))$. The compression domain rendering performs volume rendering from compressed scalar data without decompressing the entire data set, and therefore reduces the storage, computation and transmission overhead of otherwise large volume data [Hansen and Johnson(2011)].

2.8 Datasets

2.8.1 Stanford Bunny

The Stanford Bunny is a standard test model of a clay bunny of roughly 7.5 inches in height, used for essentially all computer graphics research. These topics can include polygonal simplification, compression, surface smoothing, non-photorealistic rendering and texture mapping. The model was created using zippered polygon mesh technique [Turk and Levoy(1994)] to create polygonal models from several range scans. A range scan can be defined as a grid of distance values representing the distance between the points on the physical object from the capturing device. The model is a good test model - it is fairly smooth, has manifold connectivity and consists of 69,451 triangles. For the purpose of our experiments, we have generated images for 33 views and 3 lighting directions.

Zippered Polygon Meshes

The method provides a solution for combining a collection of range of images into a single polygonal mesh that describes the outer appearance of the object. The process encompasses three steps - first, it aligns the meshes with each other using a modified iterated closest-point algorithm, second, zips together adjacent meshes to form a continuous surface that faithfully captures the topology of the object and third, computes local weighted averages of the surface position of all meshes to form a consensus surface geometry.

Matlab Renderer

Matlab Renderer is an offscreen renderer based on deferred shading. It provides a rasteriser based on z-buffering and supports texture/normal mapping, shadowing, visibility

and orthographic, perspective and perspective+distortion camera models. It is fast and offers complete feature control for texture mapping, camera parameters and shadow mapping [Bas and Smith(2019)].

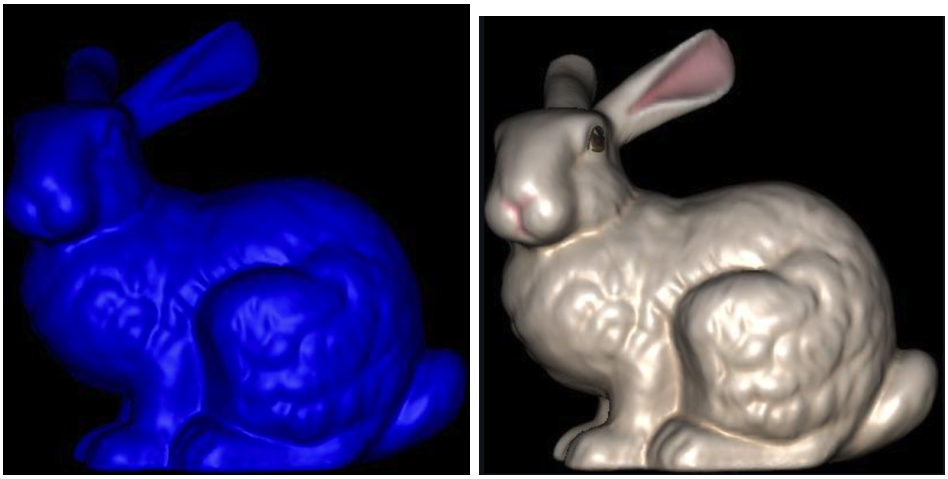


Figure 2.3: Mesh and Texture Map for Stanford Bunny used as input for Matlab Renderer

The renderer's coordinate system assigns the top left pixel as (1,1) with positive X on the right, positive Y down and positive Z on the screen. The renderer only supports the following:

1. point light sources which can either be local or distant
2. Blinn-Phong reflectance
3. scaled orthographic, perspective or perspective with distortion camera models

The details on the implementation and usage of the dataset have been provided in the Appendices section 8.1.

2.8.2 DiLiGeNT Multiview

The 'DiLiGenT-MV' dataset [Li et al.(2020b)] includes five objects: BEAR, BUDDHA, COW, POT2, and READING as shown in figure 2.4. The images have been illuminated by 96 different lights from 20 different viewpoints.

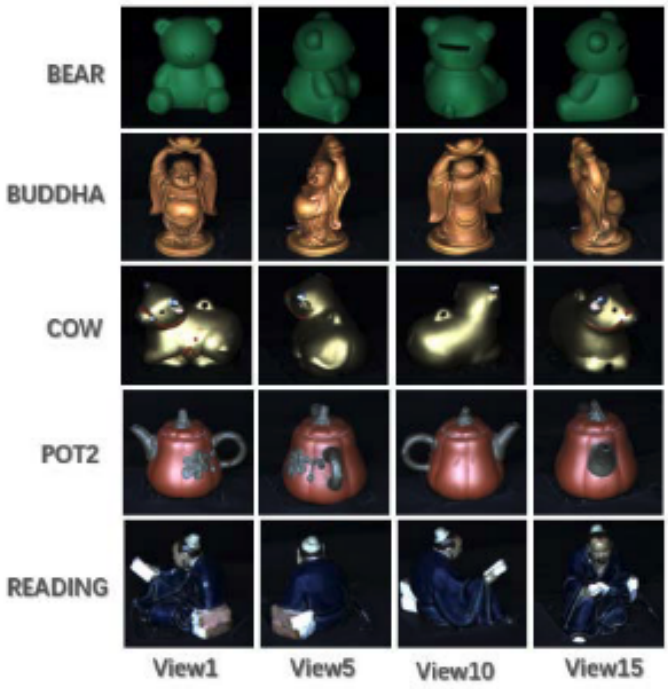


Figure 2.4: DiLiGeNT-MV Objects. (Figure taken from [Li et al.(2020a)])

Data Capture Setups

The data has been captured using two different types of setups - a studio scanner and a desktop scanner with both systems fitted with camera-synchronous LED lights for fast data capture. The studio setup uses a PointGrey Grasshopper camera whereas the desktop setup uses linear industry camera with resolutions 1200×900 and 1280×960 respectively. The images were captured viewpoint by viewpoint.

Setup Details. The studio setup consists of 72 LED lights placed uniformly in two concentric circles of diameters 400 and 600 mm with the video camera mounted at the center as shown in figure 2.5. The desktop setup consists of a similar concentric ring setup with diameters 150 and 300 mm. For all experiments, the object has been placed 400 mm away from the camera.

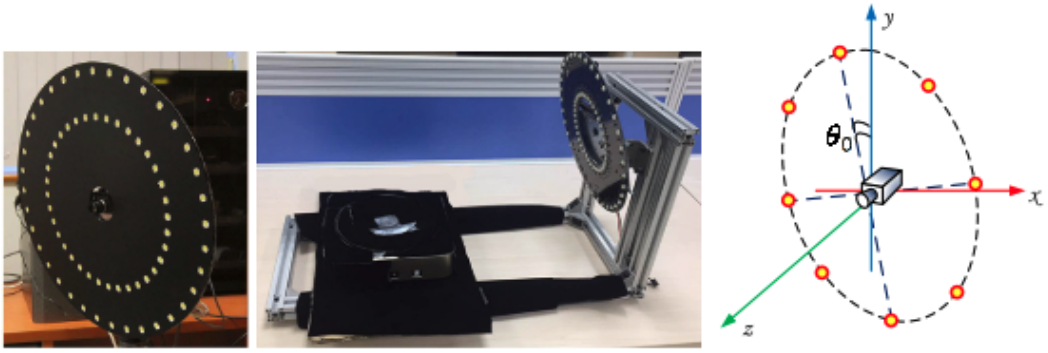


Figure 2.5: Left: the studio scanner setup. Middle: the desktop scanner setup. Right: θ_0 can be calibrated to determine the position of the LED lights in 3D space. (Figure taken from [Li et al.(2020a)])

Calibration. The calibration process has been simplified due to the known radii of the concentric ring lights for both the setups, thus, a single parameter, azimuth angle, needs calibration for calculating the precise position of 3D object. The azimuth angle can be represented as $\theta_0 + \alpha$, where α is the true azimuth angle. The light intensities are measured using a diffuse board roughly parallel to the image plane.

Shape and Reflectance Reconstruction.

The images have been captured from multiple viewpoints, and at each viewpoint, photometric stereo images under different lighting conditions have been recorded. This has been achieved by identifying iso-depth contours from the input images from every viewpoint as shown in figure 2.6. In the upcoming sections, we will dive into details about each step.

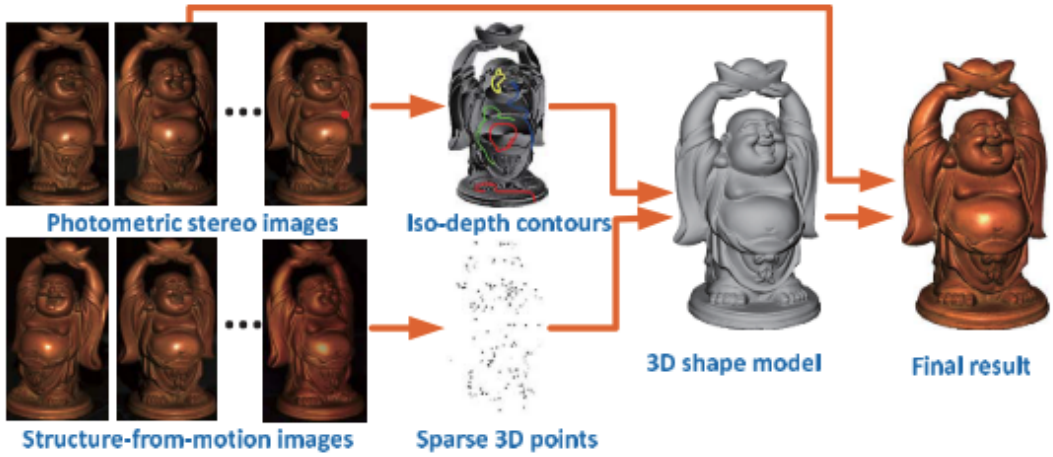


Figure 2.6: The iso-depth contours and a sparse 3D point cloud have been recovered from photometric stereo images and by structure-from-motion, respectively. (Figure taken from [Li et al.(2020a)])

Basic Iso-Depth Contour Estimation. For the estimation, a generalized algorithm has been used to make it more robust in real data. When the light is rotated on the path of the circles, the camera is able to recover the surface normal direction of the points on the isotropic surface. In the camera's local coordinate system, this plane gives the azimuth angle of the surface normal. Once an azimuth direction is computed at each pixel, iso-depth contours are generated. This has been achieved by iteratively tracing along the two directions perpendicular to the azimuth direction for every pixel.

Multi-View Depth Propagation. Employing a sparse set of 3D points has been proven extremely useful in the recreation of the object using any standard structure-from-motion algorithm. For a reconstructed 3D point x, the algorithm is proficient in searching and projecting it to all the sample images where is it visible. The following step in depth propagation is to determine the depth of point x in every pixel in an input image C_i. It generates information that is used to determine the depths of other 3D points in every input image. Finally, a median image has been computed so as to avoid any moving highlights.

For obtaining the most accurate results some issues need to be addressed, these are (i) all points are required to be sorted according to the confidence of their associated iso-depth contours and (ii) the depth propagation should not be applied in images where a 3D point

is not visible.

Reflectance Capture. In order to capture the surface reflectance, the system incorporates a general tri-variant isotropic BRDF, given the assumption that the surface reflectance can be represented by a linear combination of several (K = 2) basis isotropic BRDFs. Following this, an $N \times M$ observation matrix V has been created as follows.

$$\mathbf{V}_{N\times M} = \mathbf{W}_{N\times K}\mathbf{H}_{K\times M}$$

The observation matrix has been decomposed into mixing weight matrix W and BRDF basis matrix H. The accuracy of reflectance capture can be improved by calculating H as a subset of accurately measured 3D points.

2.9 Conclusion

In this chapter, we discussed the core concepts one requires to be acquainted with in order to understand this piece of work and the novel contribution made by it. We discussed the types of surfaces which we will be modelling through our work - Lambertian and BRDF, the mathematics supporting Photometric and Multiview Stereo and finally, image-based rendering and modelling. We concluded the chapter with a soft introduction to the datasets - Stanford Bunny and DiLiGeNT-MV used for all our training. With an in-depth understanding of the preliminaries, we are ready to dive deep into the workings of Neural Representations which is the core essence of the upcoming chapter.

Chapter 3

Literature Review

In this chapter, we will be discussing the state-of-the-art technologies on neural representations and high-fidelity object reconstruction which are the backbone of this project. We will first provide honourable mentions to some previous works in the neural 3D representation realm - Local Implicit Grid Representation and DeepSDF and in view synthesis and image-based rendering realm - soft rasterizer and DeepView, followed by Local Light Field Fusion (LLFF) which is the direct predecessor to Neural Radiance Fields. Once we provide the reader with a strong understanding of these methods, we will gently introduce NeRF and dive deep into the core mathematics behind the technique. Following this, we will provide an exhaustive categorised list of varied forms of NeRF which aim to improve a particular set of features, for instance, faster inference and training models, generalised and deformable models. Furthermore, a literature critique will provide the scope and drawbacks of some of these techniques, providing a case for a knowledge gap. We will conclude our Literature Reiview by further exploring the different methods which provide relightable functionality as it would be one of the main focuses of our novel contributions.

3.1 Neural 3D Representations

Previous works in the field of neural rendering have learned shape priors from partial or noisy data. Local Implicit Grid Representations [Jiang et al.(2020)] trains an autoencoder to handle complex and diverse indoor scenes. Continuous Single Distance Function (DeepSDF) [Park et al.(2019)] provides trade-offs across fidelity, efficiency and compression capabilities enabling high-quality shape representation, interpolation and comple-

tion. Local Deep Implicit Functions (LDIF) [Genova et al.(2020)] and Occupancy Networks [Mescheder et al.(2019)] provide a computationally efficient function for high-resolution geometry representation for any arbitrary topology. However, these representation functions are limited by a major factor – they all require access to the ground truth of the 3D geometry which is typically obtained from a synthetic 3D shape.

A step up from these techniques has been in training reconstruction models from RGB images, however, these approaches have been restricted to voxel- [Stutz and Geiger(2018), Xie et al.(2019), Gadelha et al.(2017)] and mesh-based [Kanazawa et al.(2018), Liao et al.(2018), Pan et al.(2019), Wang et al.(2018)] representations which suffer from discretization or low resolution [Niemeyer et al.(2020)]. Recent works have provided a rather relaxed dependency on ground truth 3D geometry and essentially employing only 2D images to formulate differentiable rendering functions. [Niemeyer et al.(2020)] propose to learn implicit shape and texture representations using depth gradients which are derived from implicit differentiation. This is obtained by calculating the surface intersection for each ray which is the input to a neural 3D texture field predicting a diffuse colour at a particular point. [Sitzmann et al.(2019)] propose Scene Representation Networks (SRNs) which provide a structure-aware scene representation through a continuous function mapping world coordinates to features of a local scene – achieved through a differentiable ray-marching algorithm. These techniques have quite a potential to represent complicated and high-resolution geometry, however, they are still limited to extremely low geometric complexity leading to over-smoothed renderings.

3.2 View Synthesis and Image-based Rendering

Photorealistic novel views can be reconstructed when the data sampling is dense in nature by widely popular interpolation techniques, however, when the sampling is sparse in nature it becomes difficult to predict geometry and appearances from observed images. Mesh-based representations can be directly optimised using differentiable rasterizers and pathtracers. DIB-R [Chen et al.(2019)] implements a differentiable renderer which views foreground rasterization as a weighted interpolation of local properties and background rasterization as a distance-based aggregation of global geometry – which, in turn, enables gradient calculation for all pixels in the image. Similarly, soft rasterizers [Liu et al.(2019)] renders using an aggre-

gation function which collates probabilistic contributions of all mesh triangles with respect to the rendered pixels. Monte Carlo ray tracer [Li et al.(2018)] through edge sampling provides a comprehensive solution to calculate scalar function derivates over a rendered image with respect to parameters like camera poses, lighting and scene geometry. However, these strategies require a template mesh with a fixed topology which is not possible for in-the-wild scenes.

Volumetric representations provide superior reconstruction for complex shapes and materials as they can be easily optimised through gradient-based approaches and are less prone to visually distracting artefacts. DeepView [Flynn et al.(2019)] synthesises multiplane images (MPI) from sparse camera viewpoints improving performance on object boundaries, light reflection and thin structures. Local light field fusion (LLFF) [Mildenhall et al.(2019)] proposed an algorithm by extending the traditional plenoptic sampling theory for view synthesis from an irregular grid of sample views via MPI representations. Neural Volumes [Lombardi et al.(2019)] present an encoder-decoder framework which outputs a 3D volume representation from images and a differentiable ray-marching operation for end-to-end training. Though these techniques have the potential to render high-resolution novel viewpoints, they are limited due to discrete sampling.

3.3 Local Light Field Fusion LLFF

LLFF proposes a robust deep learning technique for view synthesis from an irregular grid of sampled views to render novel views of complex real world scenes for virtual exploration. As opposed to previous works which required intractably dense view sampling, it expands sparse sampled views into a local light field via multiplane image (MPI) [Zhou et al.(2018)] scene representations and then renders new views by blending these adjacent local light fields. The approach has been proven to be practical in real life and real time scenarios like augmented reality and virtual exploration.

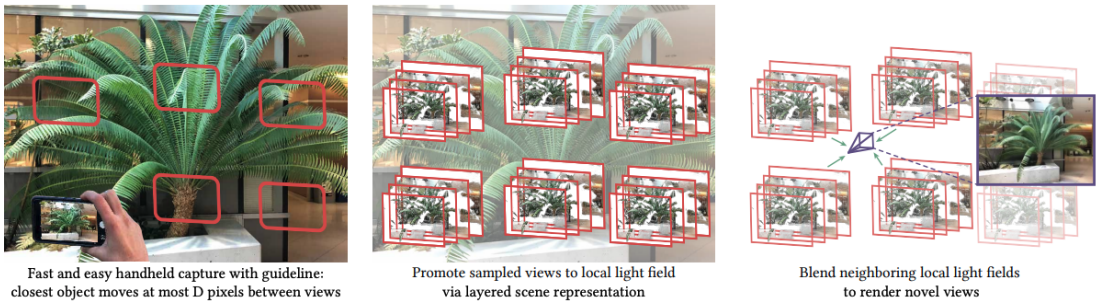


Figure 3.1: Local Light Field Fusion Overview: A method for view synthesis from a set of input images captured by a handheld camera in an irregular grid pattern (Figure taken from [Mildenhall et al.(2019)]).

LLFF Model

The LLFF model (overview provided in figure 3.1) uses an extension of plenoptic sampling theory [Chai et al.(2000)] to derive a bound specifying the density of required sample views to be provided by the user for a particular scene. This strategy reduces the number of required sample views by a factor of D^2, where D is the number of depth planes, compared to Nyquist view sampling.

Nyquist Rate View Sampling. The extension to plenoptic sampling [Chai et al.(2000)] defines the required maximum camera sampling interval $\triangle_u$ for a light field with occlusion as

$$\triangle_u \leq \frac{1}{2K_x f(\frac{1}{z_{min}} - \frac{1}{z_{max}})} \tag{3.1}$$

where K_x is the highest spatial frequency representation in the sampled light field. It is defined by the highest spatial frequency in the continuous light field B_x and the camera spatial resolution $\triangle_x$

$$K_x = \min(B_x, \frac{1}{2\triangle_x}). \tag{3.2}$$

MPI Scene Representation and Rendering. The representation consists of a set of fronto-parallel RGBα planes which are evenly sampled within a reference camera view. The novel view are generated in the local neighbourhood using alpha compositing of the colour along the rays using the "over" operator.

View Sampling Rate Reduction. Each alpha composting step increases the Fourier support by convolving the previously-accumulated light field's spectrum with the spectrum

of the occluding depth layer. Steering away from the traditional plenoptic sampling, the additional bound to the camera sampling interval $\triangle_u$ can be measured as

$$\triangle_u \leq \frac{W \triangle_x z_{min}}{2f} \tag{3.3}$$

where W is the image width in pixels of each sampled view.

View Synthesis Pipeline

As shown in the figure 3.2, the synthesis pipeline encompasses two phases - (i) a CNN to promote a captured input image to an MPI and (ii) reconstruction of novel views by blending renderings from adjacent MPIs.

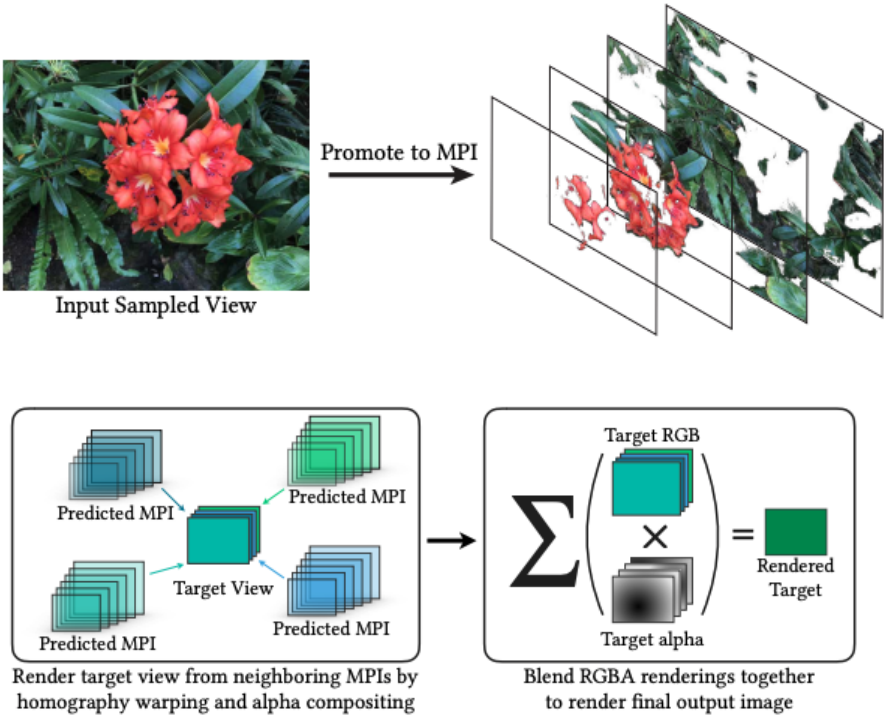


Figure 3.2: View Synthesis Pipeline. **Top**: View sample inputs to an MPI scene representation [Zhou et al.(2018)], consisting of D RGB planes at regularly sampled intervals. **Bottom**: Render novel views as a weighted combination of renderings from neighbouring MPIs, modulated by the corresponding accumulated alphas (figure taken from [Mildenhall et al.(2019)]).

MPI Prediction for Local Light Field Expansion. In order to fully utilise the potential of each capture, the pipeline provides a platform to expand each sampled view to a local

light field using an MPI scene representation. This has been achieved by taking five views as input and expanding the reference view to its four nearest neighbours in 3D space. The input images are then reprojected to D depth planes and sampled linearly in disparity within the reference view frustum and form 5-plane sweep volumes (PSVs) of size $H \times W \times D \times 3$. The CNN provides opacity value α for each MPI coordinate (x, y, d) as well as a set of 5 colour selection weights which sum up to 1 at every MPI coordinate. The predictions are improved using 3D convolutional layers instead of 2D convolutional layers which in turn, enables it to predict MPIs with a variable number of planes D.

Continuous View Reconstruction by Blending. The MPI prediction network uses a set of RGB images C_k and their camera poses p_k to create a stack of MPIs M_k for each input image. The novel views are created by homographically wrapping each RGBα MPI plane into the frame of reference of the target pose p_t and finally, alpha compositing these planes from back to front. The output from the network is RGB image $C_{t,k}$ and alpha image $\alpha_{t,k}$. The prediction performed over multiple MPIs can be eventually defined as

$$C_t = \frac{\sum w_{t,k}\alpha_{t,k}C_{t,k}}{\sum_k w_{t,k}\alpha_{t,k}} \tag{3.4}$$

where C_t is the final output after blending all rendered images $C_{t,k}$, $w_{t,k}$ are scalar blending weights. The output is normalised to make it fully opaque.

3.4 Neural Radiance Fields NeRF

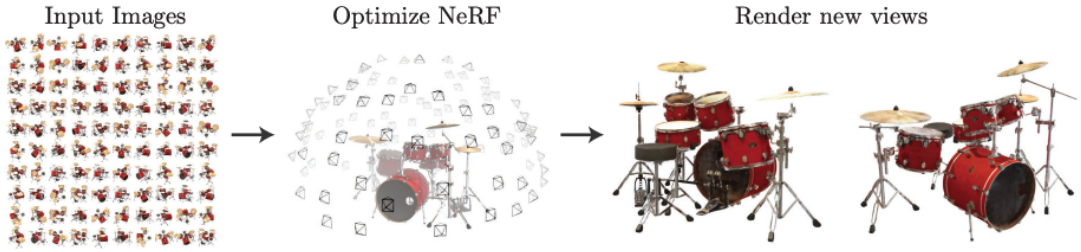


Figure 3.3: **NeRF** presents a method to optimize a continuous 5D neural radiance field representation of a scene from a set of input images. In the figure, a set of 100 input views of the synthetic Drums scene has been used to generate the 3D volume of the object (figure taken from [Mildenhall et al.(2020)]).

NeRF represents a scene using a neural volumetric representation. A continuous scene is represented as a 5D vector function with inputs 3D location $\mathbf{x} = (x, y, z)$ and 2D viewing direction (ϕ, θ) (corresponding to a 3D Cartesian unit vector $\mathbf{d}$) which outputs an emitted colour $\mathbf{c} = (r, g, b)$ and volume density σ as shown in figure 3.3. The volume density function σ is restricted to be a function only of location $\mathbf{x}$ in order to maintain multiview consistency. The network comprises a deep fully connected neural network without any convolutional layers (Multilayer Perceptron, MLP), $F_\Theta : (\mathbf{x}, \mathbf{d}) \mapsto (\mathbf{c}, \sigma)$. The network outputs the radiance emitted in each direction at each point in space along with a density at each point which acts like a differential opacity controlling the amount of radiance that is accumulated when it passes through a particular point as evident from figure 3.4 [Mildenhall et al.(2020)]. In simple terms, differential opacity along a ray can be defined as the probability of it terminating within the given volume. A large value of the Δ provides a good indication of termination as it signifies the presence of a high-density object through which light is not transmissible. Previously, such MLP architectures have been used to represent natural textured materials that can be sampled over infinite domains [Henzler et al.(2020)] or circumvent factors – like in Texture Fields – which limit the fidelity of highly textured surfaces [Oechsle et al.(2019)].

Volume Rendering

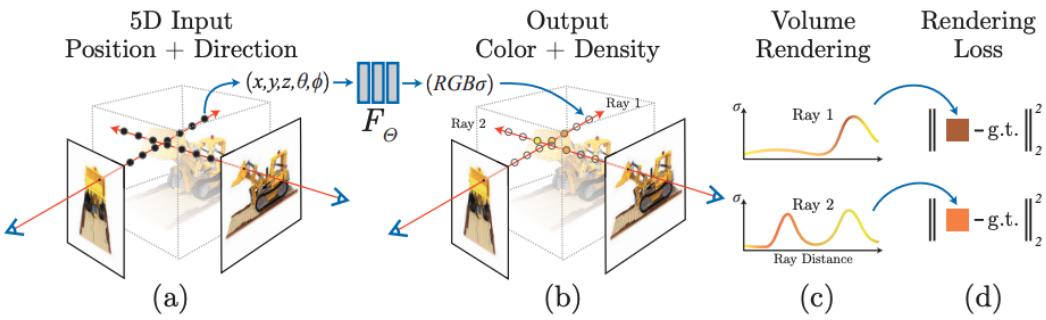


Figure 3.4: **Differential Rendering Pipeline Overview.** The image is represented as 5D coordinates which include the location and viewing direction along camera rays which are used to (a) generate corresponding colour and volume density, (b) composite into an image using volume rendering. (c) A differentiable rendering function minimises the error between the generated and ground truth images (d) and quantified using L2 loss (figure taken from [Mildenhall et al.(2020)]).

The network weights Θ are optimised such that reconstruction error to training images is minimised when the network output is passed to a differentiable volume renderer. The colour of any ray passing through a scene can be rendered using classical volume rendering techniques [Kajiya and Von Herzen(1984)]. The volume density $\sigma(\mathbf{x})$ can be interpreted as the probability of a ray terminating at a particle at location $\mathbf{x}$ - it must be noted that the function is designed such that it is differentiable. Thus, the expected colour $C(\mathbf{r})$ of camera ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ with near and far bounds t_n and t_f is:

$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), \mathbf{d})dt, \tag{3.5}$$

where

$$T(t) = \exp\left(-\int_{t_n}^{t} \sigma(\mathbf{r}(s))ds\right). \tag{3.6}$$

The function $T(t)$ represents the amount of colour accumulated along the ray from t_n to t. This can also be represented as the probability of the ray travelling from t_n to t without hitting any other particle. This continuous integral is estimated using stratified sampling where $[t_n, t_f]$ is divided into N evenly-spaced bins and values are randomly sampled from each bin as

$$t_i \sim \mathcal{U}\left[t_n + \frac{i-1}{N}(t_f - t_n), t_n + \frac{i}{N}(t_f - t_n)\right]. \tag{3.7}$$

These samples are used to estimate $C(\mathbf{r})$ with the quadrature rule [Max(1995)] as

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i(1 - \exp(-\sigma_i\delta_i))\mathbf{c}_i, \tag{3.8}$$

where

$$T_i = \exp\left(-\sum_{j=1}^{i-1} \sigma_j\delta_j\right), \tag{3.9}$$

where $\delta_i = t_{i+1} - t_i$ is the distance between adjacent samples of stratified sampling bins and $\alpha_i = 1 - \exp(-\sigma_i\delta_i)$ represents alpha compositing.

Optimisation

Volume rendering alone proves insufficient for achieving state-of-the-art rendering quality. The quality of complex scene representation in NeRF can be attributed to two factors - positional encoding and hierarchical volume sampling.

- Positional Encoding: It has been observed that the network F_Θ performs subpar when representing high-frequency variations in colour and geometry which can be attributed to deep networks being biased towards learning lower frequency functions [Rahaman et al.(2019)]. Therefore, operating on $xyz\theta\phi$ requires working with a composite function $F_\Theta = F'_\Theta \circ \gamma$ so as to enable the MLP to more easily approximate a higher frequency function [Mildenhall et al.(2020)]. γ is sinusoidal mapping from $\mathbb{R}$ into higher dimensional space $\mathbb{R}^{2L}$ and can be defined as

$$\gamma(p) = (\sin(2^0\pi p), \cos(2^0\pi p), \cdots, \sin(2^{L-1}\pi p), \cos(2^{L-1}\pi p)). \tag{3.10}$$

- Hierarchical Volume Sampling: Evaluating the network at N query points along every camera ray is computationally intensive and thus inefficient. For a more efficient evaluation, the rendering strategy [Levoy(1990), Mildenhall et al.(2020)], simultaneously optimises two networks - 'coarse' and 'fine'. A set of N_c locations are used to first compute the 'coarse' network. The output helps in an informed sampling of relevant points for further detailed computation. The sole alteration is the reconfiguration of alpha compositing colour from the coarse network as

$$\hat{C}_c(\mathbf{r}) = \sum_{i=1}^{N_c} w_i c_i, \quad w_i = T_i(1 - \exp(-\sigma_i \delta_i). \tag{3.11}$$

- Loss Function: The loss function is defined as the square of the difference between the rendered and true pixel colours for both coarse and fine renderings:

$$\mathcal{L} = \sum_{\mathbf{r}\in\mathcal{R}} \left[\left\| \hat{C}_c(\mathbf{r}) - C(\mathbf{r}) \right\|_2^2 + \left\| \hat{C}_f(\mathbf{r}) - C(\mathbf{r}) \right\|_2^2 \right] \tag{3.12}$$

  where $\mathcal{R}$ is the set of rays in each batch, and $\hat{C}(\mathbf{r})$, $\hat{C}_c(\mathbf{r})$, and $\hat{C}_f(\mathbf{r})$ are the ground truth, predicted RGB colours from the coarse and fine networks for ray $\mathbf{r}$ respectively.

3.4.1 NeRF extensions

The field of Neural Field Radiance has received significant attention due to its impressive results in capturing unbounded and bounded 360-degree scenes. According to [Martin-Brualla et al.(2021)], though NeRF works well on static images captured under controlled settings, it is un-

able to model uncontrolled real-world phenomena in images, such as variable illumination or transient occluders. NeRF++ [Zhang et al.(2020)] applies NeRF to 360-degree, large-scale, unbounded 3D scenes by applying shape radiance ambiguity and creating inverted sphere parameterization - separately modelling foreground and background. NeRF-W [Martin-Brualla et al.(2021)] introduces a system for 3D reconstruction from in-the-wild photo collections. This was achieved by introducing Generative Latent Optimization (GLO) [Bojanowski et al.(2018)] - making the approximation with respect to image-dependent radiance - and modelling observed colour as a probability distribution (isotropic normal distribution) over its value.

Faster Inference Models

Despite of the success of producing faithful representations by NeRF, its training remains challenging and time-consuming to render such representations under complex lighting such as environment maps.

Neural Transmission Functions accelerates the rendering process using monotonic transmittance along the ray providing two orders of magnitude speedup under complex conditions [Shafiei et al.(2021)]. Neural Sparse Voxel Fields (NSVF) obtains 10x faster and high-quality free-viewpoint rendering by defining a set of voxel-bounded implicit fields organized in a sparse voxel octree to model local properties in each cell and learning the underlying voxel structures with a differentiable ray-marching operation from only a set of posed RGB images [Liu et al.(2020)]. FastNeRF renders high-fidelity photorealistic images at 200Hz on a high-end consumer GPU by compact caching and efficient querying of the radiance map making it 3000 times faster [Garbin et al.(2021)]. DONeRF significantly reduces the samples required along each ray, without reducing the quality of the image rendered, by using a classification network around logarithmically discretized and spherically warped depth values is essential to encode surface locations over direct estimation of depth [Neff et al.(2021)].

DeRF [Rebain et al.(2021)] presents a solution to NeRF's computationally intensive nature using spatial decomposition where each portion of the scene is handled by smaller networks which can work together to render the complete scene. Furthermore, employing Voronoi spatial decomposition is efficient for GPU rendering and thus DeRF's inference is 3 times more efficient than NeRF. It also circumvents the issue of diminishing returns in neural rendering. AutoInt [Lindell et al.(2021)] presents an efficient learning framework using implicit neural

networks. When applied to neural volume rendering, it provides improved computational efficiency.

KiloNeRF demonstrates real-time rendering by employing thousands of minute MLPs which individually learn parts of the scene leading to a smaller and faster model [Reiser et al.(2021)]. RT-NeRF provides algorithm-hardware co-design acceleration of NeRF by alleviating pipeline inefficiency through the removal of uniform point sampling and processing invisible points redundant [Li et al.(2022a)]. Neural Light Fields (NeLF) perform rendering in one single forward pass without the need for ray-marching. It achieves $26 \sim 35$ times FLOPs reduction, $28 \sim 31$ times runtime speedup and better-rendering quality [Wang et al.(2022)].

Faster Training Models

Given a large number of input views, NeRF can still be time-consuming to train; it often takes between ten hours to several days to model a single scene at moderate resolutions on a single GPU. The training is slow due to both the expensive ray-casting operations and the lengthy optimization process. The need for such lengthy training times makes NeRF infeasible for many application scenarios.

Depth-supervised NeRF (DS-NeRF) defines a model which takes free sparse 3D points, produced while processing the structure-from-motion pipeline, as inputs for the depth supervision training. The model defines a loss which can determine ray termination, using these depth values, making DS-NeRF 2-3x faster [Deng et al.(2022)]. Direct Voxel Grid Optimization technique whose training converges in under 15 minutes for any new scene, obtains high-quality renderings with a single GPU. This has been achieved through post-activation interpolation on voxel density, for sharp surface rendering, and by imposing priors for the voxel density optimisation process [Sun et al.(2022)].

Instant Neural Graphics Primitives (NGP) reduce the cost of training by employing multiple smaller networks which are combined via a multiresolution hash table of trainable features. The method reduces the number of floating points and memory access operations providing several orders of training speedup [Müller et al.(2022)]. TensoRF employs 4D tensors to model the radiance fields of a scene along with vector-matrix decomposition and achieves high-quality reconstruction in under 10 minutes [Chen et al.(2022)].

Deformable Models

NeRF are able to reconstruct scenes with unprecedented fidelity. A common approach to reconstructing non-rigid scenes is through the use of a learned deformation field mapping from coordinates in each input image into a canonical template coordinate space. However, these deformation-based approaches struggle to model changes in topology.

Deformable NeRFs [Park et al.(2021a)] presents a method for photorealistic reconstruction of a non-rigidly deforming scene from casually captured photos and videos from mobile phones - 'nerfies'. It provides multiple techniques to improve robustness - coarse-to-fine optimization and elastic regularization of deformed fields. Though vanilla NeRF is applicable to static scenes, D-NeRF [Pumarola et al.(2021)] extends the applicability to the dynamic domain which allows reconstruction and rendering of objects under motion from a single moving camera. The time-varying deformation is performed through two modules - one learning a canonical configuration and the other learning the displacement field with respect to the canonical space.

Non-Rigid NeRF (NR-NeRF) provides a model for creating high-quality space-time geometry and appearance representation using a single handheld consumer-grade camera. This has been achieved by implementing ray bending for deformable features and a rigidity network to better constrain rigid regions of the scene [Tretschk et al.(2021)]. Neural Articulated Radiance Field (NARF) focuses on pose-dependent changes without a significant increase in computational complexity and thus is efficient and generalises well to novel poses [Noguchi et al.(2021)]. Category-Level Articulated Neural Radiance Field (CLA-NeRF) performs view synthesis and pose estimation for a 3D object in a particular category and obtains realistic deformation results [Tseng et al.(2022)]. HyperNeRF handles deformation by representing a 5D radiance field in a higher dimensional space which can smoothly handle the transition between scene configurations and obtain renderings at novel fixed scenes [Park et al.(2021b)].

Video Rendering Models

Another extension into the dynamic domain is Neural Scene Flow Fields [Li et al.(2021)] which models dynamic scenes as a time-variant continuous function of appearance, geometry and 3D scene motion. Neural Irradiance Fields [Xian et al.(2021)] enables the rendering of

video from novel viewpoints. NeRFlow enables multi-view rendering in diverse dynamic scenes, including water pouring, robotic interaction and real image dynamic view synthesis by capturing the 3D occupancy, radiance, and dynamics of the scene [Du et al.(2021)].

A Neural 3D video synthesis technique [Li et al.(2022c)] provides high-quality motion interpolation for dynamic scenes by employing a time-conditioned NeRF. The model can represent a 10-second 30 FPS multiview video recording by 18 cameras with a model size of 28MB. Another method [Gao et al.(2021)] is able to learn an implicit function from an ill-posed dynamic monocular video. It reduces the ambiguity of the solution by employing a regularisation loss leading the model to the most physically viable solution. Streaming Radiance Fields uses the explicit grid-based model to learn incremental changes with changing time frames in the dynamic scene [Li et al.(2022b)].

A fundamental obstacle to making these methods practical is the extreme computational and memory requirements caused by the required volume integrations along the rendered rays during training and inference [Huang et al.(2020)].

Generalisation Models

Generative Radiance Fields (GRAF) [Schwarz et al.(2020)] present a generative model with a multi-scale patch-based discriminator which demonstrates the synthesis of high-resolution results from unposed 2D images. The discriminator is implemented as a convolutional neural network which compares the predicted patch to a patch extracted from a real image. The framework generates higher-resolution images from voxel-based approaches but is limited to simple scenes with single objects - incorporating inductive biases could extend it to real-world images.

Closely related to this paper have been several previous efforts to incorporate physically-based image formation models into NeRF-like architectures. These differ in their modelling assumptions, input data and target applications.

3.5 Relightable NeRF Architectures

3.5.1 Neural Reflectance and Visibility Fields NeRV

NeRV introduces a technique that produces a 3D representation that can be rendered from novel viewpoints under arbitrary lighting conditions. The method takes a set of images of a scene under unconstrained known lighting conditions and outputs multiple scene properties like, volume density, surface normal, material parameters, distance to the first surface of intersection in any direction and visibility of the external environment at a particular input location as shown in figure 3.5. The model outperforms previous iterations of NeRF due to its ability to simulate direct and indirect illumination, simultaneously. This is possible through its predicted visibility and surface intersection fields [Srinivasan et al.(2021)].

