DEPARTMENT OF COMPUTER SCIENCE

3D Representation of Underwater Scenes using Neural Radiance Fields

Luca Gough

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Bachelor of Science in the Faculty of Engineering.

Thursday 9th May, 2024

Abstract

Novel view synthesis in underwater scenes presents many unexplored challenges, traditional NVS techniques assume a static scene and that the medium in which the images were captured doesn't significantly scatter or absorb light. The aim of this project is to deliver a system for generating an interactive 3D representation of underwater scenes based on a dataset of static images. This system should be optimised for delivering highly accurate novel views of a scene. Novel view synthesis has a wide range of real-world applications including but not limited to: virtual-reality, allowing the user to navigate through a virtual environment, 3D model generation and image/video editing such as scene relighting or viewpoint transformation.

Crucially this project presents a rendering approach that leverages learned depth information to improve its reconstruction accuracy. Additionally we implement several modifications to the MLP loss functions proposed by existing work including a robust losses algorithm. Finally we add a post-processing step to rectify reconstruction inaccuracies generated by the model.

Summary of work completed for this project:

- Performed comprehensive research on state-of-the-art Neural Radiance Field methods to establish a basis for this project.
- Wrote around 1000 lines of python code utilising the Nerfstudio and Pytorch libraries. An additional 500 lines of code were written as part of the project testing framework and the dataset preparation code.
- Spent over 500 hours training Neural Radiance Field models with different components and hyperparameters on different subsea datasets.

Declaration

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Taught Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, this work is my own work. Work done in collaboration with, or with the assistance of others, is indicated as such. I have identified all material in this dissertation which is not my own work through appropriate referencing and acknowledgement. Where I have quoted or otherwise incorporated material which is the work of others, I have included the source in the references. Any views expressed in the dissertation, other than referenced material, are those of the author.

Luca Gough, Thursday 9th May, 2024

Contents

List of Figures

List of Tables

List of Code Listings

Ethics Statement

This project did not require ethical review, as determined by my supervisor, Pui Anantrasirichai

Supporting Technologies

Third party software used to develop this project:

- **draw.io** - https://app.diagrams.net/ - Used for making diagrams.
- **COLMAP** - https://github.com/colmap/colmap - Structure-from-motion tool used for pose estimation.
- **nerfstudio** - https://docs.nerf.studio/ - Provides an open-source modular implementation of different NeRF (Neural Radiance Field) architectures.
- **tinycudann** - https://github.com/NVlabs/tiny-cuda-nn - Provides an implementation of a fully-fused multi-layer perceptron.

Notation and Acronyms

CV	:	Computer Vision
NVS	:	Novel View Synthesis
MVS	:	Multi View Stereo
MLP	:	Multi-Layer Perceptron
NeRF	:	Neural Radiance Fields
RGB	:	Red, Green, Blue
SfM	:	Structure-from-Motion
SGD	:	Stochastic Gradient Descent
MSE	:	Mean Squared Error
PSNR	:	Peak Signal to Noise Ratio
SSIM	:	Structural Similarity Index Measure
LPIPS	:	Learned Perceptual Image Patch Similarity
PDF	:	Probability Density Function
GPU	:	Graphical Processing Unit
VRAM	:	Video Random Access Memory
ADAM	:	Adaptive Momentum Estimation
ILRS	:	Iteratively Reweighted Least Squares
DoF	:	Degrees of Freedom

Chapter 1

Introduction

The concept of computationally generating new perspectives of a scene is not a new one, the earliest works in the 1990s focused on interpolating between corresponding pixels. In the mid 2010s advances were made in generating dense point clouds using Structure-from-Motion techniques. In recent years significant progress has been achieved in the field of novel view synthesis (NVS), the substantial advancements in GPU architectures, such as increased VRAM size and the introduction of Tensor Cores have enabled Deep Neural Networks to assume a central role in NVS research. Specifically in 2020 the seminal work on Neural Radiance Fields by Mildenhall et al. [21] kick-started a cascade of extensive novel view synthesis research.
Performing NVS for underwater scenes comes with unexplored challenges. Typically imagery captured underwater is much noisier than in open air caused by particulates in the water, it also has highly dynamic lighting due to scattering effects within the water and specular reflections from its surface. Additionally objects in the scene itself are almost always dynamic, such as fish swimming in different locations in different images, traditional NVS approaches generally assume a temporal consistency between frames.

The high-level objective of this project is to derive an approach for generating interactive 3D representations of underwater scenes given a dataset of static images. The following targets outline the methodology to achieve this objective:

1. Research literature on algorithms for Novel View Synthesis and identify an appropriate method that can be adapted for this project.
2. Modify an existing state-of-the-art approach to address the challenges associated with this project.
3. Evaluate the resulting algorithm's performance and perform an ablative study to identify salient improvements.

Chapter 2

Technical Background

This project concerns the challenge of generating novel views of an underwater scene given a high definition video dataset. The aim is therefore to identify an RGB value for each pixel in the image given a target camera pose. Prior work demonstrates the effectiveness deep neural networks for approximating high dimensional functions and critically, their ability to encode a scene volumetrically.

2.1 Novel View Synthesis

Novel View Synthesis is a field of Computer Graphics that aims to generate target images of a scene viewed from arbitrary camera poses using static images of the scene. Early approaches in NVS involved sampling from plenoptic light field functions (functions that describe the flow of light in certain directions from certain positions) [7], advances in Structure-from-Motion enabled the estimation of sparse point clouds eventually leading to 3D reconstruction techniques. However, these methods struggle to capture scenes with fine detail furthermore they generally require the scene to be static which is not the case for most underwater scenes. In the early 2020s significant progress was achieved in NVS via the application of Deep Learning techniques and classical volume rendering [21].

2.2 Multi-Layer Perceptron (MLP)

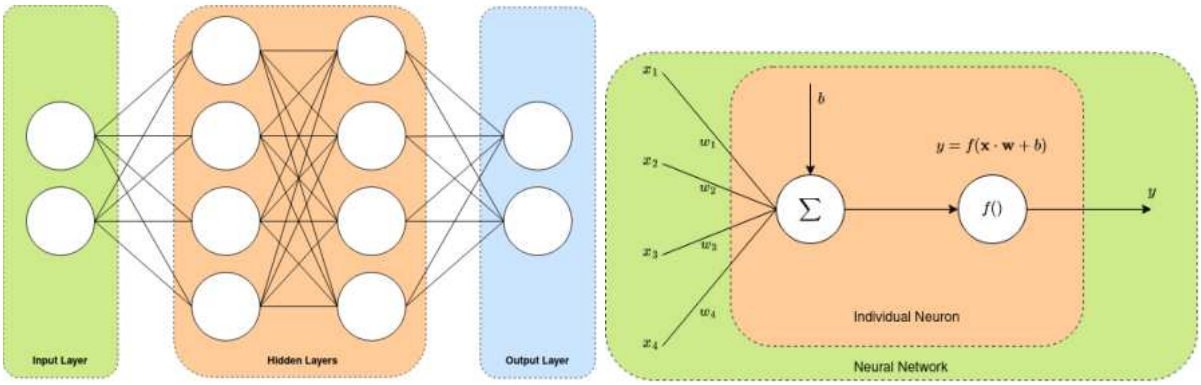
A multi-layer perceptron is a fully connected neural network, MLPs are supervised learning algorithms that act as universal function approximators [11]. This uniquely positions them for learning high-dimensional functions such as the plenoptic functions that describe light flow in a scene. Deep MLPs typically have an input layer, an output layer and several hidden layers of neurons, each hidden layer is fully connected to the layer before and after it (2.1a). Each neuron in a given layer computes its current activation as a weighted sum of the activations in the layer before, this is then skewed by some learned bias and fed into a activation function [5]. The activation function introduces non-linearity into the network allowing it to learn more complex functions. The weights and biases of the MLP can be learned via a variety of methods however for this project we will utilise Stochastic Gradient Descent (SGD) this is the process of updating the learning parameters in the direction of the gradient of the cost function. Typically the dataset is divided randomly into mini-batches aiding in accelerating convergence, allowing the algorithm to escape local minima. In the case of neural networks SGD corresponds to back-propagating the errors found in the output layer, throughout the prior layers. [28].

In a simplified example for a neural network with a single neuron per layer the following three functions correspond to: the cost function, the weighted sum of the previous layer's activation skewed by sum bias, and that same sum after passing it through some non-linear activation function for a given layer L.

$$C_0 = (a^{(L)} - t)^2 \qquad z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)} \qquad a^{(L)} = \sigma(z^{(L)}) \tag{2.1}$$

Utilising the chain rule it can be shown that the rate of change of the final error with respect to the weights at layer L is dependant on the three derivatives in equation 2.2. A similar equation can be derived for the rate of change with respect to the bias.

$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}} = a^{(L-1)} \sigma'(z^{(L)}) 2(a^{(L)} - y) \tag{2.2}$$



(a) Deep Neural Network (Deep MLP) (b) Individual Neuron Activation

Figure 2.1: **Multi-Layer Perceptron: Left:** A diagram illustrating the general structure of a multi-layer perceptron, constituting a input/output layers and multiple hidden layers **Right:** A diagram of a single neuron with inputs $x_1, \ldots, x_4$. (Made in draw.io)

The notion of back-propagating the errors from the output layer is demonstrated in equation 2.3 where the sensitivity of the cost function with respect to the activation of the previous layer is dependant on the weights at the current layer.

$$\frac{\partial C_0}{\partial a^{(L-1)}} = \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}} = w^{(L)} \sigma'(z^{(L)}) 2(a^{(L)} - y) \tag{2.3}$$

2.2.1 Deep MLP Optimisations

Training deep neural networks is a very time consuming process because they generally require much larger volumes of data than other machine learning algorithms it is therefore imperative apply certain optimisations, saving on time and cost.

Optimiser

In recent years more efficient stochastic optimisers have been identified such as Adaptive Momentum Estimation [15] which adjusts the learning rate of a given parameter as a function of the recent gradients for that parameter. Adam consists of two existing optimisation techniques Momentum and Root Mean Square Propagation (RMSProp)[9]. Momentum accelerates gradient descent by adding a fraction of the previous gradient to the current one (2.4) and RMSProp increases the learning rate in regions of shallow gradient and decreases learning rate in regions of steep gradient. RMSProp aims to prevent SGD overshooting local optima with small basins of attraction.

Equation 2.4 updates the weight increment by the weight gradient multiplied by a learning rate λ and add some fraction of the previous iterations weight increment.

$$\Delta w_i^{(L)} = \lambda \frac{\partial C_0}{\partial w_i^{(L)}} + \gamma \Delta w_{i-1}^{(L)} \tag{2.4}$$

Equation 2.5 computes a squared rolling average of the cost gradient with respect to changes in weights. This RMS value is then used to scale the learning rate hyper-parameter λ

$$AV(i) = \beta AV(i-1) + (1-\beta) \left(\frac{\partial C_0}{\partial w_i^{(L)}} \right)^2 \tag{2.5}$$

$$w_i^{(L)} = w_{i-1}^{(L)} - \frac{\lambda}{\sqrt{AV(i)}} \frac{\partial C_0}{\partial w_i^{(L)}} \tag{2.6}$$

Finally, Rectified Adaptive Momentum Estimation [18] addresses a key problem introduced by Adam's variable learning rate. Its large variance in the early stages of training can cause the model to converge on sub-optimal solutions. RAdam introduces a warm-up to a low initial learning rate, meaning the model has to train for a number of epochs before it becomes maximally exploratory of the search landscape. Additionally it toggles adaptive momentum estimation after this point.

Learning Rate Decay

The learning rate hyper-parameter of an MLP model scales the amount the model can adjust its weights. Typically it is beneficial to initialise the model with a higher learning rate than at the end a process referred to as simulated annealing. This effectively increases the model's adaptability at the beginning, allowing it to navigate away from local optima, while also preventing it from overshooting a better solution as training progresses.

Minibatching

Minibatching computes the forward pass of the neural network on multiple training examples, accumulates the error across the whole batch and then updates the network parameters. Minibatching takes advantage of modern hardware's efficient vector processing resulting in large performance gains. In addition to this it can improve the efficacy of the network. The **batch size** hyper-parameter determines the number of samples in one batch, for a dataset of size N batch sizes close to N will lead to the model converging on saddle points [3]. Values closer to 1 will generate lots of noise, because it over-fits to its single datapoint, enabling it to escape from saddle points and local minima.

2.2.2 Performance Metrics

In order to perform stochastic gradient descent we must be able to define an expression for computing a quantitative value of the error between our model's predictions and the ground truth. In the context of this project we must be able to compare the images generated by our model and the training images.

Mean Squared Error (MSE)

Mean squared error measures the average squared difference between prediction and ground truth, it refers to the average loss on the given dataset. For an $m \times n$ image MSE is defined as

$$MSE = \frac{1}{m \cdot n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (I(i,j) - T(i,j))^2 \tag{2.7}$$

Peak Signal to Noise Ratio (PSNR)

Peak Signal to Noise Ratio measures the ratio between the maximum possible amplitude of a pixel and the mean squared error of the full image. PSNR acts as an approximation to human perception of the comparison between two images.

$$PSNR = 10 \log_{10} \left(\frac{MAX^2}{MSE} \right) \tag{2.8}$$

Structural Similarity Index Measure (SSIM)

Structural Similarity Index Measure[37], quantifies the change in structural information which accounts for the fact that spatially-close pixels are strongly correlated. Unlike PSNR and MSE, SSIM extracts three salient features: contrast, luminance and structure which serves to better model human perception[34][40]. Typically SSIM is performed in fixed size windows across the image. The resulting SSIM index is between -1 and 1, a value of 1 indicates perfect similarity between the two images and -1 corresponds to perfect anti-correlation.

$$c(x,y) = \frac{2\sigma_x\sigma_y + c_2}{\sigma_x^2 + \sigma_y^2 + c_2} \qquad l(x,y) = \frac{2\mu_x\mu_y + c_1}{\mu_x^2 + \mu_y^2 + c_1} \qquad s(x,y) = \frac{\sigma_{xy} + c_3}{\sigma_x\sigma_y + c_3} \tag{2.9}$$

$$SSIM(x,y) = c(x,y) \cdot l(x,y) \cdot s(x,y) \tag{2.10}$$

- Sample mean of image x and y respectively: μ_x, μ_y
- Variance of x and y: σ_x^2, σ_y^2
- Covariance of x and y: σ_{xy}
- c_1, c_2, c_3 shifts each value by some function of the dynamic range ($2^{\#\text{Bits per pixel}} - 1$)

Learned Perceptual Image Patch Similarity (LPIPS)

LPIPS measures the perceptual similarity [41] between two images as perceived by humans. It does this by leveraging a pre-trained deep neural network to perform a feature extraction on each image. These features can then be compared with a distance measure and the resulting distance is transformed into a perceptual similarity score. Since the deep MLP was trained on a human-labelled dataset, LPIPS should better align with human perception than standard deterministic approach. The primary drawback of LPIPS is its significantly higher computational cost. LPIPS values range from $[0, 1]$, where values closer to 0 represent higher perceptual similarity.

2.3 Volume Rendering

Volume Rendering is a field of Computer Graphics concerned with generating 2D projections of a volumetric grid. We can represent a scene as a grid of voxels (3D pixels) which describe its colour and opacity. Projections of the grid can be generated using Volume Ray Casting. For each pixel on the image plane a ray is cast from the centre-of-projection of the camera, through the pixel and into the scene. The ray is then discretely sampled along its length, these samples can be composited via a transfer function which accumulates the colour and opacity along the ray. There are various techniques for volume rendering however volumetric ray casting is naturally differentiable and therefore can be learned with gradient descent.

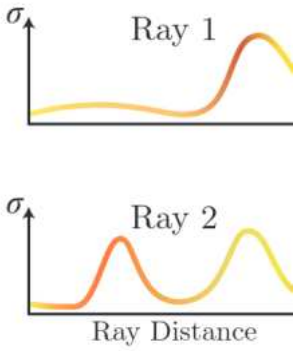


Figure 2.2: **Ray Transmittance:** Rays accumulate more colour from high density volumes closest to the ray origin. Credit: [21]

Ray starting at p_0 with direction vector u

$$r(\lambda) = p_0 + \lambda u \tag{2.11}$$

Transmittance from the centre of projection t_n to t. $\sigma(x)$ represents the volume density

$$T(t) = \exp\left(-\int_{t_n}^{t} \sigma(p_0 + \lambda u) d\lambda\right) \tag{2.12}$$

The exponential decay in equation 2.12 attempts to model the natural attenuation of light due to absorption and scattering [13]. $T(t)$ represents the probability that a ray travels from t_n to t without terminating in a volume. Finally equation 2.13 models the colour accumulated along the ray, projected onto the image plane. $c(x)$ represents the colour at some coordinate x in the voxel grid similarly $\sigma(x)$ returns the density at that coordinate.

$$C(r) = \int_{t_n}^{t_f} T(t)\sigma(r(t))c(r(t))dt \tag{2.13}$$

Since our samples are positioned at discrete intervals we approximate these integrals as a Gaussian quadrature, evaluating them as a weighted sum.

2.3.1 Sampling

In order to calculate the colour of a given pixel we must aggregate samples along its associated ray. In an ideal world we could densely sample the ray however this is computationally expensive, we must employ a sampling strategy to minimize the number of samples we need to take. Typically volume ray casting algorithms will use a stratified sampling approach, first defining a set of bounds to sample between $[t_n, t_f]$ and then dividing up the ray into N bins 2.14. Increasing the number of samples per ray will increase reconstruction quality at the cost of longer rendering times.

$$t_i \sim U\left[t_n + \frac{i-1}{N}(t_f - t_n), t_n + \frac{i}{N}(t_f - t_n)\right] = \phi_{\text{sample}}(i) \tag{2.14}$$

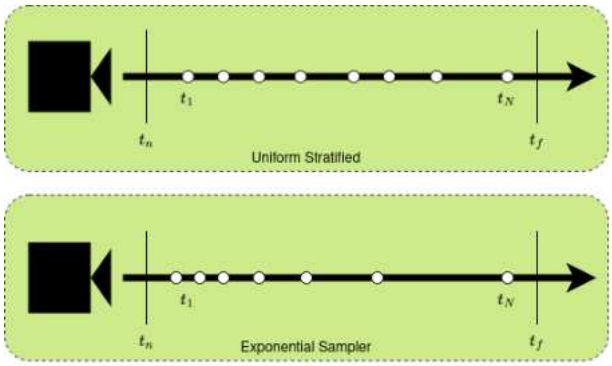


Figure 2.3: **Ray Sampling Strategies: Top:** The diagram demonstrates the sampling strategy described by equation 2.14 with uniform stratified sampling across N bins between t_f and t_n. **Bottom:** The diagram shows an exponential sampling strategy where more samples are taken at the start of the ray, reflecting the fact that volumes closer to the camera will have a greater contribution to the pixel.

2.4 Neural Radiance Fields (NeRF)

2.4.1 Overview

Neural Radiance Fields is a very new algorithm for NVS, first proposed in 2020 [21]. However since then significant advancements have been made in both performance and efficacy. The general method outlined by NeRF involves encoding all the necessary information to render a scene within the weights and biases of a Deep MLP. More specifically the MLP represents a function (2.15) with a continuous 5D coordinate input (θ, ϕ, x, y, z) which corresponds to position and a viewing direction, this describes a ray emitted from a point. The function outputs a RGB colour and a volume density. Density is typically trained independently of viewing direction preserving its consistency across different camera angles.

$$f(\theta, \phi, x, y, z) = ((r, g, b), \sigma) \tag{2.15}$$

The MLP corresponds to a field that describes the radiance of a voxel from a particular viewing direction, this field can be sampled using volumetric ray casting to render a 2D projection of the field.

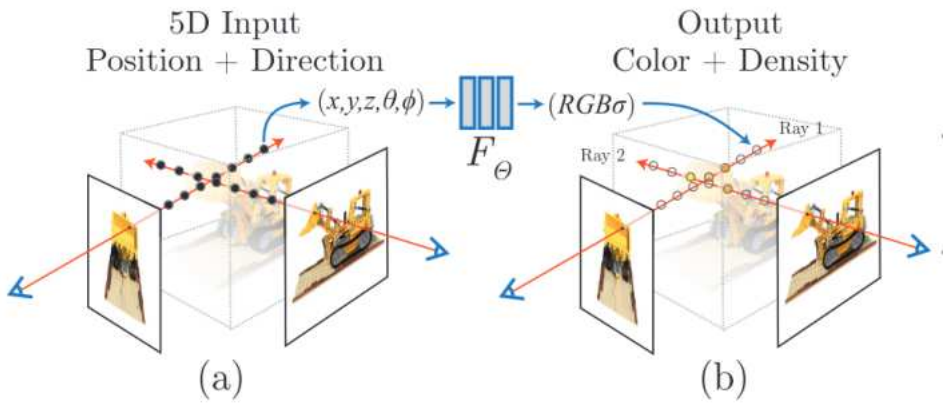


Figure 2.4: **Neural Radiance Field Function:** This diagram from the original NeRF paper [21] demonstrates the function 2.15 that encodes the colour and density information of a scene. It shows the field being sampled from two different camera angles.

2.4.2 Training

The training process for a neural radiance field consists of first generating a set of ray samples for each pixel in a dataset image with width W and height H:

$$\forall x \in \left[\frac{-W}{2}, \frac{W}{2}\right] \qquad \forall y \in \left[\frac{-H}{2}, \frac{H}{2}\right] \tag{2.16}$$

H_{wc} is the extrinsic matrix of the camera describing the transformation between world coordinates and camera coordinates with translation T and rotation R. Equation (2.18) utilises this information to compute the world coordinates for the centre-of-projection of the camera C_{world} and a given pixel on the image plane P_{world}.

$$H_{\text{wc}} = \begin{bmatrix} R^T & T \\ 0 & 1 \end{bmatrix} \qquad H_{\text{cw}} = \begin{bmatrix} R & -RT \\ 0 & 1 \end{bmatrix} \tag{2.17}$$

$$P_{\text{world}} = H_{\text{cw}} \cdot \begin{bmatrix} x \\ y \\ f \end{bmatrix} \qquad C_{\text{world}} = -RT^T \tag{2.18}$$

$f(\lambda)$ describes the set of points that lie on a ray projected from the centre of the camera through a given pixel (x, y).

$$\frac{P_{\text{world}} - C_{\text{world}}}{|P_{\text{world}} - C_{\text{world}}|} = V \qquad f(\lambda) = C_{\text{world}} + \lambda V \tag{2.19}$$

N samples are taken for each ray $f(\lambda)$. λ_i represents one of N samples distributed according to a particular sampling regime, P_i is the sample location along the ray. 2.3.

$$\forall i \in \{i | i \in \mathbb{N} \wedge i < N\} \qquad \phi_{\text{sample}}(i) = \lambda_i \qquad f(\lambda_i) = P_i \tag{2.20}$$

$$\begin{aligned} x &= P_{ix} \\ y &= P_{iy} \\ z &= P_{iz} \\ \text{roll} = r &= \arctan 2\,(V_y, V_z) \\ \text{pitch} = p &= \arctan 2\left(-V_x, \sqrt{V_y^2 + V_z^2}\right) \\ \text{yaw} = q &= \arctan 2\,(\sin(r)V_z - \cos(r)V_y, \cos(p)V_x + \sin(p)\sin(r)V_y + \sin(p)\cos(r)V_z) \end{aligned} \tag{2.21}$$

Each sample at varying points along each ray is passed as input to the MLP which will output a colour and density value for that sample. The volume renderer can now accumulate the colour and density and produce a colour value for each pixel. Thereby outputting a reconstructed version of the original dataset image based on the scene information encoded in the neural network. The model can then quantitatively evaluate its reconstruction performance using the average $\mathcal{L}2$ loss for each pixel. This loss is then backpropagated through the network allowing it to adjust it's weights with respect to their contribution to the reconstruction loss. After enough iterations of this process the multi-layer perceptron should converge on an accurate reconstruction for each image in the training dataset.

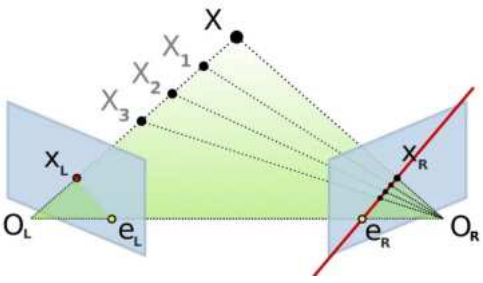


Figure 2.5: **Multi-View Stereo depth estimation**: The diagram shows the depth estimation of a point X using the epipolar lines from two camera angles.

Typically NeRF methods employ learning rate decay as the training progresses. This prevents the model from making large adjustments to the weights later in training. Meaning once the model converges on a low contrast reconstruction it can learn the more high frequency details. We can also utilise mini-batching to allow the model to generalise properly [39] in addition to training multiple batches in parallel. Theoretically the trained model can generate images from camera angles that weren't in the training dataset. For an accurate reconstruction of an evaluation image, at minimum each voxel needs appear in two non-colinear camera angles. This is because the field cannot infer depth without sampling a volume from two disparate camera angles. This approximation becomes more accurate with each camera angle that can see a particular voxel.