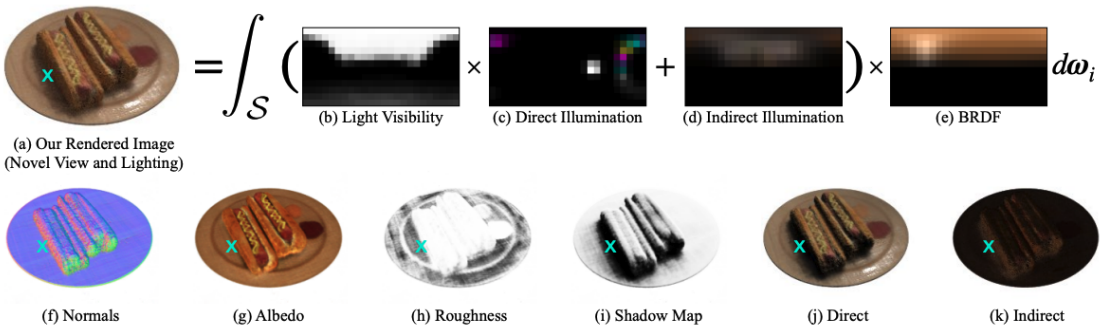


Figure 3.5: **Neural Reflectance and Visibility Fields Outputs.** For (a) a 3D location "x" as input, the NeRV model provides the volume density followed by (b) visibility for the surrounding scene coupled with (c) the direct illumination and (d) estimated indirect illumination. The outgoing radiance at the point "x" is obtained by multiplying the (e) predicted BRDF with the environment map. The bottom row shows the visualized outputs: (f) Surface normals, and BRDF parameters for (g) diffuse albedo and (h) specular roughness (i) a shadow map (j) direct and (k) indirect illumination (figure taken from [Srinivasan et al.(2021)]).

NeRV Model

NeRV extends NeRF capabilities by the introduction of a neural visibility field representation. Visibility fields model a simulation of light transportation through each point in the scene so as to supplement NeRF's volumetric representation.

Neural Reflectance Fields. Following the work by [Bi et al.(2020)] which modifies NeRF to enable relighting by representing a scene as a field of particles that emit light, NeRV

represents it as a scene of particles that reflect incoming light. When this is combined with an arbitrary lighting condition, NeRV can simulate the transport of light through the volume as it is reflected by particles until it reaches the camera using a volume rendering integral:

$$L(\mathbf{c}, \omega_\mathbf{o}) = \int_0^\infty V(\mathrm{x}(t), \mathrm{c})\sigma(\mathrm{x}(t))L_r(\mathrm{x}(t), \omega_o)\mathrm{d}t, \tag{3.13}$$

$$L_r(\mathbf{c}, \omega_\mathbf{o}) = \int_S L_i(\mathrm{x}, \omega_i)R(\mathrm{x}, \omega_i, \omega_o)\mathrm{d}\omega_i, \tag{3.14}$$

where NeRF's view-dependent emission is replaced with an integral over the sphere S of incoming direction and a reflectance function R representing the amount of light reaching from a particular direction ω_i which has been reflected towards direction ω_o.

The radiance MLP from NeRF is replace by two MLPs - "shape" MLP which outputs volume density σ and a "reflectance" MLP which outputs the BRDF parameters (3D diffused albedo $\mathbf{a}$ and 1D roughness γ) for any input 3D point: $\mathrm{F}_\Theta : x \rightarrow \sigma$, $\mathrm{F}_\Psi : x \rightarrow (\mathbf{a}, \gamma)$.

Light Transport via Neural Visibility Fields.

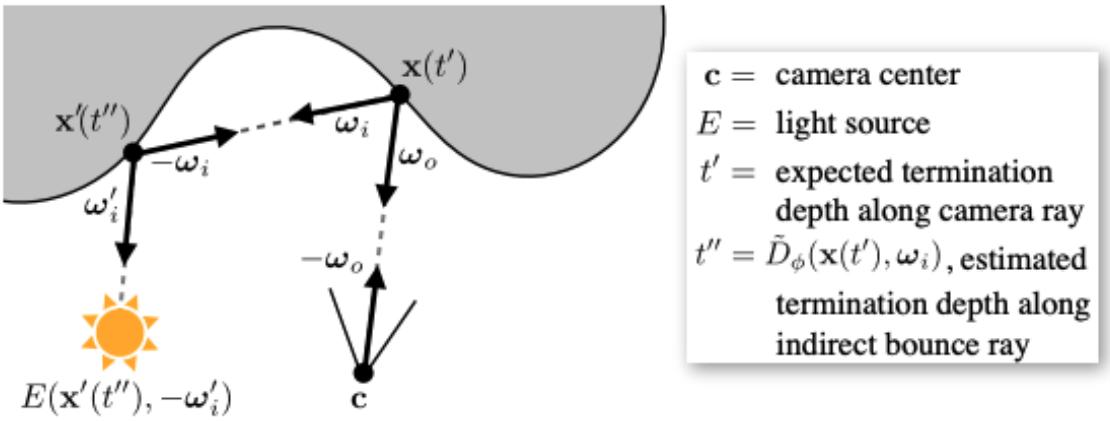


Figure 3.6: An illustration of the indirect illumination path from camera to light source. (figure taken from [Srinivasan et al.(2021)]).

The radiance MLP explained before works well for straightforward relighting but volume rendering integral estimation for general lighting scenarios is difficult to achieve due to the massive amount of computation required to train such a model. This sheer amount of computation has been mitigated through learned approximations using "visibility" MLP. The MLP outputs an approximation of the environment lighting visibility at any input location along any input direction and an approximation of the expected termination depth

of the corresponding ray: $\mathrm{F}_\Theta : (\mathrm{x}, \omega) \rightarrow (\hat{V_\theta}, \hat{D_\theta})$.

Neural Visibility Field approximations simplify the volume rendering integral by decomposing in two stages. The reflected radiance $L_r(\mathrm{c}, \omega_o)$ is divided into its direct and indirect illumination components. Next, it decomposes incident illumination L_i into (i) L_e the radiance due to the light source and (ii) L the estimated incoming radiance as shown in figure 3.6. This provides a definition of reflected radiance L_r as:

$$L_r = \int_S L_e(\mathrm{x}, \omega_i) R(\mathrm{x}, \omega_i, \omega_o) \mathrm{d}\omega_i + \int_S L(\mathrm{x}, -\omega_i) R(\mathrm{x}, \omega_i, \omega_o) \mathrm{d}\omega_i \tag{3.15}$$

The incident direct lighting L_e accounts for the environment map attenuation E due to volume density by:

$$L_e(\mathrm{x}\omega_i) = V(\mathrm{x}, \omega_i) E(\mathrm{x}, -\omega_i)$$

Rendering. For a camera ray $\mathbf{x}(t) = \mathbf{c} - t\omega_o$ passing through a NeRV, the volume rendering integration is divided into four parts:

1. The ray is divided into 256 stratified samples at which the volume densities, surface normals and the BRDF parameters are calculated.

$$\sigma = \mathbf{F}_\Theta(\mathbf{x}(t))$$

$$\mathbf{n} = \bigtriangledown_x \mathbf{F}_\Theta(\mathbf{x}(t))$$

$$(\mathbf{a}, \gamma) = \mathbf{F}_\Psi(\mathbf{x}(t))$$

2. Each point along the ray is shaded using the estimated integral of direct illumination. This is achieved by first, generating the known environment lighting $E(\mathbf{x}(t), -\omega_i)$ and multiplying it by the predicted visibility $\tilde{V}_\phi(\mathbf{x}(t), \omega_i)$ and microfacet BRDF values $R(\mathbf{x}(t), \omega_i, \omega_o)$ and integrated together.

3. Another shading of each point is performed with indirect illumination. The expected camera ray termination depth t' is used to query the visibility MLP and compute secondary surface intersection points $\mathbf{x}$'(t'') which are again multiplied with the multifacet BRDF values and integrated together.

4. The final computed pixel colour using the quadrature rule:

$$L(\mathbf{c}, \omega_o) = \sum_t V(\mathbf{x}(t), \mathbf{c})\alpha(\sigma(\mathbf{x}(t))\delta)L_r(\mathbf{x}(t), \omega_o),$$

$$V(\mathbf{x}(t), \mathbf{c}) = \exp(-\sum_{s<t} \sigma(\mathbf{x}(s))\delta), \quad \alpha(z) = 1 - \exp -z,$$

where δ is the distance between samples along the ray.

Loss calculation. For a random ray $\mathcal{R}'$, three losses are computed - visibility and expected termination depth at each location and in either direction along the ray as supervision for the visibility MLP. The training minimizes the sum of all these three losses:

$$\mathcal{L} = \sum_{\mathbf{r}\in\mathcal{R}} \left\| \tau(\tilde{L}(\mathbf{r}) - \tau(L(\mathbf{r})) \right\|_2^2 + \lambda \sum_{\mathbf{r}'\in\mathcal{R}'\cup\mathcal{R},t} \left(\left\| \tilde{V}_\phi(\mathbf{r}'(t)) - V_\theta(\mathbf{r}'(t)) \right\|_2^2 + \left\| \tilde{D}_\phi(\mathbf{r}'(t)) - D_\theta(\mathbf{r}'(t)) \right\|_2^2 \right), \tag{3.16}$$

where $\tau(x) = x/1 + x$ is a tone-mapping operator, $L(\mathbf{r})$ and $\tilde{L}(\mathbf{r})$ are ground truth and predicted camera ray radiance values respectively, $\tilde{V}_\phi(\mathbf{r})$ and $\tilde{D}_\phi(\mathbf{r})$ are the predicted visibility and expected termination depth from the visibility MLP with current weights ϕ, $V_\theta(\mathbf{r})$ and $D_\theta(\mathbf{r})$ are estimates implied by the shape MLP with current weights θ and $\lambda = 20$ (taken from [Srinivasan et al.(2021)]).

3.5.2 Neural Reflectance Decomposition NeRD

The task of decomposition is a classical graphics problem which becomes more difficult under unconstrained environmental illumination. Furthermore, most recent works of implicit representation do not provide relight abilities while resynthesis. NeRD proposes a technique to provide a combined solution to the two problems by utilising physically-based rendering to decompose the scene into spatially varying BRDF material properties. Furthermore, NeRD also presents a method to transform learned reflectance volume into relightable textured mesh for fast real-time rendering with novel illuminations [Boss et al.(2021)].

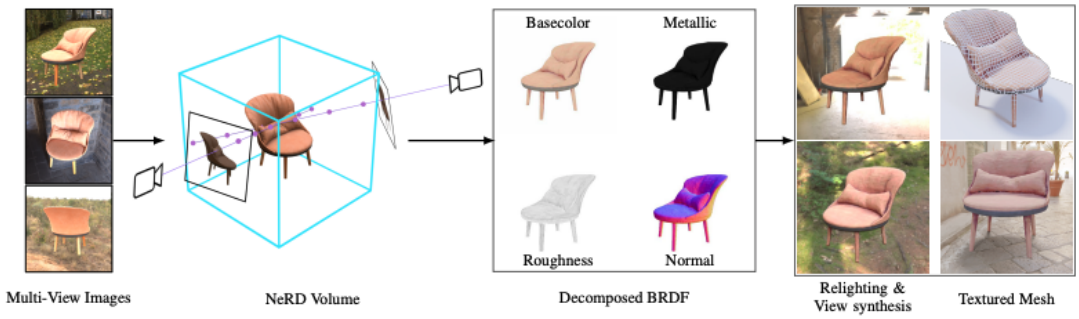


Figure 3.7: **Neural Reflectance Decomposition for Relighting.** For a set of inputs consisting of multiple views of an object under varying illumination, the NeRD model provides its geometry, spatially-varying BRDF parameters, an approximation of incident illumination and a texture mesh. These could be repurposed into renderings under novel illumination in real-time. (figure taken from [Boss et al.(2021)]).

NeRD Model

NeRD provides the functionality of optimising shape, BRDF and illumination of an image collection of an object, captured under fixed or varying lighting, together. When compared to the original NeRF, NeRD makes improvements in the finer network by converting it into a decomposition network. This network retains the lighting-independent reflectance parameters instead of the direct view-dependent colour, figure 3.7.

Problem Setup. The input is defined as a set of q images with s pixels each, $I_j \in \mathbb{R}^{s\times 3}; j \in 1, ..., q$ captured under varying lighting conditions. The model is able to learn a 3D volume with input points $\mathbf{x} = (x, y, z) \in \mathbb{R}^3$ in 3D space, BRDF parameters $\mathbf{z}_{BRDF} \in \mathbb{R}^5$ surface normals $\mathbf{n} \in \mathbb{R}^3$ and density $\sigma \in R$. Spherical Gaussian mixtures (SG) with parameters $\Gamma \in \mathbb{R}^{24\times 7}$ (24 lobes) have been employed to represent an approximation of the environment map.

Sampling Network.

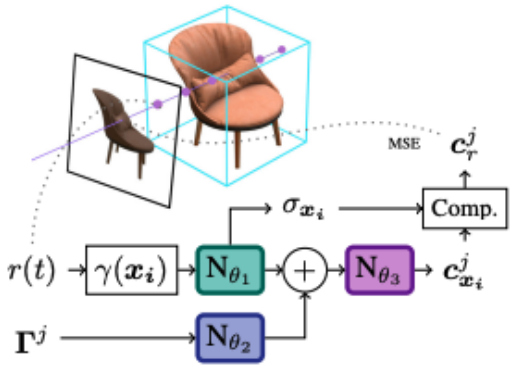


Figure 3.8: **Sampling Network.** The coarse sampling network generates a finer distribution for sampling in the decomposition network (figure taken from [Boss et al.(2021)]).

The purpose of the sampling network, figure 3.8, is to establish a sensible sampling pattern for the decomposition network. Compared to NeRF, the input for NeRD will have images with varying illumination and thus, the network requires illumination Γ^j to create light-dependent colour $\mathbf{C}^j$ for input image I_j. Finally, the estimate of view-independent and light-dependent colour at each point $\mathbf{C}^j$ can be optimized by an MSE:

$$\frac{1}{s}\sum^{s}(\hat{C}^j - C^j)^2$$

On the other hand, as the density, σ is not dependent on the lighting condition, it is extracted in the same way as presented in NeRF as $\mathbf{N}_{\theta_1}$. A second network, known as the compaction network $\mathbf{N}_{\theta_2}$ is used to encode the 24×7 dimensional SG to 16 dimensions. Both results are jointly passed to the final estimation network $\mathbf{N}_{\theta_3}$ which finally outputs the colour values.

Decomposition Network.

The decomposition network, figure 3.9, is added as a step in-between the sampling network and rendering. It estimates the view- and illumination-independent BRDF parameters $\mathbf{z}_{BRDF}$ and surface normals $\mathbf{n}$ at each point. The rendering is maintained as differentiable, such that the loss from the input colour $\hat{C}^j$ can be backpropagated to the BRDF $\mathbf{z}_{BRDF}$, the normal $\mathbf{n}$ and the illumination Γ^j. This aids in the approximation of the general rendering equation:

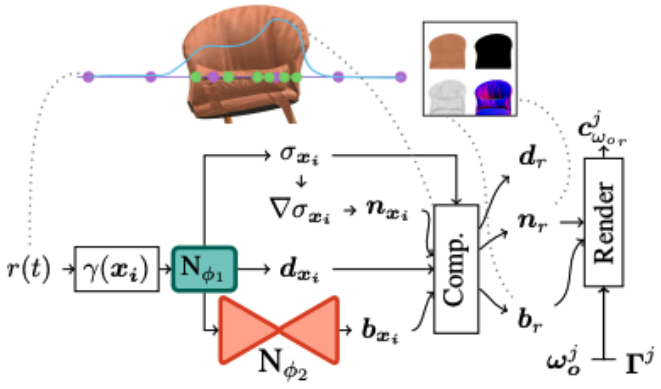


Figure 3.9: **Decomposition Network.** The network performs SVBRDF decomposition at each point in the neural volume. This allows for the generation of plausible BRDFs (figure taken from [Boss et al.(2021)]).

$$L_o(\mathbf{x}, \omega_o) = \int_\Omega f_r(\mathbf{x}, \omega_i, \omega_o) L_i(\mathbf{x}, \omega_i)(\omega_i \cdot \mathbf{n}) \mathrm{d}\omega_i.$$

In order to calculate the shading, the surface normal $\mathbf{n}$ should be computed. As opposed to learning it as one of the outputs from the network, NeRV establishes a relationship between the normal and normalised negative gradient of the density field as this is perpendicular to the implicitly represented surface:

$$\mathbf{n} = -\frac{\bigtriangledown_x \sigma}{\|\bigtriangledown_x \sigma\|}$$

The gradient obtained during optimisation of σ, provides a smoother normal from volume definition which is not possible from learned normal.

Dynamic Range, Tonemapping and Whitebalancing. In order to maintain the Low Dynamic Range of the renderings, the model provides linear outputs which ensure ground truth mapping of sRGB curve and white balance. This is achieved using the images' exposure meta-data encoded in JPEG files based on the Saturation Based Sensitivity auto-exposure calculation. The white balancing helps reduce ambiguity between illumination and material colour and restores the overall intensity of the illumination.

Mesh Extraction. The decomposition approach plays a key role in the extraction of a consistent textured mesh during real-time rendering and relighting. This is not possible for a view-dependent approach like NeRF. The process includes generating a point cloud,

computing a mesh and a texture atlas and finally completing the texture atlas with BRDF parameters.

Training and Loss The model utilises Mean Squared Error (MSE) loss between the input image and the rendering. This is approached differently for the sampling network and the decomposition network where the losses are computed on the RGB prediction and re-rendered result $\mathbf{c}^j$ respectively. The networks are trained for about 300K iterations using the Adam optimizer with a learning rate of 5e-4.

3.5.3 Neural Factorization of Shape and Reflectance-NeRFactor

NeRFactor allows the rendering of novel views of an object under arbitrary environmental lighting and the editing of the object's material properties. The model can produce 3D neural fields of surface normals, light visibility, albedos and BRDFs without supervision and utilising only the re-rendering loss, simple smoothness priors and BRDF prior learned from real-world measurements. As a result, NeRFactor recovers 3D models for free-viewpoint relighting in an unconstrained capture setup for both synthetic and real scenes which outperforms classic and state-of-the-art deep learning techniques.

NeRFactor Model

The model takes multi-view images (and their camera parameters) which are lit with only one unknown illumination condition. These are represented as a set of learned 3D fields from an optimised MLP. NeRFactor, for every 3D location $\mathbf{x}$ on the object's surface, outputs surface normal $\mathbf{n}$, light visibility in any direction $v(\omega_i)$, albedo $\mathbf{a}$ and reflectance z_{BRDF} which altogether explain the observed appearance as shown in figure 3.10.

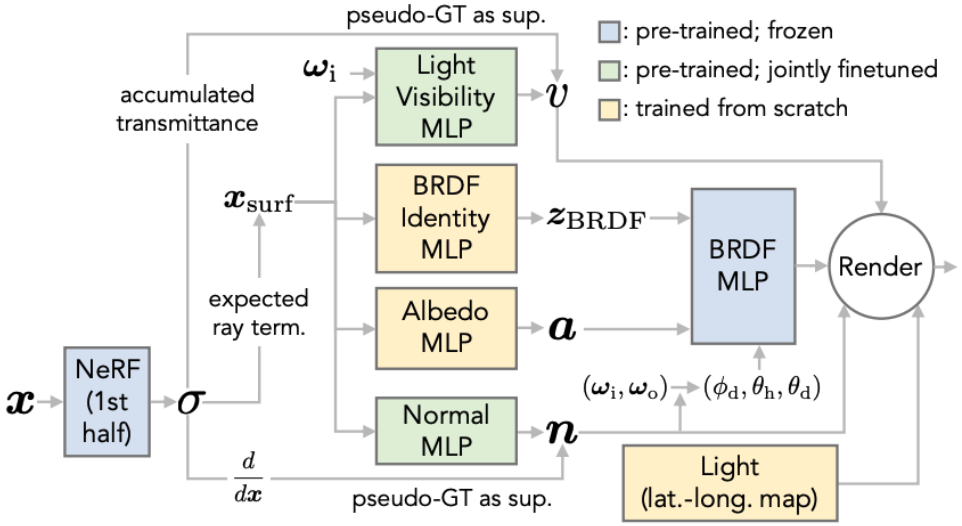


Figure 3.10: **NeRFactor Model.** NeRFactor utilises the volume density from NeRF's model as initialisation so as to provide improved predictions for (figure taken from surface normal $\mathbf{n}$, light visibility v, albedo α, BRDF latent code z_{BRDF}, and the lighting condition[Zhang et al.(2021)]).

Shape. NeRFactor's input to the model is the same as what is used in NeRF and therefore, the output from the NeRF model has been used to compute the initial geometry. As mentioned earlier, NeRF's MLP maps the 3D spatial coordinate and 2D viewing direction to the volume density at that 3D location and colour emittance at the location along the 2D viewing direction. NeRFactor optimises this mapping to calculate the expected surface location along any camera ray, the surface normal at each point on the object's surface and the visibility of light arriving from any direction at each point on the object's surface. In order to achieve the above-mentioned optimisation and re-rendering, the following steps have been added to NeRF:

Surface points. For a given camera and a trained NeRF, the termination location of a ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ which originates at camera $\mathbf{o}$ along direction $\mathbf{d}$ using volume density σ can be calculated as:

$$x_{surf} = \mathbf{o} + \left(\int_0^\infty T(t)\sigma(\mathbf{r}(t))tdt \right) \mathbf{d}, \tag{3.17}$$

where $T(t) = -\exp \int_0^t \sigma(\mathbf{r}(s))ds$ is the probability of a ray traveling distance t without being blocked. Finally, the result is repurposed such that the geometry lies on the surface extracted

from the optimised NeRF. Thus, providing efficient relighting during training and inference.

Surface normals. As mentioned previously, the surface normals can be calculated as the negative normalised gradient of volume density σ w.r.t. $\mathbf{x}$. However, such normals tend to be noisy and produce bumpy artefacts during rendering. Thus, there is a need to re-parameterise the normals using another MLP f_n which maps from any location $\mathbf{x}_{surf}$ on the surface to a denoised surface $\mathbf{n} : f_n : \mathbf{x}_{surf} \mapsto \mathbf{n}$. The optimisation of this function ensures that the output normals are (i) close to the normals produced from the pre-trained NeRF, (ii) vary smoothly in 3D space and (iii) reproduce the observed appearance of the object. Thus the loss function for the MLP can be defined as:

$$\mathcal{L}_n = \sum_{x_{\text{surf}}} \left(\tfrac{\lambda_1}{3} \left\| \boldsymbol{f}_n(x_{\text{surf}}) - \boldsymbol{n}_a(x_{\text{surf}}) \right\|_2^2 + \tfrac{\lambda_2}{3} \left\| \boldsymbol{f}_n(x_{\text{surf}}) - \boldsymbol{f}_n(x_{\text{surf}} + \epsilon) \right\|_1 \right), \tag{3.18}$$

where ϵ is a random 3D displacement from x_{surf} sampled from a zero-mean Gaussian with standard deviation 0.01 and hyperparameters λ_1 and λ_2 as 0.1 and 0.05 respectively (values taken from [Zhang et al.(2021)]).

Reflectance. The BRDF model $\mathbf{R}$ encompasses a diffuse component (Lambertian) dependent on the albedo $\mathbf{a}$ and specular spatially-varying BRDF $\boldsymbol{f}_r$ (it is defined for any location on the surface $\mathbf{x}_{\text{surf}}$ with incoming light direction ω_i and outgoing direction ω_o) learned from real-world reflectance:

$$R(\mathbf{x}_{\text{surf}}, \omega_i, \omega_o) = \frac{\boldsymbol{a}(\mathbf{x}_{\text{surf}})}{\pi} + \boldsymbol{f}_r(\mathbf{x}_{\text{surf}}, \omega_i, \omega_o). \tag{3.19}$$

NeRFactor is able to recover plausible reflectance functions by utilising two elements - first, a learned reflectance function which has been pre-trained to recreate a variety of empirically observed real-world BRDFs and second, a latent space for learning real-world BRDFs.

Albedo. The albedo a at any surface location $\mathbf{x}_{\text{surf}}$ can be defined as an MLP $\boldsymbol{f}_a : \mathbf{x}_{\text{surf}} \mapsto \boldsymbol{a}$. As the observations are single source illuminated, the model relies on simple spatial smoothness priors. The loss function for $\boldsymbol{f}_a$ is represented as:

$$\mathcal{L}_a = \lambda_5 \sum_{\mathbf{x}_{\text{surf}}} \frac{1}{3} \left\| \boldsymbol{f}_a(\mathbf{x}_{\text{surf}}) - \boldsymbol{f}_a(\mathbf{x}_{\text{surf}} + \epsilon) \right\|_1, \tag{3.20}$$

where ϵ is a random 3D perturbation, hyperparameter λ_5 is 0.05 (value taken from [Zhang et al.(2021)])

and the output from $\boldsymbol{f}_a$ is used as albedo for Lambertian reflectance.

Rendering. Mathematically, the rendering equation of NeRFactor can be defined as:

$$L_o(x, \omega_o) = \sum_{\omega_i} \left(\frac{f_a(x)}{\pi} + f'_r(f_z(x), g(f_n(x), \omega_i, \omega_o)) \right) L_i(x, \omega_i)(\omega_i \cdot f_n(x)) \triangle \omega_i \qquad (3.21)$$

where $L_o(x, \omega_o)$ is the outgoing radiance at x as viewed from ω_o, $L_i(x, \omega_i)$ is the incoming radiance, masked by the visibility $f_v(x, \omega_i)$ arriving at x along ω_i and $\triangle\omega_i$ is the solid angle of lighting sample at ω_i.

3.6 Neural Implicit Representations

The Neural Network methods, NeRF discussed earlier though provides a simple and straightforward technique to reconstruct 3D volume from a sparse set of inputs, it fails to provide sharp renderings due to complex surfaces like non-Lambertian surfaces or thin structures. This can be attributed to conspicuous noise in planar regions. Furthermore, as NeRF has been designed to synthesize novel views rather than surface reconstruction, it tends to learn the volume density field only which makes it difficult to extract high-quality surfaces.

3.6.1 Implicit Differentiable Renderer (IDR)

NeRFs provide a simple solution to the fundamental computer vision problem of learning 3D shapes from 2D images using differentiable rendering techniques. These are mostly based on ray casting, tracing or rasterization. However, 3D geometry is best represented using point clouds, triangle meshes or implicit representation defined over volumetric grids. From these, implicit representations possess the obvious advantage of being flexible while representing surfaces with arbitrary shapes and topologies without the need for ground truth mesh. The techniques developed before IDR lacked the incorporation of lighting and reflectance properties as well as were unable to deal with trainable camera locations or orientations.

Implicit Differentiable Rendering have been developed as an answer to this gap between differentiable rendering and implicit representations. IDR presents an end-to-end architecture system that can learn 3D geometries from masked 2D images and rough camera estimates without any additional supervision [Yariv et al.(2020)]. The colour of each can be represented as a differentiable function using three unknowns in a scene - the geome-

try, its appearance (a collective of all factors which define the surface light field excluding surface BRDF and lighting conditions) and the cameras. The approximation of 3D shape representation from zero level set of neural networks has been achieved for surface light fields which can be represented as continuous functions of the point on the surface, surface normal, viewing direction and global shape features for complex appearances.

IDR Model

The model should reconstruct the geometry of an object from masked 2D images with rough or noisy camera information. The geometry as the zero level set of a neural network (MLP) f,

$$S_\theta = \left\{ x \in \mathbb{R}^3 \mid f(x;\theta) = 0 \right\} \tag{3.22}$$

with learnable parameters $\theta \in \mathbb{R}^m$. The function f has been regularized using a signed distance function (SDF) model which has multiple benefits which are (i) it allows an efficient ray-casting with sphere tracing algorithm and (ii) implicit geometric regularization (IGR) favours smooth and realistic surfaces.

IDR Forward Model: For a pixel p with an associated input image, the ray through pixel p can be defined as

$$R_p(t) = \{o_p + td_p \mid t \geq 0\} \tag{3.23}$$

where $\mathbf{o}_p$ denotes the unknown center of respective camera and d_p represents the direction of the ray. The first intersection of the ray and surface, the normal and global geometry feature vector can be represented as

$$\hat{x}_p = \hat{x}_p(\theta, t) \tag{3.24}$$

$$\hat{n}_p = \hat{n}_p(\theta) \tag{3.25}$$

$$\hat{z}_p = \hat{z}_p(\hat{x}_p; \theta) \tag{3.26}$$

This, in turn, is used to define the rendered colour of the pixel $C_p = C_p(\theta, \gamma, t)$. Finally, the IDR forward model is defined as:

$$C_p(\theta, \gamma, t) = F_\Theta(\hat{x}_p, \hat{n}_p, \hat{z}_p, v_p; \gamma), \tag{3.27}$$

where F_Θ is a Multi-Layer Perceptron (MLP).

Masked rendering. Masks are binary images that indicate whether the object of interest is occupied at a single pixel. These act as valuable tools for 2D supervision for 3D reconstruction and can be represented as:

$$S(\theta, t) = \begin{cases} 1 \; R(t) \bigcap S_\theta \neq \emptyset \\ 0 \; otherwise \end{cases} \tag{3.28}$$

As the function is neither differentiable nor continuous in θ, t approximation can only be performed:

$$S_k(\theta, t) = sigmoid\left(-k \min_{t \geq 0} f(c + tv; \theta)\right), \tag{3.29}$$

where $k > 0$ is an arbitrary parameter. The equation can be differentiated w.r.t. c, v using the envelope theorem.

Loss. Given $I_p \in [0, 1]^3, O_p \in 0, 1$ are the RGB and mask values respectively, the total loss function has the form:

$$\mathcal{L}(\theta, \gamma, t) = loss_{RGB}(\theta, \gamma, t) + \rho loss_{MASK}(\theta, \gamma, t) + \gamma loss_E(\theta) \tag{3.30}$$

$$\mathcal{L}_{RGB}(\theta, \gamma, t) = \frac{1}{\mid P \mid} \sum_{p \in P^{in}} \mid I_p - L_p(\theta, \gamma, t) \mid, \tag{3.31}$$

$$\mathcal{L}_{MASK}(\theta, t) = \frac{1}{\mid \alpha P \mid} \sum_{p \in P^{out}} CE(O_p, S_{p,\alpha}(\theta, t) \tag{3.32}$$

where CE is the cross-entropy loss. The final step is to approximate f as a signed distance function with Implicit Geometric Regularization using Eikonal regularization.

3.6.2 Neural Implicit Surfaces (NeuS)

IDR produces impressive reconstruction, however, it fails to reconstruct objects with complex structures with abrupt depth changes. This is due to the fact that IDR, as mentioned earlier, only takes a single surface intersection point into consideration for each ray making the gradient too local for effective backpropagation. As a result, the optimization falls into poor local minimum when there is an abrupt change in depth in the image. Another point of concern is the requirement for mask supervision for final convergence for a valid surface.

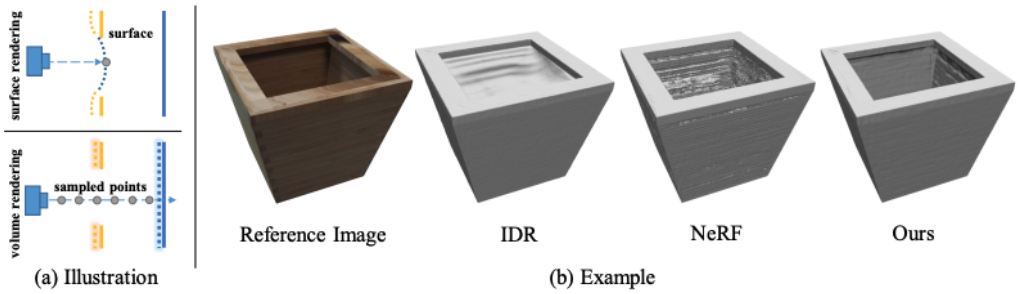


Figure 3.11: **NeuS Illustration**. (a) A diagram illustrating the difference between surface rendering and volume rendering. (b) Results from a study exhibiting renderings from different methods for toy example of a bamboo planter (figure taken from [Wang et al.(2021)]).

NeuS, a neural rendering scheme for multi-view surface reconstruction, uses the signed distance function (SDF) for surface representation (also used by IDR) and a novel volume rendering scheme to learn a neural SDF representation. However, by introducing a density distribution induced by SDF, the volume rendering provides both an accurate surface representation using a neural SDF model and a robust network training of surfaces with abrupt depth changes. One of the most important reasons for choosing NeuS is that it is capable of reconstructing complex 3D objects and scenes with severe occlusion and delicate structures without any foreground mask supervision, thus outperforming IDR [Yariv et al.(2020)] and NeRF [Mildenhall et al.(2020)], as shown in figures 3.11 and 3.12.

NeuS Model

The model is defined by first taking an input of a set of posed images I_k of a 3D object to reconstruct its surface S. The reconstructed surface is represented by a zero-level set of neural implicit SDF. The model weights are learned through a volume rendering technique which minimizes the difference between the rendered images and the input images.

Scene Representation. The scene of an object to be reconstructed is represented using two functions, (i) $f : \mathbb{R}^3 \to \mathbb{R}$ that maps a spatial position $\mathbf{x} \in \mathbb{R}^3$ to its signed distance to the object, and (ii) $c : \mathbb{R}^3 \times \mathbb{S}^2 \to \mathbb{R}^3$ which encodes the colour associated with a point $\mathbf{x} \in \mathbb{R}^3$ and viewing direction $\mathbf{v} \in \mathbb{S}^2$. These functions have been represented as two sets of Multi-Layer Perceptrons (MLPs). Finally, the surface to be reconstructed can be defined as

a set of zero-level set of its SDF:

$$S = \mathbf{x} \in \mathbb{R}^3 \mid f(\mathbf{x}) = 0 \tag{3.33}$$

Volume rendering for SDF training is introduced using a probability density function known as S-density $\phi_s(f(\mathbf{x}))$ where $f(\mathbf{x}, \mathbf{x} \in \mathbb{R}^3)$ is the signed distance function and follows logistic density distribution (chosen for computational convenience)

$$\phi_s(x) = \frac{se^{sx}}{(1 + e^{-sx})^2}$$

which is a derivative of the Sigmoid function

$$\Phi_s(x) = (1 + e^{-sx})^{-1} \quad i.e. \quad \phi_s(x) = \Phi_s'(x)$$

Successful convergence of the model and minimization of the loss function is a characteristic of the induced S-density $\phi_s(f(\mathbf{x}))$ assuming a relatively high value near the surface S.

Rendering. The ray emitted from a given pixel can be represented as $\mathbf{p}(t) = \mathbf{o} + t\mathbf{d} \mid t \geq 0$ where $\mathbf{o}$ is the centre of the camera and $\mathbf{d}$ is the unit direction of the ray vector. Using this, the accumulated colours along the ray are defined as

$$C(\mathbf{o}, \mathbf{d}) = \int_0^{+\infty} w(t)c(\mathbf{p}(t), \mathbf{d})\mathrm{d}t, \tag{3.34}$$

where $C(\mathbf{o}, \mathbf{v})$ is the colour output of the pixel, $w(t)$ is a weight for point $\mathbf{p}(t)$ and c(p(t), d) is the colour at the point $\mathbf{p}$ along the viewing direction $\mathbf{d}$.

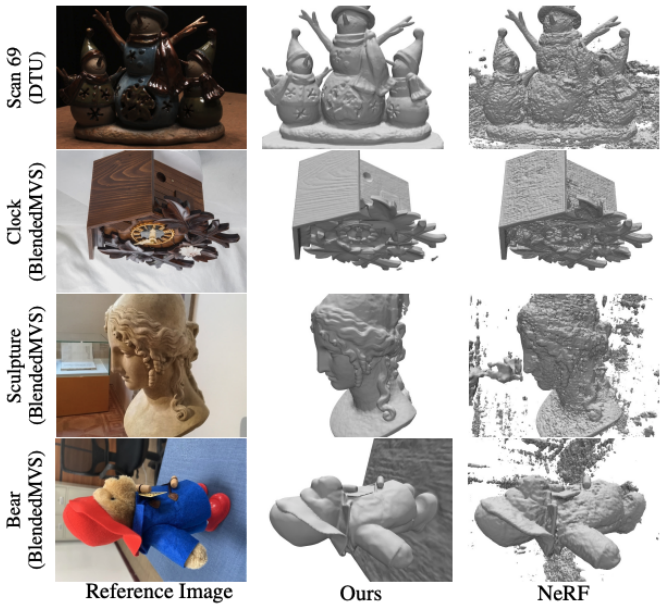


Figure 3.12: Comparisons of surface reconstruction results from various methods without mask supervision (figure taken from [Wang et al.(2021)]).

The weight function $w(t)$ should be defined such that it can aid the SDF representation from 2D images to build an appropriate connection between output colours and SDF. There are two strict requirements:

1. **Unbiased.** An unbiased weight function ensures the intersection of the camera rays with the zero-level set of SDF contributes most to the pixel colour. For a camera ray $\mathbf{p}(t)$, $w(t)$ should attain a locally maximal value at the point of intersection with the surface, $\mathbf{p}(t*)$ and

$$f(\mathbf{p}(t*)) = 0$$

showing that $\mathbf{p}(t*)$ is on the zero-level set of the SDF ($\mathbf{x}$).

2. **Occlusion-aware.** The occlusion-aware property assures that while the ray passes through multiple surfaces, the colour of the surface nearest to the camera computes the output colour. When two points t_0 and t_1 exhibit the same SDF value, the point nearer to the viewpoint would have a larger contribution to the output colour than the other point.

$$f(t_0) = f(t_1), \quad w(t_0) > 0, \quad w(t_1) > 0 \quad and \quad t_0 < t_1$$

then it should satisfy

$$w(t_0) > w(t_1)$$

Weight Function Definition. NeuS defines the function $w(t)$ as an unbiased weight function of normalised S-density values as weights

$$w(t) = \frac{\phi_s(f(\mathbf{p}(t)))}{\int_0^{+\infty} \phi_s(f(\mathbf{p}(u)))\mathrm{d}u} \tag{3.35}$$

where $\phi_s(f(\mathbf{p}(t)))$ defines the volume density in classical volume rendering format morphed to accommodate S-density values.

However, the construction is still not occlusion-aware that is if a ray intersects with two surfaces, the resulting weight function $w(t)$ will be an equal composition of colours represented by both the surfaces without any consideration for occlusions. Thus, for an additional occlusion-aware weight function, a first-order approximation of the SDF of the construction is required. This is defined using an opaque density function $\rho(t)$ to compute the weight function as

$$w(t) = T(t)\rho(t), \quad where \quad T(t) = exp\left(-\int_0^t \rho(u)\mathrm{d}u\right) \tag{3.36}$$

where $T(t)$ denotes the accumulated transmittance along the ray.

Opaque Density Function Definition. The generalised opaque density $\rho(t)$ for multiple plane intersections along the ray $\mathbf{p}(t)$ has been defined in NeuS [Wang et al.(2021)] as

$$\rho(t) = \max\left(\frac{-\frac{\mathrm{d}\Phi_s}{\mathrm{d}t}(f(\mathbf{p}(t)))}{\Phi_s(f(\mathbf{p}(t)))}, 0\right). \tag{3.37}$$

Training. The difference between the rendered colours and ground truth colours needs to be minimized without any 3D supervision but masks can be used. The global loss function can be defined as

$$\mathcal{L} = \mathcal{L}_{color} + \lambda\mathcal{L}_{reg} + \beta\mathcal{L}_{mask} \tag{3.38}$$

The colour loss $\mathcal{L}_{color}$ is defined as

$$\mathcal{L}_{color} = \frac{1}{m}\sum_k R(\hat{C}_k, C_k) \tag{3.39}$$

This is the same L1 loss as discussed before for IDR [Yariv et al.(2020)]. An additional Eikonal term has been used for the sample points to regularise the SDF

$$\mathcal{L}_{reg} = \frac{1}{nm} \sum_{k,i} (\left\| \bigtriangledown f(p_{\hat{k},i}) \right\|_2 - 1)^2. \tag{3.40}$$

where m and n are arbitrary parameters taken from [Wang et al.(2021)]. Finally, the optional mask loss $\mathcal{L}_{mask}$ is defined as

$$\mathcal{L}_{mask} = \mathbf{BCE}(M_k, \hat{O}_k) \tag{3.41}$$

where $\hat{O}_k = \sum_{i=1}^{n} T_{k,i}\alpha_{k,i}$ is the sum of weights along the camera ray and BCE is the binary cross entropy loss.

3.7 Conclusion

In this chapter, we discussed the technologies for neural representation and 3D reconstruction. We provided an in-depth explanation of NeRF and its predecessors - highlighting LLFF as they are crucial for understanding our pipeline. We further extended the reader's scope by providing a gentle introduction to multiple NeRF variants aimed to improve various aspects of the original NeRF and a following critical review to introduce the knowledge gap we have identified. There is a need for a singular system to handle the different functionality provided by models introduced before. We aim to limit the scope to object relighting and accurate surface texture reconstruction through our work, thus, we have provided detailed information about state-of-the-art works which aim to address similar problems - NeRV, NeRD, NeRFactor, IDR and NeuS. In the upcoming chapters, we will develop our idea of a singular model in a step-by-step process with the first iteration being a simple relightable model built with minimal changes to the original NeRF model.

Chapter 4

Relightable NeRF

In the previous chapter, we introduced the intricacies of Neural Radiance Fields providing us with a platform to understand and initiate minute incremental changes to the original NeRF's architecture to accommodate the new functionalities we have planned. In this chapter, we will be implementing the first novel contribution, Relightable NeRF.