2.4.3 NeRF Optimisations

Coordinate Encoders

One of the main problems with NeRF identified by the original paper [21] is that typically MLPs are biased towards learning low frequency functions [27] resulting in poor reconstruction accuracy on scenes with high frequency variations in colour. This issue was addressed in the original paper by mapping the input coordinates to a higher dimensional space before passing them as input to the neural network. Specifically they applied equation 2.22 to each component of the coordinate vector projecting them from $\mathbb{R}$ to $\mathbb{R}^{2L}$.

$$\gamma(p) = (\sin(2^0\pi p), \cos(2^0\pi p), \cdots, \sin(2^{L-1}\pi p), \cos(2^{L-1}\pi p)) \tag{2.22}$$

Later works on coordinate encoders proposed learning this mapping for each input vector, these components are optimised as a part of the standard back-propagation process in the original MLP [22]. The trainable mappings are F dimensional vectors contained in L separate grids, where L corresponds to the resolution level at each grid. For a given sample of the field, we first find the surrounding voxels at each resolution level. By hashing their coordinates we can efficiently look up the corresponding feature vectors in the L grids. These feature vectors are interpolated to the actual position of the sample and concatenated across resolution levels. The resultant high dimensional vectors enables the MLP to learn high frequency colour variations and as a consequence of the hash encoding significantly reduced training times.

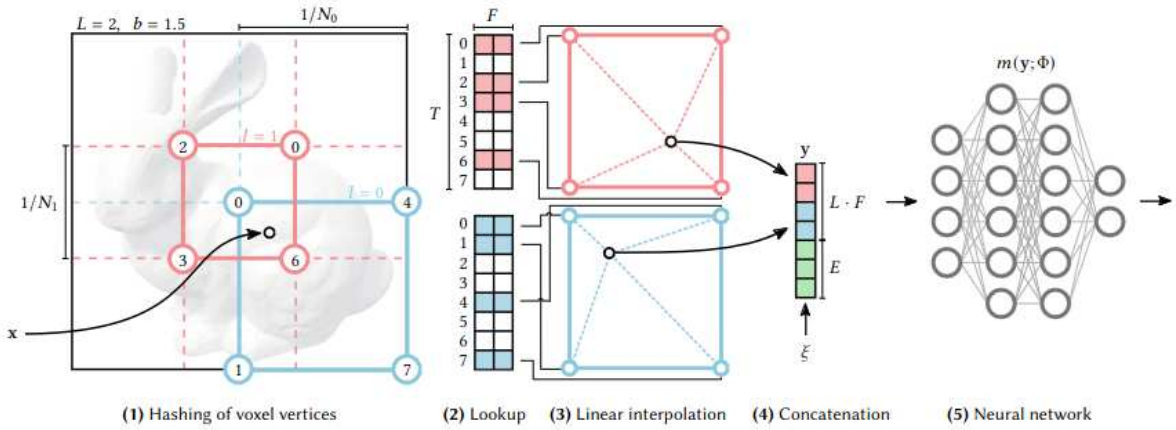


Figure 2.6: **Multiresolution Hash Encoding in 2D: 1:** A sample of the field x with it's 4 surrounding voxels in each resolution layer. **2:** The surrounding voxels' coordinates are hashed a particular grid location. **3:** The 4 feature vectors at each grid location are linearly interpolated to x. This happens for each of the L resolution grids. **4:** The resultant feature vectors are concatenated into a single vector. **5:** The high dimensional vector is passed as input to the MLP. Credit: Müller et al. [22]

Fully Fused MLP

Previous works on the NeRF model used existing libraries such as TensorFlow or Pytorch for their implementation of the core MLP. In 2021 Müller et al. at Nvidia [23] developed a bespoke implementation, known as a Fully-Fused MLP. It takes advantage of the usual structure of NeRF models, which often consist of numerous hidden layers, each with a narrow width of 64 neurons. This enables them to fit the entire weight matrix of a layer into the on-chip cache, minimising traffic to the much slower main memory. Additionally it utilises Nvidia's GPU Tensor cores which function as half-precision matrix multipliers to decrease training times by up to 1/10th of TensorFlow's implementation.

Scene Contraction

First introduced in MipNeRF-360 [1] a scene contraction bounds the field to the interior of a fixed volume, this reduces the complexity of selecting the point of termination for each ray. We can warp the space into a sphere of a given radius R. This is particularly useful for outdoor, unbounded scenes where volumes

are separated by a much greater variation in distance.

$$D(x, y, p) = \left(\sum_{i=0}^{N} |y_i - x_i|^p \right)^{1/p} \tag{2.23}$$

$$f_{\text{Spherical Contraction}}(x) = f(x, 2) = \begin{cases} x & D(x, c, 2) \leq 1 \\ \left(R - \frac{1}{D(x,c,2)}\right)\left(\frac{x}{D(x,c,2)}\right) & D(x, c, 2) > 1 \end{cases} \tag{2.24}$$

Equation (2.24) performs a spherical scene contraction for input coordinates at a distance greater than 1 from the centre of the sphere at point c. Multi-resolution has encoding described above, assumes that the voxel field is arranged in a grid with equidistantly spaced voxels. A spherical scene contraction will warp the distance between voxels, we can negate this effect by using a distance measure (2.23) with order $p = \infty$. Commonly known as Chebyshev distance. This will contract the space to a cube, ensuring the distance between voxels inside the cube is consistent in all three dimensions.

Hierarchical Sampling

As discussed in section 2.3.1, it's critical to pick an optimal ray sampling strategy in order to balance rendering times with rendering accuracy. However a static sampling function will never be optimal for all viewing directions this is because along the length of a ray, different samples will contribute varying amounts to the final pixel value. For instance, occluded areas will yield no contribution, making it inefficient to sample them. The original NeRF paper [21] optimises a sampling strategy for each ray within the training process. This is achieved by training two MLPs, a "course" and a "fine" network. The "course" network uses a fixed stratified sampling strategy (Fig. 2.3), a probability distribution is then constructed as a function of the accumulated transmittance of each sample. This PDF assigns greater probabilities to regions along the ray that have large contributions. Subsequently a second set of locations are sampled from this PDF and the "fine" network is trained on the union of both sets of samples.

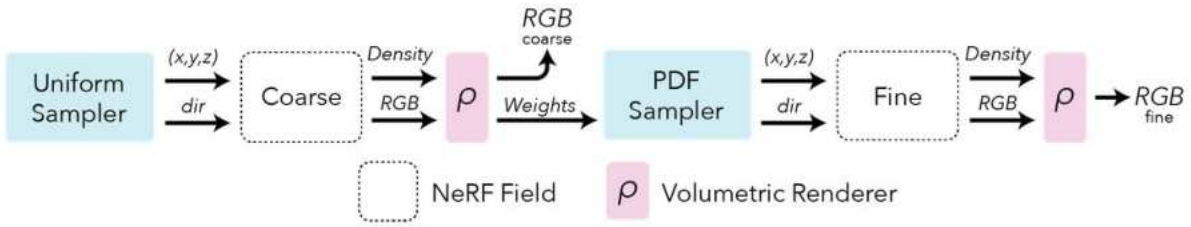


Figure 2.7: **NeRF Pipeline:** This diagram illustrates the full NeRF pipeline as described in Mildenhall et al. [21]. A uniform stratified sampler allocates an initial set of sample locations, these are evaluated in the "course" network. Each sample's contribution is represented as a PDF, used as a sampler for the "fine" network. **Credit:** nerfstudio

Pose Refinement

Errors in the pose estimation process resulting in erroneous extrinsic matrices for each training frame can lead to significant loss in detail in the reconstructed images. Additionally they can introduce artifacts as the relative positioning of volumes in each frame is now perceived as inconsistent. Pose refinement [8, 38, 12], proposes a method for optimising the camera's intrinsic matrices. The parameters encoding the 6 DoF (3 Translation, 3 Rotation) are adjusted via gradient descent minimising the $\mathcal{L}2$ reconstruction loss with respect to these parameters. Some NeRF implementations initialise these parameters with random values allowing the poses to self-calibrate. For this project we initialise the extrinsic matrix with the poses generated by COLMAP's structure-from-motion pose estimation. This allows pose refinement to correct errors in COLAMP's pose estimation.

2.5 Software

2.5.1 Nerfstudio

Nerfstudio provides a fully modular implementation of NeRFs, it exposes a comprehensive API enabling a user to easily implement their models. This project is primarily built on top of the Nerfacto model. Each Nerfstudio method consists of a 'pipeline' which is comprised of a data manager and the model. A custom model will override the base class methods which are called by Nerfstudio internally.

2.5.2 COLMAP

COLMAP [31, 32] is an open source structure-from-motion software package utilised in this project for its pose estimation features. COLMAP uses feature-extraction to identify corresponding points between frames and SfM techniques to learn the positions and orientations of the camera given a series of images.

Chapter 3

Related Works

3.1 SeaThru-Nerf

SeaThru-Nerf [17] released in December 2023 addresses key issues in representing underwater scenes with NeRF, specifically they model the absorption and scattering of light caused by the water by altering their rendering equations to include these effects. These equations contain parameters which are learned by a separate MLP (σ^{bs} = Backscatter Density, σ^{attn} = Attenuation Density, c^{med} = Medium Colour).

$$\hat{C}(r) = \sum_{i=1}^{N} \hat{C}_i^{\text{obj}}(r) + \sum_{i=1}^{N} \hat{C}_i^{\text{med}}(r) \tag{3.1}$$

$$\hat{C}_i^{\text{obj}}(r) = T_i^{\text{obj}} \cdot \exp\left(-\sigma^{\text{attn}} t_i\right) \cdot \left(1 - \exp\left(-\sigma_i^{\text{obj}} \delta_i\right)\right) \cdot c_i^{\text{obj}} \tag{3.2}$$

$$\hat{C}_i^{\text{med}}(r) = T_i^{\text{obj}} \cdot \exp\left(-\sigma^{\text{bs}} t_i\right) \cdot \left(1 - \exp\left(-\sigma^{\text{bs}} \delta_i\right)\right) \cdot c^{\text{med}} \tag{3.3}$$

Whilst this model performs well in static scenes with temporally consistent objects and lighting, it fails to generate an accurate representation of scenes with moving objects such as fish. Any static NeRF method will represent dynamic objects with inconsistent positioning and semi-transparent density (3.1 left). This is because the object occupies a different position in each image meaning it's sampled with a much lower density from different viewing angles. Another major issue with most NeRF models is the existence of 'floaters' clouding the generated view. 'Floaters' are foggy artifacts in the scene. This issue occurs when the MLP assigns a non-zero density to voxels without objects in the original scene.

Figure 3.1: **SeaThru-Nerf Weaknesses: Top:** Low density fish volumes. **Bottom:** Floating artifacts.

Chapter 4

Project Execution and Implementation

4.1 Introduction

This chapter explains the contributions made by this study, as well as how those contributions can be optimised. The first section 4.2 of this chapter covers the dataset used to train the machine learning model on including pre-training computation and dataset reduction. The next section 4.3 describes and explains the baseline model that this study builds upon, utilising many of the techniques outlined in the technical background 2. The subsequent sections cover the modifications made to this model with the goal of improving reconstruction accuracy. All the models trained for this project in both development and analysis we're trained on a PC provided by the University of Bristol the specification of which contains an Nvidia GTX 3090 GPU with 24GB VRAM, Intel Core i7-11700K CPU and 64GB at 3200MHz of system memory.

Technical contributions:

- Dataset reduction algorithm for removing redundant video frames, thereby reducing overall training times. Including a tool for estimating a projected dataset reduction percentage.
- Rendering approach utilising learned depth information to re-weight samples as a function of their distance from the closest volume to the camera.
- A Nerfstudio implementation of the Robust losses algorithm presented by [29]. This paper shows promising results for reconstruction accuracy with the primary focus of removing floating artifacts.
- A modified implementation of the gradient scaled loss function described by [25] which utilises depth information to rectify algorithm for scenes that include volumes in close proximity to the camera.
- A post processing sharpening step with the goal of addressing issues with representing high frequency detailing suffered by most generic NeRF methods.

4.2 Dataset

Due to the nature of NeRF models a new MLP must be trained for each scene, this requires a new dataset of images for each scene. However a key benefit of NeRF is that they don't require many images per scene, for a spherical scene around 150-200 images from different viewpoints (Depending on the complexity of the scene) may be sufficient to generate accurate reconstructions. This dataset comprises of a single high-definition (1920x1080) video for each scene. Each video is recorded as a diver circles around a focal point within the scene, with the camera focused on that point. All the videos are captured in shallow, relatively clear water.


Figure 4.1: **Example dataset images:** The images above show two frames from two separate subsea datasets. Both videos contain around 1000-2000 frames captured by a diver circling a central object (Left: rainbow watch, Right: Coral).

4.2.1 Pose Estimation

NeRF's sampling process requires that rays are projected into the scene, in order to do this accurately we must know the position and orientation of the camera in 3D space at each frame. Since the calibration information is not provided in the original dataset we must generate this. For this project COLMAP was used to calculate the pose information of each frame. COLMAP uses traditional structure-from-motion techniques as well as feature matching to estimate the relative position and orientation for all the images in the dataset. Nerfstudio (used in this project for the base NeRF model implementation) utilises COLMAP to generate a *transforms.json* file which contains a homogeneous transformation matrix for each frame.

```
{
    "frames": [
        {
            "file_path": "images/frame_00000.jpg",
            "transform_matrix": [
                [ 0.99, 0.01, -0.12, 0.31 ],
                [ -0.12, -0.027, -0.99, 7.79 ],
                [ -0.01, 0.99, -0.02, 5.03 ],
                [  0.00, 0.00,  0.00, 1.00 ]
            ]
        }
    ]
}
```

Listing 4.1: **COLMAP transforms.json example:** The transforms.json file contains a homogeneous (world to camera) transformation matrix for each frame which can be loaded into the NeRF program and used to compute the origin and direction for each raycast.

4.2.2 Frame Selection

In the interest of reducing training times we can also reduce the size of our input training dataset, we can achieve this by either down-scaling each image or by reducing the total number of images by removing redundant frames. Pairs of frames that have a high PSNR may be considered redundant. Both methods will reduce the number of rays projected per training epoch and therefore reducing training times. For this project our datasets consist of 24 frames-per-second video with slow camera movements therefore neighboring frames are generally almost identical. This means that for each frame in the dataset we can

compare its PSNR value with the subsequent frames, if that value is greater than a given threshold we can discard the frame. Since pairs of containing most of the same details will have a higher PSNR. This threshold value determines the percentage reduction, however it will be different for every dataset. Since it requires some trial and error to ascertain a suitable PSNR value for a given proportion of reduction, this project also provides a tool for estimating these values. We observe that for datasets with slow moving relative optical flow (camera movements or objects moving in the video), the relationship between the PSNR threshold and percentage reduction is linear. Leveraging this assumption we can compute the percentage reduction given two separate PSNR thresholds and interpolate the values in between. Note that we perform pose estimation before the dataset reduction, this increases the chance of successfully identifying poses for each frame. The code that performs this reduction can be found in appendix E. This reduction will result in a loss of reconstruction accuracy 4.1.

```
D:\> python filter_data.py --path=[PATH_TO_DATA] -e
PSNR Threshold = 16.069240522582696 ---> Kept 74.46%
PSNR Threshold = 12.069240522582696 ---> Kept 5.57%
Interpolating...

| % of dataset retained | PSNR Threshold
|-----------------------|----------------
| 30%                   | 13.49
| 50%                   | 14.65
| 70%                   | 15.81
| 90%                   | 16.97
```

Listing 4.2: **PSNR threshold interpolation:** This bash terminal snippet shows the output of the predicted dataset reduction for a given PSNR threshold.

% of original dataset	PSNR	SSIM	LPIPS	Runtime (HH:MM:SS)
90%	23.89	0.83	0.10	04:45:28
70%	23.81	0.82	0.11	04:45:36
50%	23.06	0.80	0.12	03:32:00
30%	22.62	0.78	0.13	03:02:03

Table 4.1: **Dataset size's effect on reconstruction accuracy:** It's clear that as the dataset is reduced in size the reconstruction accuracy is gradually reduced along with training runtimes. The reduction in training time is however less than expected, this indicates that the forward pass of the model (once per ray sample) contributes to a much smaller proportion of the runtime than the backward pass (once per batch).

COLMAP will also generate resolution downscaled versions of the dataset. Training with these datasets will result in the output resolution being downscaled and therefore these are more useful for hyperparameter tuning where many iterations of the model have to be trained in quick succession. The downscaled versions of the images are stored in directories with the format 'Images_[SCALING]':

```
Dataset
├── Images
├── Images_2
├── Images_4
├── Images_8
└── transforms.json
```

4.2.3 Dataset Partitioning

When training a new model, typically a dataset is partitioned into three disjoint groups. The first of which is the training set, this data is used exclusively for training the model. A validation set is utilised while fine tuning the model's hyper-parameters, it provides a quantitative evaluation of the model's current performance. Finally an evaluation set is used to evaluate the model's final reconstruction accuracy, this can be useful for identifying over-fitting. For this project we are using an 80:10:10 split for training:validation:evaluation. We perform this split after the frame selection process and add the discarded frames from that process to the evaluation dataset. This is because the evaluation execution time is negligible compared to training time so having a larger evaluation dataset makes little difference to performance.

4.3 Pipeline Architecture

The work in this project aims to define a basis of viability of the NeRF model for representing subsea scenes. Whilst the underwater datasets contain challenging artifacts compared to open air video, the fundamental novel view synthesis method remains consistent. We hypothesise that modifications to the loss function and/or rendering approach will address these challenges.

4.3.1 Model

Building on the work of nerfstudio and their nerfacto model this project takes (Num. of Images) $\times$ (Image Width) $\times$ (Image Height) or $(N \times W \times H)$ inputs of dimension $\theta \in \mathbb{R}^6$. Corresponding to 3 spatial dimensions and 3 directional dimensions. Our model outputs in the form $\phi \in \mathbb{R}^4$, 3 dimensions for RGB and 1 for density σ. Initial ray samples are generated from a piecewise sampler, allocating fewer samples as the step size from the camera increases. The model feeds these samples into a proposal sampler to learn locations of samples that contribute significantly to the final image. The proposal sampler implements the multi-resolution hash encoding [23], transforming the sample inputs to effectively learn high frequency colour variations. The proposal sampler forwards highly contributing samples to the Neural Radiance Field. The renderer uses the output of the radiance field to generate a reconstructed image, the accuracy of this reconstructed is evaluated by the model's loss function. This loss is then utilised to update both the proposal network and the radiance field via backpropagation.

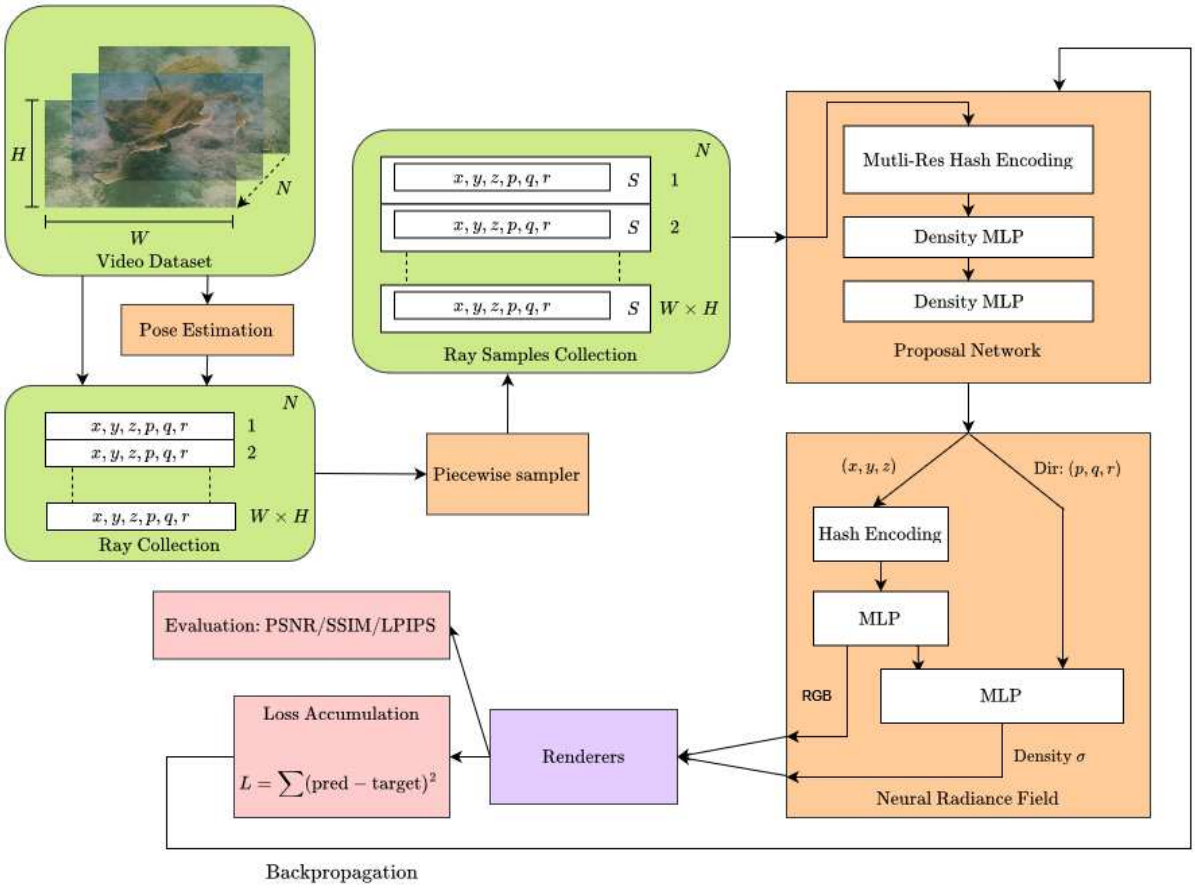


Figure 4.2: **Pipeline Architecture Flow Chart:** Starting with the initial dataset of N images a collection of rays is generated from the estimated pose of each frame. The piecewise sampler then distributes S samples along each rays. These samples are fed into the proposal network which redistributes samples to learned higher density locations. The Neural Radiance Field then generates an RGB value and density for each sample and is forwarded to the renderer which accumulates density and colour values along each rays, producing an image. These images are compared to the target images and the resultant loss is back-propagated down the pipeline.

Once trained the 'field' encodes the radiance at each voxel and therefore can generate novel views of the scene given a position and orientation input. Typically the model can only do this accurately if multiple images contain a target voxel.

4.3.2 Hyperparameters

Hyperparameters are the typically untrainable parameters that specify details about the model and its training process. For this project the 'field' utilises a deep MLP with 3 hidden layers, each with 128 dimensions wide. This structure enables us to leverage the high performance Fully-Fused MLP implementation developed by NVIDIA [23]. Each of the hidden layers uses a ReLU activation function, the output layer uses a sigmoid. All of the MLPs use the 'Rectified Adam' optimiser. (2.2.1, [15]). Each MLP has its own learning rate decay function but in general we use an exponential decay function with an initial warm-up period. A full list of hyperparameters can be found in Appendix A.

4.4 Renderer

This project presents a novel modification to the standard NeRF renderer. It aims to learn a depth map of the scene which can be used to weight ray samples relative to their distance from the point of maximum absorption along the ray. Traditional NeRF methods are intrinsically biased towards up-weighting samples closer to the camera. This is because neighbouring rays have a closer spread nearer to the camera (4.3). The impact of this is evident in figure (3.1 right) where the lower half of the image is much clearer due to it being closer to the camera. The proposed method identifies the weighted median density point along a ray, constructing a depth map of the scene (figure 4.4).

Figure 4.4: **Depth Map Comparison: Left:** An image representing the learned depth map (After 3500 iterations) using median absorption. Darker blue implies a voxel is closer to the camera. **Right:** Training image corresponding to the depth map image.

The weighted median will locate the point at which the cumulative sum of the transmittance (eq. 4.1) values exceeds 50% of the total value. Transmittance computes a particular sample's 'contribution' to a particular ray's colour as a function of it's density.

$$\text{Transmittance} = T(d) = \exp\left(-\int_{s=0}^{d} \sigma(x_s)ds\right) \quad (4.1)$$

Since almost all the absorption occurs on the front face of the object this method effectively identifies the depth along each ray. Because this depth map relies on learned density values, the initial map is likely to be highly inaccurate and thus erroneously bias the densities along each ray. Hence we choose to utilise the depth weightings after 3500 iteration steps, this proved long enough to accurately learn the depth map for each training image. Utilising the depth points we can define a continuous function centered on this point that models the absorption on the front face of the opaque solid. This can be approximated as a Gaussian function (eq. 4.2) where d = predicted depth and s is the sample location along the ray.