As discussed previously in Chapter 3, the appearance of an object is characterised by the Bidirectional Reflectance Distribution Function, self-occlusion and reflection towards the viewer. However, the geometry-inclined model, NeRF's architecture is unable to render the physical properties (normals, albedo and shadow maps) of these lighting scenarios. The introduction of illumination-aware architecture for NeRF would enable it to seamlessly bake in the captured object into any artificial or real scene with given illumination information and would be a key tool in creating hyper-realistic object placement in VR or AR simulations. Thus, this chapter aims to explicitly define the lighting conditions by introducing a light-dependent component to the model as well as the training set.

We will be introducing an approach which requires an extra set of inputs - the images are now required to have poses as well as illumination information and an attached pose and illumination direction matrices which are compatible with Relightable NeRF. In terms of architecture, the changes are made in the input to the networks - in addition to the view direction, we incorporate light direction as inputs to the Geometry network and Colour network.

4.1 NeRF Revisited

The original NeRF model can be written as two MLPs:

1. $f_{\text{geo}}^{\Theta_{\text{geo}}} : \mathbf{x} \mapsto (\mathbf{z}, \sigma)$ with learnable parameters Θ_{geo} which maps a 3D location $\mathbf{x} = (x, y, z)$ to a latent code $\mathbf{z} \in \mathbb{R}^d$ and density $\sigma \in \mathbb{R}_{\geq 0}$.

2. $f_{\text{col}}^{\Theta_{\text{col}}} : (\mathbf{z}, \mathbf{d}) \mapsto \mathbf{c}$ with learnable parameters Θ_{col} which maps the viewing direction $\mathbf{d} \in \mathbb{R}^3$, $\|\mathbf{d}\| = 1$, and latent code to a colour $\mathbf{c} = (r, g, b)$.

In general, the first MLP $f_{\text{geo}}^{\Theta_{\text{geo}}}$ captures the geometry of the scene in the form of volumetric density which depends only on position. The second MLP $f_{\text{col}}^{\Theta_{\text{col}}}$ additionally depends on view direction and captures the appearance of the scene under a *single, fixed (but arbitrary) illumination condition.* As the network f_Θ performs subpar when representing high-frequency variations in colour and geometry which can be attributed to deep networks being biased towards learning lower frequency functions [Rahaman et al.(2019)]. Therefore, operating on $xyz\theta\phi$ requires working with sinusoidal functions which are composite functions $f_\Theta = f_\Theta^{'} \circ \gamma$ such that the MLP can easily approximate a higher frequency function [Mildenhall et al.(2020)]. γ is sinusoidal mapping from $\mathbb{R}$ into higher dimensional space $\mathbb{R}^{2L}$ and can be defined as

$$\gamma(p) = (\sin(2^0\pi p), \cos(2^0\pi p), \cdots, \sin(2^{L-1}\pi p), \cos(2^{L-1}\pi p)). \tag{4.1}$$

The precise architecture used in the original method is shown in Figure 4.1. We now describe more of the specific details used in the original model, many of which are critical for good performance, in the following subsections.

4.1.1 Initialising NeRF Model

In order to initialise the NeRF model, we require input channels with embedded sinusoidal functions which enhance the learning capabilities of the model. This makes the number of input channels as five - three for RGB and two for view direction. Following this, two models are created - the *coarse* model and the *fine* model. Both the models are similar except for the fine model taking N_importance (the number of points to be considered in between sampling points where the scene density is higher) into consideration. For a 360°dataset, settings

like `no_ndc` (Normalised Device Coordinate system which is only used for grid dataset) and `lindisp` (linear displacement) are crucial - we will discuss them later.

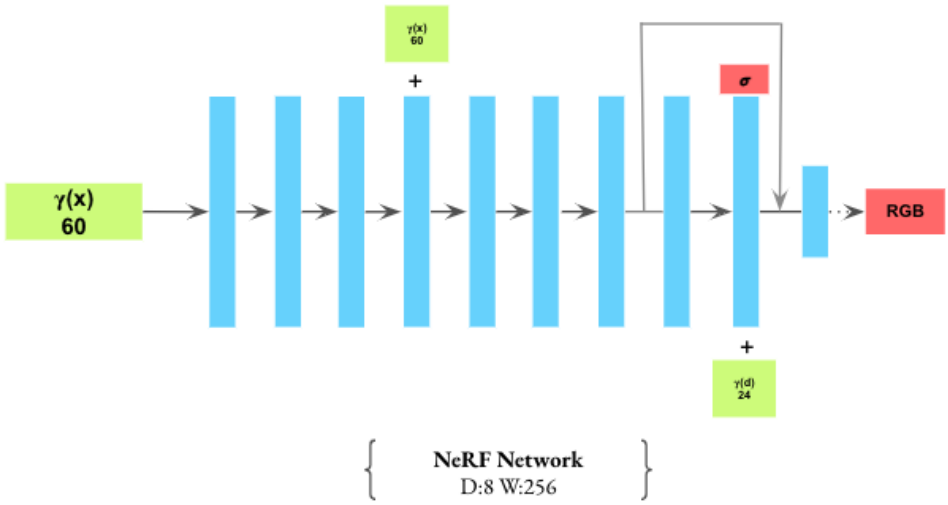


Figure 4.1: **Vanilla NeRF Architecture**: The model consists of two networks - the **Geometry Network** and **Colour Network** with 256 hidden units in each layer. The layers are fully connected with ReLU activation represented by black arrows except for the layer with skip connection shown by the light gray arrow and sigmoid activation shown by the dotted arrow. The inputs include the 3D location **x** and view direction **d** shown in green boxes in their positional encoded forms $\gamma(x)$ and $\gamma(d)$. The outputs include the colour **RGB** and volume density α shown in red boxes (figure adapted from [Mildenhall et al.(2020)]).

The positional encoding of the input location ($\gamma(\mathbf{x})$) is passed through 8 fully-connected ReLU layers, each with 256 channels. It also includes a skip connection that concatenates this input to the fifth layer's activation. An additional layer outputs the volume density σ (which is rectified using a ReLU to ensure that the output volume density is non-negative) and a 256-dimensional feature vector. This feature vector is concatenated with the positional encoding of the input viewing direction ($\gamma(\mathbf{d})$), and is processed by an additional fully-connected ReLU layer with 128 channels. A final layer (with a sigmoid activation) outputs the emitted RGB radiance at position **x**, as viewed by a ray with direction **d**.

4.1.2 Ray Batching

Before the training, NeRF generates multiple batches of rays with random ray origins and directions. The randomness is dependent on (i) camera height H, (ii) camera width W

which determines the grid where rays originate from (iii) camera focal length f and (iv) camera-to-world matrix which computes the ray origin ro and direction rd in this grid. The RGB values and training images rgb are also added to the final batch of the form `[(N-1)*H*W, ro+rd+rgb, 3]` where $(N-1)*H*W$ represents the total number of pixels in the dataset, $ro+rd+rgb$ represents the total number of rays in the batch and the value three represents each of the RGB channels.

4.1.3 Training and Loss Optimisation

Once the training begins, NeRF makes predictions for colour, disparity and accumulated opacity at every iteration step. This is performed using the `render()` function which requires the following inputs - (i) dimensions of the image `H*W`, (ii) focal length of the pinhole camera, (iii) ray batch, (iv) camera-to-world transformation matrix and (v) near-far boundaries.

Volume rendering is performed on every individual ray by sampling linearly in inverse depth (determined by `N_samples` and `lindisp` configuration values) and evaluating the coarse model at those points. The `N_importance` determines the additional sampling points (where the scene information is dense) to evaluate the fine model. The hierarchical sampling helper built on the `weights`, proves to be extremely useful in determining this order. The output provided by both models includes an estimated colour of RGB ray `rgb_map`, a disparity map `disp_map`, the sum of weights along each ray `acc_map`, weights assigned to each sampled colour `weights` and the estimates distance to object `depth_map`.

The predicted colour is compared with the training data to compute an image loss which is the mean squared error (MSE) and a peak signal-to-noise ratio (PSNR) - performed for both fine as well as coarse models. The loss is then utilised as one of the variables for optimisation using ADAM optimiser.

4.2 Relightable NeRF

As a first step towards endowing NeRF with an understanding of how appearance varies with illumination, we propose a straightforward extension to produce a relightable version of NeRF. Given appropriate training data, we can simply condition the second MLP (the colour network) on a parametric representation of illumination.

The parametric representation could, in principle, be chosen from several possibilities. These could include spherical harmonic lighting [Green(2003)] or spherical Gaussian lighting [Huo et al.(2020)]. The sole requirement is that we have paired training images, that is, for each input image we must know the corresponding ground truth lighting parameters that describe the lighting in that image.

We select the simplest possible lighting environment, where the objects are lit with a single-point light source and the direction of the lighting is known. This is also consistent with photometric stereo datasets. Under this assumption, our proposed model represents scenes with the following two MLPs:

1. $f_{\text{geo}}^{\Theta_{\text{geo}}} : \mathbf{x} \mapsto (\mathbf{z}, \sigma)$ with learnable parameters Θ_{geo} which maps a 3D location $\mathbf{x} = (x, y, z)$ to a latent code $\mathbf{z} \in \mathbb{R}^d$ and density $\sigma \in \mathbb{R}_{\geq 0}$.

2. $f_{\text{col}}^{\Theta_{\text{col}}} : (\mathbf{z}, \mathbf{d}, \mathbf{s}) \mapsto \mathbf{c}$ with learnable parameters Θ_{col} which maps the viewing and lighting directions $\mathbf{d} \in \mathbb{R}^3$, $\|\mathbf{d}\| = 1$ and $\mathbf{s} \in \mathbb{R}^3$, $\|\mathbf{s}\| = 1$, and latent code to a colour $\mathbf{c} = (r, g, b)$.

The colour network, $f_{\text{col}}^{\Theta_{\text{col}}}(\mathbf{z}, \mathbf{d}, \mathbf{s})$, is now clearly related to the underlying physics. Ignoring shadowing and global illumination effects, it effectively represents the BRDF when evaluated with incident direction $\mathbf{s}$ and outgoing direction $\mathbf{d}$. However, it must be noted that we impose no physical constraints or priors on this network. It remains a black box that is free to produce any values as long as they closely fit the training data. Meaningful interpolation between observed light/viewer configurations or even extrapolation beyond them relies entirely on the smoothness of the underlying MLP representation.

The precise architecture used in the original method is shown in Figure 4.2. We now describe more specific details of our idea.

4.2.1 Initialising Relightable NeRF Model.

Following the vanilla NeRF model, we create 27 input channels with embedded sinusoidal functions. These include 3^2 channel each for input RGB values, view direction and illumination direction. The values are fed into a Relightable NeRF model which consists of two MLP networks - a view-dependent Geometry Network and a view+light dependent Relightable Network.

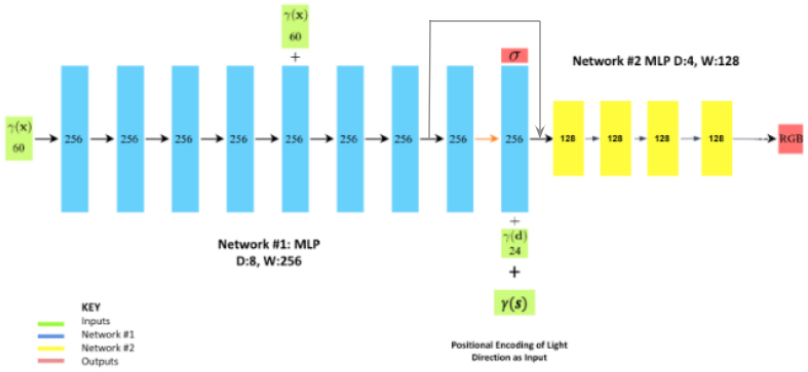


Figure 4.2: **Relightable NeRF architecture**: The model consists of two networks - the **Geometry Network** and **Relightable Network** with 256 hidden units in each layer. The layers are fully connected with ReLU activation represented by black arrows except for the layer with skip connection shown by the light gray arrow. The inputs include the 3D location **x**, view direction **d** and light direction **s** shown in green boxes in their positional encoded forms $\gamma(x)$ $\gamma(d)$ and $\gamma(s)$. The outputs include the colour **RGB** and volume density α shown in red boxes.

The Geometry Network follows the same configurations as vanilla NeRF as shown in figure 4.1. The new Relightable Network consists of four hidden layers which takes a concatenated input of the previous Geometry Network's output latent code z, view directions and illumination direction and outputs the raw RGB values.

4.2.2 Ray Batching.

As discussed earlier in the vanilla NeRF section, NeRF generates multiple batches of rays with random ray origins and directions. We further introduce an additional channel of light direction origin which, after being stacked over by the training images, takes the form `(N-1)*H*W, ro+rd+ld+rgb, 3]`.

The further details on the implementation and usage of the code have been provided in the Appendices section 8.2.

4.3 Experiments

We will now be discussing our results from experimentation on varied datasets. In addition to the image sets provided by NeRF (Fern and LEGO), we tested creating the neural 3D

representations on multiple datasets available in the public domain and some custom images shot from hand-held devices. The image sets are captured either in grid form or in inward facing 360° view.

4.3.1 Ablation Study for Vanilla NeRF

Study 1: In-the-wild Data

In this study, we examine the proficiency of the vanilla NeRF pipeline on multiple coarse and inaccurately recorded datasets. We captured these in-the-wild data using a variety of small everyday objects from an iPhone XS camera. Each object has about $15 - 20$ input images with camera poses data generated through COLMAP. The input image dataset for an object can be planar or stereo.

1. **Potted Plant**: The dataset is planar like the one used during the vanilla NeRF initial run, thus, we followed the default configurations. The reconstruction is shown in figure 4.3.

2. **Teabox**: This is a stereo dataset and thus require additional configuration setting. We use the recommended flags `--no_ndc --spherify --lindisp`. The reconstruction is shown in figure 4.4.

We replicated the results for grid as well as 360° capture for in-the-wild-images. The reconstruction for grid are plausible, however, the 360° data tends to generate cloudy artefacts. This is because while the object of concern does rotate, the background remains fixed. It also helped us identify the cause of similar artefacts observed for DiLiGenT object reconstruction.

Figure 4.3: NeRF reconstruction for grid capture: potted plant

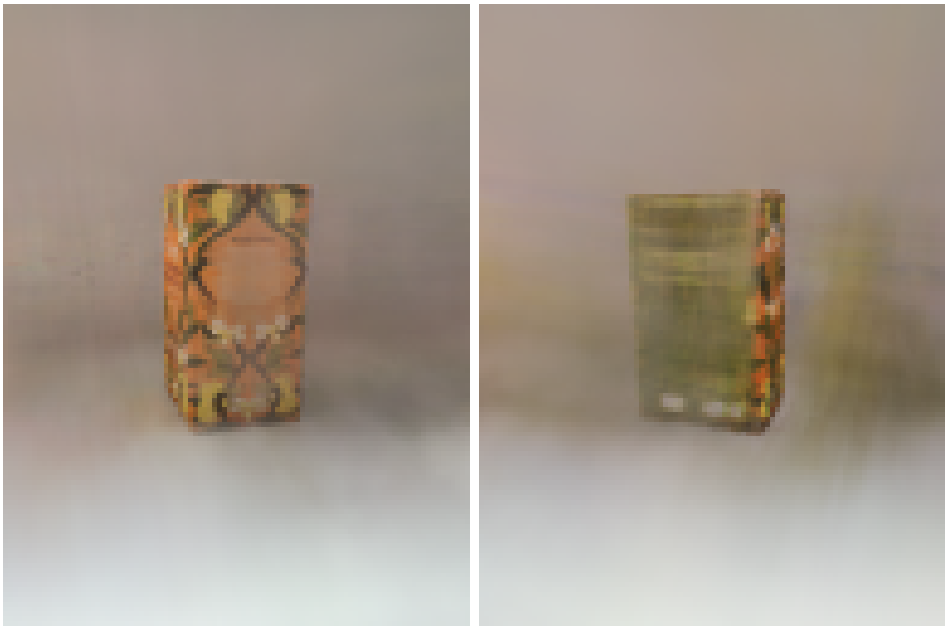


Figure 4.4: NeRF reconstruction for 360 °capture: teabox

Study 2: Synthetic Data Stanford Bunny

We use the MatlabRenderer [Bas and Smith(2019)] to generate a dataset compatible with NeRF. Due to its synthetic nature, there is no background noise or artefacts. The NeRF reconstruction for this has proved to be the perfect starting point for a relightable version of NeRF. The reconstruction is shown in figure 4.5.

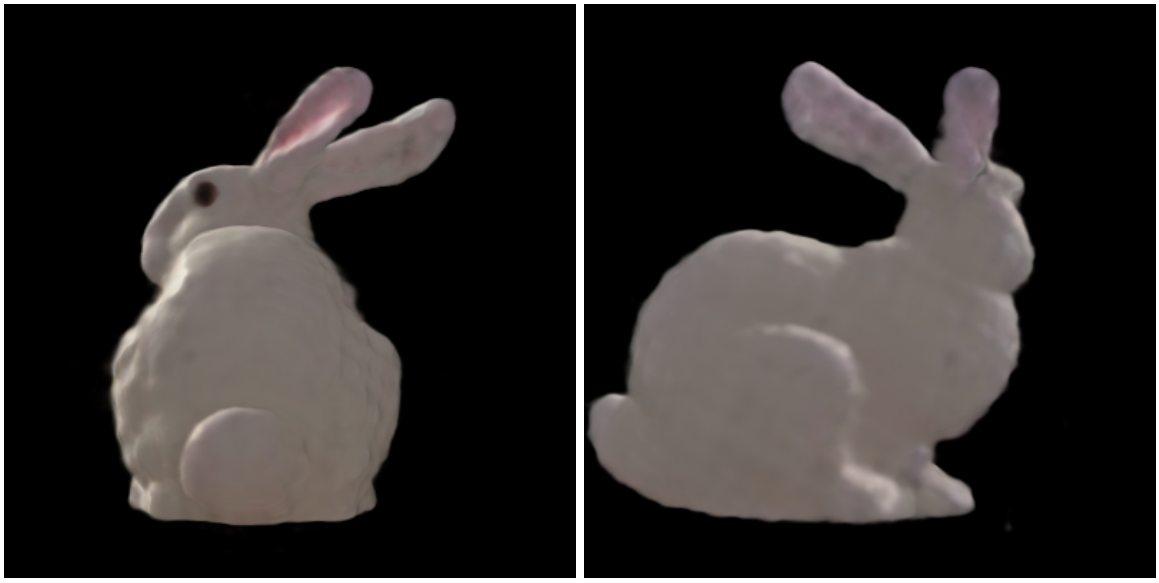


Figure 4.5: NeRF reconstruction for 360°capture: bunny

Study 3: Multiview Photometric Stereo Data - DiLiGeNT-MV

The DiLiGenT-MV is a photometric stereo dataset containing object images with complex BRDFs taken from 20 different views. Each viewpoint image is in a 16-bit integer PNG format with a resolution of 612×512 from 96 different lighting directions. The 'ground truth' mesh is provided in a ply format and the calibrating information is stored as a Matlab matrix (acquired using 'Camera Calibration Toolbox for Matlab(Jean-Yves Bouguet)' [Bouguet(2022)]) [Li et al.(2020b)]. For our reconstruction, we adopted two directions - vanilla NeRF pipeline in section 4.1 and building custom LLFF output with known poses (as explained in section 8.2).

In this study, we examine the ability of vanilla NeRF to learn and output renderings for a stereo dataset with accurate poses and compare them with target renderings and mesh reconstructions from conventional methods. We have selected a comparatively difficult-to-reconstruct object known as 'ReadingPNG' from the dataset due to its surface's difficult geometry and specular nature (as mentioned in Section 2.8.2). The experimentation was performed in an incremental process to investigate the ideal settings for training. The process includes the following steps:

- **Raw Images**: These images have been used without any preprocessing and as the dataset is stereo in nature, we have employed additional configuration parameters

`--no_ndc --spherify --lindisp` as used for the teabox. The original images are also not well-illuminated. We have achieved convincing renderings using vanilla NeRF, however, these are riddled with hazy artefacts. It has been suspected that these might occur due to the unevenly lighted background highlighted in the renderings.

- **Averaged Images**: The images for each of the 96 illumination settings have been averaged to obtain a new dataset with 20 images with 20 poses. The process alleviated the under-lit object images. The rendering obtained showed a little improvement from the raw images in terms of the legibility of the objects, however, the hazy artefacts are still prevalent.

- **Masked Images**: The DiLiGeNT-MV objects have been provided with additional mask files which allow us to overlay the masks on the original image to remove the background from the training process. We employed two studies using masks - white mask background and black mask background. The white mask background training does not provide great results as the saturated white pixels overwhelms the training process and leads to a premature convergence to a plain white background without learning any details about the object. The black background training provides better results as compared to the averaged images with fewer hazy artefacts as shown in figure 4.6.

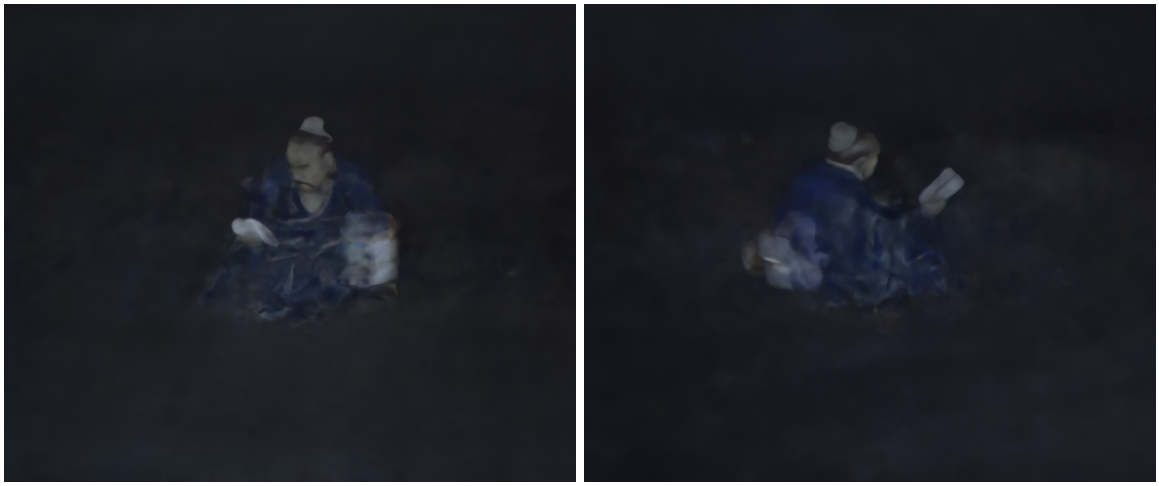


Figure 4.6: NeRF reconstruction for DiLiGenT

Limitations of NeRF for Multiview Stereo Data. Though vanilla NeRF provides convincing

renderings with minor preprocessing steps, these are nowhere close to the renderings provided by state-of-the-art traditional photometric stereo methods. Therefore, we construct it on top of the vanilla NeRF architecture to make our Relightable NeRF model more stereo data-friendly.

4.3.2 Ablation Study for Relightable NeRF

Synthetic Stanford Bunny Data

Study 1: Dataset Size. The first challenge we wanted to address was the speed of training for the original NeRF model. With the complete dataset of 96 illuminations and 20 camera poses, it has been difficult to try a multitude of settings for an ablation study. Thus, we downsized our dataset to **Mini Dataset**, figure 4.7, and **Micro Dataset**, figure 4.8 which contain 3 illumination settings and 1 illumination setting for 20 camera poses respectively. It must be noted that the micro dataset has been used to test the original NeRF's speed only where we do not require lighting information. Both datasets significantly improve the speed of training as compared to the full dataset for the original NeRF implementation due to a significant reduction in the number of input images to be processed and learned from. From a relightable NeRF standpoint, the micro dataset is not useful as it is not photometric in nature. Thus, the minimum viable dataset for the relightable training would be the mini dataset with 3 illumination directions. A further investigation on modified datasets with the number of lighting directions 6, 12 and 24, does not show a significant improvement in model performance. Therefore, the final dataset of choice is with three illumination directions per viewpoint. This configuration has been used for every ablation study succeeding the study.

Figure 4.7: Mini Dataset - 3 illumination directions

Figure 4.8: Micro Dataset - 1 illumination direction

Study 2: Model Depth. The model depth **D** defines the number of layers in the geometry network, MLP1. The original NeRF model uses **D** as 8. In this study, we have experimented

Figure 4.9: Results for ablation study on model depth, **D** as 1, 2 and 4.

with different depths so as to find a middle ground between faster training and quality rendering and, thus, have experimented with values 1, 2 and 4. The rendering quality remains similar among these depth values as shown in figure 4.9. Finally, for faster training, we have set the depth **D** value as 2 for succeeding studies.

For MLP2, we have performed similar experimentation, where we have trained with **D** values as 2, 4 and 6, keeping **D** for MLP1 as 2. Though the renderings for the **D** as 6 are of the highest quality, it comes with a longer training duration and higher complexity structure, thus, it is comparatively beneficial to continue using **D** as 4 for MLP2.

Study 3: Textured Data. In order to judge the ability of our relightable model to handle structures with higher complexity, we introduced streaks of varied bright colours to the object texture file. Qualitatively speaking, the model is able to handle the complexities provided by the colours as shown in figures 4.10 and 4.11, however, it must be noted that the intensity of the specular highlights sometimes don't match and this can be alleviated by training for more epochs.

Study 4: Boundary Factor. The boundary factor determines the measurements of the visual hull cube within which the marching cube algorithm will generate the mesh of the object. The experiment was performed in order to fix the rendered mesh. Visually speaking, the rendered mesh was a matrix of multiple small objects of focus (bunny) evenly spaced out in a cube space as shown in figure 4.12. In this study, we experimented with the Boundary Factor bd_factor values - 0.01, 1.0, and 10.0 to reduce the number of objects in the field of view to one object. However, changing the boundary factor does not affect the size of our

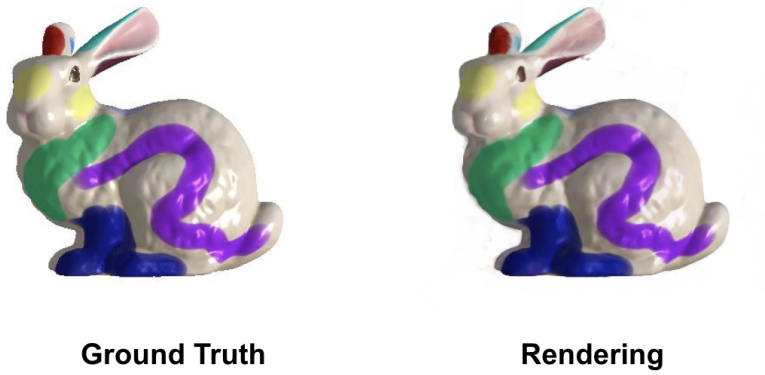


Figure 4.10: Textured Rendering

Figure 4.11: 360°Relighting of Textured Stanford Bunny

field of view and requires a change of other unknown parameters at the time which we later come to understand is the spherify radius explained below.

Study 5: Spherify Radius. After multiple debug runs, the spherify radius parameter was found as the culprit parameter for changing the field of view for the rendered mesh. In this study, we experimented with varied values ranging from 100 to 0.01. The initial value provided by vanilla NeRF implementation is calculated as follows

$$radius = \sqrt{\frac{\sum x_{trans}^2 + y_{trans}^2 + z_{trans}^2}{N_{poses}}} \; \forall \; poses$$

where x_{trans}, y_{trans} and z_{trans} are the $[x, y, z]$ values for the translational matrices for every pose and N_{poses} is the total number of poses in the training set. The formula fails to compute the correct radius for the stereo dataset, therefore, we relied on the trial and error method to find the most appropriate radius value. This is the **most sensitive parameter** for our model and it differs from dataset to dataset.

Radius above 1.0: We trained our model on radius values 100, 10, 5.0 and 1.0 and concluded that the field of view for these radii, the renderer does not reduce the matrix to a single

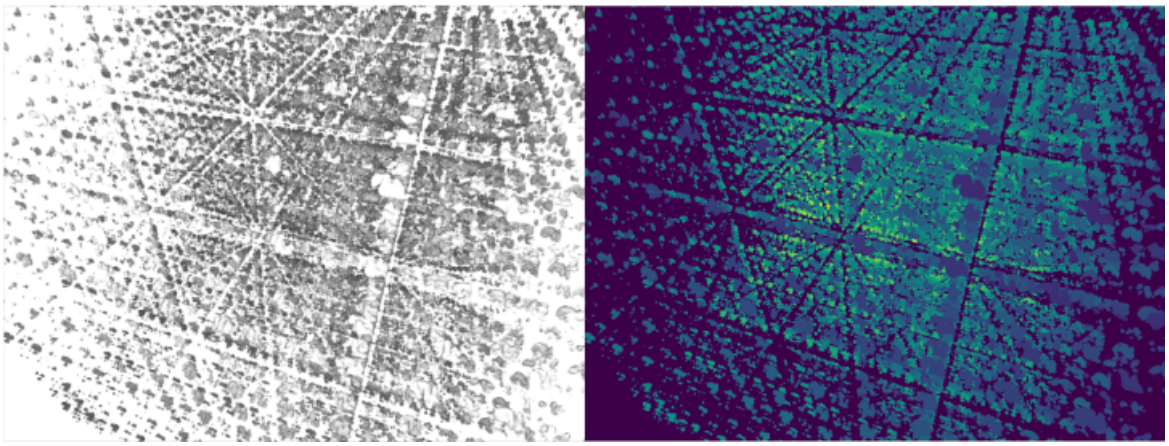


Figure 4.12: Matrix Mesh for a large grid: The grid is a snapshot of the mesh extracted after training. Each individual point in the mesh ensemble is an individual object rendered. The model provides this result due to the incorporation of sinusoidal positional encoding which repeats the data after a fixed interval (as reported by the fading object as we move away from the actual rendering of the object). This issue can only be resolved by magnifying the focus cube of the marching cubes algorithm which is achieved by modifying the spherify radius parameter.

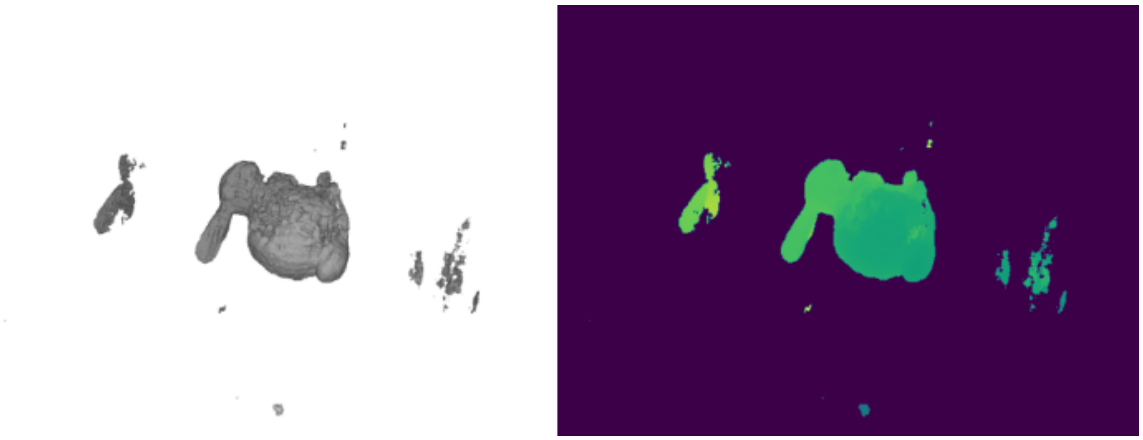


Figure 4.13: Rendering with Spherify Radius 0.35

object of interest and includes some remnants of the repeated object as shown in figure 4.13.

Radius ranging 1.0 to 0.5: We trained our model on radius values 0.85, 0.75 and 0.5 and concluded that the field of view for these radii, the renderer significantly reduces the matrix to a few objects in the matrix.

Radius ranging 0.5 to 0.1: We trained our model on radius values 0.35 (figure 4.13), 0.25, 0.175, 0.15 and 0.1. We concluded that the field of view for 0.175 (figure 4.14) and 0.15 are the best fits for the object. The values above 0.175 render a mesh with a single object with

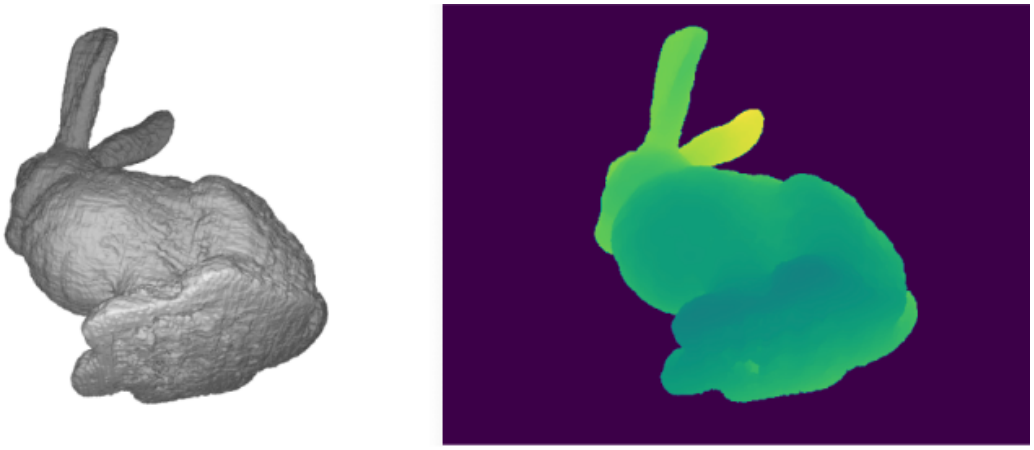


Figure 4.14: Corrected Spherify radius for DiLiGeNT-MV dataset: Once we are able to find the correct value for spherify radius for a particular dataset (here 0.35) we are able to get rid of the repeated rendering of the object

floating artefacts and the ones below 0.15 render a zoomed-in mesh which contains part of the object.

Multiview Photometric Stereo Data - DiLiGeNT-MV

The ablation study performed for the Stanford Bunny provided valuable parameter intel and proved to be beneficial for any future dataset. We, therefore, do not require running an ablation study for the succeeding datasets and followed the best parameter settings obtained from this study. However, the spherify radius is the only parameter which required tuning for the new dataset. We followed the Ablation Study 5 and obtained a radius value of 0.0425 for every DiLiGeNT-MV object. We have showcased the results obtained for all the DiLiGeNT-MV objects in figures 4.15, 4.16, 4.17, 4.18, 4.19 and 4.20.

Generalisability of the Ablation Studies to other datasets and objects

The ablation study in this work has been performed to investigate the contribution of individual components, features and hyperparameters to our baseline Neural Apparent BRDF Fields model. We have iteratively identified data parameters like dataset size and data complexity; and model hyperparameters like model depth and boundary radius and provided qualitative results for various models. The previously tuned parameters on Stanford Bunny and DiLiGeNT-MV datasets could be directly used for any other dataset, however, we would consider these in more detail in the sections below:

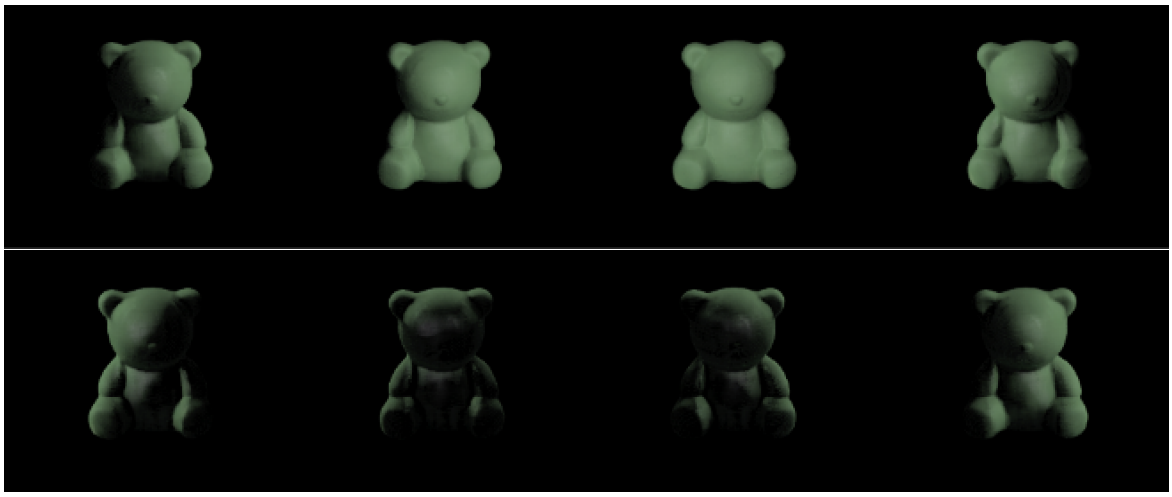


Figure 4.15: Bear Object with Single Pose Relighting

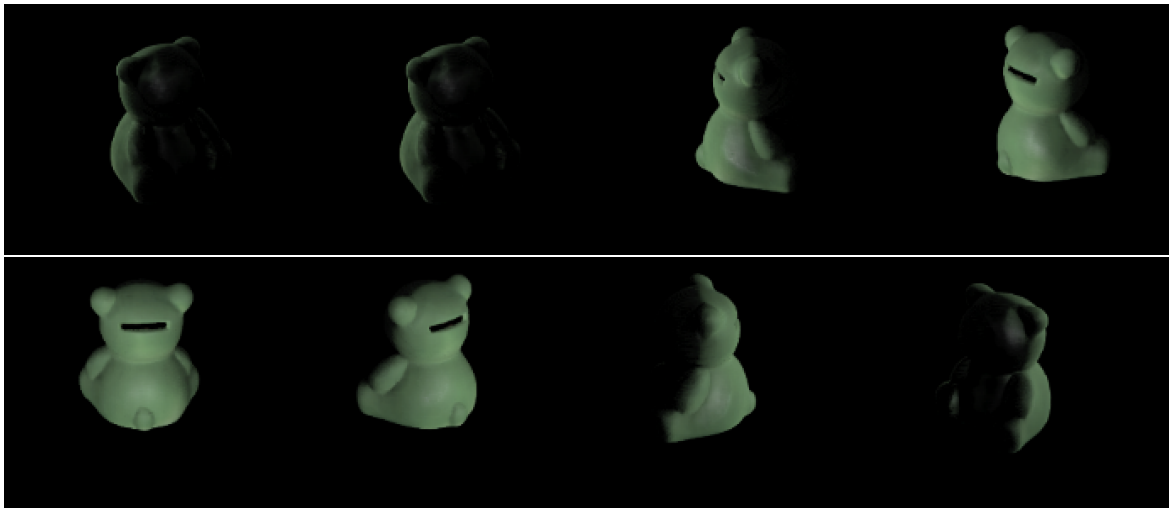


Figure 4.16: Bear object 360°viewpoints with novel distant point light source

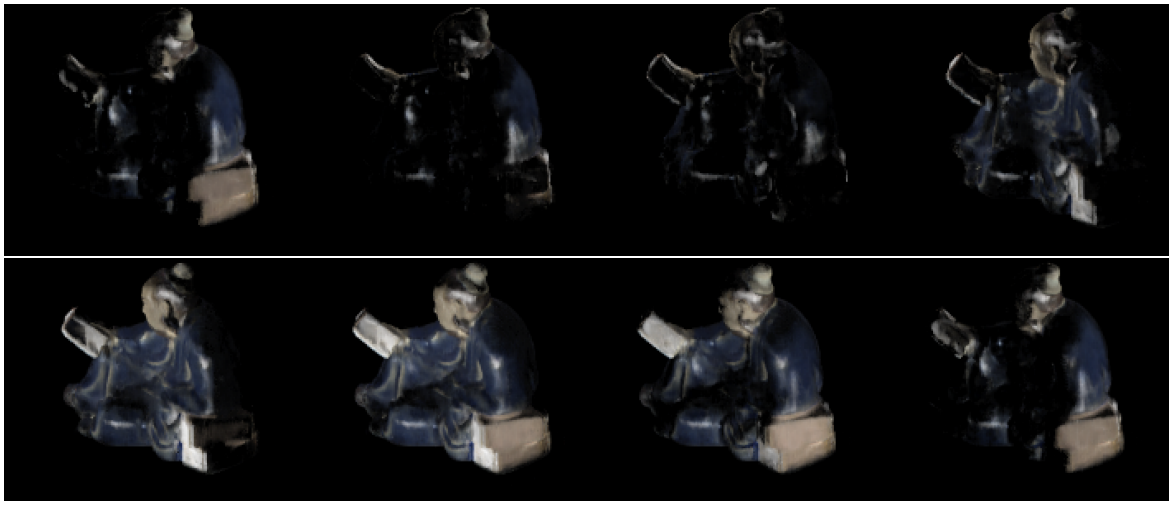


Figure 4.17: Reading Object with Single Pose Relighting

Figure 4.18: Reading object 360°viewpoints with novel distant point light source

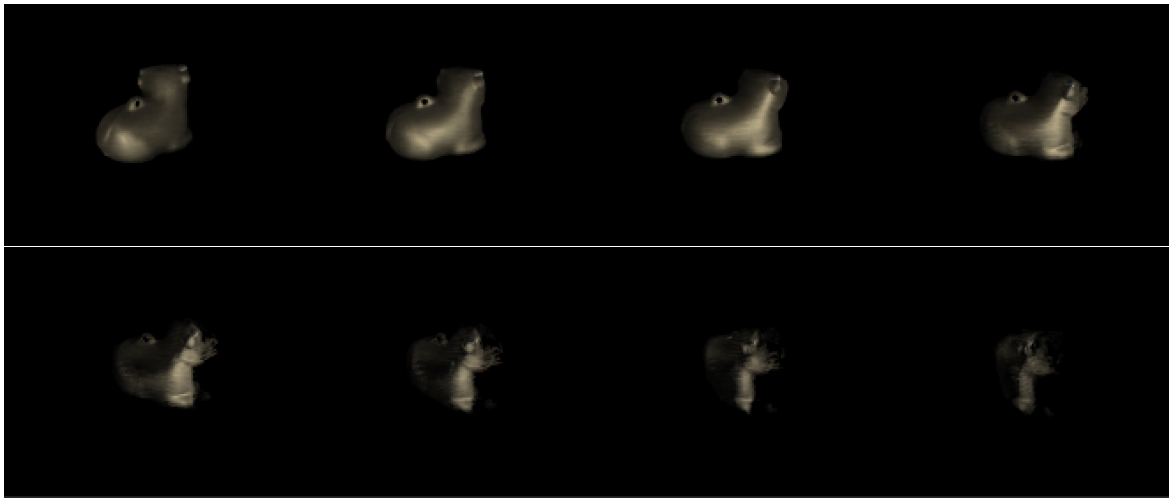


Figure 4.19: Cow object 360°viewpoints with novel distant point light source

Figure 4.20: Buddha Object with Single Pose Relighting

1. **Model Architecture:** Our model is trained to accurately mimic the data and provide a plausible relighted rendering, thus, we believe that objects with similar complexity would not require any change in the architecture. However, if we would like to model different types of textures, it would require some alterations to accommodate a suitable physics-based model.