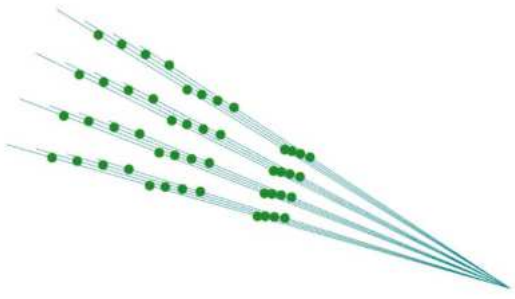


Figure 4.3: **Ray Spread Diagram:** A diagram illustrating the increasing spread of ray samples further away from the camera.

$$w(s) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-(s-d)^2}{2\sigma^2}} \tag{4.2}$$

The Gaussian distribution is particularly useful in this context because it's enclosing area is 1. This implies that the total ray transmittance remains consistent both before and after applying the weightings. This has the effect of redistributing the observed density, down-weighting low density regions before and after the learned depth point. This property is evident in figure (4.3) where r models the sample density.

$$\int_{-\infty}^{\infty} w(s)ds = \lim_{r\to\infty}\left(\sum_{s=0}^{N}\frac{w(s/r)}{r}\right) = 1 \tag{4.3}$$

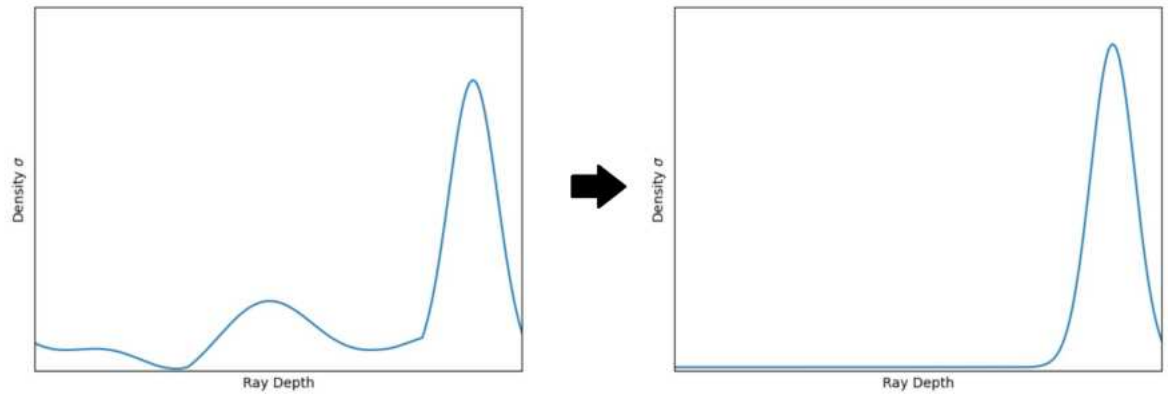


Figure 4.5: **Ray density/depth function:** This diagram illustrates the redistribution of density values described by equations 4.2 and 4.3. This approach aims to remove semi-transparent artifacts in the foreground by fitting the samples' transmittance to a gaussian distributed centered on the front face of the opaque solid in the background.

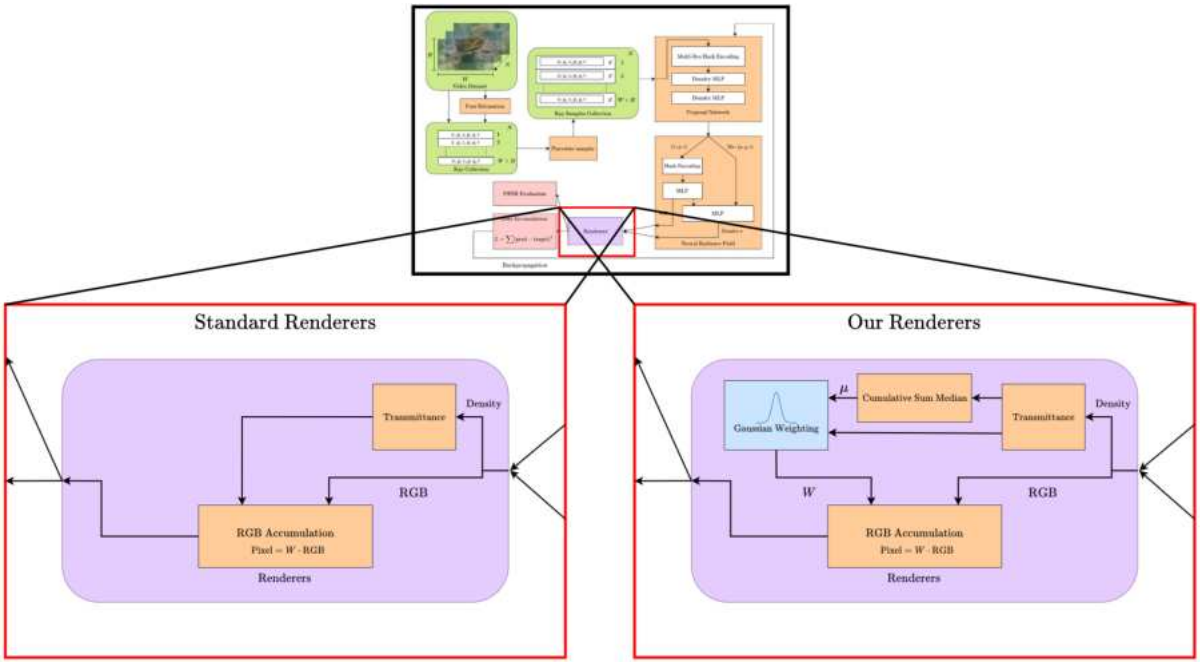


Figure 4.6: **Renderer Architecture:** This diagram demonstrates how the new rendering pipeline integrates with the existing model. The transmittance values are refitted to a Gaussian distribution the results of which are used to compute the RGB accumulation for each pixel.

4.4.1 Improvements

Standard Deviation

An initial obvious issue with this approach is the hyper-parameters that it introduces, namely the value chosen as the standard deviation for the ray absorption curve. For simplicity we can bound the ray length to 1. Choosing a smaller value leads to large blocks of a single colour due to errors in the depth map estimation. If the model erroneously places the point of median absorption before the actual face of the volume, a smaller standard deviation results in most of the density being redistributed among voxels in the water inside of the volume. Leading to large continuous blocks of blue pixels in front of the volume's face. Conversely if the standard deviation is too large and the absorption is spread over a large area of the ray we can expect to see more of the floater artifacts resulting from erroneously assigning density to voxels between the camera and the nearest volume. After extensive testing setting this value to between 0.3-0.6 generally resulted in the highest PSNR.

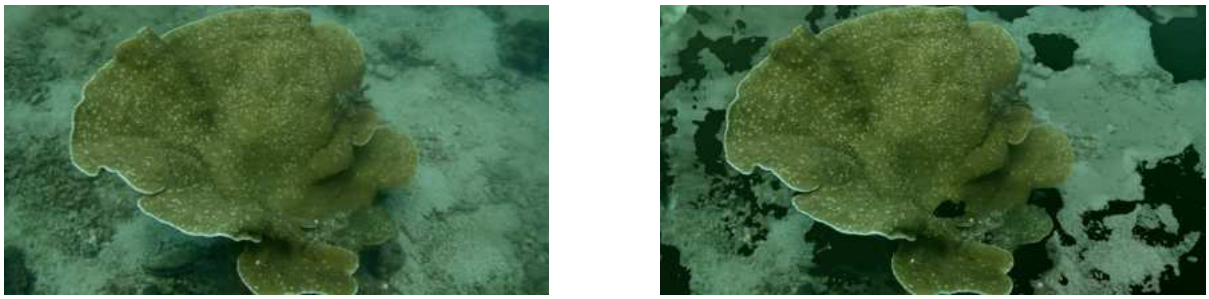


Figure 4.7: **Left:** Image generated with a standard deviation of 0.1. **Right:** Image generated with a standard deviation of 0.45. The dark blobs in the right image demonstrate the affect of utilising an absorption curve with too small a spread.

Absorption Curve Standard Deviation	PSNR (Higher is better)
0.1	22.39
0.3	23.10
0.5	23.15
0.7	23.01

Table 4.2: **Absorption Curve Standard Deviation vs PSNR:** From this table we can see that the optimal standard deviation value lies between 0.3 and 0.5 for this particular dataset.

Far Plane Depth

Another major issue with this method is obvious when areas of the target image do not contain a volume before the 'far plane'. The 'far plane' is a hyperparameter value that restricts the ray sampler, meaning sample positions are bound between the camera ('near plane') and the 'far plane'. If minimal density is observed along a ray the point of median absorption may be close to the camera as the majority of the observed density is attributed to absorption in the water. As a consequence the resultant pixel colours will be noisy as the colour of voxels in the medium is essentially the same as it's value before training. We can observe this effect in figure 4.8, a large contiguous block is assign the same depth in top of the image, the corresponding reconstruction contains significant noise in this region. Note that the red pixels in the centre of the image are errors in the depth estimation. In order to prevent this issue from occurring, we can disable the depth-based re-weighting if the ray fails to accumulate a total density greater than a certain threshold value. In this case we apply a weighting of 1 to each density sample. The renderer will then allocate the remaining density to the background RGB value (last few samples on the ray).

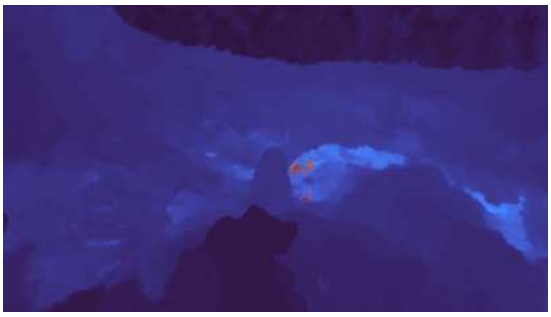


Figure 4.8: **Left:** Subsea dataset frame. **Right:** Corresponding median absorption depth map. Dark blue implies a voxel is closer to the camera, light blue means a voxel is far away. The dark blue area in the top right of the depth map is a region where there are no large volumes before the 'far plane' and so scattering and the water's absorption contribute to the majority of the observed density.

An advantage of this approach is its inherent ability to remove dynamic objects from the resultant image. This is because dynamic objects will naturally have a lower learned density than static structures, hence why the fish appears semi-transparent in figure 3.1. A disadvantage of this method is that it assumes there are no low density static structures in the foreground of the scene. In practice such an object is highly unlikely to exist in an underwater dataset. This approach also assumes that the background is mostly static. We can expect some subsea datasets to include dynamic background objects such as seaweed. We could choose to address this issue by integrating our renderer with a dynamic model however we discuss in section 5.1.2 why this isn't feasible.

Base Value

In order to scale the effective power of the new rendering approach we introduce a 'base value' parameter which acts as the minimum weight which can be assigned to a particular sample. This value attempts to account for error in the depth estimation process by enabling the model to assign some density to all samples along each ray. Using values closer to 1 decreases the proposed model's ability to mitigate floating artifacts and remove dynamic objects from the reconstructions. In section 5.2.2 we analyse the affects of selecting difference base values on the reconstruction quality. This modification does violate the conservation of total transmittance yielded from an unmodified Gaussian distribution with an area under the curve of 1. However, this fact does not significantly affect the final reconstructions because in regions devoid of floating artifacts or dynamic objects most of the total density is already assigned to the background volume. Conversely in regions containing these artifacts we can expect our model to reduce the total accumulated density since we are reducing the density assigned to the foreground voxels. Unless stated otherwise for all the experiments in this study we use a base value of 0.2 this value was obtained via trail-and-error across multiple datasets.

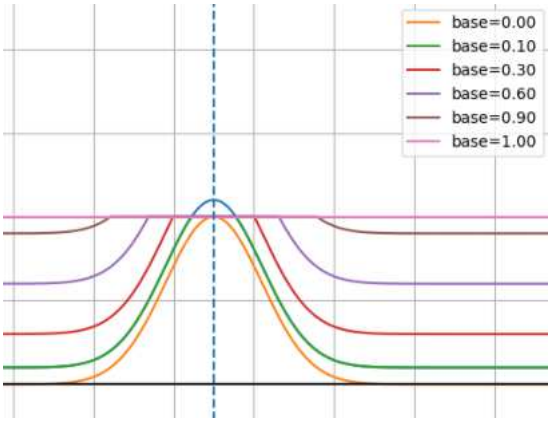


Figure 4.9: **Density re-weighting Gaussian curve:** This graph shows the distribution applied to the density samples with different base weight values.

4.5 Loss Function

All machine learning methods require a measure that quantitatively describes the error between the model's predictions and the ground truth values. For artificial neural networks this error or loss value is back-propagated through the network adjusting weights and biases within the network that contributed to the error. The standard NeRF implementation uses MSE for its loss function. This function describes how 'correct' our model's prediction is compared to the ground truth target image. MSE simply assumes a linear relationship between predictions and targets. The squared error is computed for each ray $r(t) = o + td$, the average of these values comprises the MSE loss function.

$$\mathcal{L} = \sum_{i=0}^{N} \mathcal{L}_{\text{rgb}} \tag{4.4}$$

$$\mathcal{L}^r_{\text{rgb}} = ||\hat{C}(r) - C(r)||_2^2 \tag{4.5}$$

While MSE accurately describes the absolute error for each predicted data-point, it applies an even weighting to both outliers and inliers. In the later stages of training MLPs can be extremely sensitive to outliers because of the vanishing gradient problem [10, 2]. This occurs because as the training progresses we can expect the loss magnitude to decrease, essentially slowing the training process over time. This is because after each iteration we update each weight by an amount proportional to the partial derivative of the total loss with respect to the weight. As the activation functions becomes saturated their derivatives reduce to zero. Leading to very little loss actually backpropagating through the network. This implies that outlier predictions occurring towards the end of the training process will greatly affect the existing weights and reduce the reconstruction quality. In the context of underwater scenes, these outliers manifest as floating artifacts generated by erroneously sampled densities in the medium between the camera and the background. These densities can be attributed to features such as particles and back-scattering within the water.