2. **Feature Importance and Robustness:** Features are highly specific to a particular dataset, therefore, there is a need for major intervention and revision for ablation study for a new dataset.

3. **Hyperparameter Sensitivity:** The only sensitive hyperparameter is the 'spherify radius'.

4. **Transferability:** The training from one dataset to another is not possible as we are trying to overfit our data for the best reconstruction.

4.4 Conclusion

In this chapter, we introduced our first novel contribution Relightable NeRF. This has been achieved by incorporating an extra set of lighting direction input and remodelling the NeRF architecture such that the information can be fed into the Geometry and Colour Networks. This is an upgrade from the original NeRF making the model view- as well as illumination-dependent while providing legible results for novel viewpoints and illumination directions close to what has been seen in the training set. The model, thus, provides a simple yet efficient approach to baking in objects into VR and AR settings by treating the colour network as a black box. The learning is, therefore, unspecified and undefined from the user's perspective.

The black box approach, though easy to implement, falls short in rendering novel viewpoints and illumination directions which are far off from the ones present in the training dataset as well as modelling various features of surface reflectance. We have also observed floating artefacts for our renderings as well as incorrect shadows and lighting with unknown origins. Therefore, there is a need to replace the black box with a system which enables a clear definition of properties to be learned by the Colour Network.

In the upcoming chapter, we will address this issue by upgrading the black box model through the introduction of individual Multi-Layer Perceptron models to render different features of an object. The models will include - a Geometry Network, a BRDF Network and a Shadow Network which would train in unison to provide an improved RGB rendering and 3D reconstruction from the Relightable and original NeRF.

Chapter 5

Neural Apparent BRDF Fields for Photometric MV-Stereo

In this chapter we introduce explicit modelling of geometry and reflectance into a NeRF-based representation and use it to tackle the multiview photometric stereo problem. This problem provides both geometric- and illumination-calibrated images and we question whether the NeRF representation is able to estimate accurate shape from such data. While a naive lighting dependency can be introduced by simply conditioning the appearance of the NeRF on light direction (as done in the previous chapter), this leads to very poor view synthesis extrapolation when the light source is far from the directions seen during training. Moreover, such an approach doesn't allow explicit modelling of the relationship between geometry and appearance via the BRDF.

Instead in this method, as shown in Fig. 5.1, we will explicitly regress surface normal direction with the geometry network and decompose the NeRF colour network into networks for BRDF (with suitable physical priors) and shadowing (i.e. we learn the shadowing function for each point in the scene to avoid costly ray casting through the neural volume). We will train the network using the same volume rendering and appearance losses as the original NeRF but use multi-illuminant data with known light source direction as present in a photometric stereo setting. We refer to the local BRDF with the effect of shadowing baked in as the 'apparent' BRDF (i.e. a function parameterised and used in the same way as the BRDF but which captures non-local effects).

The method presented in Chapter 5.1 improves extension to NeRF simply conditions the

colour network on a point source light direction, $\mathbf{s} \in \mathbb{R}^3$, $\|\mathbf{s}\| = 1$, by replacing the second network with: $f_{\text{col}}^{\Theta_{\text{col}}} : (\mathbf{z}, \mathbf{v}, \mathbf{s}) \mapsto \mathbf{c}$. We must note that the latent code $\mathbf{z}$ must store an encoding of the apparent BRDF at a given point in the scene. Appearance under arbitrary illumination environments can be obtained by integrating the output of this network over incident illumination directions. To train such a network we require training images with both viewpoint variation and variation in illumination direction provided by a point light source of known direction.

Such a naive extension neglects several underlying physical properties of surface reflectance which strongly limits its ability to accurately interpolate and extrapolate to new lighting directions. We, therefore, propose a series of modifications to create a NeRF architecture that follows a physical image formation model closely.

5.1 Neural Apparent BRDF Fields Model

Building towards neural apparent BRDF fields is an incremental process with multiple implementations. The method proposes to tackle the multiview photometric stereo problem using an extension of Neural Radiance Fields (NeRFs), conditioned on light source direction. It tackles multiple problems including point source relighting, surface texture using Bidirectional Reflectance Surface Distribution (BRDF) as well as shadow prediction through a single model. The Neural Apparent BRDF Fields model can be written as three MLPs:

1. $f_{\text{geo}}^{\Theta_{\text{geo}}} : \mathbf{x} \mapsto (\mathbf{z}, \sigma)$ with learnable parameters Θ_{geo} which maps a 3D location $\mathbf{x} = (x, y, z)$ to a latent code $\mathbf{z} \in \mathbb{R}^d$ and density $\sigma \in \mathbb{R} \geq 0$.

2. $f_{\text{BRDF}}^{\Theta_{\text{BRDF}}} : (\mathbf{z}, \mathbf{v}, \mathbf{s}) \mapsto \mathbf{c}$ with learnable parameters Θ_{col} which maps the viewing and lighting directions $\mathbf{v} \in \mathbb{R}^3$, $\mathbf{s} \in \mathbb{R}^3$, $\|\mathbf{v}\| = 1$ and latent code to a colour $\mathbf{c} = (r, g, b)$.

3. $f_{\text{shad}}^{\Theta_{\text{shad}}} : (\mathbf{z}, \mathbf{s}) \mapsto [0, 1]$ with learnable parameters Θ_{shad} which maps the lighting directions $\mathbf{s} \in \mathbb{R}^3$, $\|\mathbf{s}\| = 1$ and latent code to scalar value $\in [0, 1]$.

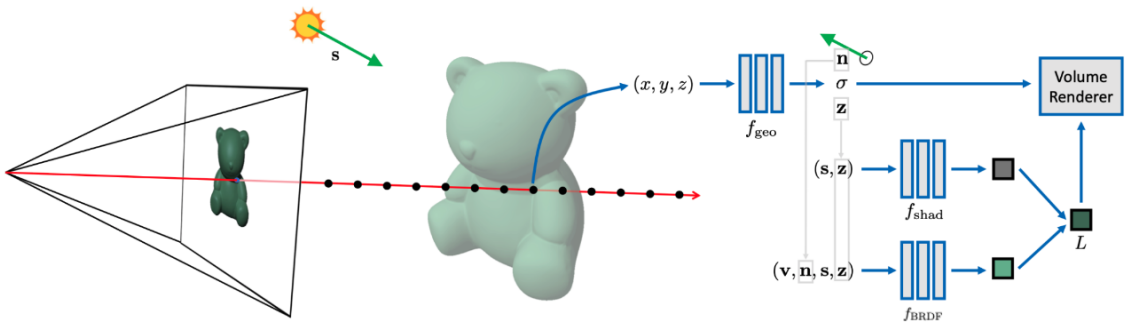


Figure 5.1: We replace the view-conditioned black box radiance predicted by a NeRF [Mildenhall et al.(2020)] with a physically-based image formation model. The geometry network predicts density and surface normal direction at each position in the volume. A neural BRDF and shadow network predict reflected scene radiance for the point which is then volume rendered using predicted density as for NeRF. The resulting model is relightable while also improving geometric information using multi-light observations (figure taken from [Asthana et al.(2022)]).

5.1.1 Geometry network

Surface reflectance is determined by the local surface orientation where the light strikes the surface. Thus, in order to reason about this explicitly, first, we extend the initial geometry network to regress not only density but also surface normal direction: $f_{\text{geo}}^{\Theta_{\text{geo}}} : \mathbf{x} \mapsto (\mathbf{z}, \mathbf{n}, \sigma)$, where $\mathbf{n} \in \mathbb{R}^3$, $\|\mathbf{n}\| = 1$, is the unit length surface normal (in practice we regress an unconstrained 3D vector and normalise to unit length in-network).

An alternative is to derive the normal from the gradient of the density field as done in [Srinivasan et al.(2021)]. This explicitly ties the two geometric representations together. However, we found that allowing the network to directly regress the surface normal direction enables the reconstruction of higher frequency details in the normal map and reduces noise artefacts in the normal maps caused by using second-order derivatives. This is far more suited to the task of multiview photometric stereo where high-quality per-view normal maps may be a useful output in their own right. However, we still use the network to separately learn normals as the normals that are obtained by differentiating density are very noisy as they are a second derivative in nature and calculating loss on such values does not provide

decent results.

A classical Computer Graphics technique, Bump Mapping has been used to provide a surface much greater geometric detail than in actuality on a triangle mesh. It is based on a special type of texture map which produces the illusion that the surface varies in height (though the surface is flat) using the changes in brightness of the reflected light at high frequency [Lengyel(2019)]. Taking inspiration from this, our approach could be assumed as a neural bump mapping where we effectively learn a bump map for the normals to be used at each point on the surface.

5.1.2 Neural BRDF

Second, we use a network to explicitly model the BRDF: $f_{\text{BRDF}}^{\Theta_{\text{BRDF}}} : (\mathbf{n}, \mathbf{s}, \mathbf{v}, \mathbf{z}) \mapsto \mathbf{a}$, where $\mathbf{a} \in \mathbb{R}^3$ represents the BRDF value for each of the RGB channels. The latent code $\mathbf{z}$ is used to encode information about the BRDF at the scene point. The learnable parameters of the network, Θ_{BRDF}, are optimised such that the network can represent the range of BRDFs observed in the scene, as parameterised by $\mathbf{z}$. A BRDF is not an arbitrary function - it must obey some physical constraints. We satisfy two of these: 1. positivity: satisfied by using an activation function that can only output positive values, 2. reciprocity: we parameterise the incident and outgoing directions using an angular representation that is itself reciprocal (i.e. unchanged by switching $\mathbf{s}$ and $\mathbf{v}$). Satisfying the third physical constraint, conservation of energy, we leave as future work due to the difficulty of imposing integral constraints on neural networks [Weber et al.(2021)].

As a special case, this network can be a fixed function representing a known parametric BRDF. In this case, we use a learnable network to map the latent code to the parameters of the BRDF: $f_{\text{BRDF}} : \mathbf{z} \mapsto \mathbf{p}$, where $\mathbf{p} \in \mathbb{R}^P$ contains the specific parameters for the chosen BRDF model. As a specific example, the Lambertian model has $P = 3$ parameters for the RGB diffuse albedo. Then the BRDF becomes a non-learnable function $f_{\text{BRDF}} : (\mathbf{n}, \mathbf{s}, \mathbf{v}, \mathbf{p}) \mapsto \mathbf{a}$. We explore this variant in our experimental results.

5.1.3 Shadow prediction

Modelling reflectance using a BRDF means we could only describe local reflectance effects. We equip our model with additional power to allow it to describe the effect of shadows.

These are the most significant non-local effect encountered in our data. The combination of BRDF and shadowing means we model a partially decomposed apparent BRDF. Our neural BRDF model will learn some non-local effects such as interreflection while the gross changes caused by cast shadows can be explained by the shadow network.

Of course, exact cast shadows can be computed from a NeRF by ray casting, but this is expensive. So, we follow the idea in NeRV [Srinivasan et al.(2021)] and instead replace ray casting by a learnt approximation. Specifically, we train an MLP that predicts a scalar soft shadow value from light source direction and the NeRF latent code: $f_{\text{shad}}^{\Theta_{\text{shad}}} : (\mathbf{s}, \mathbf{z}) \mapsto [0, 1]$. A soft shadow factor can be defined as the representation of occlusion or self-cast shadow through a value between 0 and 1. It must be noted that the value should not be a hard value of 0 (RGB channels [0,0,0]) or 1 (RGB channels [1,1,1]) as it will be considered as a hard shadow. We exhibit the soft shadow phenomena in the figure 5.8(bottom). For fixed latent code, this MLP then represents the light source visibility function.

5.1.4 Camera behaviour and nonlinear appearance loss

Surface reflectance can have high dynamic range (for example, specularities on glossy objects can be orders of magnitude brighter than diffuse reflections). For this reason, similar to NeRF in the dark [Mildenhall et al.(2022)], we let our model learn high dynamic range radiance in a linear space. We then apply nonlinear tone-mapping (in our case, a simple gamma correction) before computing the error to tone-mapped training images. This significantly outperforms computing errors in linear space where the bright regions completely dominate the error. Operating in a nonlinear tone-mapped space ensures the reconstruction better approximates minimisation of a perceptual error.

5.1.5 Rendering

In order to compute the reflected radiance at scene point $\mathbf{x}$ of a single white point light source from direction $\mathbf{s}$ of unit intensity towards the viewer in direction $\mathbf{v}$, our pointwise rendering equation is:

$$L^{\Theta}(\mathbf{x}, \mathbf{s}, \mathbf{v}) = f_{\text{BRDF}}^{\Theta_{\text{BRDF}}}(\mathbf{n}(\mathbf{x}), \mathbf{s}, \mathbf{v}, \mathbf{z}(\mathbf{x})) f_{\text{shad}}^{\Theta_{\text{shad}}}(\mathbf{s}, \mathbf{z}(\mathbf{x})) \max(0, \cos\theta_i), \tag{5.1}$$

where $(\mathbf{n}(\mathbf{x}), \mathbf{z}(\mathbf{x})) = f_{\text{geo}}^{\Theta_{\text{geo}}}(\mathbf{x})$, $\theta_i = \arccos(\mathbf{n}(\mathbf{x}) \cdot \mathbf{s})$ and $\Theta = (\Theta_{\text{geo}}, \Theta_{\text{BRDF}}, \Theta_{\text{shad}})$ are all the learnable parameters in the model. We use this reflected radiance quantity in place of the view-conditioned radiance in the original NeRF model and volume render using the estimated density σ in the same way, i.e.

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i(1 - \exp(-\sigma_i \delta_i)) L^{\Theta}(\mathbf{x}_i, \mathbf{s}, \mathbf{v}). \tag{5.2}$$

5.1.6 Mask supervision

For individual objects with known foreground masks, we find performance is substantially improved by supervising the network using the mask (for the DiLiGeNT dataset, these masks have been already provided with the input images). This takes two forms. First, we can penalise any density along rays that do not pass through the object mask. We use the image masks to divide the set of rays $\mathcal{R}$ into two disjoint subsets for object foreground $\mathcal{F}$ and background $\mathcal{B}$. We then introduce a silhouette loss that encourages zero density outside the visual hull of the object:

$$\mathcal{L}_{\text{sil}} = \sum_{\mathbf{r} \in \mathcal{B}} \sum_{i=1}^{N} \sigma_i. \tag{5.3}$$

This is equivalent to the shape-from-silhouette cue and serves to directly constrain the geometry of the object. Second, we mask the appearance loss to only penalise appearance errors inside the object: $\mathcal{L}_{\text{app}} = \sum_{\mathbf{r} \in \mathcal{F}} \|\hat{C}(\mathbf{r})^{1/2.2} - C(\mathbf{r})\|^2$, where $C(\mathbf{r})$ are the ground truth values and we use a fixed gamma of 2.2 (our model is sensitive to the gamma value).

5.2 Implementation

In Figure 5.5 we show the overall architecture of our model. For the geometry network, we follow NeRF and use an MLP with 8 layers, 256 hidden units per layer and ReLU activation. The dimensionality of the latent vector $\mathbf{z}$ is 256 (the choice of value has been taken from the original NeRF architecture). Both the BRDF and shadow networks are MLPs with 2 layers, 128 hidden units per layer (the choice of value has been taken from the original NeRF architecture) and ReLU activation. For the surface normal $\mathbf{n}$, we regress an unconstrained 3D vector then rescale to unit length. For our angular representation of BRDF geometry, we use a simple two angle representation: the incident angle $\theta_i = \arccos \mathbf{n} \cdot \mathbf{s}$ and the half

angle $\theta_h = \arccos \mathbf{n} \cdot \mathbf{h}$ where $\mathbf{h} = (\mathbf{s} + \mathbf{v})/\|\mathbf{s} + \mathbf{v}\|$ is the halfway vector. For the variant with fixed (Lambertian) BRDF, we replace the BRDF network with a single linear layer to map $\mathbf{z}$ to 3 values and use sigmoid activation to compute RGB diffuse albedo.

We follow the original NeRF and apply positional encoding to input position $\mathbf{x}$. Since we use view direction $\mathbf{v}$ for the computation of angles for BRDF calculation, we do not apply positional encoding to this. We also follow the original NeRF optimisation in using both a coarse and fine network combined with stratified sampling.

We work entirely in the normalised NeRF coordinate system. This rescales camera poses such that the object lies approximately inside the unit cube with cameras lying approximately around the equator. We convert light source directions to this coordinate system meaning that the normal maps we estimate are in NeRF world coordinates. Where needed, we convert these to camera coordinates via the rotational component of the camera extrinsics.

5.2.1 Lambertian Model

The first intermediate stage of building towards neural apparent BRDF fields network is a Lmabertian Model which is simplest choice for representing diffused reflection. The model is designed to to reflect light equally in all directions when rendered. In order to upgrade the Relightable NeRF, the steps below have been followed:

- **Basic Lambertian Model**: The Relightable MLP2 from the previous model is replaced by a basic Lambertian model which consists of two additional outputs from the Geometry Network - Normal and Albedo maps. Normals and Albedo are of shape **batch size** $\times$ 3 and are crucial for understanding and properly representing shadows and illumination for a far-off light source. It must be noted that at this intermediate step, the model is not relightable in nature. The previous model assumed the bottleneck feature vector of 256 dimensions from MLP1 as a Blackbox whose inner workings are not known to us.

- The **Normals** provide us with information about the vector perpendicular to the face (in 3D space) at the specific pixel. This is extracted as a dense output of 3 layers from MLP1 and normalised to maintain the bounds of the output as $[-1, -1]$.

- We replace the unknown Blackbox nature of latent code with a known property - the Albedo output of 3 dense layers. The **Albedo** provide us with information on the base

colour that defines the diffuse colour or reflectivity of the surface at the specific pixel. It must be noted that Albedo values are unaffected by view or light direction.

- An intermediate output, θ_i is defined as the angle between normal and the angle of light incidence. The **cosine of** θ_i is obtained as the dot product of input light directions and Normal values from the network. This foundation value is essential for defining the effect of illumination on a Lambertian surface.

- The final **RGB values** are calculated as, per the definition of Lambertian model, the dot product between θ_i and albedo values.

- The outputs from the network include the density value σ, normal map **n**, albedo map **a** and the final RGB values **c**.

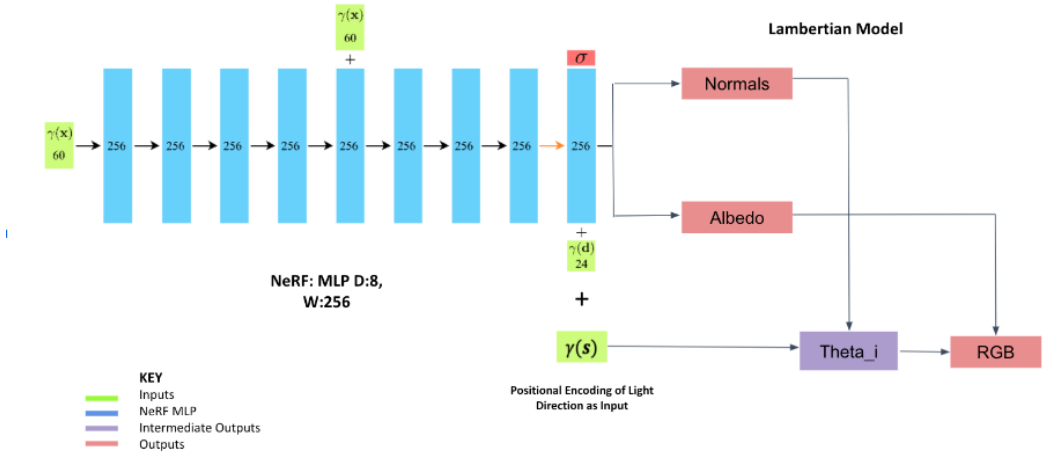


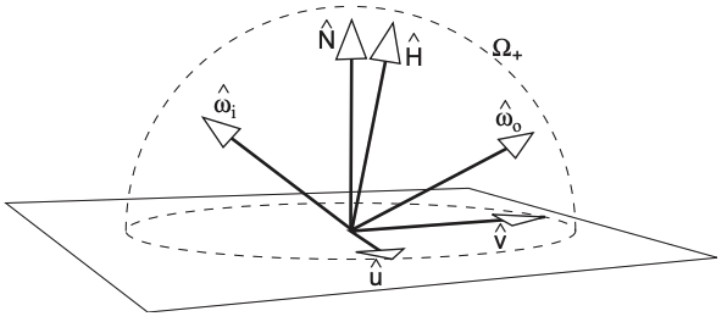
Figure 5.2: **Lambertian NeRF architecture**: Input vectors are shown in green, MLP hidden layers are shown in blue, intermediate outputs are shown in purple, output vectors are shown in red, and the number inside each block signifies the vector's dimension. The layers are represented as fully-connected layers with ReLU activations (black arrows), orange arrows indicate layers with no activation and "+" denotes vector concatenation.

Half-Vector Extension

The Half-Vector extension has been performed as an addition to the simple Lambertian model to fine-tune the implementation of BRDF representations in terms of θ_i and ϕ_i. Multiple models have shown empirically that it is not possible to predict visually plausible BRDF for phenomena like off-specular reflection using just the parameters explained in the base

Lambertian model and require extra components for the BRDF. A model based on incident direction and hallway vector yields more visually plausible results [Edwards et al.(2006)]. The half-vector can be defined as a unit vector that is equidistant to the incident light ray and reflected light ray. It can be represented as:

$$\hat{H} = \frac{\hat{\omega_i} + \hat{\omega_o}}{\|\hat{\omega_i} + \hat{\omega_o}\|}$$



Symbol	Meaning
$\hat{\omega_i}$	Direction from surface toward source of incident light
$\hat{\omega_o}$	Direction from surface toward sampler of outgoing light
$\hat{N}$	Unit-length surface normal vector
$\hat{u}, \hat{v}$	Orthonormal basis for local surface orientation
$\vec{H}$	Halfway vector ($\hat{\omega_i} + \hat{\omega_o}$)
$\hat{H}$	Unit-length halfway vector $\vec{H}/\|\vec{H}\|$
$\vec{h}$	Halfway vector after transformation into another domain
Ω_+	Unit hemisphere above the surface
$d\Omega(\hat{x})$	Differential solid angle in the $\hat{x}$ direction
$d\mu(\vec{h})$	Differential measure (e.g., area) on the $\vec{h}$ domain

Figure 5.3: Half-vector notations (figure taken from [Edwards et al.(2006)]).

The implementation encompasses creating a vector which is the sum of view direction and illumination direction ray batches, $\mathbf{h}$. The sum is then normalised $\mathbf{h}_{norm}$ and used to compute the dot product with normalised Normal map $\mathbf{n}_{norm}$. The dot products

$$\cos\theta_v = \mathbf{n}_{norm} \cdot \gamma(\mathbf{v})$$

$$\cos\theta_h = \mathbf{n}_{norm} \cdot \mathbf{h}_{norm}$$

are sent as input to the second MLP (same as in relightable NeRF) instead of the input view and light directions. We have successfully implemented this in all our further iterations.

Loss Function Extensions

We have employed a variety of losses to improve the learning process and, in turn, the final renderings. Each loss brings a separate set of functionality to `total_loss` and a varied combination of each works best for every dataset we have trained. The extent of influence by any form of loss can be modified using factor arguments, these include alpha loss, binary loss, piecewise loss, silhouette density loss and mask appearance loss. The details of the implementation are provided in section 8.3.

5.2.2 Shadow Prediction Model

The second intermediate stage of building towards neural apparent BRDF fields network is the addition of a shadow prediction model which, together with the BRDF properties, can provide an illusion of 'apparent' BRDF fields. In order to upgrade our basic Lambertian model, the following steps have been followed:

- In addition to the Normals and Albedo map outputs from the Geometry network, a set of latent code of 256 hidden dimensions (the choice of value has been taken from the original NeRF architecture) has been used to represent the **BRDF parameters**
- The second MLP is the **Shadow Prediction Mode**l which encompasses two dense layers of width (number of hidden units) as 128. The inputs to the network include the BRDF parameters and θ_i. The output from this model is a single dense layer with the proposed shading factors.
- The initial result from the model converged all the RGB values to zero, providing a blank rendering. This was due to the convergence of the shading factor to zero. In order to obtain results in a sensible range, a bias initialiser has been used which sets the mean of the initial value as 5.0 and the standard deviation value as 0.05 for the output layer (these values are arbitrary and our model is not sensitive to them). Finally, these values are processed through a **sigmoid activation** function which takes any real value as input and outputs values in the range of 0 to 1. The larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to 0.0.
- The non-shaded RBG values have been obtained by computing the dot product between

the Albedo map values and θ_i. This value is then used to obtain the shaded colour values by calculating its dot product with the shading factor obtained from the Shadow Prediction model i.e. points with a shadow factor close to 1 exhibit little to no shading and vice versa.

- The outputs from the network include the **density value** α, **normal map** N, **albedo map** A, shadow factor **S** and the final radiance shadow + RGB.

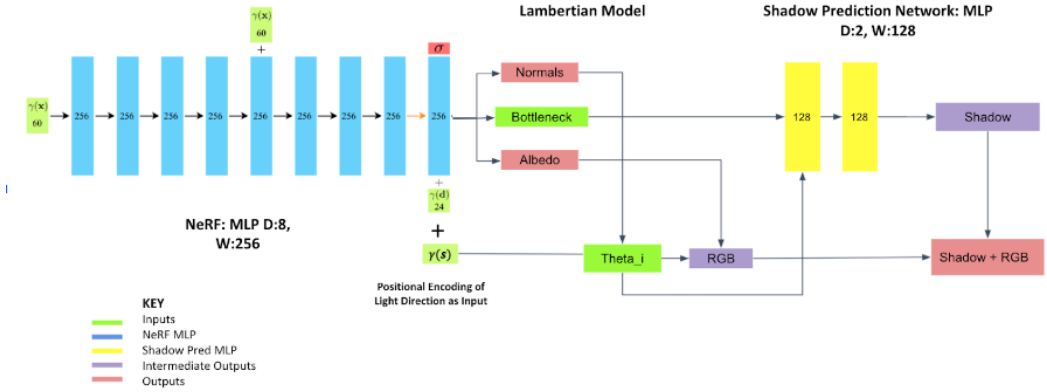


Figure 5.4: **Lambertian + Shadow prediction NeRF architecture**: Input vectors are shown in green, Geometry Network MLP1 hidden layers are shown in blue, Shadow Prediction MLP2 hidden layers are shown in yellow, intermediate outputs are shown in purple, output vectors are shown in red, and the number inside each block signifies the vector's dimension. The layers are represented as fully-connected layers with ReLU activations (black arrows), orange arrows indicate layers with no activation and "+" denotes vector concatenation.

5.2.3 Neural Apparent BRDF Fields Model

The final model in the development of vanilla NeRF to accurately represent the rendered surface texture and illumination is the neural apparent BRDF fields with the benefit of being learnable. The following steps have been taken to enhance our previous model:

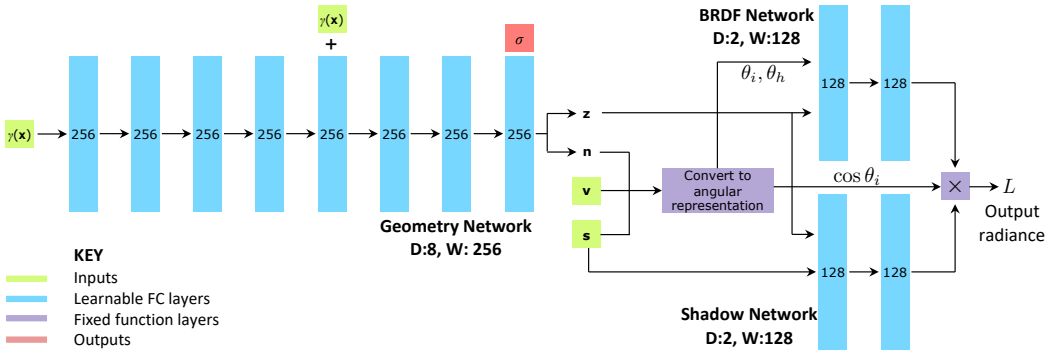


Figure 5.5: **Neural Apparent BRDF Fields Architecture**: Our model is a combination of three learnable MLPs and fixed functions. The latent parameters **z** act as a bottleneck from which BRDF parameters and local shadowing function is inferred by the BRDF and shadow networks respectively (figure taken from [Asthana et al.(2022)]).

- Our model is a combination of three learnable MLPs and fixed functions.
- The **Geometry Network** regresses density as well the surface normals which are better at reconstructing higher frequency details, reducing noise and are better suited for multi-view stereo tasks. The network has eight layers with 256 hidden units each. It takes RGB values, view-direction ray batch and light-direction ray batch as input with positional encodings off to reduce complexity. The network outputs features including volume density σ, Normals map **n** and Albedo map **a**.
- The **BRDF Network** explicitly models the BRDF values for each of the RGB channels and is designed such that it follows the physical constraints of positivity, reciprocity and conservation of energy. We satisfy two of these:

  1. **Positivity:** The constraint is satisfied by using an activation function that can only output positive values.
  2. **Reciprocity:** We parameterise the incident and outgoing directions using an angular representation which maintains the incident and reflectance angles from the surface as equal.

  Satisfying the third physical constraint, **Conservation of Energy**, we leave it as future work due to the difficulty of imposing integral constraints on neural networks. In terms of the architecture, the network encompasses two dense layers with 128 nodes each.

- The **Shadow Network**, as mentioned before, uses the shadow factor technique from NeRV [Srinivasan et al.(2021)] where soft scalar values are predicted by the model. The method is proven to be less expensive than traditional ray casting used by NeRF. It enables a combination of BRDF and shadow factor which models a partially decomposed apparent BRDF. The network consists of two dense layers with 128 hidden units each. The output is a set of soft shadow factors as mentioned before in the Shadow Prediction Model.

- The dot product between the shadow factor and the illumination-aware RBG values provides the final shaded pixel colour values from the network.

- The outputs from the network include the density value σ, normal map $\mathbf{n}$, albedo map $\mathbf{a}$, shadow factor S and the final output radiance $\mathbf{L}$.

The further details on the implementation and usage of the code have been provided in the Appendices section 8.3.

5.3 Experiments

5.3.1 Summary

View synthesis and relighting Our trained model can be used to synthesise novel views with arbitrary (potentially unseen) viewpoint and lighting direction. In Figure 5.6, column 1 we show rendering results of our model. For comparison, we show the closest image in the training set in the column 2. It is crucial to also note that we are able to produce more extreme lighting directions that predict realistic shadows and position of specularities. In columns 3 and 4 we show the normal and shadow maps obtained by volume rendering the normal and shadow predictions. Figure 5.7 shows qualitative results for the variant in which we used a fixed (Lambertian) BRDF. In this case, the model predicts only diffuse albedo parameters with no learnable BRDF. We retain the learnable shadow network. While these results provide plausible albedo estimates, they are not able to synthesise non-Lambertian effects such as specular reflections.

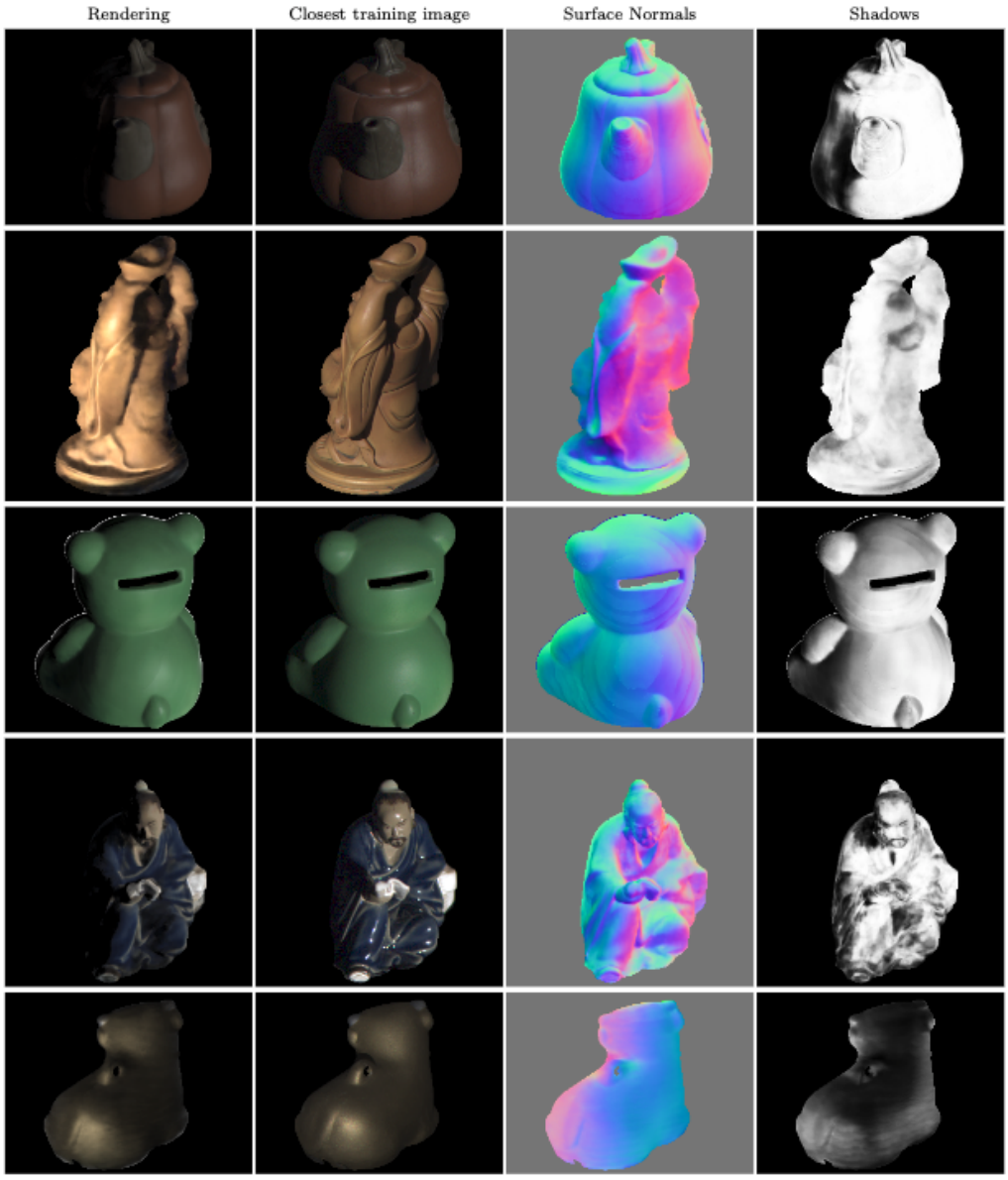


Figure 5.6: Qualitative results for Diligent-MV [Li et al.(2020b)] using neural BRDF and shadow networks. From left to right: rendering from the trained model under novel lighting and viewpoint, closest training image, estimated surface normal and shadow maps. It must be noted that the aim of this work is to render the most accurate possible mesh and feature maps (normals, albedo and shadow) from the given data and not produce novel views reconstruction i.e. we have utilised every ground truth image from the training set. Thus, we comparing a rendered image with the corresponding ground truth view, we have not exluded the ground truth from the training set. (figure taken from [Asthana et al.(2022)]).

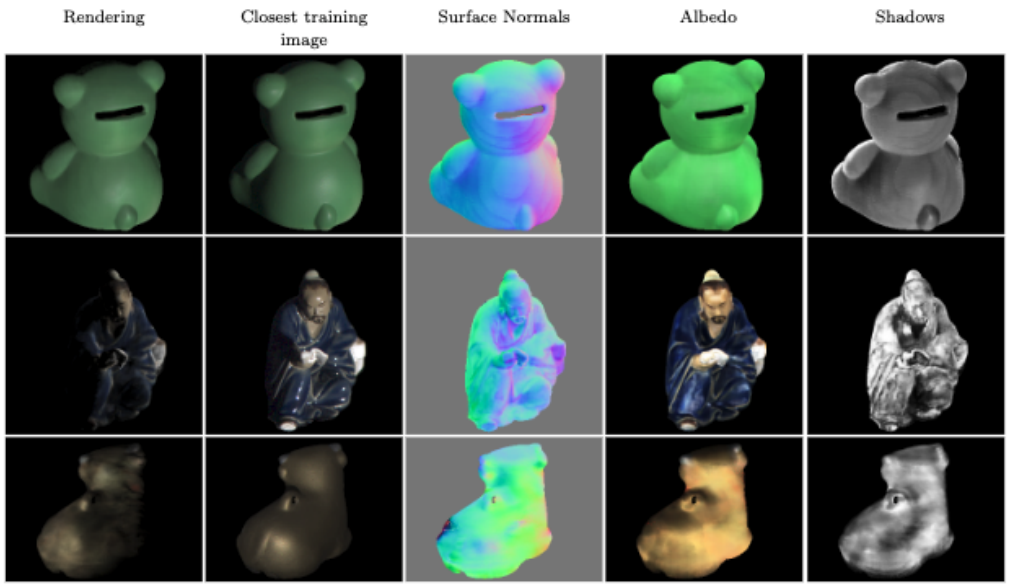


Figure 5.7: Qualitative results for Diligent-MV [Li et al.(2020b)] using Lambertian and shadow networks. From left to right: rendering from the trained model under novel lighting and viewpoint, closest training image, estimated surface normal, diffuse albedo and shadow maps (figure taken from [Asthana et al.(2022)]).

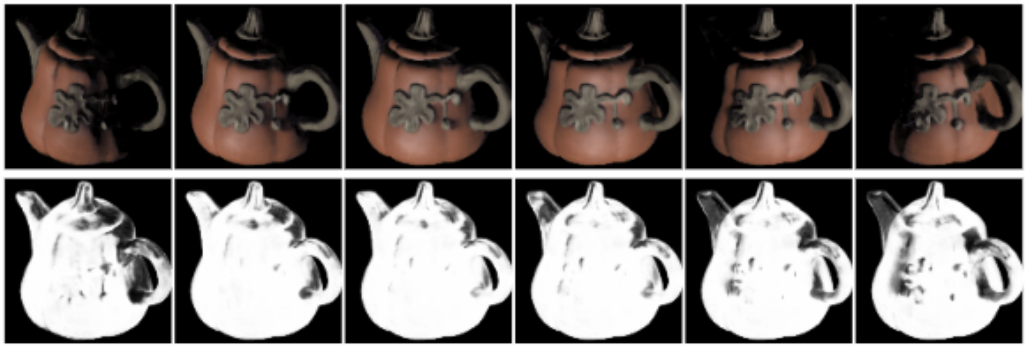


Figure 5.8: **Relighting result** (top) and the corresponding rendering of the shadow prediction only (bottom) (figure taken from [Asthana et al.(2022)]).

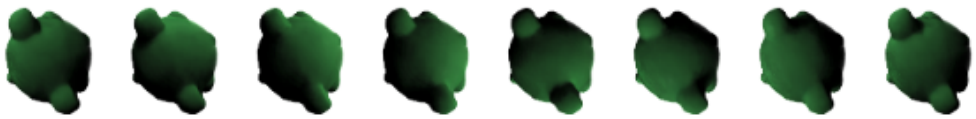


Figure 5.9: Relighting under extreme pose and illumination extrapolation. We show a top-down view of the bear object with 360°light rotation around the object (figure taken from [Asthana et al.(2022)]).

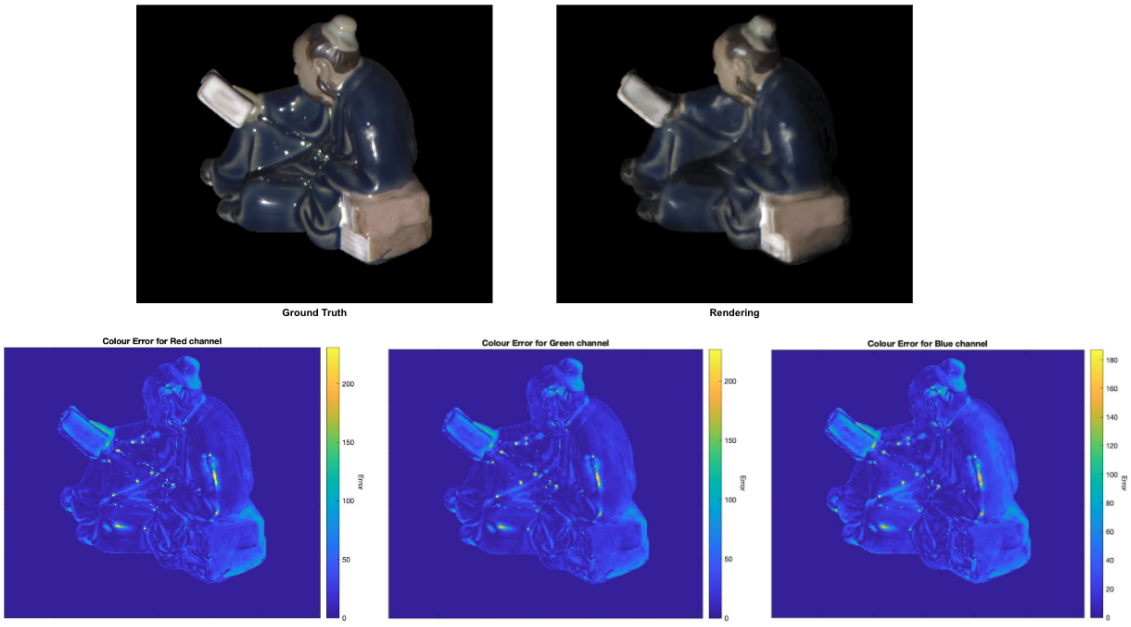


Figure 5.10: Colour Error Heatmaps for Red, Blue and Green channels for object Reading

In Figure 5.8, we show the effect of systematically varying light source direction as the light rotates around the front of the object. We show shadow maps in the second column. Note the prediction of the cast shadow by the handle onto the teapot. We emphasise that this dependency is learnt and not ray cast. Finally, in Figure 5.9, we show an extreme extrapolation example. The training views form a ring around the equator of the object. We synthesise a top-down view 90 degrees from the training view directions. Similarly, the synthesised 360 rotating light is 90 degrees from the virtual view direction, much greater than observed during training.

Shape estimation In Table 5.1, we show quantitative shape estimation errors in terms of angular error between estimated and ground truth surface normal maps. The normal maps can also be inspected visually in Figures 5.6 and 5.7. While the numerical errors are larger than state-of-the-art methods, particularly for the more challenging materials, our qualitative results show that we are able to estimate detailed and smooth shapes using our model.

Training and Code Our version exhibits faster training with convergence for DiLiGenT objects reaching convergence at 100K iterations as opposed to vanilla NeRF which converges

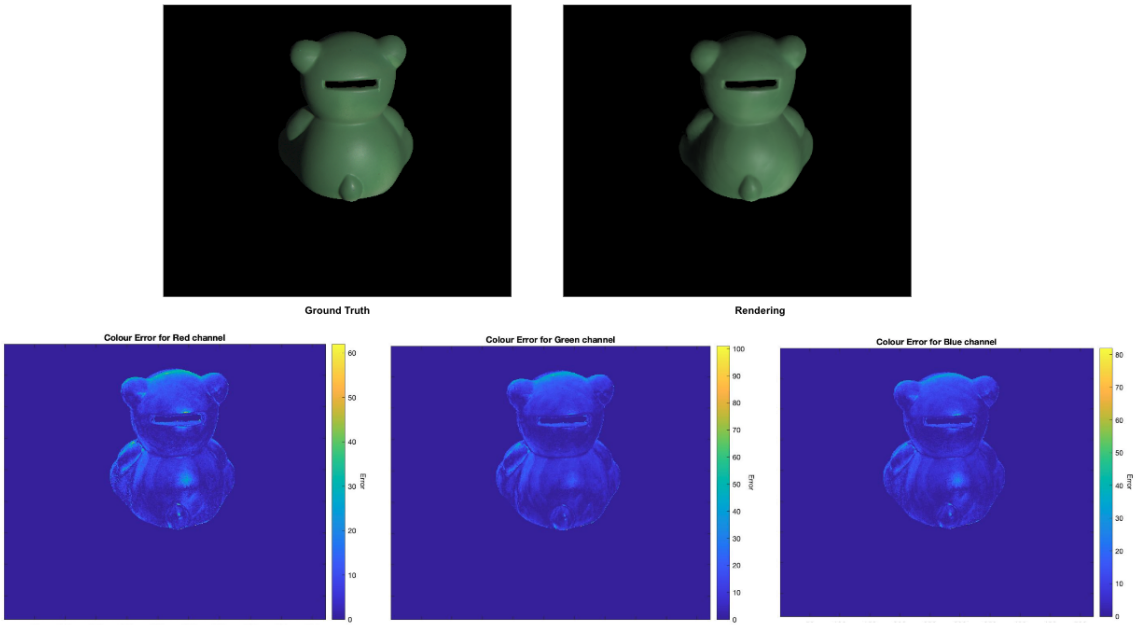


Figure 5.11: Colour Error Heatmaps for Red, Blue and Green channels for object Bear

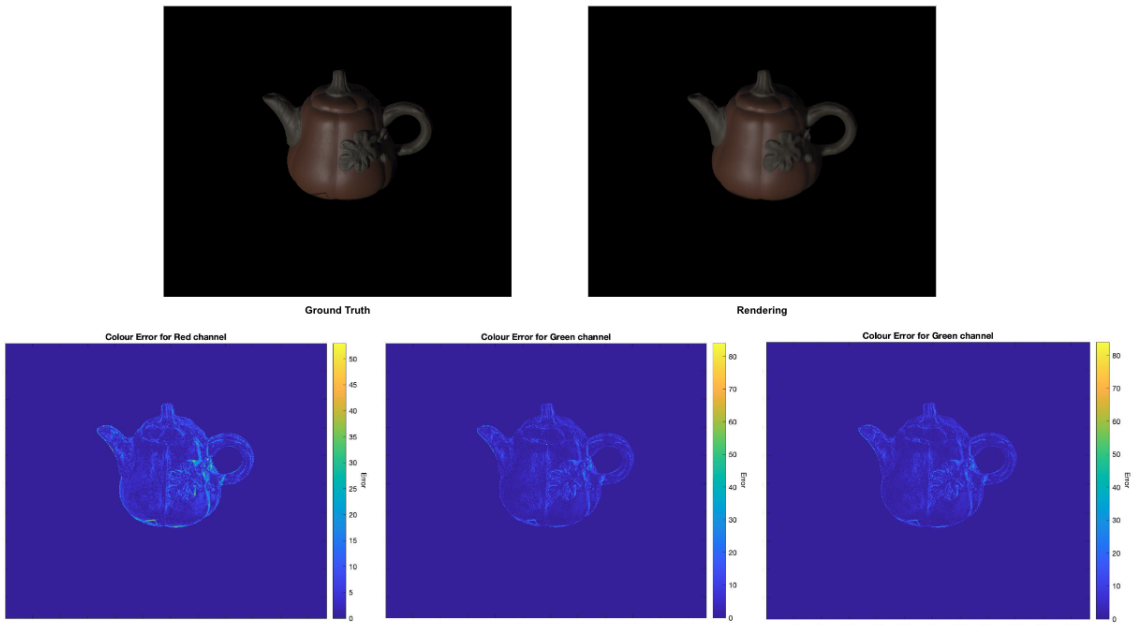


Figure 5.12: Colour Error Heatmaps for Red, Blue and Green channels for object Pot

after 200k iterations. These could be attributed to the removal of positional encoding to the view direction input and reducing the bottleneck to 3 from 256 in our Lambertian as well as learnable BRDF networks. Please refer to our GitHub repository for more information.

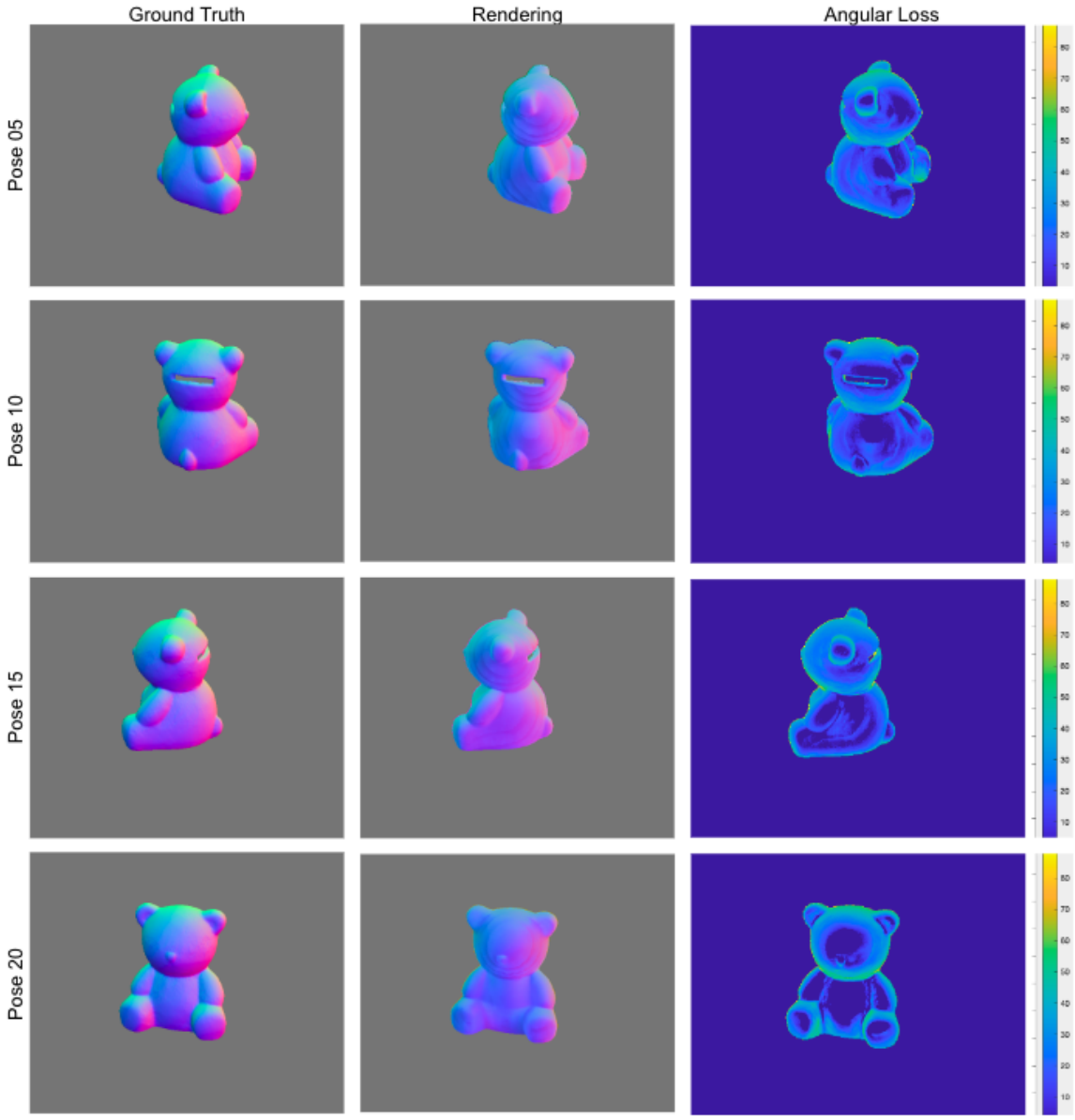


Figure 5.13: Angular Error Heatmaps for object Bear

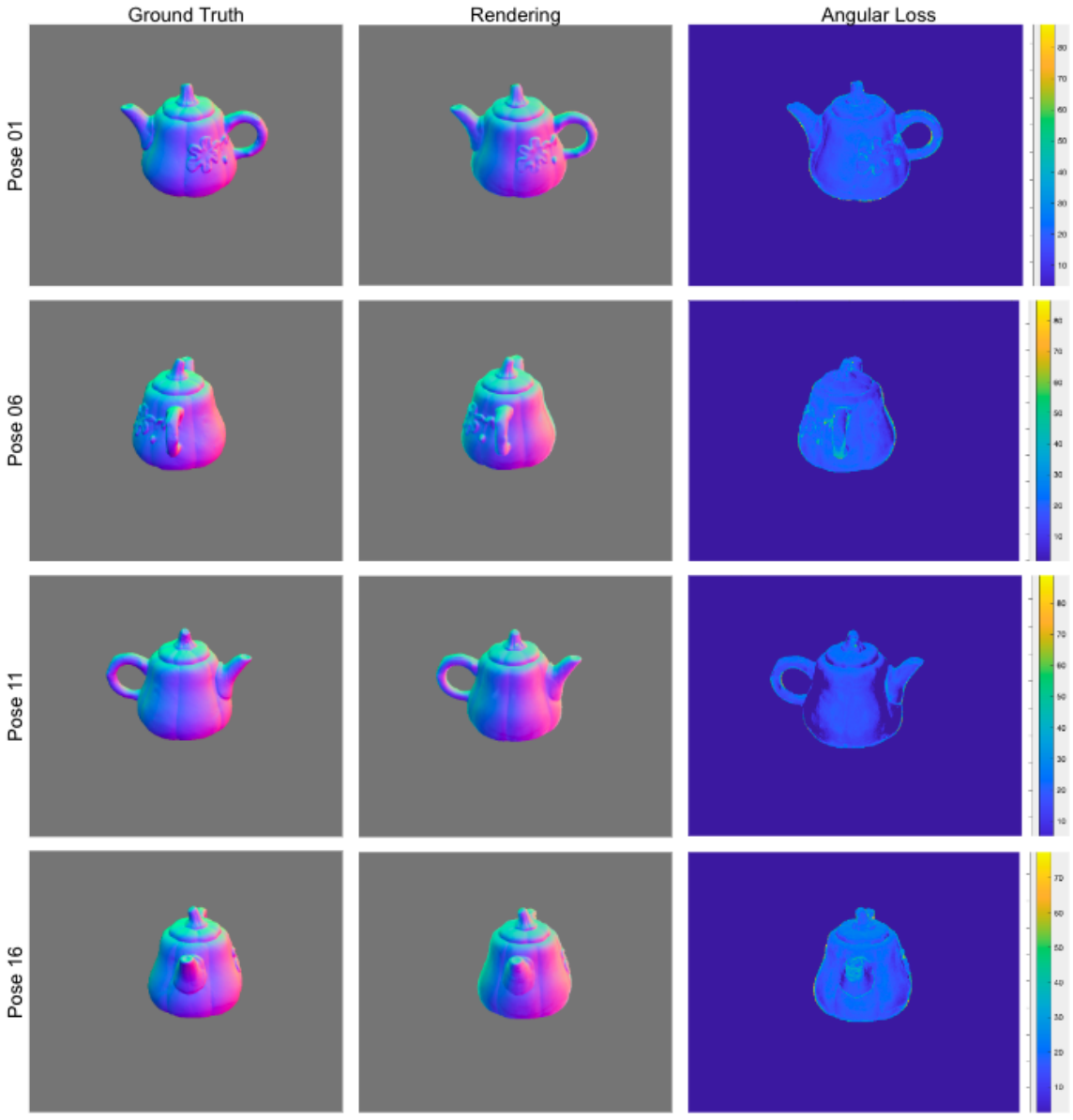


Figure 5.14: Angular Error Heatmaps for object Pot

5.3.2 Ablation Study

Study 1: Architecture Development

The study aims to build the optimal structure to represent Lambertian surfaces. We start by first reducing the `bottleneck` from the Geometry Model from 256 to 6 so as to constrain the outputs from the model (potentially to somewhat model BRDF parameters). We further replace view direction dot product to half vector dot product, however, these do not provide any improvement from our relightable results.

The next set of changes include reducing the `bottleneck` to 3 to mimic albedo maps, and further removing the Colour Network. The new colour output is the product of albedo map and light direction vectors. This drastic change provides very poor results as shown in figures

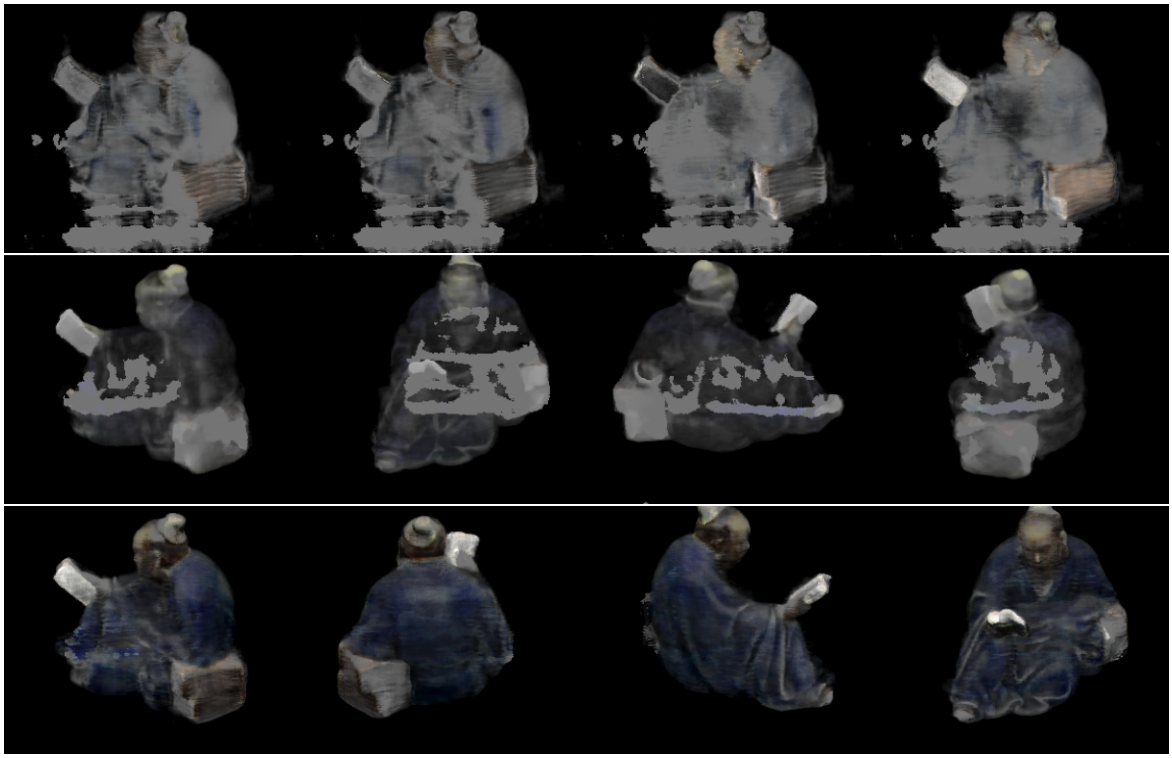


Figure 5.15: **Lambertian Model Architecture Development:** The study aims to build the optimal structure to represent Lambertian surfaces. **Top:** Figure shows the result from the reduction of bottleneck from the Geometry Model from 256 to 6 so as to constrain the outputs from the model (potentially to model BRDF parameters); **Middle:** Figure shows the result of further replacement of the view direction dot product to half vector dot product; **Bottom:** Figure shows the results from next set of changes include reducing the bottleneck to 3 to mimic albedo maps, and removing the Colour Network. The new colour output is the product of the albedo map and light direction vectors.

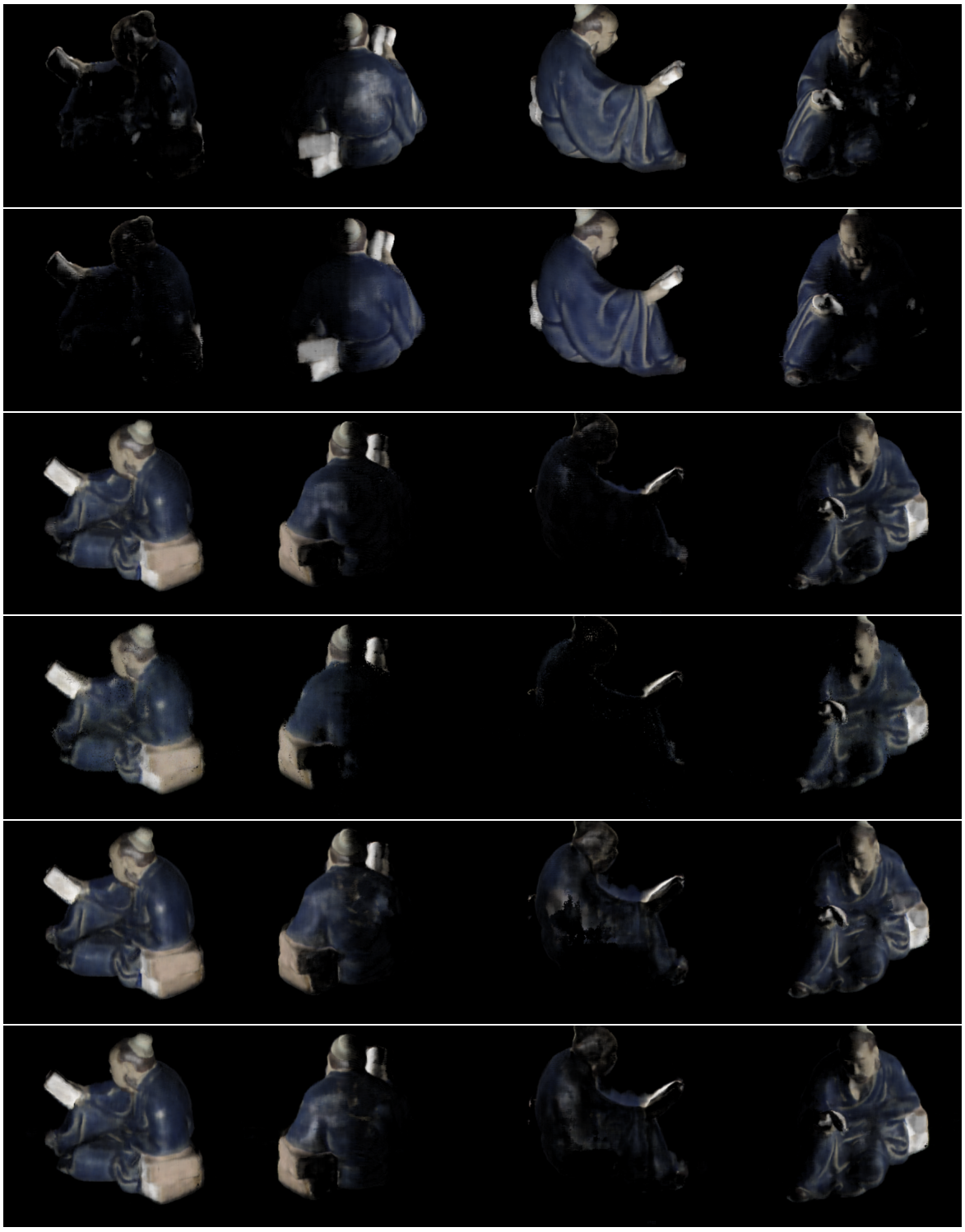


Figure 5.16: Qualitative Results for Density Loss. Top to Bottom: 0.1, 0.5, 1.0, 1.5, 2.0 and 2.5. The figure shows the renderings obtained from the ablation study on Density Loss. Qualitatively speaking, the optimal weightage has been found to be 1.5 as evident from the lack of cloudy artefact on the shadow side of the object.

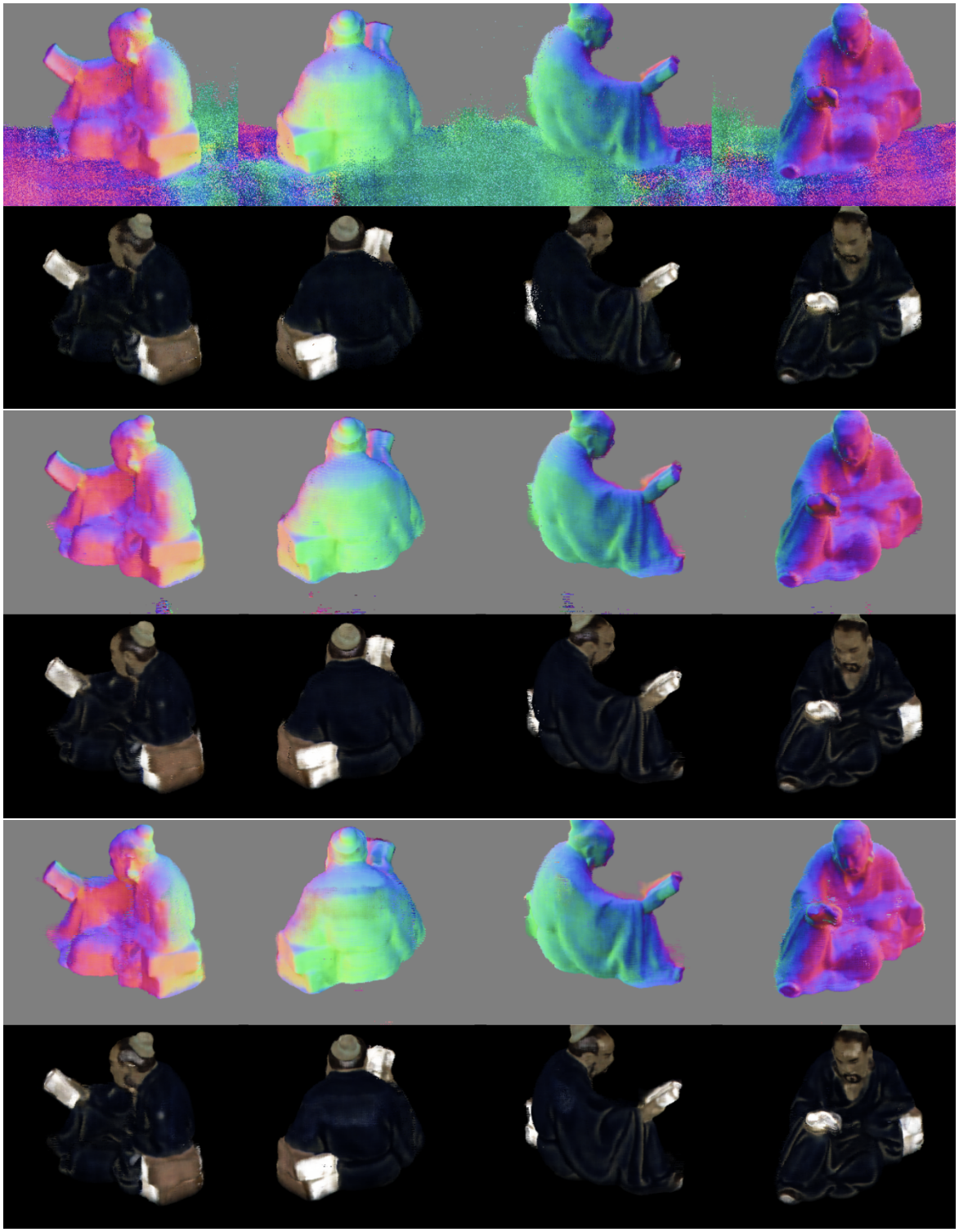


Figure 5.17: Qualitative Results for Loss Ablation Study I: Normals and Albedo Map. Top to Bottom: Density Loss 1.5 and Multires 20; Density Loss 1.5 and Multires 10; Silhouette Density Loss 5.0

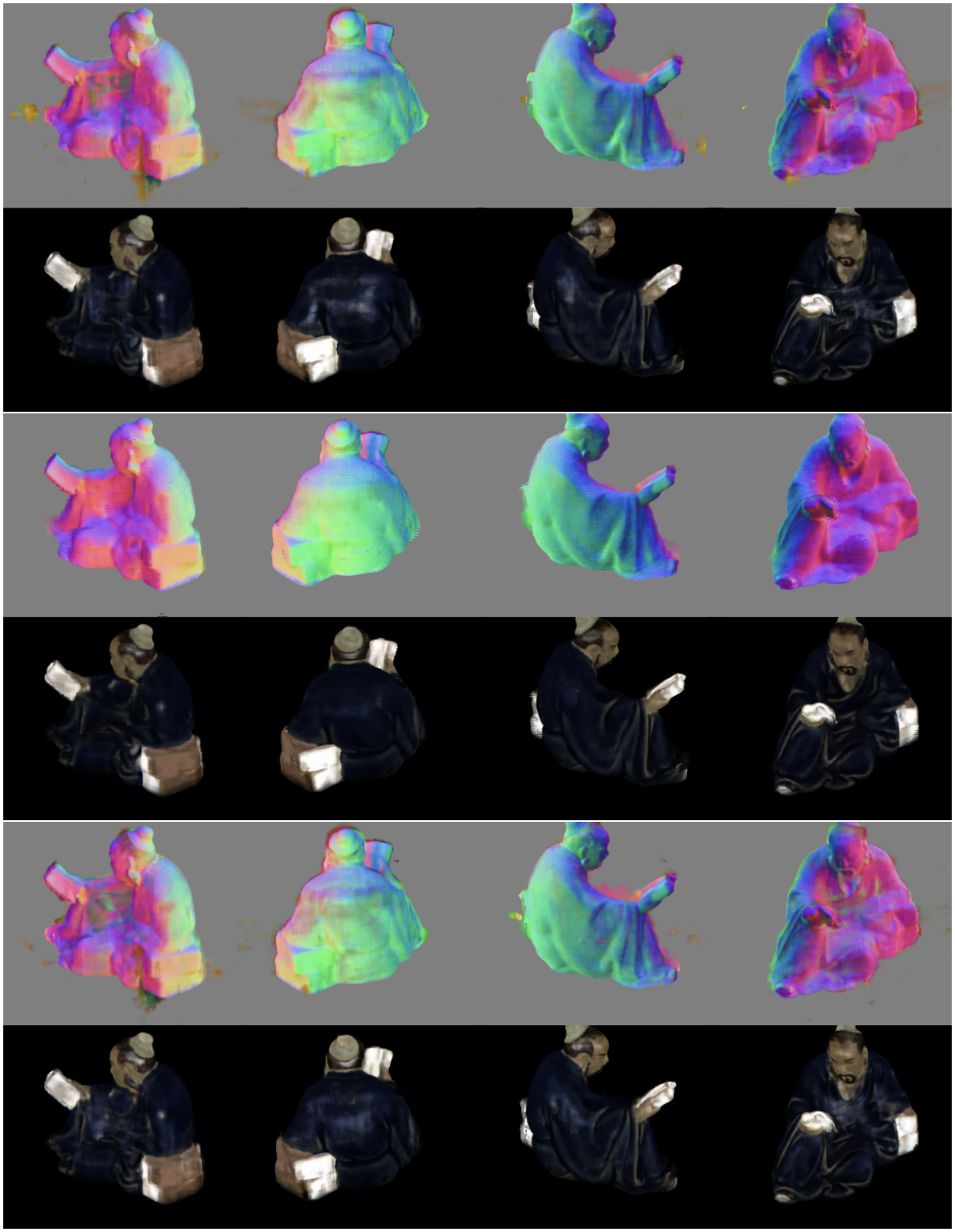


Figure 5.18: Qualitative Results for Loss Ablation Study II: Normals and Albedo Map. Top to Bottom: Density Loss 2.5 and Silhouette Density Loss 2.5; Silhouette Density Loss 2.5 and Binary Loss 1.0; Density Loss 2.5, Piecewise Loss 4.0 and Multires 12

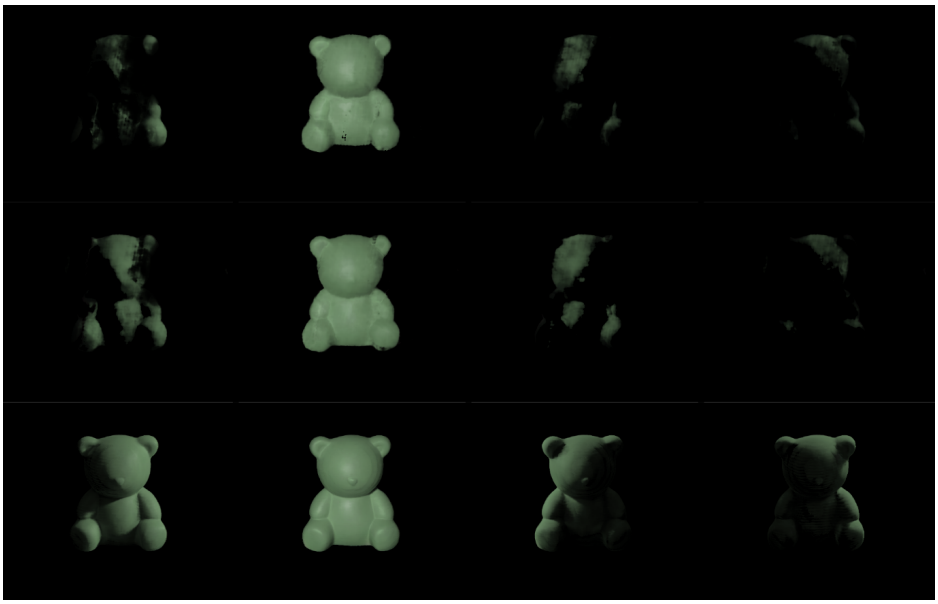


Figure 5.19: Shadow Prediction Architecture Development I. Top to Bottom: Ray Noise standard deviation 5, 10 and Multires 8 for Standard Shadow Prediction Model defined above

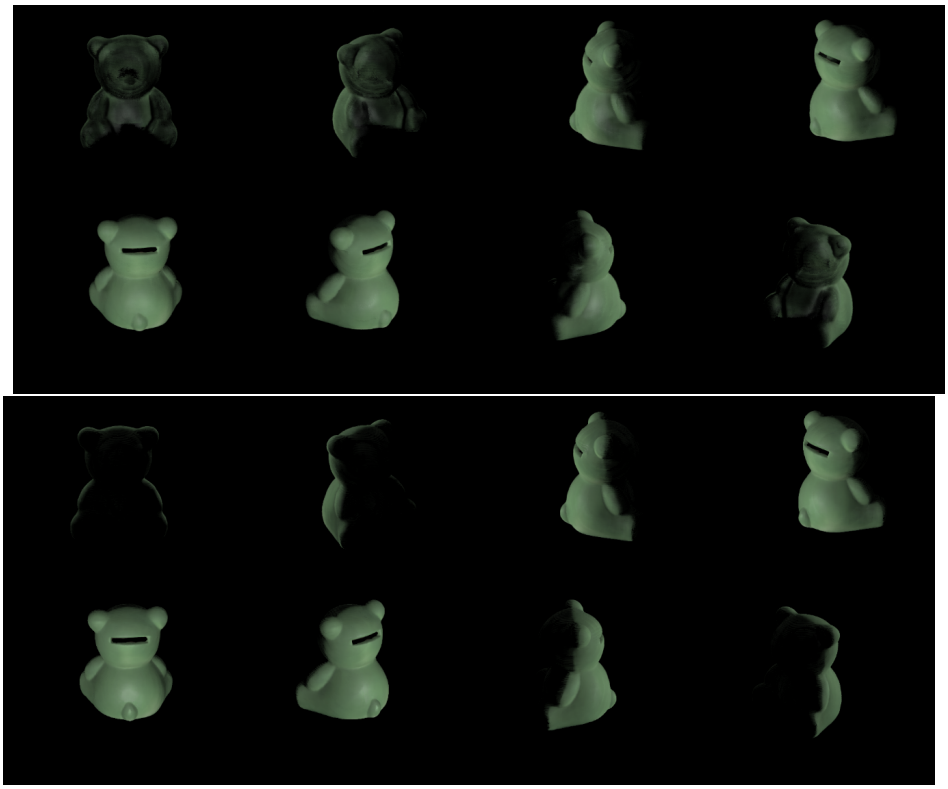


Figure 5.20: Shadow Prediction Architecture Development II. Top to Bottom: Best Lambertian Model and Standard Shadow Prediction Model

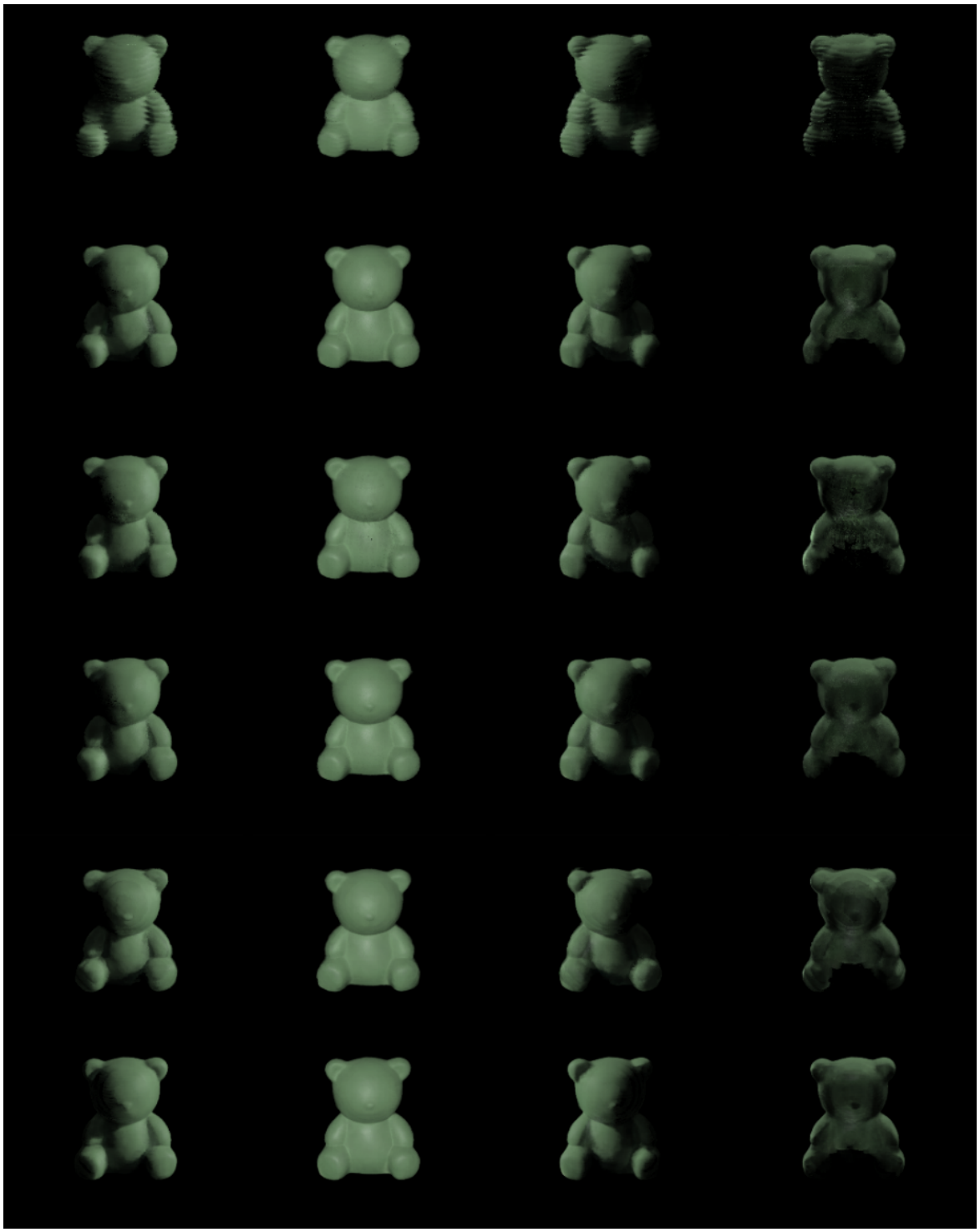


Figure 5.21: Loss Variants and Ray Noise Ablation Study. Top to Bottom: Binary Loss 0.002 Ray Noise 0.001; Binary Loss 0.0001 Ray Noise 0.005, Binary Loss 0.001 Ray Noise 0.005; Binary Loss 10 Ray Noise 0.005; Binary Loss 10 Ray Noise 0.05; Binary Loss 10 Ray Noise 1.5

5.15, 5.19 and 5.20.