4.5.1 Robust Losses

Recent work has demonstrated the benefits of utilising a 'robust' loss function for reducing the model's sensitivity to outliers [29]. A robust loss function will down-weight outliers which in the context of NeRF corresponds to inconsistencies between viewing angles. The primary difficulty with this approach is a developing a method for identifying outliers without mis-classifying high frequency details as outliers. RobustNeRF achieves this by applying a 3x3 smoothing kernel to the $\mathcal{L}2$ loss (eq. 4.6), this attempts to model the spatial smoothness of the outliers. Thresholding the result of this convolution removes high frequency details. This result is then used to label each 8x8 neighborhood as an outlier or an inlier, Based on the convolution applied to the surrounding 16x16 patch, leveraging the spatial locality of outliers (eq. 4.7). Finally the $\mathcal{L}2$ loss is then re-weighted using the computed patch weights (eq. 4.8). This algorithm is generally known as Iteratively Re-weighted Least Squares [36] (IRLS).

$$\mathcal{W}(r) = (\mathcal{L}^r_{\text{rgb}} * \mathcal{B}_{3\times3}) \geq T_B \tag{4.6}$$

$R_N(r)$ defines an $N \times N$ neighbourhood around r. $T_B = 0.5$ and $T_R = 0.6$. A represents some aggregation function for a given neighbourhood.

$$\omega(R_8(r)) = A_{s\in R_{16}(r)}(W(s)) \geq T_R \tag{4.7}$$

$$\mathcal{L}^r_{\text{robust}} = \omega(R_8(r)) \cdot \mathcal{L}^r_{\text{rgb}} \tag{4.8}$$

Our model integrates this robust loss algorithm after a specified number of epochs. This is chosen to further prevent the loss function kernel removing high frequency colour variations while the training is in a more exploitative phase. The robust losses algorithm also increases loss for photo-metric inconsistencies such as dynamic objects. Meaning it should aid in the effort to remove the semi-transparent fish. In practice this Robust loss algorithm is very difficult to fine tune and its hyper-parameters were extremely dataset dependant. Specifically erroneously chosen 'Blurred' image and 'Residuals' thresholds which attempt to classify outliers and inliers can result in extremely low fidelity reconstructions.

4.5.2 Gradient Scaled Losses

Another approach for eliminating floater artifacts is to up-weight loss gradients sampled far away from the camera. This method, first described in [25], assumes that the floating artifacts result from allocating a higher proportion of loss gradient to near-camera volumes. This aligns with our assumption leveraged in our depth re-weighting rendering approach 4.4. Effectively this method scales the loss gradient with respect to colour or density as a function of the samples distance from the camera.

$$\nabla_{\text{grad scaled}} = \frac{\partial C_0}{\partial w_i^{(L)}} \cdot \min(1, |c - p|^2) \tag{4.9}$$

Equation 4.9 describes this method applied to the RGB loss gradient ∂C_0 with respect to a particular weight $\partial w_i^{(L)}$, for a sample at position p from a camera positioned at c. This method leverages the inverse square relationship between sample density and distance from the camera, this is further illustrated in figure 4.3. In practice this method is reasonably effective at visibly removing floater artifacts 4.10.

Figure 4.10: **Left (PSNR=27.97):** Image generated by the current model with gradient scaled losses. **Middle (PSNR=27.37):** Image generated from the same pose by the current model without gradient scaled losses **Right:** Ground truth image. It clear to see that the centre image contains many floating artifacts

However the weakness of this approach is evident when rendering images in which the view is dominated by near-camera solid volumes. The down-weighting of gradients near to the camera causes means that errors in near-camera predictions are not learnt by the network and therefore it fails to reconstruct high frequency colour variations for these scenes. For this reason the model with gradient scaled losses results in an average evaluation PSNR of 21.1 compared to the regular model which achieves 21.5.

In an effort to overcome this issue my implementation pre-computes a depth map before applying the gradient scaling. We then apply the gradient scaling only if the average depth value is above some threshold. Unless stated otherwise, for all of the following experiments in this study we use a threshold value of 0.02, this value was obtained via trail-and-error across multiple datasets. This will disable gradient scaling for poses with many near-camera high density voxels.

	Video 1			Video 2		
Model	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
Nerfacto	29.88	0.80	0.17	19.52	0.46	0.37
Renderer (mine)	31.79	0.86	0.15	21.09	0.62	0.29
Renderer (Mine) + Gradient Scaling	31.35	0.85	0.16	21.12	0.63	0.29
Renderer (Mine) + Depth-Aided-Gradient-Scaling	31.60	0.85	0.26	21.06	0.63	0.29

Table 4.3: **Gradient Scaled Losses Comparison Table:** This table shows the average PSNR and LPIPS values for across all evaluations image for 3 different subsea datasets. It's evident that my model without gradient scaling performs best however the depth-aided gradient scaling approach does mitigate some of the near-camera error effects.

4.6 Post Processing

This subsection describes the algorithms applied to the trained model's output to improve its reconstruction quality. These modifications do not interact with the model's training process since they may introduce colour inaccuracies which would generate erroneous RGB loss.

All NeRF models are fundamentally limited by the underlying hardware, because of this they must make compromises in output quality in order to reduce training times. A key example of this is the samples-per-ray hyper-parameter, this determines how many samples are distributed along each ray [33]. The more samples we take the better our reconstruction quality. However for each ray sample we must perform a forward pass of each of the neural networks, thus significantly increasing rendering times. This reduction in sample density results in a reduction in ability to represent high frequency colour variations; leading to blurry regions in the place of high frequency details. Note that ray sample density comprises one of many performance-reconstruction accuracy compromises. Another source of blur is intrinsic to the NeRF model's architecture. Specifically as outlined in the original NeRF paper [21] multi-layer perceptrons typically struggle to accurately learn high frequency functions [16], this effect is known as the spectral bias. As explained in section 2.4.3 spectral bias is generally mitigated by utilising a coordinate encoder, however Rahaman et. al. [27] demonstrates that the bias is never fully eliminated.

Figure 4.11: **Left:** Dataset evaluation frame **Right:** Corresponding generated frame, blur is clearly visible along the edges of objects, such as the watch in the top left as well as the detailing on the surface of the coral.

We can recover some of this detail by applying a post-process sharpen to each generated image. Specifically we can utilise an unsharp mask. An unsharp mask works by extracting the low pass component of the input image using a Gaussian blur convolution [6]. Subtracting the low pass component from the original image yields the high pass signal, this encodes the high frequency detail in the image. Finally we can add the re-scaled high pass signal to the original image which will sharpen the image by enhancing the edges. Scaling the high pass signal by some value k.

$$f_{\text{smooth}}(x,y) = \frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix} * f(x,y) \tag{4.10}$$

$$f_{\text{sharp}}(x,y) = f(x,y) + k(f(x,y) - f_{\text{smooth}}(x,y)) \tag{4.11}$$

One issue with this method is that it will result in a loss of colour accuracy around edge pixels, introducing an effect known as halo artifacts where the subtracted blurred image subtracts vibrant edge pixels from the background colour. This leads the pixel colour to overshoot the target colour. We can reduce this effect by producing a sharpened image and blending it with the original image. For this reason it's also important to fine tune the amount of sharpening we perform, specified by k. Since this method is directly amplifying the high frequency detailing in the image it will result in colour inaccuracy across the image and therefore will reduce the PSNR and SSIM scores of the reconstruction. This is because colour accuracy is the basis for both of these evaluation methods in the form of Mean Squared Error.

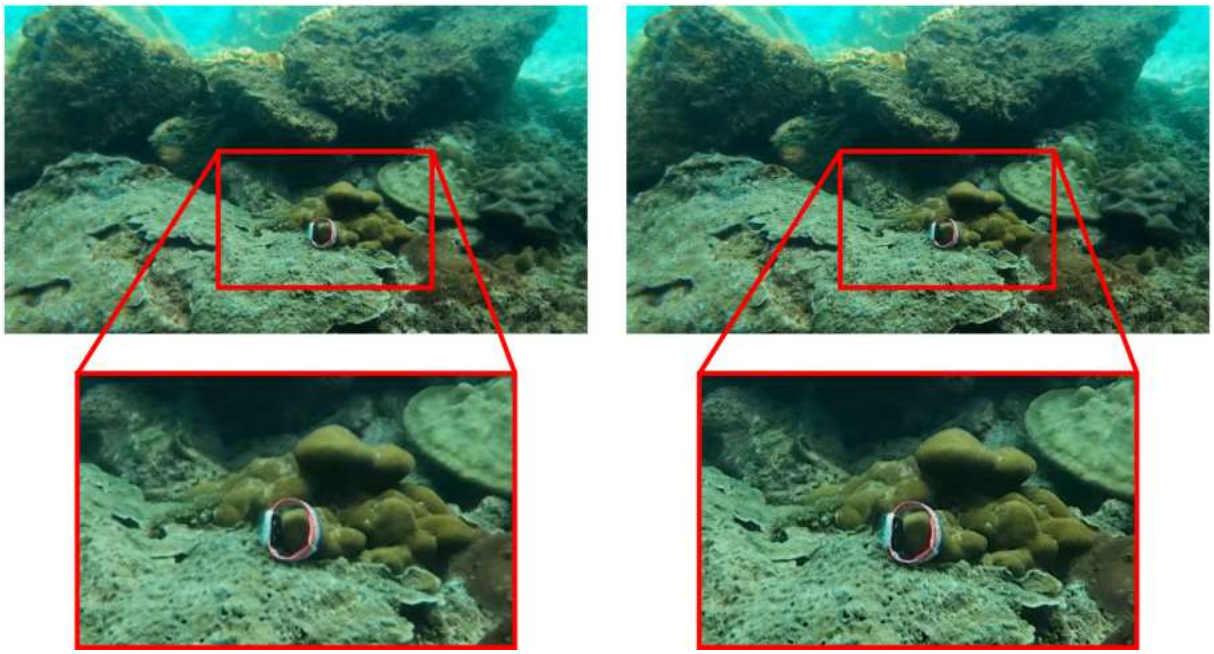


Figure 4.12: **Left:** Evaluation frame generated by the NeRF model before the unsharp masking post processing step. **Right:** The same evaluation frame after applying the unsharp mask with a k value of 1.5. It's clear that the edges along the watch and pieces of coral are enhanced, additionally the features on the surface of the coral are much better defined.

4.7 Contribution Summary

- As a part of the dataset preparation process our model attempts to eliminate redundant frames from the training process by analysing the frames in order and comparing neighbouring frames similarity. In order to support this process we provide a tool which estimates the projected dataset reduction percentage for a given similarity threshold.
- The salient contribution of this project is the renderer architecture that leverages learned depth information to re-weight the sampled density values (as shown in figure 4.5). We further optimise this approach by rectifying the output if the measured depth is past the maximum length of each ray.
- Additionally this project implements the robust losses algorithm first proposed by RobustNeRF [29]. Similarly we integrate a loss gradient scaling described in [25] which compensates for the intrinsic sampling bias for near-camera volumes, generated by the inverse square relation between ray depth and sample separation.
- Finally we attach a post-processing step with the goal of recovering the detail lost due to NeRF's limited ability to represent high frequency detailing.

Chapter 5

Critical Evaluation

5.1 Evaluation of Key Decisions

This section presents an analysis of key decisions made throughout the project including the evaluation of alternative options in comparison to the selected approach. In some cases this includes an experimental analysis of alternative methods supported by a review of the theoretical material.

5.1.1 Base Model

The base model forms a foundation on top of which we can build our contributions, it was particularly important to assess the viable candidates as this would significantly affect the implementation and performance of future features. I opted to use Nerfstudio's [35] nerfacto model [24]. This is because nerfacto implements several existing state-of-the-art NeRF optimisation techniques including but not limited to: multi-resolution hash encoder 2.4.3, course and fine sample proposal networks 2.4.3 and utilises Nvidia's Fully Fused MLP [23]. Nerfstudio itself provides a modular framework for developing a fully customisable NeRF model whilst exposing an API to standard components.