Table 5.1: Surface normal estimation errors. For each object in each pose, we render normal maps from our neural model and compute mean angular error over the foreground region. In the last column, we show results averaged over poses.

Pose	**Pot**	**Bear**	**Reading**	**Buddha**	**Cow**
01	13.4	22.3	22.4	22.9	47.6
02	13.1	22.2	18.3	23.7	51.1
03	13.2	22.6	18.5	23.4	59.5
04	13.3	22.5	17.5	24.4	65.4
05	13.0	21.8	17.4	24.5	65.8
06	12.4	20.5	17.8	26.3	64.3
07	11.6	19.4	17.3	29.2	63.7
08	11.1	19.0	17.2	27.7	62.2
09	11.1	19.5	17.0	25.9	57.7
10	11.1	20.6	17.0	21.4	50.6
11	11.3	22.1	16.9	18.5	49.4
12	11.8	22.9	16.8	19.7	56.1
13	12.2	23.7	16.0	22.7	65.3
14	12.6	24.0	16.0	25.8	69.9
15	12.7	24.0	15.8	26.5	66.4
16	12.7	23.7	15.5	26.4	61.5
17	12.7	23.1	15.4	25.7	65.6
18	12.2	22.2	15.4	23.2	69.3
19	12.3	21.7	15.4	22.3	64.5
20	12.4	21.3	16.0	24.2	53.7
Mean	12.3	22.0	17.0	24.2	60.5

Study 2: Density Loss Factor

The study aims to improve the renderings provided by the Lambertian model by changing the value of the alpha loss or density loss factor to suit the model. The default value is 1

and therefore we have tried experimenting with values ranging from 0.1 to 5.0. Through this qualitative experimentation, we figured the optimum weightage to be 1.5. In figure 5.16, we have shown the effects of different density factor values.

Study 3: Multires Positional Encoding and Sampling Points

In this study, we extend the multires factor which is defined as the logarithm of maximum frequency for positional encoding for the 3D location. The factor is set to 10 with 64 sampling and importance points which provides decent renderings. We trained our model with combinations like `--multires 20 --N_sample 64`, `--multires 12 --N_sample 256`, `--multires 14 --N_sample 512`, `--multires 12 --N_sample 256` and `--multires 14 --N_sample 256`. The aim was to use the maximum amount of computing power for the highest frequency of positional encoding. The best set was `--multires 20 --N_sample 64` which we kept for all the later experiments.

Study 4: Loss Variants In this study, we have developed a variety of losses to improve the renderings - RGBs, normal maps and albedo maps. We have experimented with multiple combinations and variances, some of which include `--silhoutte 1.0`, `--silhoutte 5.0`, `--binary 1.0`, `--piecewise 4.0`, `--silhoutte 2.5 --binary 1.0`, `--binary 0.5`, `--silhoutte 0.5 --binary 1.0` and `--silhoutte 1.0 --binary 1.0`.

We finally arrive at the conclusion that different objects require completely different loss combination sets to achieve the best renderings. We have shown this from two examples in figures 5.16, 5.17, 5.18 and 5.21. We also showcase the shadow renderings from the network in figure 5.22. We furthermore provide colour loss and angular loss heatmaps for DiLiGeNT-MV objects in figures 5.10, 5.11, 5.12, 5.13 and 5.14.

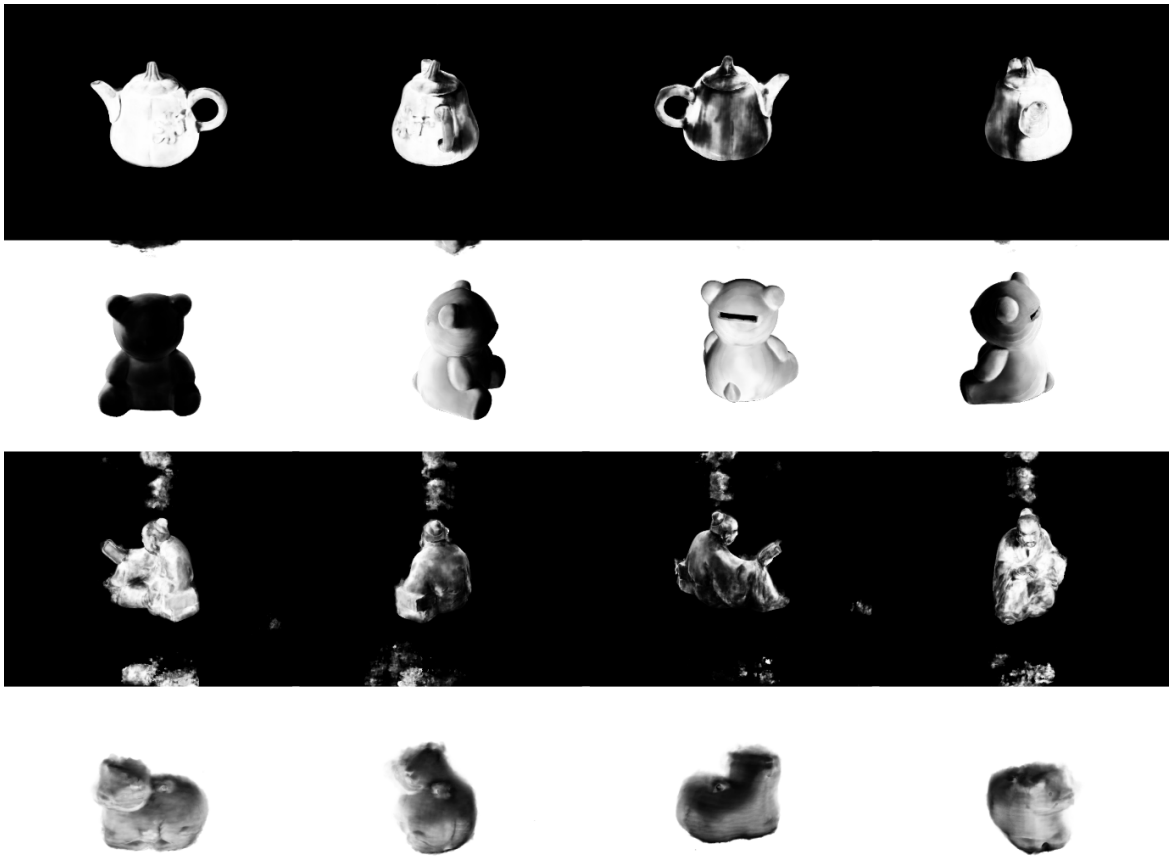


Figure 5.22: Neural Apparent BRDF Shadow Results

5.4 Conclusion

In this chapter, we presented the first attempt to use the neural volumetric density representation of NeRF combined with neural estimates of intrinsic geometric and material surface properties for the task of multiview photometric stereo. This has been achieved through incremental implementation of surface reflectance properties from a plain Lambertian model to train Lambertian surfaces (the Bear object in DiLiGeNT-MV) to incorporating a BRDF network for modelling more complex non-Lambertian surfaces (the Reading and Pot objects in DiLiGeNT-MV) and finally shadow network to correctly render self-casting and occlusion shadows.

We have shown that the approach is feasible and allows view synthesis extrapolation far from the viewpoints and light source directions seen in the training data. Though a better result could be achieved if more data was available which spreads the camera in a hemispherical

view rather than a circular ring around the object of interest. While the metric accuracy of the shape estimates does not yet match that of hand engineered multiview photometric stereo solutions, we believe there is potential to do so. We believe that the current accuracy is held back by the soft density representation (we will explain more in the upcoming chapter 6).

For objects comprising hard surfaces, a representation that directly represents surfaces is likely to provide better results. For example, the combination of a neural implicit surface with volume rendering [Wang et al.(2021)] shows great promise and could be easily combined with our image formation model and rendering process. Imposing additional constraints on the neural BRDF, either by explicitly enforcing conservation of energy or by learning a shared model across multiple objects or even a BRDF database is also likely to help constrain the problem and reduce ambiguities. Finally, the angular input to the neural BRDF is a topic for further exploration. We used a simple incident angle/half angle representation which can model basic non-Lambertian phenomena. Alternatives such as the Rusinkiewicz angles [Rusinkiewicz(1998)] might offer improved BRDF modelling.

Chapter 6

Neural Implicit Fields for Merging Monocular Photometric Stereo

Neural Apparent BRDF Fields presented in Chapter 5 allows view synthesis extrapolation far from the viewpoints and light source directions seen in the training data. However, the metric accuracy of the shape estimates does not match that of hand-engineered multi-view photometric stereo solutions. The single biggest contributor to the low quality of the recovered surfaces - particularly the extracted meshes - is the underlying soft density representation. The objects we study do not exhibit volumetric effects. They are well modelled as surfaces with local BRDF-based reflectance. Non-binary density allows the neural field to incorrectly explain appearance by using semi-transparent layers in front of the surface. Equally critically, we resorted to regressing surface normal direction with an independent neural field with no constraint to relate this to the underlying geometry represented by density. This makes the problem more ambiguous: an incorrect geometry can be made to appear correct with an incorrect normal. It also introduces additional variables to optimise since the parameters of the surface normal neural field must also be learnt.

In this chapter we replace the binary density model of NeRF with a neural signed distance function. We follow the volume rendering technique of NeuS [Wang et al.(2021)] which converts the SDF value to a soft density that can be used for NeRF-style volume rendering. As the width of the soft density distribution is gradually reduced, the rendering tends towards a hard surface-based rendering, leading to reconstruction of fine details without floating cloud artefacts. Even more beneficially for our application, the surface normal of a

signed distance function is well-defined and can be efficiently computed by using autograd [Maclaurin et al.(2015)] to compute the gradient of the SDF with respect to input position. We are therefore able to use this in our normal-based losses which then directly influence the reconstruction of the surface geometry.

We use this framework to tackle a different problem to the previous two chapters. While the problem is simpler (it does not require modelling of surface reflectance properties or illumination) it is sufficient to demonstrate the benefits of moving to an SDF-based representation and still has significant practical utility in the context of multiview photometric stereo or single view shape estimation methods more generally. Many shape-from-x methods deliver image space estimates of surface geometry and material properties. Often, geometry is in the form of a normal map, i.e. each pixel stores the local surface orientation at the point that projects to that pixel, and material properties in the form of one or more maps of intrinsic reflectance parameters, for example diffuse albedo. Example methods that deliver such estimates include:

1. A light stage [Ma et al.(2007)] in which spherical gradient illumination and polarisation-based reflectance separation is used to estimate per-view normal maps and diffuse/specular albedo maps.

2. Monocular photometric stereo [Woodham(1980)]. As described in previous chapters, such methods estimate a normal map and (usually) an albedo map from a sequence of images with fixed viewpoint and varying point light source direction. If this process is repeated from multiple viewpoints, they provide multiview normal and albedo maps.

3. Single image inverse rendering methods [Yu and Smith(2019)] estimate normal and albedo maps and illumination from single images. When applied to multiview image datasets or video they again provide multiview normal and albedo maps.

We propose to use our neural SDF based representation to tackle the problem of merging output from such methods into a single, unified model. The input to our method is a set of calibrated multiview normal and albedo maps to which we fit a neural SDF and neural field for diffuse albedo. Once trained, a mesh-based model can be extracted from the neural SDF and, thanks to its surface-based nature, this leads to considerably improved output geometry.

The rationale for this approach is that errors in single view normal or albedo maps could be corrected via information from other views, from multiview consistency constraints, from other multiview cues (such as silhouettes) or from the smoothing/interpolation behaviour of the SDF neural field. We expect that the unified model and potentially the single view normal/albedo maps rendered from the unified model will be more accurate than those from single viewpoints alone.

We begin in Section 6.1 by revising some of the methodological and implementation details of the original NeuS model. Then in Section 6.2 we describe the theory underlying our proposed model, including the normal map and albedo map losses used to fit to multiview normal and albedo maps.

6.1 NeuS Revisited

The NeuS model can be written as four MLPs:

1. $f_{\text{SDF}}^{\Theta_{\text{SDF}}} : \mathbf{x} \mapsto (\mathbf{z}, d)$ with learnable parameters Θ_{SDF} which maps a 3D location $\mathbf{x} = (x, y, z)$ to a latent code $\mathbf{z} \in \mathbb{R}^d$ and the signed distance $d \in \mathbb{R}$. d represents the distance from $\mathbf{x}$ to the closest point on the surface. The surface is defined implicitly by the set of all points with zero SDF, i.e. $\mathcal{S} = \{\mathbf{x} | f_{\text{SDF}}^{\Theta_{\text{SDF}}}(\mathbf{x}) = 0\}$.

2. $f_{\text{col}}^{\Theta_{\text{col}}} : (\mathbf{z}, \mathbf{x}, \mathbf{v}) \mapsto \mathbf{c}$ with learnable parameters Θ_{col} which maps the position $\mathbf{x}$, viewing direction $\mathbf{v} \in \mathbb{R}^3$, $\|\mathbf{v}\| = 1$ and latent code $\mathbf{z}$ to a colour $\mathbf{c} = (r, g, b)$.

3. $f_{\text{NeRF}}^{\Theta_{\text{NeRF}}} : \mathbf{x} \mapsto (\mathbf{c}, \rho)$ with learnable parameters Θ_{NeRF} which is a conventional NeRF used to model the background which maps position and viewing direction to a colour and density for standard NeRF volume rendering.

4. $f_{\text{Var}}^{\Theta_{\text{Var}}} : \sigma^{\mathbf{2}} \mapsto \mathbf{s}$ with a single dense layer of ones multiplied by the variance value, such that the only learnable component is the variance value which is used to adjust the width of the blurred density around the SDF zero level set.

6.1.1 Network Architecture

NeuS consists of two MLPs to encode SDF and colour respectively. The signed distance function $f_{\text{SDF}}^{\Theta_{\text{SDF}}}$ is modelled by an MLP that consists of 8 layers with 256 hidden units

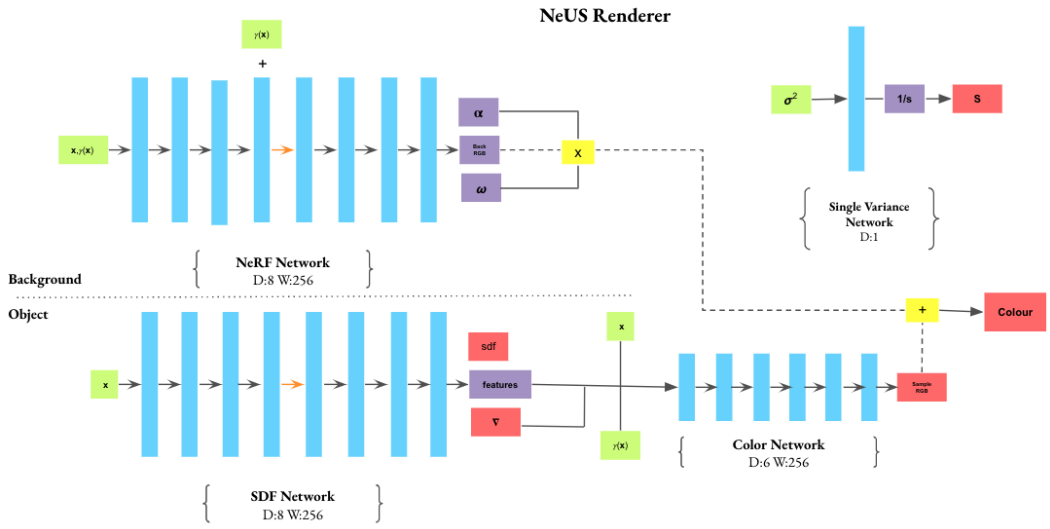


Figure 6.1: NeuS Architecture

each. The activation function is Softplus with $\beta = 100$ (taken from [Wang et al.(2021)]) for all hidden layers. A skip connection is used to connect the input with the output of the fourth layer. The function $f_{\text{col}}^{\Theta_{\text{col}}}$ for colour prediction is modelled by an MLP with 4 layers (256 hidden units), which takes not only the spatial location $\mathbf{x}$ as inputs but also the view direction $\mathbf{v}$, the normal vector of the SDF, and a 256-dimensional latent code from the SDF MLP. Positional encoding is applied to spatial location $\mathbf{x}$ with 6 frequencies and to view direction $\mathbf{v}$ with 4 frequencies. Weight normalization has been utilised to stabilize the training process. Two additional MLPs are used in practice. Since background content may not be well handled by a surface based model, a conventional NeRF is stored in $f_{\text{NeRF}}^{\Theta_{\text{NeRF}}}$ and only evaluated for background pixels lying outside the foreground mask. For our purposes we do not use this component since we are not interested in modelling background. Finally, $f_{\text{Var}}^{\Theta_{\text{Var}}}$ is a trivial single layer network that maps a learnable variance value to all sample points being evaluated. This allows the optimisation process to gradually reduce the width of the density distribution around the surface as training progresses. This means that predictions early in the training process will be blurry, providing useful information for training while the scene is coarsely reconstructed. Then later the variance can be reduced as the model approaches an exact reconstruction of the images.

6.1.2 Ray Batching

During training, NeuS generates batches of rays that are rendered and compared to the corresponding pixel. To generate a random batch for a given camera, first a batch of random pixel coordinates within the image are chosen. Each defines a ray from the camera centre towards the pixel. The ray origin and direction is augmented by additional information used within the loss computation: (i) the RGB colour of the pixel, (ii) mask flag for the pixel indicating whether it is foreground or background and (iii) the camera-to-world matrix which allows conversion between world and camera coordinates. Origin, direction and colour are 3D while mask flag is 1D meaning the final batch has size [Batchsize $\times$ 10] (the camera-to-world matrix is shared for the whole batch).

6.1.3 Alpha and Colour Computation

In the implementation, there are two types of sampling points - the sampled section points $\mathbf{q}_i = \mathbf{o} + t_i\mathbf{v}$ and the sampled mid-points $\mathbf{p}_i = \mathbf{o} + \frac{t_i+t_{i+1}}{2}\mathbf{v}$, with section length $\delta_i = t_{i+1}t_i$. The alpha value α_i can be computed as

$$\max(\frac{\Phi_s(f(\mathbf{q}_i)) - \Phi_s(f(\mathbf{q}_{i+1}))}{\Phi_s(f(\mathbf{q}_i))}, 0)$$

where Φ_s is the sigmoid function. The colour c_i can be computed as the mid-point $\mathbf{p}_i$.

6.1.4 Hierarchical Sampling

In the first instance, 64 points are uniformly sampled along the ray, then an importance sampling has been conducted for k = 4 times. The coarse probability estimation in the i-th iteration is computed by a fixed s value, which is set as 322^i (the value and calculations have been adopted from the original NeUS [Wang et al.(2021)] architecture). In each iteration, additional 16 points are sampled. Therefore, the total number of sampled points for NeuS is 128. For the 'w/o mask' setting, an extra 32 points outside the sphere have been sampled. If a background NeRF is being used then the outside scene is rendered according to the NeRF++ model.

6.2 A neural SDF model for merging multiview normal and albedo maps

For our task, we retain the SDF neural field of the original NeuS model and the same SDF volume rendering technique. The SDF represents geometry and the derivative of the SDF with respect to position defines the surface normal direction. Specifically, for a point $\mathbf{x}$ that lies on the surface, i.e. where $f_{\text{SDF}}^{\Theta_{\text{SDF}}}(\mathbf{x}) = 0$ then the unit surface normal direction is given by:

$$\mathbf{n}(\mathbf{x}) = \frac{\nabla f_{\text{SDF}}^{\Theta_{\text{SDF}}}(\mathbf{x})}{\|\nabla f_{\text{SDF}}^{\Theta_{\text{SDF}}}(\mathbf{x})\|}. \tag{6.1}$$

Note that this surface normal will be defined in world coordinates.

We replace the view-dependent colour network with a view-independent diffuse albedo network (since diffuse albedo is an intrinsic property of the surface that does not change with view direction):

$$f_{\text{albedo}}^{\Theta_{\text{albedo}}} : (\mathbf{z}, \mathbf{x}) \mapsto \mathbf{a}, \tag{6.2}$$

with learnable parameters Θ_{albedo} which maps the position $\mathbf{x} \in \mathbb{R}^3$, $\|\mathbf{v}\| = 1$ and latent code to an RGB colour albedo $\mathbf{a} = (r, g, b)$. The purpose of modelling and learning albedo in addition to geometry is that albedo is an intrinsic and view-independent quantity. It should therefore provide a strong signal for correct correspondence between views while multiple observations of the same surface point will allow noise and systematic errors to be reduced in the fitted volumetric representation. In addition, by learning a unified single model of albedo from the multiview albedo maps, we end up with a merged geometric model augmented with diffuse albedo such that the final model can be re-rendered in any pose and lighting condition.

6.2.1 Normal map loss

As input, we are provided with normal maps $\mathbf{n}_i(\mathbf{u})$, $i \in \{1, \ldots, K\}$ from K different viewpoints, where $\mathbf{u} \in \{1, \ldots, W\} \times \{1, \ldots, H\}$ is a discrete pixel coordinate in an image of size $W \times H$. $\mathbf{n}_i(\mathbf{u}) \in \mathbb{R}^3$, $\|\mathbf{n}_i(\mathbf{u})\| = 1$ contains the unit surface normal estimated for pixel $\mathbf{u}$ by photometric stereo applied to the varying-illumination images in viewpoint i. Each camera viewpoint is calibrated such that we know the pose $[\mathbf{R}_i \ \mathbf{t}_i]$ in terms of a rotation matrix $\mathbf{R}_i \in SO(3)$ and translation vector $\mathbf{t}_i \in \mathbb{R}^3$ that converts world to camera coordinates.

We compute per-pixel surface normal vectors from our neural SDF representation via volume

rendering. While blending multiple surface normal vectors does not have a direct geometric intepretation, as NeuS training converges the variance parameter that defines how density is derived from signed distance tends towards zero. This means that the volume rendering represents a hard surface. In this case, we would expect only the surface normal at the first visible surface point to contribute to the final result and so it is meaningful to compare a volume rendered surface normal to one estimated from a monocular photometric method.

Specifically, to volume render the surface normal at pixel coordinate $\mathbf{u}$ corresponding to viewing direction $\mathbf{v}(\mathbf{u})$ for the ith camera with centre $\mathbf{c}_i = -\mathbf{R}_i^T\mathbf{t}_i$ we compute:

$$\hat{\mathbf{n}}(\mathbf{u}, \mathbf{c}_i) = \int_0^\infty w(t)\mathbf{n}(\mathbf{c}_i + t\mathbf{v}(\mathbf{u}))dt. \tag{6.3}$$

The weight function $w(t)$ is defined exactly as in the NeuS method [Wang et al.(2021)]. The volume integral is discretised exactly as in NeRF.

Since a linear combination of unit vectors will not necessarily yield a unit vector, we re-normalise:

$$\bar{\mathbf{n}}(\mathbf{u}, \mathbf{c}_i) = \frac{\hat{\mathbf{n}}(\mathbf{u}, \mathbf{c}_i)}{\|\hat{\mathbf{n}}(\mathbf{u}, \mathbf{c}_i)\|}. \tag{6.4}$$

The surface normal vectors in the normal maps are supplied in the camera coordinate system since the light source directions used to estimate the surface normal maps are specified relative to the camera's viewing direction. Hence, to compare volume rendered and target surface normals, we must rotate the volume rendered normals to the camera coordinate system:

$$\bar{\mathbf{n}}_{\text{cam}}(\mathbf{u}, \mathbf{c}_i) = \mathbf{R}_i\bar{\mathbf{n}}(\mathbf{u}, \mathbf{c}_i) \tag{6.5}$$

Finally, we can compute the surface normal loss at a pixel $\mathbf{u}$ in the ith camera. We choose to maximise the dot product (minimise the negated dot product) between the volume rendered and target normals as we found minimising the angular error to be unstable:

$$\mathcal{L}_{\text{normal}}(\mathbf{u}, i) = -\bar{\mathbf{n}}_{\text{cam}}(\mathbf{u}, \mathbf{c}_i) \cdot \mathbf{n}_i(\mathbf{u}) \tag{6.6}$$

6.2.2 Albedo map loss

We are also provided with (colour) diffuse albedo maps for each viewpoint, such that $\mathbf{a}_i(\mathbf{u}) \in \mathbb{R}^3$ contains the albedo value at pixel coordinate $\mathbf{u}$ in camera i. We volume render diffuse

albedo values from our neural field in a similar manner to the surface normals:

$$\hat{\mathbf{a}}(\mathbf{u}, \mathbf{c}_i) = \int_0^\infty w(t) f_{\text{albedo}}^{\Theta_{\text{albedo}}}(\mathbf{z}, \mathbf{c}_i + t\mathbf{v}(\mathbf{u}))dt. \tag{6.7}$$

We now compute L1 loss between volume rendered and target albedo at a pixel $\mathbf{u}$ in the ith camera:

$$\mathcal{L}_{\text{albedo}}(\mathbf{u}, i) = \|\hat{\mathbf{a}}(\mathbf{u}, \mathbf{c}_i) - \mathbf{a}_i(\mathbf{u})\|_1. \tag{6.8}$$

6.2.3 Eikonal loss

We retain the Eikonal loss from the original NeuS method. The purpose of this loss is to ensure that the neural field $f_{\text{SDF}}^{\Theta_{\text{SDF}}}(\mathbf{x})$ corresponds to a true SDF which must have a gradient magnitude of 1 everywhere (since moving a distance of 1 in the direction opposite to the surface should increase the signed distance by 1). This loss is defined as:

$$\mathcal{L}_{\text{Eikonal}}(\mathbf{x}) = (\|f_{\text{SDF}}^{\Theta_{\text{SDF}}}(\mathbf{x})\| - 1)^2. \tag{6.9}$$

6.2.4 Silhouette loss

The silhouette loss encourages zero density for rays passing through pixels that lie outside the foreground mask and accumulated density equal to one for pixels that lie inside the foreground mask. On its own, this loss would amount to shape-from-silhouette. In practice, the effect of this loss is to push the SDF zero level set out of the background regions such that it aligns with the mask boundaries. Specifically, this is computed per-pixel as:

$$\mathcal{L}_{\text{Mask}}(\mathbf{u}) = \text{BCE}(M(\mathbf{u}), O(\mathbf{u})), \tag{6.10}$$

where $O(\mathbf{u}) \in [0, 1]$ is the accumulated density (i.e. sum of the blending weights) along the ray, $M(\mathbf{u}) \in \{0, 1\}$ is the ground truth mask value at pixel $\mathbf{u}$ and BCE is the binary cross entropy loss. This loss is minimised when the silhouette of the rendered SDF exactly matches the ground truth binary mask in each view.

6.2.5 Overall objective

For a given batch, we sum the normal map loss, albedo map loss and silhouette loss over all pixels in the batch and sum the Eikonal loss over all sample points along every ray in the

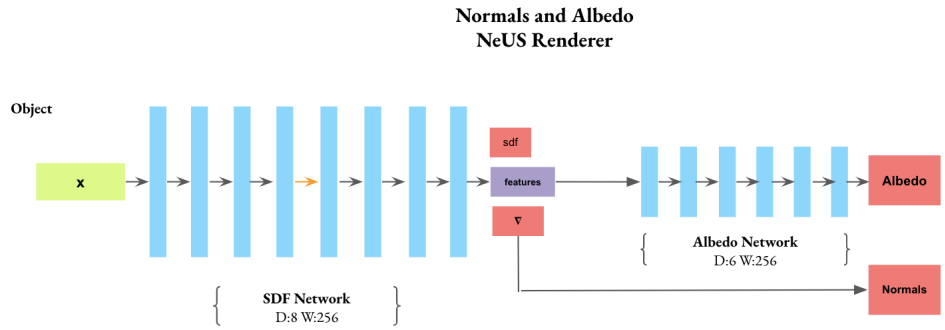


Figure 6.2: Normals and Albedo NeuS Architecture

batch.

6.2.6 Mesh extraction

Once the model is trained, we can extract a triangle mesh textured with per-vertex diffuse albedo colours.

Specifically, we use marching cubes [Lorensen and Cline(1987)] applied to the neural SDF $f_{\text{SDF}}^{\Theta_{\text{SDF}}}(\mathbf{x})$ which delivers a set of N vertices $\mathbf{X}_j \in \mathbb{R}^3$, $i \in \{1, \ldots, N\}$ and T triangles. We simply evaluate the albedo network at the positions given by the vertex coordinates in order to assign an albedo value to that vertex: $\mathbf{A}_j = f_{\text{albedo}}^{\Theta_{\text{albedo}}}(\mathbf{z}, \mathbf{X}_j)$. The textured mesh can now be rendered by any conventional renderer that accepts per-vertex material properties.

The further details on the implementation and usage of the code have been provided in the Appendices section 8.4.

6.3 Experiments

We will now be discussing our results from experimentation on varied datasets. In addition to the image sets provided by NeuS, we tested creating the neural 3D representations on multiple datasets available in the public domain. The image sets are captured in inward facing 360° view only.

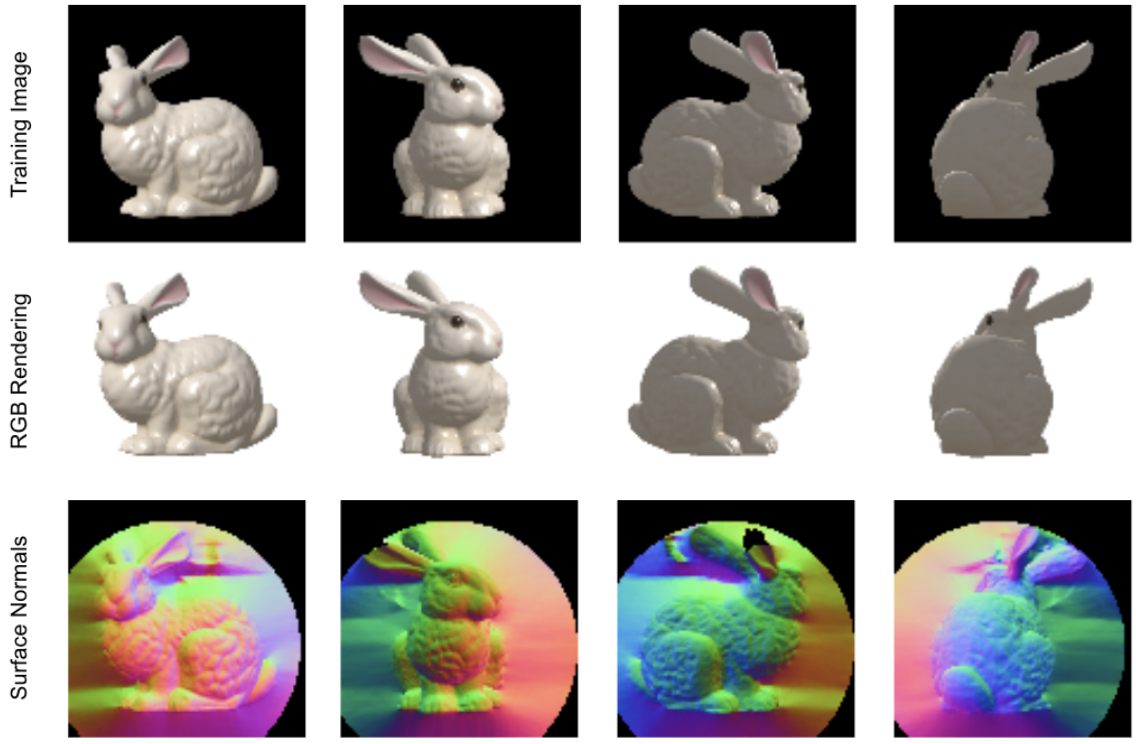


Figure 6.3: Qualitative Results for Stanford Bunny. Top to Bottom: Training Image, RGB Rendering and Surface Normals

6.3.1 Ablation Study for NeuS

Study 1: Multiple Datasets

This study examines the proficiency NeuS pipeline on multiple photometric stereo datasets. Each object has about 15 to 20 input albedo and normals map images with camera poses provided with the images.

1. **Stanford Bunny:** The synthetic data has been used with the same configuration of 360°rotated object as for Relightable NeRF 4 and Neural BRDF Fields 5 but with a single distant point light source. The results presented by NeuS possess a smoother surface. The qualitative results are shown in figure 6.3.

2. **USC Face:** The USC Face dataset consists of fifteen viewpoints of an actual face from various angles - six for the right profile and nine for the left profile. The face is captured in a lightstage which allows direct measurement of a diffuse albedo map and surface normal map. We use these estimates as the input to our method. This is the first

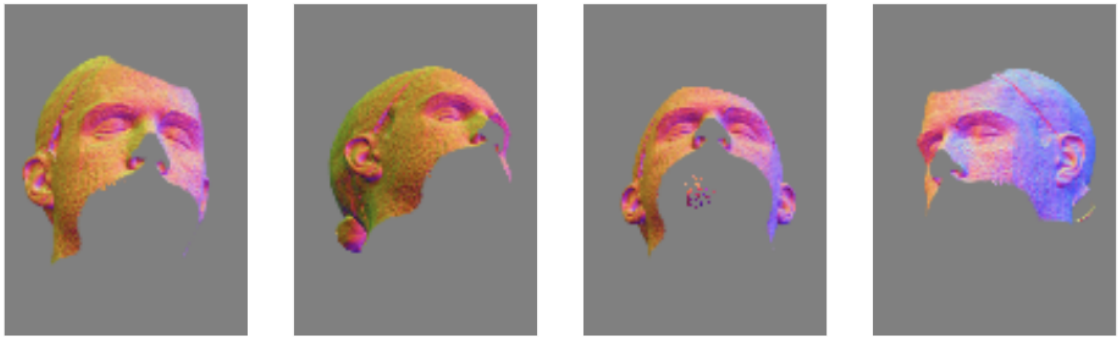


Figure 6.4: Normals Map Results for USC Face with interpolation issue

study we have performed to analyse the suitability of NeuS for human subjects. We encountered an issue while merging the left and right profiles of the face - attributed to the small number of images in the dataset and a large difference in every pose which makes the interpolation process for new poses very difficult. This is shown in figure 6.4.

We overcame this issue by dividing the dataset into two groups - `USC_FACE_L` and `USC_FACE_R` for left and right profiles, respectively. This arrangement helped achieve great results for both sub-datasets as shown in figure 6.5, however, merging of both results was not achieved. We believe the undertaking would require a larger set of images with comparatively closer poses for a faithful 360°reconstruction.

3. **DiLiGeNT-MV:** This dataset provides images under 96 illumination directions from 20 viewpoints. We give the 96 images for one viewpoint to a monocular photometric stereo method. We take the diffuse albedo map estimates from [Smith and Fang(2016)] and the surface normal maps from [Li et al.(2020b)]. These provide the noisy input to our method. We have also employed the black background mask for removing the noise during the training process. The qualitative results are shown for the DiLiGeNT-MV object, Reading in Figure 6.6.

Mask Application

The validation rendering and normals output images have been masked for better appearance and to avoid background noise from `color_error` calculation. This has been achieved by

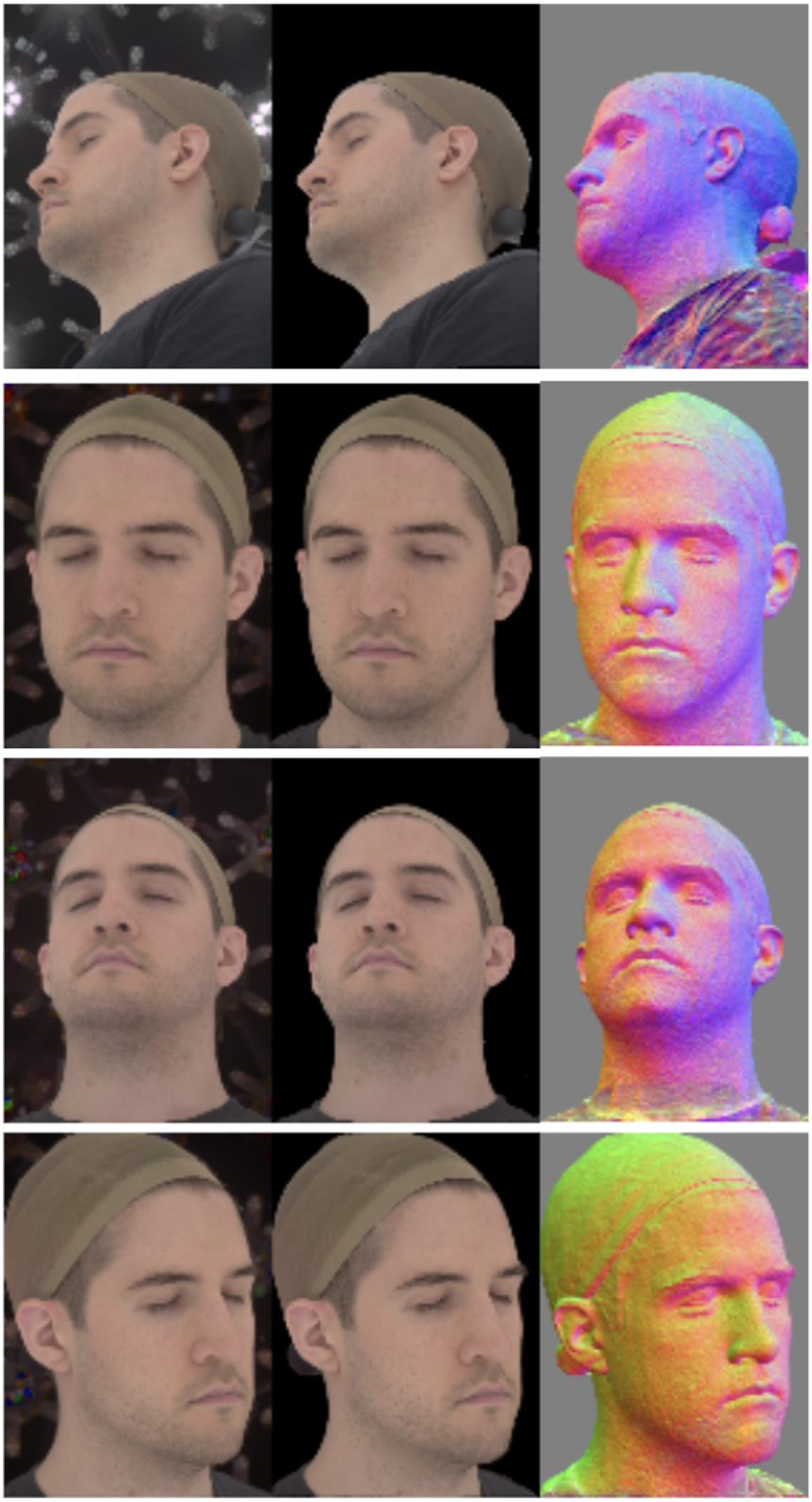


Figure 6.5: Qualitative Results for USC Face. Left to Right: Target albedo map, Rendering of estimated albedo and Surface Normals

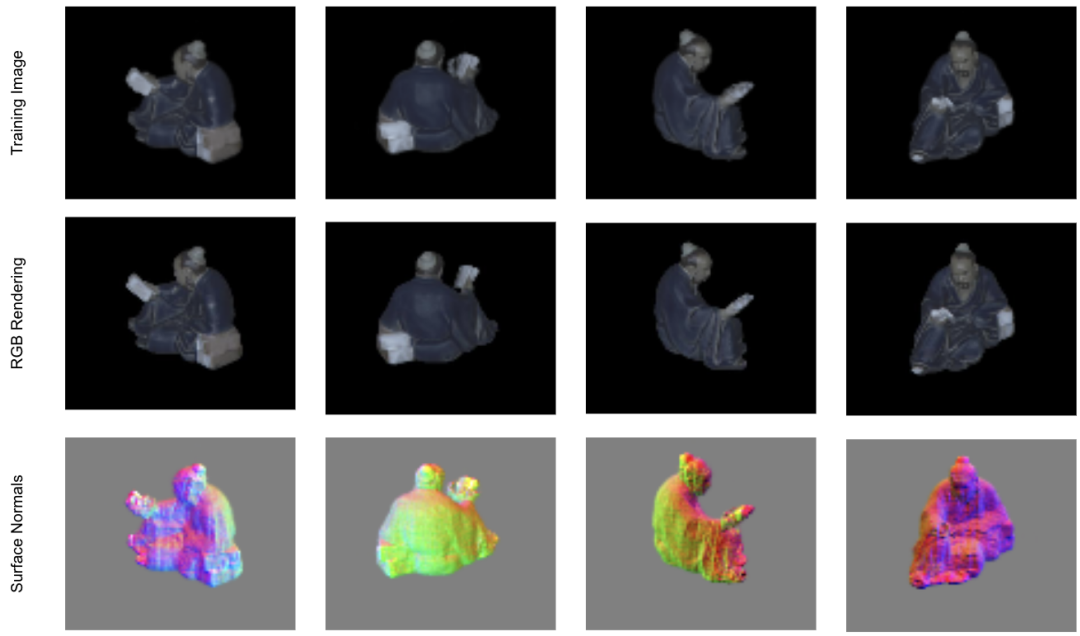


Figure 6.6: Qualitative Results for Reading Object. Top to Bottom: Target albedo map, Rendering of estimated albedo and Surface Normals

multiplying the mask of shape `(512,1)` to `color_fine` and `true_rgb` of size `(512,3)`.

Bounding Box Boundary

While experimenting with various parameters, the bounding box boundary for Stanford Bunny exhibited a multicoloured sphere around the normals rendering as shown in Figure 6.7.

The cause of the phenomena was unknown, therefore, we embarked on a debugging cycle. The first step was to remove a rotation matrix applied to the rendered normal maps. The rotation matrix can be defined as an inverse of the rotation part of the camera poses, `inv(pose_all[idx, :3, :3])`, matrix supplied in the dataset. The second step was to convert all NumPy arrays to tensors which would avoid any incompatibility while applying tensor operations. The third step was to normalise the normals rendered at every step which would force the sum of the squares of the x, y and z components to 1 and also to swap axes x and z. The final step was to remove the `'inside_sphere'` to achieve the results without the multicolour circle around the object.

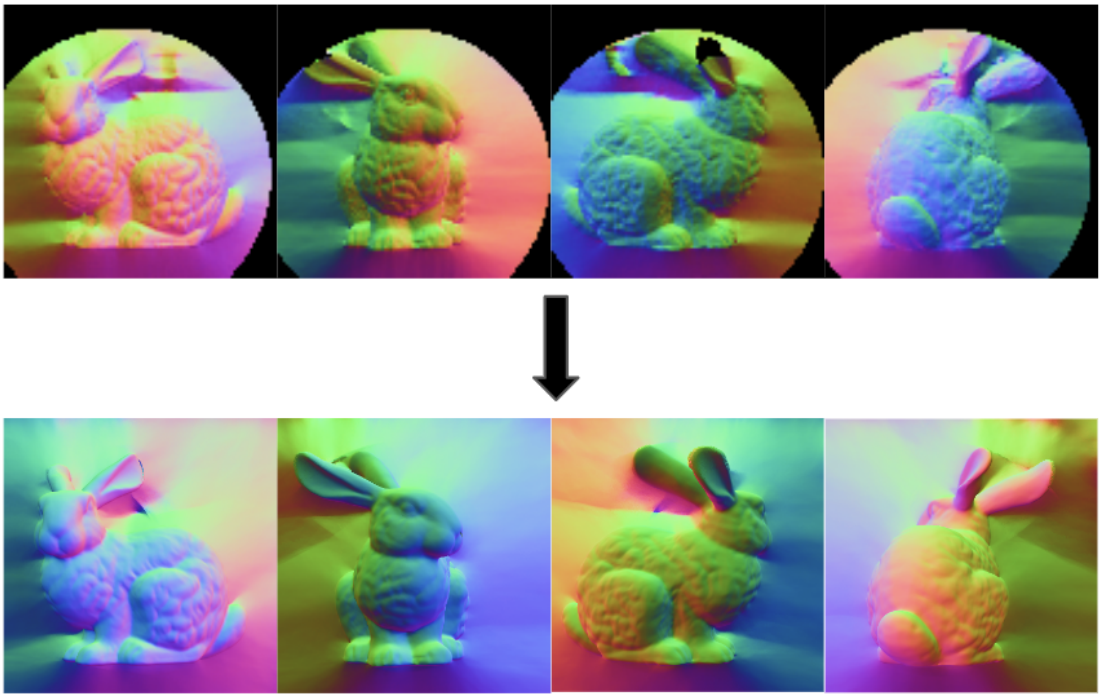


Figure 6.7: Fixed Inner Sphere and Axes for Normal Maps

6.3.2 Ablation Study for Normals and Albedo NeuS

The ablation study is designed to optimise the Normals and Albedo NeuS model and fine-tune the hyperparameters for training any dataset.

Study 1: Weighted Normal Loss

The study examines the change of normal loss weightage factor on the overall loss and improvement of renderings from the network. From the various studies and training performed on weights ranging from 0.0001 to 1, we conclude that the best results are achieved from factor 0.5 as shown in figure 6.8.

Study 2: View Direction Dependency

The study examines the effect of removing positional encoding view direction on the renderings. The quality of final renderings is not affected, however, the training process speeds up considerably.

Figure 6.8: Renderings with weighted normal loss. Left to Right: 0.0, 0.5, 1.0, 2.0, 4.0

Figure 6.9: **Qualitative Results for Angular Error**. Left to Right: Raw Normals from NeuS, Ground Truth Normals, Mask, Rotated Normals

Figure 6.10: **Stanford Bunny Mesh.** Left to Right: Ground Truth Mesh, Rendering of our reconstructed mesh and comparison between both Meshes

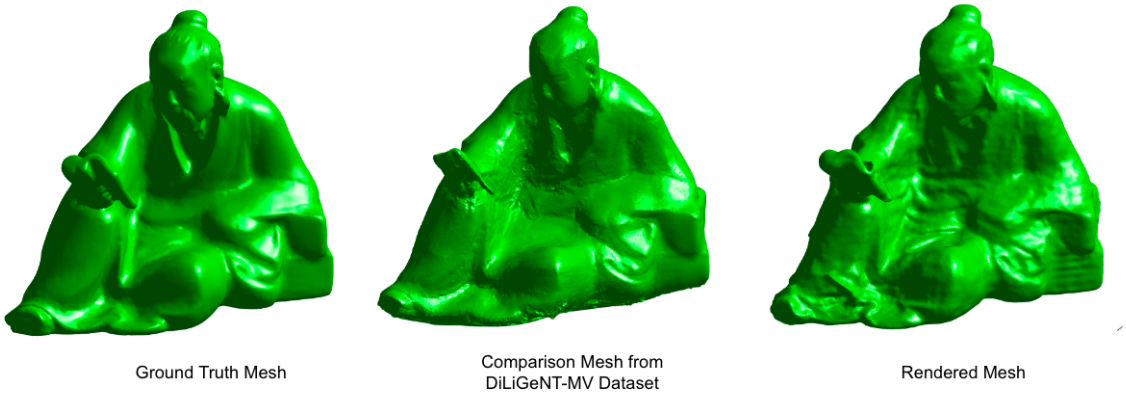


Figure 6.11: **Results for Reading Object.** Left to Right: Ground Truth Mesh, Comparison reconstruction from DiLiGeNT-MV and our reconstruction.

Figure 6.12: **Results for Pot2 Object.** Left to Right: Ground Truth Mesh, Comparison reconstruction from DiLiGeNT-MV and our reconstruction.

Study 3: Number of Sampling and Importance Points

The study examines the effect of change in the number of sampling and importance points on the rendered mesh. For both, the default was set to 64 in the original NeuS implementation. We experimented and trained with values 64, 128 and 256 for both the number of samples and importance points in all the possible combinations, however, there was no improvement in the renderings or the mesh. Thus, we reverted back to 64 as the default for all our succeeding training.

Study 4: Hyperparameters for SDF Network

- **Number of Hidden Layer Nodes for SDF Network.** The study examines the effect of change in the number of hidden layer nodes of the SDF network on the

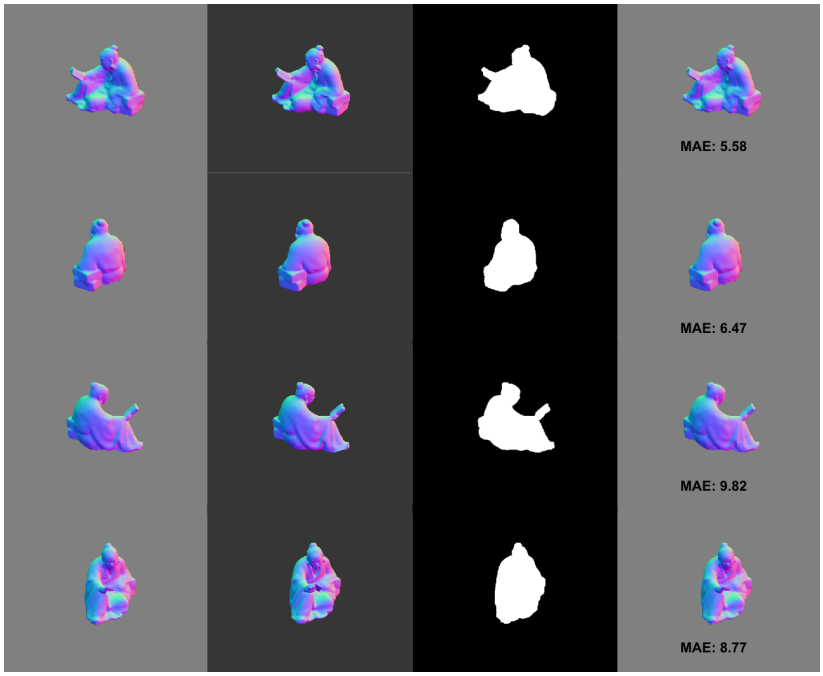


Figure 6.13: **Mean Angular Error for DiLiGeNT-MV object Reading.** Left to Right: Ground Truth Normals, Mask, Normals from NeuS for Poses 5, 10, 15, 20

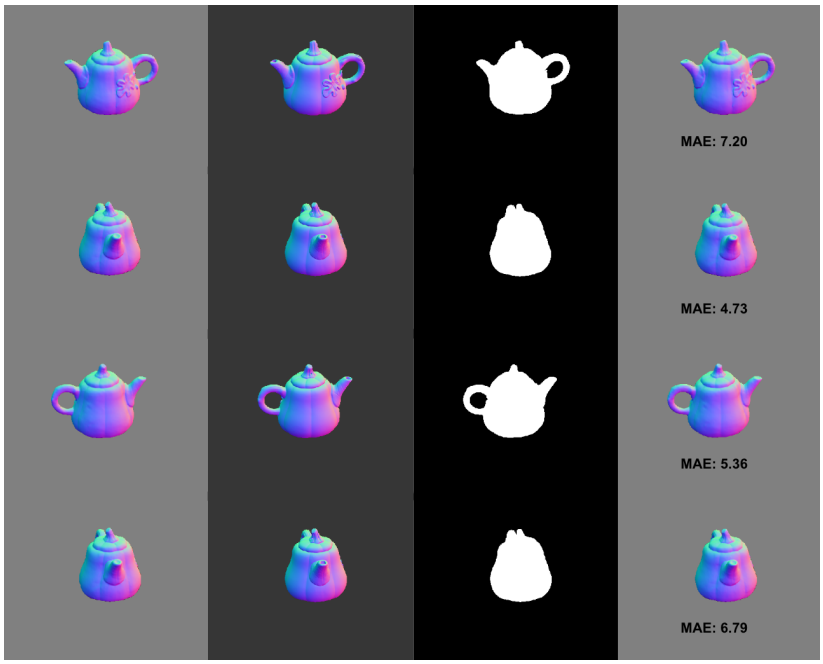


Figure 6.14: **Mean Angular Error for DiLiGeNT-MV object Pot2.** Left to Right: Ground Truth Normals, Mask, Normals from NeuS for Poses 5, 10, 15, 20

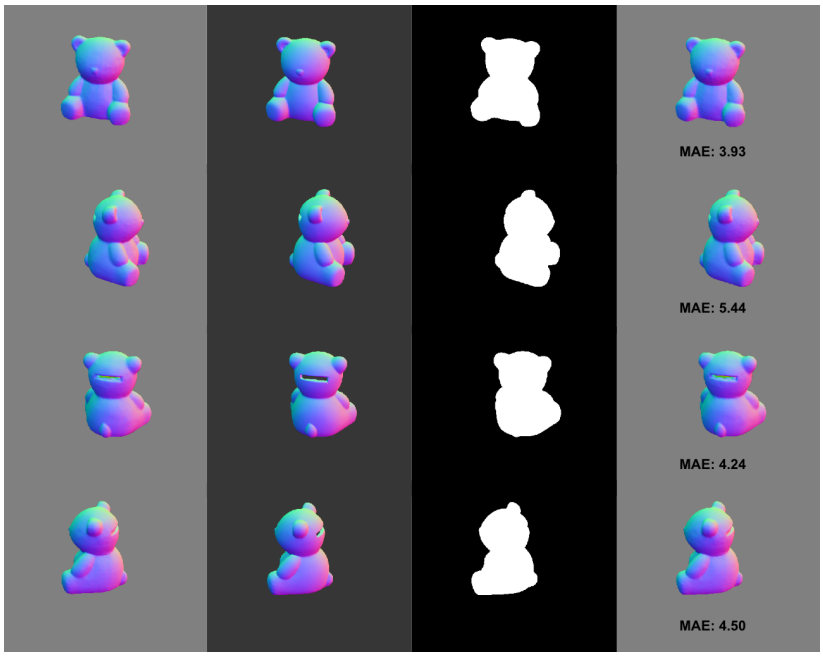


Figure 6.15: **Mean Angular Error for DiLiGeNT-MV object Bear.** Left to Right: Ground Truth Normals, Mask, Normals from NeuS for Poses 5, 10, 15, 20

Figure 6.16: **Relighted Mesh for DiLiGeNT-MV Objects.**

Figure 6.17: Results for number of hidden nodes 64 and 128 for normal loss weight 0.0 and 2.0

renderings and mesh. We experimented and trained with values 64 and 128 for normal loss weight 0.0 and 2.0. We found that there were minimal changes in the rendering output as shown in figure 6.17.

- **Number of Hidden Layers for SDF Network.** The study examines the effect of change in the number of hidden layers of the SDF network on the renderings and mesh. We experimented and trained with values, 2 and 4, for normal loss weighted at 0.0 and 2.0. We found that there were minimal changes in the rendering output as shown in figure 6.18.

- **Skips for SDF Network.** The study examines the effect of change in the number of skips for the SDF network on the renderings and mesh. We experimented and trained with values, 8 and 16, for normal loss weighted at 0.0 and 2.0. We found that there were minimal changes in the rendering output as shown in figure 6.19.

- **Multires for SDF Network.** The study examines the effect of change in the multires value for the SDF network on the renderings and mesh. We experimented and trained with values, 3 and 12, for normal loss weighted at 0.0 and 2.0. We found that there were minimal changes in the rendering output as shown in figure 6.20.

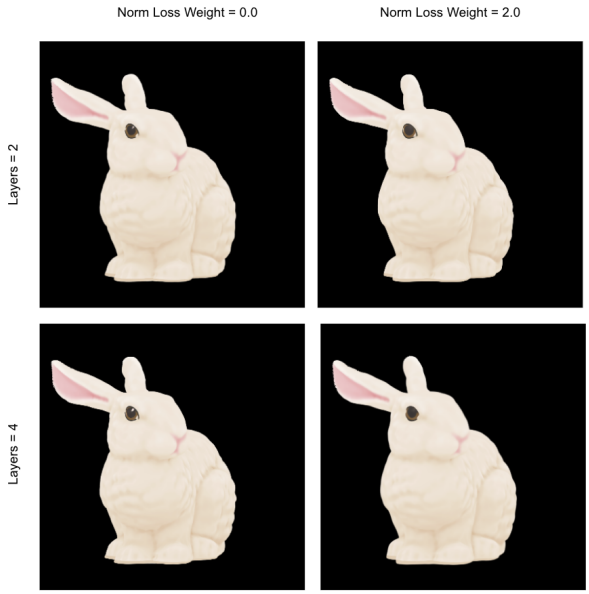


Figure 6.18: Results for number of hidden layers 2 and 4 for normal loss weight 0.0 and 2.0

Figure 6.19: Results for number of skips 8 and 16 for normal loss weight 0.0 and 2.0

Figure 6.20: Results for multi-resolution values 3 and 12 for normal loss weight 0.0 and 2.0

Quantitative evaluation

We computed two quantitative results for each object - Mean Angular Error and Mesh Distance Error. The mean angular error for the ith view is computed as the angle between ground truth and rendered surface normal vectors, averaged over all foreground pixels in the ith view:

$$\mathrm{MAE}(i) = \frac{1}{\mathcal{F}_i} \sum_{\mathbf{u} \in \mathcal{F}_i} \arccos\left[\bar{\mathbf{n}}_{\mathrm{cam}}(\mathbf{u}) \cdot \mathbf{n}_i(\mathbf{u})\right], \tag{6.11}$$

where $\mathcal{F}_i$ is the set of foreground pixels for the ith view.

For the mesh distance error, we follow the same procedure as for the error metrics reported in the original Diligent-MV benchmark. This amounts to the following process. For each vertex in the ground truth mesh, find the 50 nearest neighbours in the reconstructed mesh and average their Euclidean distances from the ground truth vertex. Then average this quantity over all ground truth vertices. The purpose of using 50 nearest neighbours rather than one is to smooth the mesh distance error and make it less sensitive to noisy vertices or small spikes.

For the Stanford Bunny, our best results provide a Mean Angular Error of 10.6593 and Mesh Distance Error as $5.7006e^{-4}$ as shown in Figures 6.9 and 6.10. For DiLiGeNT objects Pot, Bear & Reading, our best results provide a Mean Angular Error of 6.13, 4.66 & 7.59 and Mesh

Distance Error of 0.8258, 4.1347 & 15.0778 and the rest of the results are shown in Tables 6.1, 6.2 and 6.3. Additional Normal Maps and Meshes are provided in Figures 6.13, 6.14, 6.16, 6.11 and 6.12. Table 6.4 shows our mean angular errors as compared to multiple traditional photometric stereo methods and exhibits better performance in the majority of cases. Note specifically that our input normal maps come from the method of Li [Li et al.(2020b)]. In two out of three cases, our merging process reduces the error of the estimated surface normals showing that the multiview information helps improve the quality of the reconstruction.

Table 6.1: Comparison of Surface Normals Estimation Errors for **DiLiGeNT-MV object Pot** for Neural Apparent BRDF and NeuS techniques.

Pose	**NABF**	**NeuS**
Pot		
01	13.4	8.07
02	13.1	6.79
03	13.2	7.27
04	13.3	7.72
05	13.0	7.20
06	12.4	6.72
07	11.6	6.36
08	11.1	5.41
09	11.1	5.50
10	11.1	4.73
11	11.3	5.02
12	11.8	4.97
13	12.2	5.14
14	12.6	5.08
15	12.7	5.36
16	12.7	5.88
17	12.7	5.99
18	12.2	6.24

Continued on next page

Table 6.1 – continued from previous page

Pose	NABF	NeuS
19	12.3	6.43
20	12.4	6.79
Mean	12.3	6.13

Table 6.2: Comparison of Surface Normals Estimation Errors for **DiLiGeNT-MV object Bear** for Neural Apparent BRDF and NeuS techniques.

Pose	NABF	NeuS
Bear		
01	22.3	4.77
02	22.2	4.62
03	22.6	4.68
04	22.5	4.35
05	21.8	3.93
06	20.5	4.02
07	19.4	4.40
08	19.0	4.82
09	19.5	5.08
10	20.6	5.44
11	22.1	5.50
12	22.9	5.28
13	23.7	4.48
14	24.0	4.56
15	24.0	4.24
16	23.7	4.35
17	23.1	4.40
18	22.2	4.81
19	21.7	4.94

Continued on next page

Table 6.2 – continued from previous page

Pose	NABF	NeuS
20	21.3	4.50
Mean	22.0	4.66

Table 6.3: Comparison of Surface Normals Estimation Errors for **DiLiGeNT-MV object Reading** for Neural Apparent BRDF and NeuS techniques.

Pose	NABF	NeuS
Reading		
01	22.4	9.37
02	18.3	8.49
03	18.5	7.15
04	17.5	5.12
05	17.4	5.58
06	17.8	5.60
07	17.3	6.47
08	17.2	6.12
09	17.0	6.39
10	17.0	6.47
11	16.9	6.79
12	16.8	7.02
13	16.0	7.72
14	16.0	9.72
15	15.8	9.82
16	15.5	9.36
17	15.4	9.88
18	15.4	7.97
19	15.4	8.05

Continued on next page

Table 6.3 – continued from previous page

Pose	NABF	NeuS
20	16.0	8.77
Mean	17.0	7.59

Table 6.4: Mean Normal Angular Errors (IN DEGREE) ON **DiLiGeNT-MV objects Pot, Bear & Reading** for various Traditional Photometric Stereo Methods (LS [Woodham(1980)], IA14 [Ikehata et al.(2014)], ST14 [Shi et al.(2013)], HS15 [Han and Shen(2015)], SH17 [Shen et al.(2016)], ZK19 [Zheng et al.(2019)], PJ16 [Park et al.(2016)] and LI [Li et al.(2020b)]) against Ours.

	LS	IA14	ST14	HS15	SH17	ZK19	PJ16	LI	Ours
Pot	14.65	8.77	8.78	9.80	8.43	8.09	15.31	6.79	**6.13**
Bear	8.39	7.11	6.12	5.12	5.31	4.65	12.63	**4.45**	4.66
Reading	19.80	14.19	13.63	14.56	13.00	12.77	12.23	8.74	**7.59**

6.4 Conclusion

In this chapter, we showed how to use a neural implicit surface and neural field for albedo in order to merge the output of monocular photometric stereo across multiple views. This provides a method that can take the output of any monocular photometric stereo algorithm and merge it into a single unified mesh with per-vertex albedo. This can be seen as a modern take on the classical problems of mesh and texture stitching.

Compared to the results in the previous chapters, the use of an SDF-based representation has greatly increased the quality of the reconstructed surface such that we are able to improve upon state-of-the-art photometric stereo, at least for some objects.

The neural SDF provides a natural representation to utilise multiple constraints within the same framework. The normal loss encourages the normal to the recovered SDF to match the normals estimated by photometric stereo. The albedo loss encourages the albedo neural field

to accurately reconstruct the estimated albedo maps. Evaluating these losses across views encourages multiview consistency and helps to average out errors that occur in one view. The SDF itself introduces a smoothness prior that means it does a good job of interpolating over regions with unreliable normals and/or albedo and even extrapolating to fill unseen regions of the object.

The results in this chapter show great promise for the use of neural SDFs in a photometric stereo framework. However, in the presented method, the monocular photometric stereo method is completely independent from the merging. A promising direction would be to combine the method from the previous chapter with the representation in this chapter in order to solve multiview photometric stereo from scratch as an end to end process.

Chapter 7

Conclusion and Future Work

In this work, we extended the application of Neural Radiance Fields to render from overlapping planar dataset to multiview photometric stereo data. We also introduce a very simple relightable implementation of NeRF which can easily model standard 360°renderings from novel viewpoints. Furthermore, we extend our work to improve the appearance of rendered mesh surfaces through the signed distance function as applied in Neural Implicit Surfaces.

In the preliminaries chapter, we first introduced the theoretical concepts of the Lambertian model and the Bidirectional Reflectance Distribution function which we are trying to model. We defined the core principles and constraints of a BRDF surface including the rule of reciprocity, the rule of positivity and the law of conservation of energy. We also introduced the Stanford Bunny and DiLiGeNT-MV datasets which acted as objects to train on for all models discussed in the work.

In the literature review chapter, we provided a detailed explanation of the predecessors of NeRF which include neural 3D representations, view synthesis through image-based rendering and Local Light Field Fusion. We provide an in-depth explanation of the process of rendering novel views through NeRF. Due to the massive boom of NeRF-related technique emerging in 3D graphics modelling in the early 2020s, we have enumerated an exhaustive library of models aimed at different tasks, some of them include faster inferencing and training, handling deformation and relighting, overall object generalisation and video rendering. We conclude by critically analysing these techniques and fetching multiple gaps in knowledge expandable through our work.

Our first novel contribution, Relightable NeRF as mentioned in chapter 4, delivered a simple

approach to transform a vanilla NeRF to view- and light-dependent. This was achieved by incorporating distant point light source direction information in the training set along with images and poses and sending the information as input to the relightable multi-layer perceptron in a positional encoded form. We treat this relightable multi-layer perceptron as a black box and do not explicitly define the outputs from the network. The results from our ablation study on Stanford Bunny and DiLiGeNT-MV exhibit a drawback in our method which shows that the model is proficient at replicating novel viewpoints close to the ones in the training set, however, it falls short when the viewpoints are far off. This has been attributed to the occurrence of cloudy artefacts and the failure to model self-cast shadows and geometry. We resolve the aforementioned issue through our succeeding work.

The second novel contribution, Neural BRDF Fields for multiview photometric stereo as mentioned in chapter 5, presents a method to tackle the multiview photometric stereo problem using an extension of Neural Radiance Fields, conditioned on light source direction. We explicitly regress the surface normal direction with the geometry network and decompose the second network into networks for BRDF (with suitable physical priors) and shadowing. The model is a combination of three learnable MLPs and fixed functions - Geometry Network, BDRF Network and Shadow Network. We have shown that the approach is feasible and allows view synthesis extrapolation far from the viewpoints and light source directions present in the training data. While the metric accuracy of the shape estimates does not yet match that of hand-engineered multiview photometric stereo solutions, we believe there is potential to do so. We believe that the current accuracy is held back by the soft density representation which is the core idea for our final novel contribution.

Our final novel contribution, Neural Implicit Fields for Merging Monocular Photometric Stereo, as mentioned in chapter 6, aims to improve the surface texture of the geometrical mesh rendered by the network. We utilise the original NeUS work as our starting base which we convert into Normals and Albedo NeUS by replacing RBG images with albedo and additional normal maps for baked-in BRDF features. We, furthermore, implemented weighted normal loss to the total loss to explicitly learn from the normal maps provided in the dataset. Finally, there are multiple improvements to the model we would have liked to implement, however, due to time constraints we leave as a suggestion for future work discussed later.

In conclusion, the main aim of the work is to extend the implementation of Neural Radiance Fields into the multiview photometric stereo domain, while achieving distant light source relighting and accurate geometry reconstruction. This was achieved through a set of incremental changes from a simple relightable model (able to produce novel viewpoint reconstruction to some extent) to a Lambertian model (able to accurately model Lambertian surfaces) to a Neural BRDF (following BRDF surface constraints and accurately model novel viewpoints far off from the training set) to finally, a partial implementation of Neural Implicit Surface for Merging Monocular Photometric Stereo, providing noise-free surface geometry reconstruction. For general ethical consideration, as we are using publicly available data we are adhering to the terms and conditions of the license agreement and any ethical issues regarding the future applications of this work has been discussed under each future work category.

Future Work

Since commencing the work for this PhD, there has been an explosion of interest in work seeking to extend the original NeRF idea. Some of these methods consider relighting while others lean towards physics-based decomposition within the framework of NeRF and therefore, there are multiple overlapping ways in which the work in this thesis could be extended.

Furthering our work from the Neural Implicit Fields, there are multiple avenues of future work which could be explored. These include full implementation of the Neural BRDF framework within the NeUS framework, training the model on a variety of datasets which are different from Stanford Bunny and DiLiGeNT-MV and creating a VR/AR front-end application which is able to place these objects in different environments. They are discussed more in detail below:

Neural BRDF Fields framework within the NeUS framework

Chapter 6 switched from using the NeRF density representation to a neural signed distance function. In the context of merging the output of a monocular photometric stereo algorithm, this proved to be extremely beneficial for producing surfaces with higher quality and accuracy than those in Chapters 4 and 5. We believe that this is due to the change in surface representation, with SDFs being much more appropriate for representing surfaces than soft

density which is better suited to representing volumetric effects.

Therefore, the most obvious and promising line of future work would be to combine these two strands of work: the surface representation from Chapter 6 with the multiview photometric stereo method from Chapter 5, providing a standalone end-to-end multiview photometric stereo algorithm that does not rely on the output of another algorithm. Specifically, this would involve augmenting the SDF network with a learnable BRDF network (to replace the albedo network) and a shadow network to model occlusions.

Training with different datasets

For our experimentation and training, we have focused on small objects with highly accurate poses and illumination conditions. This opens up new avenues to train our network with in-the-wild scenes with inaccurate pose and illumination information or highly detailed objects which require models to pick up the minutest of details like face data. The recommendation for an in-the-wild dataset is NeRF-ORS [Rudnev et al.(2022)] which consists of various landmarks around Europe with various illumination conditions and for face data are Multi-PIE [Gross et al.(2010)] and USC Face [Li et al.(2020a)] already used in chapter 6. This would require running our current models and performing ablation studies to recalibrate parameters that are dataset-sensitive. While using face dataset, it would be extremely necessary to adhere to the ethical guidelines in handling such data. Certain VR/AR/MR applications require test subjects to endure motion sickness symptoms as part of their training process. Such scenarios might be harmful for subjects with specific medical conditions and could aggravate them as well. It is therefore paramount to assess the significance of such experimentation and choose test subjects accordingly. Also in the wild data which might have individuals not part of the data (tourists at a busy historical monument), it is essential to safeguard the identities of individuals by either blurring or removing them, by in-painting the background, from the dataset.

Relaxing lighting requirements in training data

In Chapters 4 and 5 we make the assumption that training images are captured "one light at a time" (i.e. the only illumination present is a single point light source with known direction). This restricts the applicability of the current methods to datasets captured under lab-constrained conditions. To address the previous suggestion for future work, we would also

require modifying the methods to allow for more complex and/or unknown lighting in the training images. This can be achieved by introducing complex light scenarios to the existing bunny or DiLiGeNT-MV datasets and representing those through methods like spherical harmonic lighting [Green(2003)] or spherical Gaussian lighting [Huo et al.(2020)].

Rendering with extended light sources

The methods in Chapters 4 and 5 allow for rendering novel views with arbitrary point light source direction. However, it would be straightforward to use these trained models to render novel views with more complex lighting. As an example, rendering with an environment map amounts to summing over many point light sources, with each pixel in the environment map providing a direction and lighting colour. Many evaluations of our relightable renderer would allow such renderings to be created. This could be achieved by following NeRV [Srinivasan et al.(2021)] like methods that incorporate surface rendering techniques like NeUS [Wang et al.(2021)].

Front-end VR/AR application

An appropriate way to conclude this piece of work can be a front-end application. This application should be able to load up all the trained models and datasets. Here, we can create a VR/AR simulation where all the trained objects can be placed in different types of environments. A significant challenge in this direction would be adapting the NeRF or NeuS representation for real-time rendering, however, we have recently witnessed exciting upcoming work which supports and have been able to execute the idea with finesse [Chen et al.(2023)]. It must be noted that the application could be used to realistically places objects and human subjects in a false scenario for spreading misinformation or creating controversies (similar to DeepFakes), thus, it is paramount for the researcher to adhere to safety guidelines and deter such acts as much as possible.

List of References

[Asthana et al.(2022)] Meghna Asthana, William Smith, and Patrik Huber. 2022. Neural apparent BRDF fields for multiview photometric stereo. In *Proceedings of the 19th ACM SIGGRAPH European Conference on Visual Media Production*. 1–10.

[Bas and Smith(2019)] Anil Bas and William A. P. Smith. 2019. What Does 2D Geometric Information Really Tell Us About 3D Face Shape? *International Journal of Computer Vision* 127, 10 (2019), 1455–1473.

[Bi et al.(2020)] Sai Bi, Zexiang Xu, Pratul Srinivasan, Ben Mildenhall, Kalyan Sunkavalli, Miloš Hašan, Yannick Hold-Geoffroy, David Kriegman, and Ravi Ramamoorthi. 2020. Neural reflectance fields for appearance acquisition. (2020).

[Bojanowski et al.(2018)] Piotr Bojanowski, Armand Joulin, David Lopez-Pas, and Arthur Szlam. 2018. Optimizing the Latent Space of Generative Networks. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer Dy and Andreas Krause (Eds.). PMLR, 600–609. https://proceedings.mlr.press/v80/bojanowski18a.html

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

[Bouguet(2022)] Jean-Yves Bouguet. 2022. Camera Calibration Toolbox for Matlab. (May 2022). https://doi.org/10.22002/D1.20164

[Chai et al.(2000)] Jin-Xiang Chai, Xin Tong, Shing-Chow Chan, and Heung-Yeung Shum. 2000. Plenoptic sampling. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. 307–318.

[Chen et al.(2022)] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. 2022. TensoRF: Tensorial Radiance Fields. In *European Conference on Computer Vision*. 333–350.

[Chen et al.(2019)] Wenzheng Chen, Huan Ling, Jun Gao, Edward Smith, Jaakko Lehtinen, Alec Jacobson, and Sanja Fidler. 2019. Learning to predict 3d objects with an interpolation-based differentiable renderer. *Advances in Neural Information Processing Systems* 32 (2019).

[Chen et al.(2023)] Zhiqin Chen, Thomas Funkhouser, Peter Hedman, and Andrea Tagliasacchi. 2023. Mobilenerf: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 16569–16578.

[Deng et al.(2022)] Kangle Deng, Andrew Liu, Jun-Yan Zhu, and Deva Ramanan. 2022. Depth-supervised nerf: Fewer views and faster training for free. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 12882–12891.

[Du et al.(2021)] Yilun Du, Yinan Zhang, Hong-Xing Yu, Joshua B Tenenbaum, and Jiajun Wu. 2021. Neural radiance flow for 4d view synthesis and video processing. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE Computer Society, 14304–14314.

[Edwards et al.(2006)] Dave Edwards, Solomon Boulos, Jared Johnson, Peter Shirley, Michael Ashikhmin, Michael Stark, and Chris Wyman. 2006. The halfway vector disk for BRDF modeling. *ACM Transactions on Graphics (TOG)* 25, 1 (2006), 1–18.

[Flynn et al.(2019)] John Flynn, Michael Broxton, Paul Debevec, Matthew DuVall, Graham Fyffe, Ryan Overbeck, Noah Snavely, and Richard Tucker. 2019. Deepview: View synthesis with learned gradient descent. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2367–2376.

[Gadelha et al.(2017)] Matheus Gadelha, Subhransu Maji, and Rui Wang. 2017. 3d shape induction from 2d views of multiple objects. In *2017 International Conference on 3D Vision (3DV)*. IEEE, 402–411.

[Gao et al.(2021)] Chen Gao, Ayush Saraf, Johannes Kopf, and Jia-Bin Huang. 2021. Dynamic view synthesis from dynamic monocular video. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 5712–5721.

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

[Genova et al.(2020)] Kyle Genova, Forrester Cole, Avneesh Sud, Aaron Sarna, and Thomas Funkhouser. 2020. Local deep implicit functions for 3d shape. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 4857–4866.

[Green(2003)] Robin Green. 2003. Spherical harmonic lighting: The gritty details. In *Archives of the game developers conference*, Vol. 56. 4.

[Gross et al.(2010)] Ralph Gross, Iain Matthews, Jeffrey Cohn, Takeo Kanade, and Simon Baker. 2010. Multi-pie. *Image and vision computing* 28, 5 (2010), 807–813.

[Han and Shen(2015)] Tian-Qi Han and Hui-Liang Shen. 2015. Photometric stereo for general BRDFs via reflection sparsity modeling. *IEEE Transactions on Image Processing* 24, 12 (2015), 4888–4903.

[Hansen and Johnson(2011)] Charles D Hansen and Chris R Johnson. 2011. *Visualization handbook*. Elsevier.

[Henzler et al.(2020)] Philipp Henzler, Niloy J Mitra, and Tobias Ritschel. 2020. Learning a neural 3d texture space from 2d exemplars. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 8356–8364.

[Huang et al.(2020)] Ruo Huang, Shelby McIntyre, Meina Song, Zhonghong Ou, et al. 2020. An attention-based latent information extraction network (ALIEN) for high-order feature interactions. *Applied Sciences* 10, 16 (2020), 5468.

[Huo et al.(2020)] YC Huo, SH Jin, T Liu, Wei Hua, Rui Wang, and HJ Bao. 2020. Spherical Gaussian-based Lightcuts for Glossy Interreflections. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, 192–203.

[Ikehata et al.(2014)] Satoshi Ikehata, David Wipf, Yasuyuki Matsushita, and Kiyoharu Aizawa. 2014. Photometric stereo using sparse bayesian regression for general diffuse surfaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36, 9 (2014), 1816–1831.

[Ikeuchi(2021)] Katsushi Ikeuchi. 2021. *Computer vision: A reference guide*. Springer.

[Jiang et al.(2020)] Chiyu Jiang, Avneesh Sud, Ameesh Makadia, Jingwei Huang, Matthias Nießner, Thomas Funkhouser, et al. 2020. Local implicit grid representations for 3d scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 6001–6010.

[Kajiya and Von Herzen(1984)] James T Kajiya and Brian P Von Herzen. 1984. Ray tracing volume densities. *ACM SIGGRAPH computer graphics* 18, 3 (1984), 165–174.

[Kanazawa et al.(2018)] Angjoo Kanazawa, Shubham Tulsiani, Alexei A Efros, and Jitendra Malik. 2018. Learning category-specific mesh reconstruction from image collections. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 371–386.

[Lengyel(2019)] Eric Lengyel. 2019. *Foundations of Game Engine Development: Rendering*. Terathon Software LLC.

[Levoy(1990)] Marc Levoy. 1990. Efficient ray tracing of volume data. *ACM Transactions on Graphics (TOG)* 9, 3 (1990), 245–261.

[Li et al.(2022a)] Chaojian Li, Sixu Li, Yang Zhao, Wenbo Zhu, and Yingyan Lin. 2022a. RT-NeRF: Real-Time On-Device Neural Radiance Fields Towards Immersive AR/VR Rendering. *IEEE/ACM International Conference on Computer-Aided Design (ICCAD 2022)* (2022).

[Li et al.(2022b)] Lingzhi Li, Zhen Shen, Li Shen, Ping Tan, et al. 2022b. Streaming Radiance Fields for 3D Video Synthesis. In *Advances in Neural Information Processing Systems*.

[Li et al.(2020b)] Min Li, Zhenglong Zhou, Zhe Wu, Boxin Shi, Changyu Diao, and Ping Tan. 2020b. Multi-view photometric stereo: A robust solution and benchmark dataset for spatially varying isotropic materials. *IEEE Transactions on Image Processing* 29 (2020), 4159–4173.

[Li et al.(2020a)] Ruilong Li, Karl Bladin, Yajie Zhao, Chinmay Chinara, Owen Ingraham, Pengda Xiang, Xinglei Ren, Pratusha Prasad, Bipin Kishore, Jun Xing, et al. 2020a. Learning formation of physically-based face attributes. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 3410–3419.

[Li et al.(2022c)] Tianye Li, Mira Slavcheva, Michael Zollhoefer, Simon Green, Christoph Lassner, Changil Kim, Tanner Schmidt, Steven Lovegrove, Michael Goesele, Richard Newcombe, et al. 2022c. Neural 3D Video Synthesis From Multi-View Video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 5521–5531.

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

[Li et al.(2021)] Zhengqi Li, Simon Niklaus, Noah Snavely, and Oliver Wang. 2021. Neural scene flow fields for space-time view synthesis of dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 6498–6508.

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

[Lindell et al.(2021)] D. B.* Lindell, J. N. P.* Martel, and G. Wetzstein. 2021. AutoInt: Automatic Integration for Fast Neural Volume Rendering. In *Proc. CVPR*.

[Liu et al.(2020)] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. 2020. Neural sparse voxel fields. *Advances in Neural Information Processing Systems* 33 (2020), 15651–15663.

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

[Lombardi et al.(2019)] Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. 2019. Neural volumes: learning dynamic renderable volumes from images. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 1–14.

[Lorensen and Cline(1987)] William E Lorensen and Harvey E Cline. 1987. Marching cubes: A high resolution 3D surface construction algorithm. *ACM siggraph computer graphics* 21, 4 (1987), 163–169.

[Ma et al.(2007)] Wan-Chun Ma, Tim Hawkins, Pieter Peers, Charles-Felix Chabert, Malte Weiss, and Paul Debevec. 2007. Rapid acquisition of specular and diffuse normal maps from polarized spherical gradient illumination. In *Proceedings of the 18th Eurographics conference on Rendering Techniques*. 183–194.

[Maclaurin et al.(2015)] Dougal Maclaurin, David Duvenaud, and Ryan P Adams. 2015. Autograd: Effortless gradients in numpy. In *ICML 2015 AutoML workshop*, Vol. 238.

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

[Max(1995)] Nelson Max. 1995. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics* 1, 2 (1995), 99–108.

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

[Mildenhall et al.(2022)] Ben Mildenhall, Peter Hedman, Ricardo Martin-Brualla, Pratul P Srinivasan, and Jonathan T Barron. 2022. NeRF in the Dark: High Dynamic Range

View Synthesis from Noisy Raw Images. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 16169–16178.

[Mildenhall et al.(2019)] Ben Mildenhall, Pratul P Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. 2019. Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 1–14.

[Mildenhall et al.(2020)] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2020. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *ECCV*.