5.1.2 Alternatives

Gaussian Splatting

Gaussian Splatting is a method developed in 2023 by Kerbl et. al [14]. This model priorities real-time rendering over reconstruction quality. The method they propose, takes a sparse point cloud representation of the scene as input and represents each point as a three dimensional Gaussian. This set of Gaussian's is equivalent to our radiance field. This implies that instead of optimising the learned radiance of each voxel, they optimise the characteristics of each 3D gaussian (mean, covariance matrix, colour and density). The advantages of this representation become clear during the rendering phase. First and foremost the 3D Gaussian representation is extremely compact compared to the neural radiance field because a Gaussian only exists where some geometry has been identified. This means that rendering requires far fewer samples per pixel. Additionally in the original paper they utilise a fast tile-based rendering approach, culling low confidence Gaussians before sorting them in depth order. Transmitted colour can then be accumulated as usual.

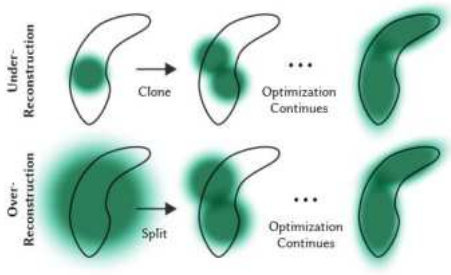


Figure 5.1: **3D Gaussian Optimisation:** This diagram illustrates how the 3D gaussians, initialised at the points described by the spare point cloud, are optimised to fit the geometry in the scene. Credit: Kerbl et. al [14]

We opted not to use Gaussian splatting for several reasons, the first of which is that Gaussian splatting typically achieves an equivalent or lower reconstruction quality than the current state-of-the-art NeRF techniques. Additionally rendering speed is not a priority for this project as we aim to maximise reconstruction quality. The other major drawback of Gaussian splatting it has much greater VRAM requires, usually greater than 20GB.

Dynamic NeRF (K-Planes)

Nerfacto assumes that the background is mostly static. We can expect some subsea datasets to include dynamic background objects such as seaweed. Our rendering model could be adapted for a Dynamic NeRF architecture however in practice DNeRF does not achieve the same reconstruction quality as static NeRF. An ideal model would be capable of accurately representing dynamic objects at arbitrary time steps. Dynamic Nerf methods such as K-Planes [4] and DNerf [26] propose algorithms capable of this. K-Planes works by representing the voxel field as 6 intersecting orthogonal planes (three for space (xy, xz, yz) and three for spatial-temporal variation (xt, yt, zt)). In reality, for non-synthetic data, static methods are able to achieve a much higher PSNR and require less memory. For this reason we opted to optimise a static NeRF method to remove the low-density dynamic objects, this is suitable for most of our subsea datasets as in general the dynamic objects are relatively small. Figure 5.2 shows an image generated by 'K-Planes' and it's corresponding image in the dataset. The K-Planes image is visibly much blurrier. Quantitatively speaking, K-Planes averaged a PSNR of 22.5 across 3 separate subsea datasets. In contrast nerfacto [24] averages a PSNR of 27.4. This is consistent with the results established in [4] where K-Planes achieves a lower reconstruction quality than Instant-NGP [23] for scenes mostly containing static volumes.

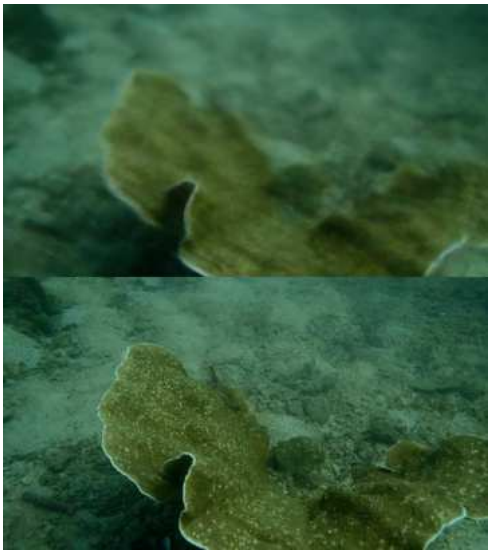


Figure 5.2: **K-Planes Evaluation:** The top image was generated by the K-Planes model after 30,000 iteration steps. It's evident that the model fails to represent high frequency details.

5.2 Testing and Analysis

This section describes the testing process and presents a detailed analysis of the resulting data, evaluating the model's strengths and weaknesses. For all of the following experiments we use the default nerfacto model as our baseline.

5.2.1 Convergence

In this section we will evaluate the model's ability to converge on an accurate reconstruction for the evaluation and training datasets. We will analyse the affect of specific modifications to the MLP architecture. We will utilise PSNR, SSIM, LPIPS and $\mathcal{L}2$ loss as metrics, measuring the model's convergence. Each model is trained on a Nvidia GTX 3090 with 24GB of VRAM, for 100,000 epochs.

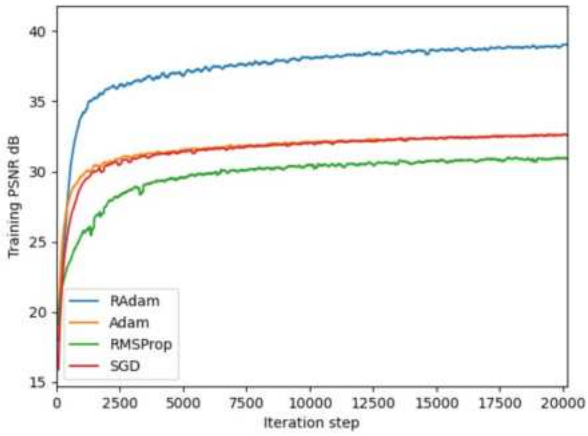
Stochastic Optimiser

The stochastic optimiser defines how the model updates its weights, the primary goal of the optimizer is to modify the weight updates in order to minimise loss. Each of the four stochastic optimisers tested in this section are explained in the technical background section for MLPs 2.2.1. In theory the optimiser can influence both convergence speed and overall model accuracy (every stable model's training reconstruction accuracy will plateau after enough iterations). Table 5.1 shows the results of our experimentation with the four stochastic optimisers. As expected RAdam performs the best for both video datasets, this is generally because of RAdam's rectified learning rate, reducing the variance of the learning rate in the early stages of training. This prevents the model from overshooting the optimal solution. Another interesting finding is that for video 2, SGD (no optimiser) fails to generate accurate reconstructions. Specifically it generates very low contrast blurry images. Indicating that the model requires a more complex optimiser to help it escape from local optima. This result is consistent with Schmidt et. al [30] (Deep-ML optimiser analysis) whose experiments show that SGD typically performs poorly at generative tasks (compared to classification/Prediction). In the case of NeRF this is because SGD has no perception of the 'trend' of the gradient and therefore can easily settle in local minima that RMSProp or Adam would otherwise 'skip' over. This analysis is reinforced by the graphs in figures 5.3 a and b, which show the the PSNR of the reconstructed images for the first 20,000 iterations. It's evident that RAdam takes longer to fully plateau

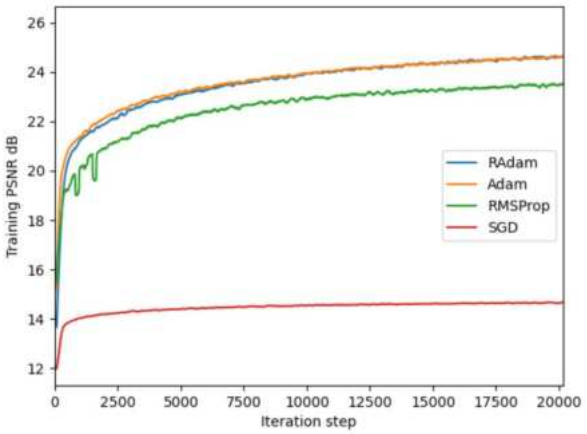
than the other optimisers this is due to its warmup phase. It's also clear that RMSProp consistently performs worse than Adam and RAdam, this is because it doesn't have a momentum component and therefore can erroneously get stuck in local optima.

	Video 1			Video 2		
Optimiser	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
SGD	31.55	0.86	0.14	13.92	0.23	0.90
RMSProp	30.65	0.84	0.16	21.45	0.63	0.28
Adam	31.57	0.86	0.14	21.24	0.62	0.28
RAdam	32.36	0.87	0.11	21.28	0.62	0.28

Table 5.1: **Stochastic Optimiser Experimentation:** This table shows the average PSNR, SSIM and LPIPS across the evaluation images after training with four different stochastic optimisers on 2 different scenes. Each model was run for 100,000 iterations. It's clear that RMSProp, Adam and RAdam all perform within ±2dB PSNR. However RAdam consistently performs the best.

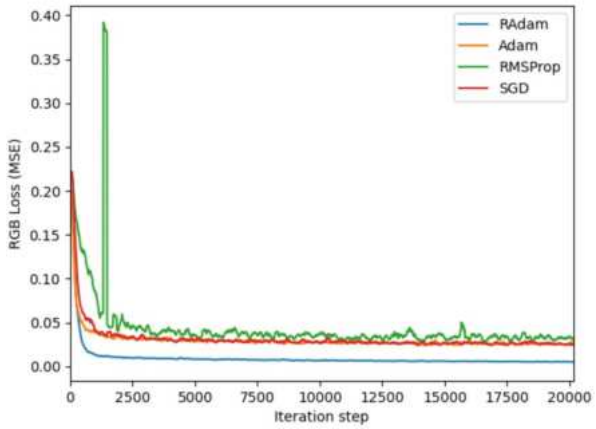


(a) **Video 1 training:** RAdam achieves a significantly higher PSNR, however it does take much longer to plateau than the other three optimisers



(b) **Video 2 training:** SGD (No optimiser) converges on a local minima, however it fails to produce accurate reconstructions.

Figure 5.3: **Video 1 and 2 training PSNR plots:** The left and right graphs show the training image reconstruction PSNR over time for video 1 and video 2 respectively. Both graphs plot PSNR for 20,000 iteration steps. RAdam generally achieves the highest reconstruction accuracy conversely SGD can perform very poorly depending on the scene.



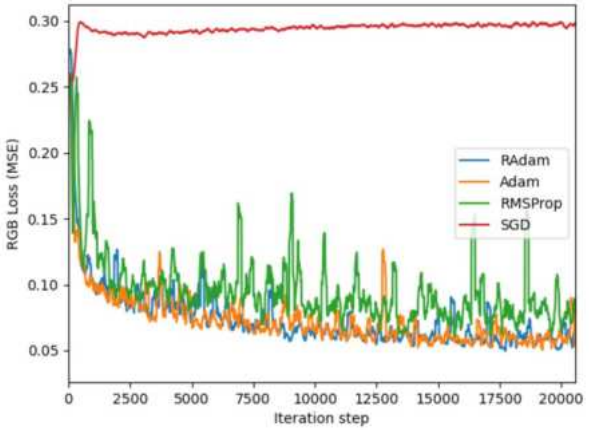
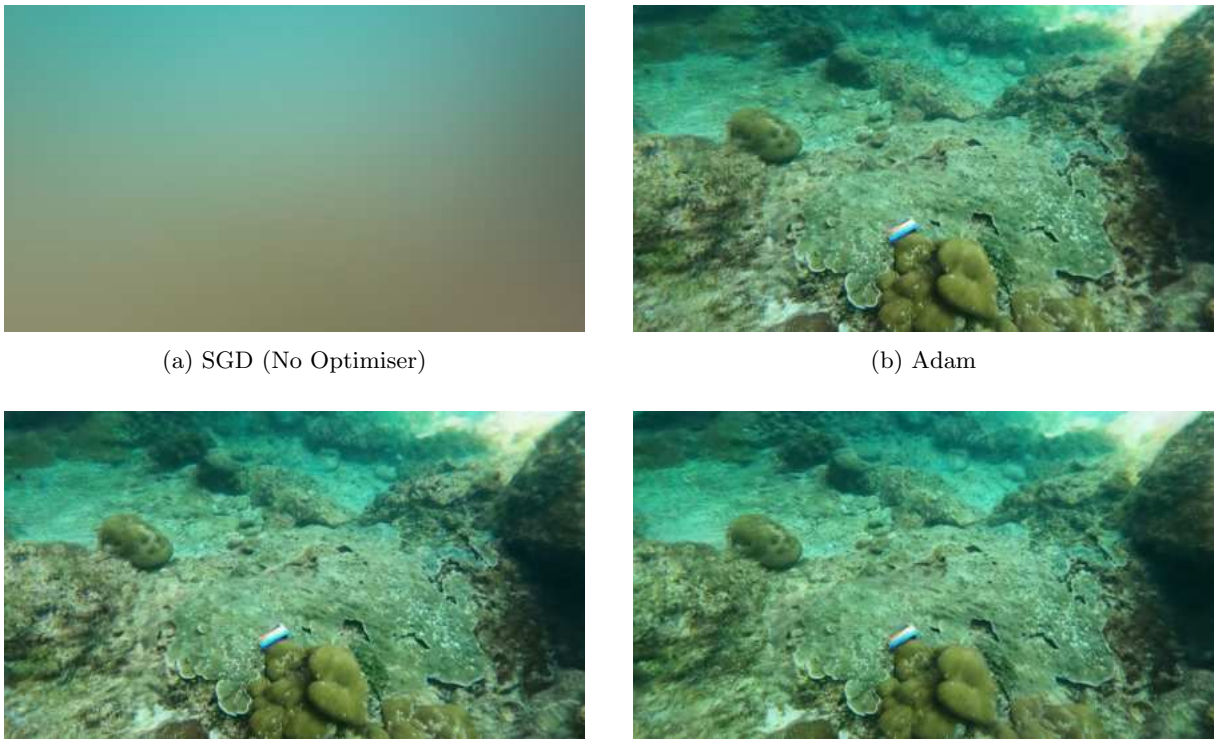


Figure 5.4: **RGB Mean Squared Error Loss for video 1 and 2:** The Left and right graphs shows the RGB $\mathcal{L}2$ loss for videos 1 and 2 respectively. Both graphs plot MSE loss for 20,000 iteration steps.