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

[Neff et al.(2021)] Thomas Neff, Pascal Stadlbauer, Mathias Parger, Andreas Kurz, Joerg H. Mueller, Chakravarty R. Alla Chaitanya, Anton S. Kaplanyan, and Markus Steinberger. 2021. DONeRF: Towards Real-Time Rendering of Compact Neural Radiance Fields using Depth Oracle Networks. *Computer Graphics Forum* 40, 4 (2021). `https://doi.org/10.1111/cgf.14340`

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

[Noguchi et al.(2021)] Atsuhiro Noguchi, Xiao Sun, Stephen Lin, and Tatsuya Harada. 2021. Neural articulated radiance field. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 5762–5772.

[Oechsle et al.(2019)] Michael Oechsle, Lars Mescheder, Michael Niemeyer, Thilo Strauss, and Andreas Geiger. 2019. Texture fields: Learning texture representations in function space. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 4531–4540.

[Pan et al.(2019)] Junyi Pan, Xiaoguang Han, Weikai Chen, Jiapeng Tang, and Kui Jia. 2019. Deep mesh reconstruction from single rgb images via topology modification networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 9964–9973.

[Park et al.(2016)] Jaesik Park, Sudipta N Sinha, Yasuyuki Matsushita, Yu-Wing Tai, and In So Kweon. 2016. Robust multiview photometric stereo using planar mesh parameterization. *IEEE transactions on pattern analysis and machine intelligence* 39, 8 (2016), 1591–1604.

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

[Park et al.(2021a)] Keunhong Park, Utkarsh Sinha, Jonathan T Barron, Sofien Bouaziz, Dan B Goldman, Steven M Seitz, and Ricardo Martin-Brualla. 2021a. Nerfies: Deformable neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 5865–5874.

[Park et al.(2021b)] Keunhong Park, Utkarsh Sinha, Peter Hedman, Jonathan T Barron, Sofien Bouaziz, Dan B Goldman, Ricardo Martin-Brualla, and Steven M Seitz. 2021b. HyperNeRF: a higher-dimensional representation for topologically varying neural radiance fields. *ACM Transactions on Graphics (TOG)* 40, 6 (2021), 1–12.

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

[Rahaman et al.(2019)] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. 2019. On the spectral bias of neural networks. In *International Conference on Machine Learning*. PMLR, 5301–5310.

[Rebain et al.(2021)] Daniel Rebain, Wei Jiang, Soroosh Yazdani, Ke Li, Kwang Moo Yi, and Andrea Tagliasacchi. 2021. Derf: Decomposed radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 14153–14161.

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

[Rudnev et al.(2022)] Viktor Rudnev, Mohamed Elgharib, William Smith, Lingjie Liu, Vladislav Golyanik, and Christian Theobalt. 2022. NeRF for Outdoor Scene Relighting. In *European Conference on Computer Vision (ECCV)*.

[Rusinkiewicz(1998)] Szymon M Rusinkiewicz. 1998. A new change of variables for efficient BRDF representation. In *Eurographics Workshop on Rendering Techniques*. Springer, 11–22.

[Sanjeev(2014)] J Koppal Sanjeev. 2014. Lambertian reflectance. *Computer Vision: A Reference Guide; Ikeuchi, K., Ed.; Springer: Boston, MA, USA* (2014), 441–443.

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

[Shafiei et al.(2021)] Mohammad Shafiei, Sai Bi, Zhengqin Li, Aidas Liaudanskas, Rodrigo Ortiz-Cayon, and Ravi Ramamoorthi. 2021. Learning Neural Transmittance for Efficient Rendering of Reflectance Fields.

[Shen et al.(2016)] Hui-Liang Shen, Tian-Qi Han, and Chunguang Li. 2016. Efficient photometric stereo using kernel regression. *IEEE Transactions on Image Processing* 26, 1 (2016), 439–451.

[Shi et al.(2013)] Boxin Shi, Ping Tan, Yasuyuki Matsushita, and Katsushi Ikeuchi. 2013. Bi-polynomial modeling of low-frequency reflectances. *IEEE transactions on pattern analysis and machine intelligence* 36, 6 (2013), 1078–1091.

[Sinha(2014)] Sudipta N Sinha. 2014. Multiview Stereo.

[Sitzmann et al.(2019)] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. 2019. Scene representation networks: Continuous 3d-structure-aware neural scene representations. *Advances in Neural Information Processing Systems* 32 (2019).

[Smith and Fang(2016)] W. A. P. Smith and F. Fang. 2016. Height from Photometric Ratio with Model-based Light Source Selection. *Computer Vision and Image Understanding* 145 (2016), 128–138.

[Srinivasan et al.(2021)] Pratul P Srinivasan, Boyang Deng, Xiuming Zhang, Matthew Tancik, Ben Mildenhall, and Jonathan T Barron. 2021. NeRV: Neural reflectance and visibility fields for relighting and view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 7495–7504.

[Stutz and Geiger(2018)] David Stutz and Andreas Geiger. 2018. Learning 3d shape completion from laser scan data with weak supervision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1955–1964.

[Sun et al.(2022)] Cheng Sun, Min Sun, and Hwann-Tzong Chen. 2022. Direct Voxel Grid Optimization: Super-fast Convergence for Radiance Fields Reconstruction. *CVPR* (2022).

[Tretschk et al.(2021)] Edgar Tretschk, Ayush Tewari, Vladislav Golyanik, Michael Zollhöfer, Christoph Lassner, and Christian Theobalt. 2021. Non-rigid neural radiance fields: Reconstruction and novel view synthesis of a dynamic scene from monocular video. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 12959–12970.

[Tseng et al.(2022)] Wei-Cheng Tseng, Hung-Ju Liao, Yen-Chen Lin, and Min Sun. 2022. CLA-NeRF: Category-Level Articulated Neural Radiance Field. In *ICRA*.

[Turk and Levoy(1994)] Greg Turk and Marc Levoy. 1994. Zippered polygon meshes from range images. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*. 311–318.

[Wang et al.(2022)] Huan Wang, Jian Ren, Zeng Huang, Kyle Olszewski, Menglei Chai, Yun Fu, and Sergey Tulyakov. 2022. R2L: Distilling Neural Radiance Field to Neural Light Field for Efficient Novel View Synthesis. In *ECCV*.

[Wang et al.(2018)] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. 2018. Pixel2mesh: Generating 3d mesh models from single rgb images. In *Proceedings of the European conference on computer vision (ECCV)*. 52–67.

[Wang et al.(2021)] Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. 2021. NeuS: Learning Neural Implicit Surfaces by Volume Rendering for Multi-view Reconstruction. In *Advances in Neural Information Processing Systems*.

[Weber et al.(2021)] Patrick Weber, Jeremy Geiger, and Werner Wagner. 2021. Constrained neural network training and its application to hyperelastic material modeling. *Computational Mechanics* 68 (2021), 1179–1204.

[Westlund et al.(2002)] Harold B Westlund, Gary W Meyer, and Fern Y Hunt. 2002. The role of rendering in the competence project in measurement science for optical reflection and scattering. *Journal of research of the National Institute of Standards and Technology* 107, 3 (2002), 247.

[Woodham(1980)] Robert J Woodham. 1980. Photometric method for determining surface orientation from multiple images. *Optical engineering* 19, 1 (1980), 139–144.

[Xian et al.(2021)] Wenqi Xian, Jia-Bin Huang, Johannes Kopf, and Changil Kim. 2021. Space-time neural irradiance fields for free-viewpoint video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 9421–9431.

[Xie et al.(2019)] Haozhe Xie, Hongxun Yao, Xiaoshuai Sun, Shangchen Zhou, and Shengping Zhang. 2019. Pix2vox: Context-aware 3d reconstruction from single and multi-view images. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2690–2698.

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

[Yu and Smith(2019)] Ye Yu and William AP Smith. 2019. InverseRenderNet: Learning single image inverse rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Zhang et al.(2020)] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. 2020. Nerf++: Analyzing and improving neural radiance fields. (2020).

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

[Zheng et al.(2019)] Qian Zheng, Ajay Kumar, Boxin Shi, and Gang Pan. 2019. Numerical reflectance compensation for non-lambertian photometric stereo. *IEEE Transactions on Image Processing* 28, 7 (2019), 3177–3191.

[Zhou et al.(2018)] Tinghui Zhou, Richard Tucker, John Flynn, Graham Fyffe, and Noah Snavely. 2018. Stereo magnification: learning view synthesis using multiplane images. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–12.

Chapter 8

Appendicies

8.1 Appendix I: Matlab Renderer

8.1.1 Pipeline Overview

1. `MR_rasterise_mesh` function performs z-buffering on the projected mesh and returns a face buffer i.e. the triangle index per pixel and a weight buffer i.e. barycentric weight for three triangle vertices per pixel. The buffers are then used to interpolate a screen space value. The inputs to the function are shown in figure 2.3.

2. `MR_render_mesh` is a wrapper for all the underlying functions and executes the entire rendering pipeline.

3. `renderparams.normalMode = 'perVertexNormals'`

4. `renderparams.normalMode = 'normalMap'`

8.1.2 Basic Rendering of Stanford Bunny.

The code below shows rendering with default parameters:

```
obj = MR_obj_read('data/StanfordBunny.obj');
[cameraparams,renderparams] = MR_default_params(obj.V,400);
render = MR_render_mesh(obj.F,obj.V,cameraparams,renderparams);
figure; imshow(render)
```

The UV texture mapping is introduced through `MR_render_mesh`

```
renderparams.VT = obj.VT;

renderparams.FT = obj.FT;

renderparams.textureMode = 'useTextureMap';

renderparams.texmap = im2double(imread('data/StanfordBunny.jpg'));
```

8.2 Appendix II: Relightable NeRF Implementation

In order to obtain a vanilla NeRF output for a set of images without any poses it must follow a few basic steps. These include:

- The set of static scene images are taken as input to COLMAP to get 6-DoF camera poses and near/far depth bounds for the scene. This reconstructed model is saved as `project.ini` with data files `cameras.bin`, `images.bin` and `points3D.bin`
- The bin files are taken as input for the Local Light Field Fusion (LLFF) which outputs a NumPy array named `poses\_bounds`
- The `poses\_bounds` array and the static scene images act as input for NeRF.

Understanding camera placements

The coordinate convention for NeRF $[-y, x, z]$ is different from the conventional $[x, y, z]$ that the DiLiGenT dataset follows. Therefore, in order to avoid any discrepancy which might fail NeRF reconstruction, we assessed the camera placement in 3D and 2D coordinate systems for sample NeRF (fern) and DiLiGenT datasets. The camera coordinates are calculated from rotational matrix $\mathbf{R}$ and translational matrix $\mathbf{t}$ as

$$\mathbf{c} = -\mathbf{R}^T\mathbf{t} \tag{8.1}$$

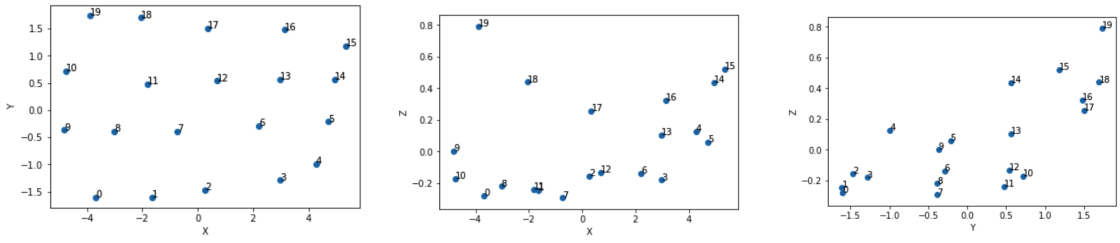


Figure 8.1: Camera Placement for NeRF sample dataset - Fern

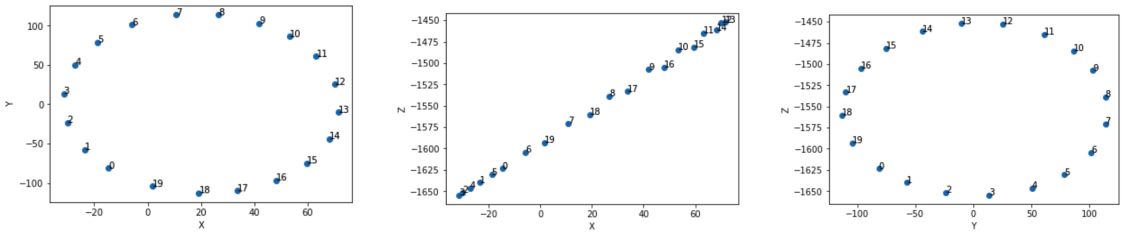


Figure 8.2: Camera Placement for DiLiGenT dataset

From the figures 8.1 and 8.2, we show that the camera placement for the fern dataset is a grid and for DiLiGenT is a ring and we confirmed the convention conversion from $[x, y, z]$ to $[-y, x, z]$.

Custom LLFF output for DiLiGenT image sets with known poses:

With the coordinate convention confirmed, we attempt to create `poses_bounds` array which consists of the camera intrinsics, poses and near/far z values in the specified order explained in 3.4. This circumvents the COLMAP and LLFF (8.2.1) runs of the NeRF pipeline which can be cumbersome for certain datasets. We use the camera calibration information provided in `Calib_Result` Matlab file from which we take values from **KK** (intrinsics), **R** and **t**. For each image in the set, we create a world-to-camera transform `T_w2c` represented as $\begin{bmatrix}\mathbf{R} & \mathbf{t}; & 0 & 0 & 0 & 1\end{bmatrix}$. Taking an inverse converts it to the camera-to-world transform `T_c2w`. The intrinsics - width, height and focal length are appended to `T_c2w` transform vertically and the entire matrix is flattened. The near/far z values are calculated from the mesh data which are appended as the last two values.

Preprocessing

The objects in DiLiGenT-MV dataset seem to be faintly lit (due to 96 single lighting in individual image) and therefore appear quite dark to capture any features. Furthermore, as the objects are very smooth and lack texture, and thus, the sparse reconstruction in COLMAP tends to fail. Therefore, we decided to average across all 96 lighting directions for each viewpoint and apply gamma correction of 1/2.2. Furthermore, we converted the images from 16-bit PNG to 8-bit PNG format to match NeRF's input format scheme.

Editing Images

The NeRF output for DiLiGenT data exhibits noticeable artefacts in the images and more prominently in videos. We concur that this is a result of the background in the images. Therefore, we removed the background entirely by adding a black mask or a white mask.

8.2.1 Training Vanilla NeRF on New Datasets

The first task of our experimentation was to replicate the results for the default data referred to as 'fern' in LLFF and NeRF. This was performed so as to obtain a better understanding of the entire pipeline and further train our own dataset.

Extraction of camera poses from COLMAP and LLFF

The raw images can be downloaded from the resources provided by the NeRF's author. The dataset comes in a set of images taken from different camera poses which present a planar view of the fern scene. These are 20 high-resolution images - around 3 MB each. In order to train on NeRF, we require an intrinsic and extrinsic matrix known as `poses_bounds` and require it to be in a fixed format which requires running LLFF and COLMAP for the final product.

Figure 8.3: Generated point cloud and camera poses from COLMAP

The first step for running LLFF is to create a virtual environment with all the prerequisites which could successfully run the snippet `image2poses.py`. The snippet runs COLMAP in the background to generate the poses but we have also tried to run COLMAP with the same setting in a GUI to understand the process better. Figure 8.3 shows the point cloud generated from the images.

COLMAP

The sparse reconstruction for the static scene is obtained following multiple steps. First, `feature_extractor` command recomputes feature from the image set to create a sparse map. The `feature_extractor` step finds sparse feature points in the image and describes their appearance using numerical descriptors. If the images have been captured from the same physical camera with an identical zoom factor, using the `SHARED INTRINSICS` option is recommended in COLMAP documentation.

```
$ DATASET_PATH=/path/to/dataset


$ colmap feature_extractor \
   --database_path $DATASET_PATH/database.db \
   --image_path $DATASET_PATH/images   \
   --ImageReader.single_camera 1
```

The next step in the process is feature matching and geometric verification which identifies the corresponding feature points in all images. As the number of images for our set is few, this is most accurately achieved by employing 'Exhaustive Matching' (matching every image against others in the set).

```
$ colmap exhaustive_matcher \
   --database_path $DATASET_PATH/database.db
```

The final step is a sparse reconstruction which is built using incremental reconstruction - extending the scene by registering new images and triangulating new points. If the image set is not registered into the same model, COLMAP may reconstruct multiple models to accommodate all. These models can be collated together at a later stage.

```
$ mkdir $DATASET_PATH/sparse
```

```
$ colmap mapper \
    --database_path $DATASET_PATH/database.db \
    --image_path $DATASET_PATH/images \
    --output_path $DATASET_PATH/sparse \
    --Mapper.num_threads 16 \
    --Mapper.init_min_tri_angle 4
```

The bin files - `cameras`, `images` and `point3D` obtained from the sparse reconstruction are used in the next step.

LLFF

LLFF creates a value matrix for each image in the dataset which is compatible with vanilla NeRF. Each value matrix consists of 17 values - the rotational matrix $\mathbf{R} \in \mathbb{R}^{3\times3}$ and translational matrix $\mathbf{t} \in \mathbb{R}^{3\times1}$, the intrinsics focal length, image height and width and near/far bounds. LLFF uses the `cameras.bin` file to calculate three intrinsics - focal length, image height and width.

The `images.bin` file's data is used to calculate the rotational matrix $\mathbf{R} \in \mathbb{R}^{3\times3}$ and translational matrix $\mathbf{t} \in \mathbb{R}^{3\times1}$. Lastly, `points3D.bin` file consists of all points from the sparse model - a visibility test is added to remove all points not in the frame and the near/far bounds are calculated using the remaining. These values are calculated for each image and saved as an array to be input into NeRF.

Once LLFF receives the required data in the format mentioned, it attempts to read these files and retrieve information like `camera_model`, `camera_id`, `height`, `width`, `focal_length` and remaining parameters specific to each image. Next, the remaining parameters are connected to their corresponding images and point clouds for further processing.

The data corresponding to each image is retrieved in the form of two matrices - rotation matrix $\mathbf{R}$ and translational matrix $\mathbf{t}$ which are then combined to create `world2camera` transformation matrix for each image in the dataset. The convention followed by LLFF is

the right-handed coordinate system of rotation i.e. from the point of view of the camera, the three axes are `[-y,x,z]` or `[down, right, backwards]`. As this does not match with the COLMAP's system `[x,-y,-z]` or `right, down, forwards]`, LLFF has to account for this conversion too.

LLFF retrieves point cloud data from `points3D.bin` which consists of `point3D_id`, `xyz` coordinates, `rgb` colour of the point and corresponding `image_id`. The data is stored neatly in a tuple list for 3D object depth calculations.

In order to obtain the near and far depth points, LLFF creates a visibility array which consists of all the points visible from a particular camera pose. These are, at first, changed from the world coordinate system to the camera coordinate system using the corresponding transformation matrix and then the closest and farthest points are picked from the 0.1 percentile and 99.9 percentile respectively. The final step involves repackaging all these data points in a list of lists.

8.2.2 Preprocessing for NeRF

The pre-requisites for training on NeRF is to have a set of 20-30 images (in planar or stereo form) and a `poses_bounds` matrix with all the 17 corresponding image values stacked together. Once these are met, the NeRF `run_nerf.py` starts initial preprocessing by downsampling each image (optional), loading the `poses_bounds` matrix and converting it to NeRF's coordinate system `[-y,-x,-z]` and rescaling the boundaries according to a factor (optional).

Recentering

In essence, recentering is performed to move the focus centre of the image to match the camera centre at a particular camera pose. The process includes creating an average pose matrix for the entire dataset which has the following - (i) the centre as the mean of the translation matrix, (ii) a forward vector which is the normalised mean of the z-axis in the rotation matrix and (iii) an up vector representing the mean of -x-axis in the rotation matrix. The camera-to-world matrix is obtained from the average pose matrix by reshuffling it to the form `([y, -x, z, centre],1)`, a 4×4 matrix. The camera intrinsics are then concatenated at the end of the camera-to-world matrix to create the final transformation matrix. This is finally multiplied by each pose in the dataset to obtain the new recentred poses.

Spherify.

The spherify setting is crucial for our multiview photometric stereo datasets, Stanford bunny and DiLiGeNT-MV. The method takes the translational matrix as the ray origin and the z-axis from the rotation matrix as the ray direction as inputs to the minimum line distance function. Using this function, NeRF finds a point which is the intersection point of all rays as well as the centre of the new NeRF coordinate system. Next, it creates another transformation matrix (similar to recentering) in the form `([y, -x, z, centre],1)` and multiplies it to every pose in the dataset.

An additional resizing factor is incorporated which would be useful for the future when the 3D mesh is constructed. Using the centroid calculated from the poses, NeRF generates a new set of poses which form a complete circle around the object and have 120 points - these will be used to train NeRF.

8.2.3 Pipeline Enhancement for Relightable NeRF

We will now be enumerating the steps involved to upgrade our pipeline to complement the Relightable NeRF model.

- Generate multilight and multipose data;
- Convert the data into lights and poses matrix which could be understood by NeRF;
- Upgrade configuration files to include light direction data settings
- Incorporate illumination; direction data in NeRF preprocessing in the same way as the view direction data;
- Extending NeRF model to include illumination direction as one of the inputs;
- Create new helper functions to render images and videos for multiple illumination settings;

Updating Configurations

In order to incorporate easy-to-manipulate properties of Relightable NeRF, we add config parameters as:

`--lightdirsdir` input data directory for light direction

`--spherifydir` input data director for the transformation matrix from world to NeRF

`--use_lightdirs` boolean data point to turn light direction on/off

`--lightdir` single light direction input for multi-pose rendering

`--multilightdirs` list of light directions input for multi-light rendering for a single pose

`--binary_density_factor` a value from 0.0 to 1.0 defining the influence of binary density loss on total loss at every iteration

`--sil_density_factor` a value from 0.0 to 1.0 defining the influence of silhouette density loss on total loss at every iteration

`--piecewise_density_factor` a value from 0.0 to 1.0 defining the influence of piecewise density loss on total loss at every iteration

Data Loading.

The illumination direction data is of the format $\mathbf{N} \times d \times 3$ where $\mathbf{N}$ is the number of images in the dataset, d is the number of light directions in each image (assuming all images have the same number of illumination points) and 3 is the x, y and z coordinates in the world coordinate system. The $d \times 3$ data for each image is saved in a `.txt` file in a separate folder. Relightable NeRF loads these files in a NumPy array. As the coordinates are in the world coordinate system, we convert them to NeRF coordinate system using the transpose of camera-to-world transformation matrices.

Helper Functions.

Given the increased complexity of the model with the introduction of illumination direction, multiple helper functions have been employed to assist in the transition. Some of the major functions are listed below:

- `render_relight()`: The function returns, for a given batch of rays, predicted RGB values for all rays, a disparity map - inverse of depth map, accumulation opacity density along each ray and raw output values from the model at every iteration.
- `run_network_relight()`: The function prepares the raw input - RBG values, view direction and illumination direction by creating the sinusoidal maps of shape $batch_size \times 3^2$ for each and concatenating them for a single large input for the model.
- `network_query_fn_relight()`: The function initialises `run_network_relight()`.

- `create_nerf_relight()`: The function creates the coarse and fine models by instantiating Relightable NeRF using runner `run_network_relight()`.
- `render_relight_path()`: The function renders individual poses for a 360°view for a single distant light source and stitches them together in a `.mp4` video.
- `render_relight_path_multilight_onepose()`: The function renders individual illumination for a 360°view for a single pose and stiches them together in a `.mp4` video.

Mesh Extraction.

The 3D reconstruction from rendering is obtained as a mesh with faces and vertices. Once a model is trained, the mesh extractor requires the `config` file and model weights related to the training session. The mesh extractor follows the steps below:

- Re-creating and re-initialising the Relightable NeRF model using the arguments from `config` file
- A two-dimensional grid space in the range $[-1.0, 1.0]$ from which a batch of rays required to run the network is created
- The `network_query_fn` performs a single run to obtain the network output values for each ray in the batch using the saved weights. The size of this chuck is fixed to 1024×64
- The raw output data is then reshaped and passed through a filter which replaces any negative values with zeros to obtain sigma values
- The marching cubes library `mcubes` is used to calculate the vertices and triangles, taking sigma and an arbitrary threshold value (in this case 10) as inputs
- The `export_obj` function creates the final mesh using the vertices and triangles computed earlier.
- Finally, the transformation matrix from NeRF coordinate system to the world coordinate system is applied to obtain the final mesh. This can be viewed in any system for processing and editing 3D triangular meshes like MeshLab.

8.3 Appendix III: Neural Apparent BRDF Fields Implementation

Loss Function Extensions

We have employed a variety of losses to improve the learning process and, in turn, the final renderings. Each loss brings a separate set of functionality to `total_loss` and a varied combination of each works best for every dataset we have trained. The extent of influence by any form of loss can be modified using factor arguments These include:

- **Alpha Loss**: The alpha loss is based on the volume density α which represents a likelihood value density of each sample point along every ray in the batch. A higher value at a sample point in the ray exhibits a higher probability of light being absorbed at that point. The loss helps us identify whether a ray has been terminated in the volume and thus, does not contribute to the final colour.

- **Binary Loss**: The Binary loss is based on the boundary between the object of interest and the background - this is determined by the individual mask corresponding to each input image. The 'binary' name comes from the fact that any density identified as the background (flag value 1 or True) is ignored during the calculation of this loss. The α density values, discussed in *alpha loss* are converted to a logarithmic value and follow the format below:

$$\textbf{binary\_loss} = \log(\alpha + \epsilon) + \log(1 - \alpha + \epsilon)$$

  where ϵ is a small value added to α for avoiding a log for 0. The loss helps us identify whether a given ray passes through the object of interest and contributes to the final colour of the pixel.

- **Piecewise Loss**: The piecewise loss, a variation of alpha loss implementation, determines the extent of influence of total `alpha_loss` value depending on the value itself. A simple pseudo implementation is shown below:

```
if alpha_loss < 0.25:
    alpha_loss = 4.0*alpha_loss
elif alpha_loss < 0.75:
```

```
        alpha_loss = 1.0
    else:
        alpha_loss = 4.0*(1.0-w_alpha_loss)
```

The factored loss calculated from the pseudo code above has been added to the `total_loss`. The loss does not prove to be instrumental in improving the total loss or the renderings.

- **Silhouette Density Loss**: The silhouette density loss, a variation of alpha loss implementation, determines the influence of `alpha_loss` depending on the origin of a particular ray. A snippet of pseudo implementation is shown below:

```
for j in range(0,target_s.shape[0].value):
    if target_s[j,0].numpy()==0.0 and target_s[j,1].numpy()==0.0
    and target_s[j,2].numpy()==0.0:
        if alpha[j,...].numpy().all()!=0.0:
            alpha[j,...] = 0.0*alpha[j,...]
```

The code checks whether a pixel consists of 0.0 in all three RGB channels, if so, it is classified as background. A ray originating from a pixel specified as 'background' would not be taken into account while calculating the total `alpha_loss`. The loss is instrumental in eliminating the contribution to the total loss from background pixels, however, it requires restructuring (remove nested for loops) so as to improve its efficiency.

- **Mask Appearance Loss**: The mask appearance loss is the restructured version of silhouette density loss that determines the influence of `alpha_loss` depending on checking whether the being tested pixel is classified as 'object' or 'background'. A snippet of pseudo implementation is shown below:

```
mask = np.where((target_s[:,0].numpy()==1.0)
& (target_s[:,1].numpy()==1.0)
& (target_s[:,2].numpy()==1.0),0.0,1.0) # inverted mask
ma_loss = tf.reduce_mean(tf.square((rgb-target_s)*mask))
```

This method is an improved version of silhouette density loss as it reduces the computational time and resources required to process nested for-loops by the clever use of

NumPy arrays.

8.3.1 Helper Functions

Every helper function requires data from a completed experiment's folders - `model_xyz.npy`, `model_fine_xyz.npy` and `optimizer_xyz.npy` as well as the specific architecture model used for training.

The data loading and model creation tasks are performed as mentioned in the vanilla NeRF implementation in section 8.2. The next step is to specify the rendering's dimensions - set at 512×612 and calls the `render_relight()` function for each of 20 poses and 32 light directions. We employ separate helper function for each type of rendering to maintain clarity in our pipeline.

- `get_albedos.py`: The script renders 360-deg `albedos.png` files for any trained model - both `multipose+single_lightdir` and `singlepose+multi_lightdir`.
- `get_normals.py`: The script renders 360-deg `normals.png` files for any trained model - both `multipose+single_lightdir` and `singlepose+multi_lightdir`.
- `get_rgbs.py`: The script renders 360-deg `rgbs.png` files for any trained model - both `multipose+single_lightdir` and `singlepose+multi_lightdir`.
- `get_shadows.py`: The script renders 360-deg `shadows.png` files for any trained model - both `multipose+single_lightdir` and `singlepose+multi_lightdir`.
- `top_view_renderer.py`: This script renders top-head view for any trained model.
- `extract_mesh.py`: This script renders a mesh in `.obj` or `.stl` format using marching cubes techniques as explained in section 8.2.3.
- `angular_error.py`: This script outputs the total angular error calculated as the mean squared error (MSE) between true normals and rendered normals for all poses for any trained model.

8.4 Appendix IV: NeUS Implementation

8.4.1 Runner Class and Objects

- A `Runner` class has been used to define the properties and functions. For every new training session, an instance of `Runner` is created.
- Every configuration setting for the training session is assigned to its corresponding properties in the `Runner` class.
- **NeRF Network.** The class `NeRF` defines this network. It consists of 8 fully connected layers with 256 nodes and includes the option of embedded layers and using view direction for the training. The model follows the same architecture as vanilla NeRF explained in section 4.1. The network is used for modelling the background.
- **SDF Network.** The class `SDFNetwork` define the network with 8 layers with 3 input nodes, 256 nodes in each of the hidden layers and 257 output nodes. The nodes are initialised from values obtained from a Normal distribution with $\mu = -\frac{\sqrt{\pi}}{\sqrt{layer\_dim}}$ and $\sigma = 0.0001$. The network also provides the functionality of embedded and skip layers. The activation applied for each layer is Softplus which is a smooth approximation to the ReLU function and can be used to constrain the output of a machine to always be positive. This can be defined element-wise as

$$\text{Softplus}(x) = \frac{1}{\beta} * \log(1 + \exp(\beta * x))$$

  where β is a formulation value set at 100 as default. The model is used to define the surface of the object of focus.
- **Single Variance Network.** The network consists of a single layer with a variance parameter which is set to an initial value of 0.3.
- **Rendering Network.** The network consists of 6 linear layers with the input layer containing 9 input vectors with 256 features, the hidden layers containing 256 nodes and the output layer with 3 output vectors. All the hidden layers utilise the ReLU activation function and the output layer the Sigmoid activation function. The renderer is set to `idr` mode which allows the input channel to include `points`, `view_directions`, `normals` and `feature_vectors`.

- **NeuS Renderer.** The class initialises and prepares for the training session with all the networks mentioned before.

8.4.2 Preprocess Camera Parameters using IDR

The Implicit Differentiable Renderer (IDR) provides a preprocessing module which converts any image data with accurate poses and mask details to a matrix system compatible to train with NeuS.

Setup

The code is compatible with `Python 3.7` and `PyTorch 1.2`. In addition, it requires `NumPy`, `pyhocon, Plotly, scikit-image, trimesh, imageio, OpenCV`, `Torchvision`. An environment, with all the preliminaries installed, can be easily created by using an `environment.yml` file as follows:

```
conda env create -f environment.yml
conda activate idr
```

In order to run IDR on new data, it requires image and mask directories, as well as `cameras.npz` file containing the appropriate camera projection matrices. For running the script to generate a `cameras.npz` file that contains the suitable normalization matrix following commands are executed:

```
cd ./code
python preprocess_cameras.py --source_dir [DIR PATH]
```

Normalisation

- The number of normalisation points is set at 100 for noise-free images
- For every mask image in the dataset, mask points are created that define whether the pixel is the object or the background. These points are flagged 1 (for the object) or 0 (for the background) and stored in an array.
- For every camera, the `world_mat` is extracted and stored for further processing.
- The normalisation function requires a set of 2D object masks and camera projection matrices (`P_i=K_i[R_i |t_i]` where `[R_i |t_i]` is world to camera transformation).

- The fundamental matrices are computed for a set of two camera pairs, named 1 and 2. The matrix transforms the points from an image of camera 2 to a line in the image of camera 1. This is performed by applying Singular Value Decomposition which generalizes the eigendecomposition of a square normal matrix with an orthonormal eigenbasis to any $m \times n$ matrix. The value acts as camera 2's center which is then multiplied by camera 1's `world_mat` to obtain epipole. The Fundamental matrix F is defined as:

```
epipole=P_1@P_2_center
epipole_cross=np.zeros((3,3))
 F = epipole_cross@P_1 @ np.linalg.pinv(P_2)
```

  It is returned back to the normalisation function as a set of paired camera fundamental matrices.

- A subset of 2D points from camera 0 (default camera position in reference to which other camera positions are calculated) is chosen at random and checked for its maximum and minimum depth in the other camera setups. The 2D point is discarded if there is no intersection of relevant depth. For simplicity following constraints have been enforced for maximum and minimum depth:

```
abs(min_d) < 0.00001
max_d_all < min_d_all + 1e-2
```

- For the selected 2D points, another round of normalisation is applied which requires the centroid and scale values with respect to the refined visual hull. The function creates a 3D mesh grid and places the selected points in each camera position in the grid and calculates the centroid with respect to this 3D point cloud. These values encompass the `scale_mat` matrix.

Deliverables

A cameras file contains for each image a projection matrix (named "world_mat_i"), and a normalization matrix (named "scale_mat_i").

Camera projection matrix. A 3x4 camera projection matrix, $\mathbf{P} = \mathbf{K}[\mathbf{R}\ \mathbf{t}]$ projects points from 3D coordinates to image pixels by the formula: $\mathbf{d}$[x; y; 1]=$\mathbf{P}$[X; Y; Z;1] where $\mathbf{K}$ is a

3x3 calibration matrix, [**R t**] is 3x4 a world to camera Euclidean transformation, [X; Y; Z] is the 3D point, [x;y] is the 2D pixel coordinates of the projected point and d is the depth of the point. The "world_mat" matrix is a concatenation of the camera projection matrix with a row vector of [0,0,0,1] (which makes it a 4x4 matrix).

Normalisation matrix. The normalization matrix is used to normalize the cameras such that the visual hull of the observed object is approximately inside the unit sphere.

8.5 Training Vanilla NeuS on Synthetic Dataset

We begin our practical experimentation by training a vanilla NeuS model on our own synthetic dataset in order to verify that our coordinate systems and camera parameters are correct for the NeuS implementation conventions.

8.5.1 Setup

The code is compatible with `Python 3.8+` and `PyTorch 1.8`. In addition, it requires `NumPy`, `pyhocon, trimesh, icecream, tqdm, SciPy, OpenCV` and `PyMCubes`. An environment, with all the preliminaries installed, can be easily performed by using an `requirements.txt` file as follows:

```
git clone https://github.com/Totoro97/NeuS.git
cd NeuS
pip install -r requirements.txt
```

Given all the data preparation is complete, the training is performed using:

```
python exp_runner.py --mode train --conf ./confs/wmask.conf --case <case_name>
```

8.5.2 Training and Loss Optimisation

We now describe some more specific details about how the vanilla model and the accompanying losses are implemented in practice.

Render

- The input to the function is the object of `NeusRenderer` class, rays origin array, rays direction array, and near and far limit values.

- The batch size is determined by the length of the rays origin array.
- A region of interest is created in the form of a unit sphere with the maximum and minimum distance determined by the far and near values. The space is divided into equal sections depending on the number of samples specified during the training.

Up-sample

- The step is executed if the training session explicitly defines the `n_importance` as a positive value. It defines new sampling depending on the density of information along a sample ray.
- The function takes rays origin matrix, rays direction matrix, SDF network object as input and returns new sampling points as output.
- The `up_sample` function updates the batch size and the number of samples. This is implemented using the Hierarchical Sampling as mentioned in section 6.1.4.

Background Model

- The background is rendered using a version of NeRF, NeRF++. The function takes rays origin matrix, rays direction matrix, NeRF network object as input and returns the RGB colour values, alpha density matrix and updated weights for the network.

Render Core

- The core renderer function takes rays origin matrix, rays direction matrix, SDF network object, Deviation network object, Colour Network object and background RGB, alpha and sample colour matrices as input.
- The gradient and sampled colour values are recalculated using the SDF network and Colour Network objects.
- For better convergence, the `cos_anneal_ratio` is set to grow from 0 to 1 in the beginning training iterations. The updated estimate SDFs are calculated using this ratio.
- Enabling the setting 'render with background' allows updating the previously calculated alpha and sampled colour matrices by merging with the newly computed values above.

- The function finally outputs the updated colour, SDF, gradient, weight, and gradient error values at the end of each iteration.

Losses

In addition to the Eikonal loss, further losses like the colour error (for `n_samples` model), colour fine error (for `n_importance` model), peak signal-to-noise ratio `psnr` and mask loss. The losses are defined as

- **Colour Error**: The error is defined as the difference between estimated colour values from the network and ground truth RGB values provided by the dataset. The error is multiplied by the mask matrix to discount the background from the final error value.

- **Colour Fine Loss**: The loss utilizes L1 regularisation loss for Colour Error defined above. The L1 loss measures the absolute mean error between the estimate and true values. The unreduced loss can be described as

$$l(x, y) = Ll_1, ..., {l_N}^T,\ l_n = |x_n - y_n|,$$

  where x and y are tensors of arbitrary shapes with a total of n elements each, N is the batch size and reduction is set to `'sum'`.

- **Mask Loss**: The loss utilises Cross Entropy loss for the weights in the network. It is useful for identifying whether a pixel is classified as 0 or 1 in the mask matrix and also provide the probability. The unreduced loss can be described as

$$l(x, y) = Ll_1, ..., {l_N}^T,\ l_n = -\sum_{c=1}^{C} w_c \log \frac{\exp(x_{n,c})}{\sum_{c=1}^{C} \exp(x_{n,i})} y_{n,c}$$

  where x is the input, y is the target, w is the weight, C is the number of classes, and N spans the mini-batch dimension. The reduced version for `'sum'` is defined as

$$l(x, y) = \sum_{n=1}^{N} l_n$$

Training Setup

The neural networks have been trained using the ADAM optimizer. The learning rate is first linearly warmed up from 0 to 5×10^4 in the first 5k iterations and then controlled by the cosine decay schedule to the minimum learning rate of 2.5×10^5. Each model has been trained for 300k iterations for 14 hours (for the 'w/ mask' setting) and 16 hours (for the 'w/o mask' setting) in total on a single Nvidia 2080Ti GPU.

8.5.3 Normals and Albedo NeuS

We now describe implementation specifics for our variant of NeuS that we use to fit to multiview normal and albedo maps.

We follow the NeuS original implementation model which consists of two MLPs to encode SDF and colour respectively. The signed distance function is modelled by an MLP that consists of 8 hidden layers with 256 nodes. The activation function is Softplus with $\beta = 100$ for all hidden layers. A skip connection is used to connect the input with the output of the fourth layer. The albedo network is modelled by an MLP with 4 hidden layers with the size of 256, which takes not only the spatial location as input but also a 256-dimensional feature vector from the SDF MLP (this improves efficiency by allowing the SDF network to encode some local scene information that can be used by the albedo network).

Input Data. This version of the dataset requires three separate folders `albedos`, `normals` and `masks` along with the file `cameras_sphere` containing the information about transformation matrices for every camera pose.

Dataset Object. The Dataset class has been updated for our Normals and Albedo model by adding three new properties `self.normals_lis`, `self.normals_np` and `self.normals` to handle different stages of loading normal maps and converting them into a model-compatible form. The properties `self.images_lis`, `self.images_np` and `self.images` have been repurposed to accommodate for the albedo maps as `self.albedos_lis`, `self.albedos_np` and `self.albedos`.

Ray Batching. During training, Normals and Albedo NeuS generates batches of rays in the same way as the original NeuS. However, this time the information associated with each ray is: (i) an RGB diffuse albedo value rather than a colour value, (ii) the target surface normal vector in camera coordinates, (iii) the foreground mask flag. Including ray origin and

direction, this means that a batch has size [Batchsize x 13] (three value for origin, direction, albedo and normal, one for mask).

Losses. We have implemented two new types of losses for the Normal maps we have included in the dataset.

- `norm_loss`: The loss computes the difference between the normals rendered and the ground truth normals at every iteration. The normals rendered are obtained by matrix multiplying the `gradients` and `weights`, summing them and finally normalising the entire matrix. The ground truth normals matrix is also normalised before computing the error. The loss itself is simply the negated dot product between all pairs of estimated and ground truth normals. The loss is finally multiplied with the mask matrix to discount the effect of background pixels over the `norm_loss`.

- `angular_loss`: The angular loss is normal loss converted in degree. This provides us with a numerical value which could be used to assess the realness of the renderings - lower angular error signifies better rendering.