Another interesting result from these experiments is that the training PSNR plateau for RAdam is around 6dB less than its evaluation PSNR performance for video 1. This indicates that for this dataset RAdam is overfitting to the training partition which suggest that we may be able to achieve a higher evaluation PSNR using a $\mathcal{L}2$ regulariser.

Figure 5.4 shows the MSE RGB Loss curves for the first 20,000 training epochs. A key feature of these graphs is the noise exhibited by each loss curve, crucially RMSProp is much nosier than the rest. This is because it doesn't have a notion of 'momentum' and therefore can 'bounce' around the solution space. With this assumption in mind SGD should also demonstrate this noisy loss curve behaviour, however it does not. This occurs because it can easily sit in areas with very small local gradients since it doesn't have the adaptive learning rates of RMSProp.

Qualitative Analysis



(a) SGD (No Optimiser)

(b) Adam

(c) RAdam

(d) RMSProp

Figure 5.5: **Evaluation image reconstruction with 4 different optimisers: (a)** Image reconstruction after training without an optimiser, the model clearly fails to converge on a solution capable of representing any detail. **(b, c)** Adam and RAdam produce very similar results with Adam generating a slightly worse colour accuracy. **(d)** RMSProp results in slightly blurrier details on the surface of the coral.

In summary the optimiser greatly affects the reconstruction performance especially across different datasets. For some complex scenes such as video 2 with several different camera paths, several volumes and lots of high frequency details SGD may even fail to produce a proper reconstruction. Overall RAdam tends to perform the best according to PSNR, SSIM and LPIPS. For some scenes there is little difference between Rectified Adam and standard Adam in terms of reconstruction quality. Rectified Adam can take more epochs to converge due to its warm-up weighting on it's learning rate.

Learning rates

The learning rate hyperparameter for an MLP determines its weight's sensitivity to the estimated loss. Selecting an appropriate learning rate is vital for a particular model's convergence. Large learning rates can lead to instability, too small values may result in the model getting stuck in local optima. Choosing an optimiser with an adaptable learning rate can mitigate some of the effects of selecting a poor learning

rate, however the optimiser will require a base learning rate. The optimiser's scheduler determines how the learning rate modified as a function of the number of time-steps elapsed. The scheduler our model uses incorporates a linear warm-up period which increases the learning rate to a maximum value initially. After this maximum value we transition to an exponential decay function which will slowly anneal the learning rate over the course of the training process.

	Video 1			Video 2		
Learning Rate	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS
0.001	10.69	0.14	0.92	16.31	0.25	0.59
0.01	31.79	0.86	0.15	21.09	0.62	0.29
0.1	25.32	0.83	0.18	19.69	0.62	0.30

Table 5.2: **Learning rate variation:** This table exhibits the results of training the same model with three different base learning rate values. A value of 0.01 performs achieves the highest reconstruction quality for both datasets. Values greater than this proved to produce too much instability and conversely values much smaller than this took much longer to converge.

Early Stopping

A machine learning model will typically overfit to its training data if trained for enough iterations. This introduces a loss of generality during the later stages of training. Generality in the case of Neural Radiance Fields refers to the ability to generate images from poses that are not in the training dataset (Novel View Synthesis). The problem of overfitting in NeRF manifests where a voxel may have different colours from different viewing angles, due to lighting or scene occlusions (note that this doesn't apply to learned density because we assume that it is independent of viewing direction). If we train for too long we may lose colour accuracy for viewing angles not included in the training dataset. We can easily fix this problem by employing an early stop check every N iterations. This consists of computing a performance metric on the evaluation dataset and checking if it has decreased since the previous evaluation. This ensures that if the model starts losing it's generality it can stop training. This approach acts as a form of regularisation. Figure 5.6 demonstrates the loss in generality caused by training for longer than necessary. The PSNR evaluation loses 0.5dB from it's peak value highlighting a loss in reconstruction accuracy over time. In contrast the LPIPS graph (0 is best) is monotonically decreasing.

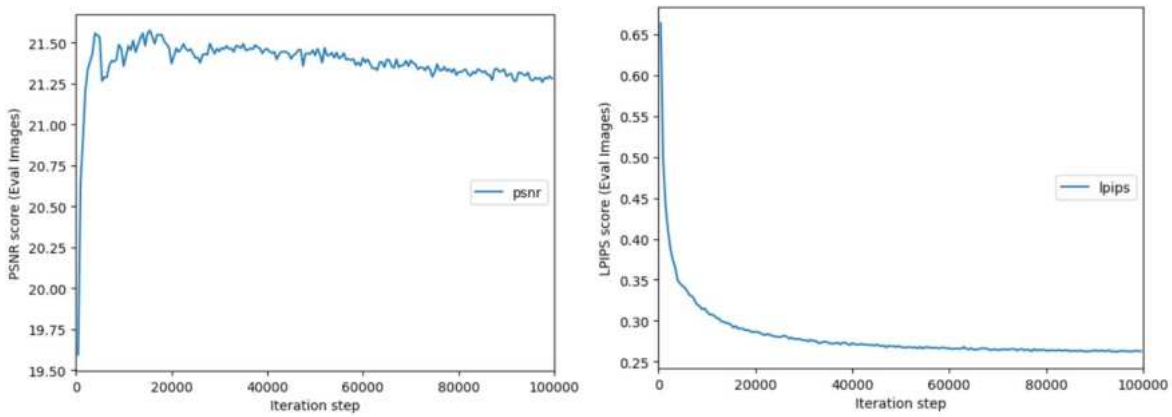


Figure 5.6: **Performance Metrics for the full evaluation dataset:** Both graphs show a performance metric plot evaluated on the entire evaluation dataset every 2000 iteration steps. The PSNR graph (left) shows a clear decrease in generality of around 0.5dB over the course of the training process.

This is because LPIPS measures a perceptual difference where the underlying neural network is able to implicitly capture features such as textures and edges as well as semantic differences. The difference in trend between the two plots indicates that the loss in generality may only be in the form of colour accuracy (measured by PSNR). However, this can still be mitigated by employing early stopping as LPIPS plateaus and PSNR begins decreasing. Additionally in most cases this will reduce training times.

5.2.2 Renderer

This section presents a detailed analysis of the performance of the renderer modifications made in this project. These modifications add several hyper-parameters to the model that require fine tuning depending on the structure and the scale of the scene being modelled. For example the standard deviation (spread) of the distribution, tuning this value depends on the relative scale of the scene. The other major hyper-parameter introduced with the new rendering approach is the base density re-weighting value which determines the minimum weight that the model can apply to a sampled density value. This value scales the effective 'power' of this rendering approach. The graph in figure 4.9 demonstrates the distribution used to re-weight the density samples with different base weight values. Our algorithm clamps the output values between 0 and 1 ensuring that with a base value of 1 each density sample is scaled with a value of 1. Selecting a base value that is too close to zero can result in colour inaccuracies due to erroneous depth estimation. This assumption is reinforced by the results in table 5.3 where a base value of 0.0 leads to a reduced reconstruction quality.

	Video 1			Video 2		
Base Value	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
0.0	31.71	0.86	0.15	20.85	0.61	0.30
0.1	31.75	0.86	0.15	21.29	0.62	0.29
0.2	31.79	0.86	0.15	21.09	0.62	0.29
0.3	31.74	0.86	0.15	20.91	0.61	0.30

Table 5.3: **Density re-weighting base value variation:** This table shows the results of varying the base value used in the density re-weighting distribution.

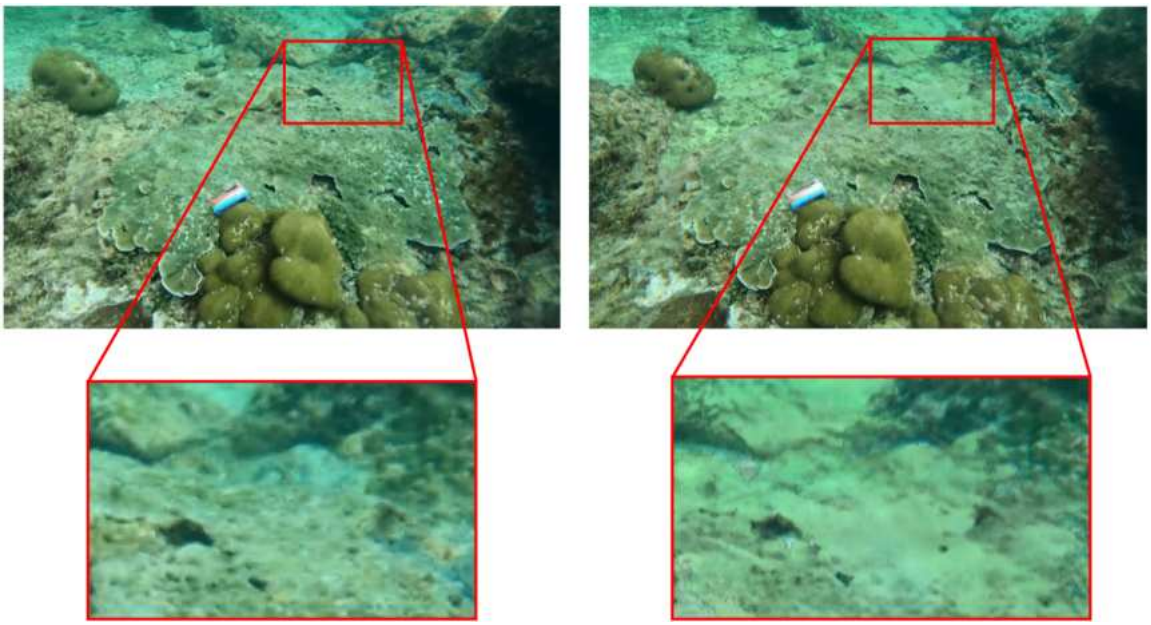


Figure 5.7: **Density re-weighting base value qualitative comparison:** The left image shows an example of a reconstructed image using a renderer with a base value of 0.2. In contrast in the right side image shows a reconstructed image using a renderer base value of 0.0.

This effect can be seen visually in figure 5.7 which shows a comparison of two reconstructed images. The image utilising a renderer with a base value of 0.0 contains much less fine detail in the background. This occurs when the depth estimation point occurs before the surface of the volume and the pixel colour is rendered as the colour of the water instead. We see similar reconstruction artifacts resulting from very small values for the standard deviation of the re-weighting distribution which is consistent with our explanation for how the artifact occurs. This is because with small standard deviation values if the depth estimation is slightly incorrect most of the density weighting is allocated to samples before the front surface of the background volume. Utilising a larger value compensates for this by ensuring that the density is spread across multiple samples. This indicates that one of the major drawbacks of our

approach is that it is very sensitive to its hyper-parameters and it's overall reconstruction quality for a given scene depends on fine tuning these parameters. This process can be very time consuming as each model generally takes 3 hours to train. For the following experiments in this study we use a base value of 0.1.

Comparison with existing methods

This subsection provides a quantitative and qualitative comparison of our density re-weighting rendering method and existing methods such as Nerfstudio's Nerfacto model [24, 35] (the baseline for our implementation) and the SeaThru NeRF model [17]. The SeaThru model provides a renderer that generates images from the trained model capable of removing the colour of the water from the scene, for these comparisons we will use the standard renderer because the water-removal will alter the colour accuracy and 'reduce' the quantitative performance of their model. Table 5.4 shows the average reconstruction quality of the evaluation dataset for two different subsea videos. This data shows that the proposed model and SeaThru NeRF produces reconstructions with a similar colour accuracy (evaluated by PSNR and SSIM), however the proposed model yields a better perceptual similarity score. This is potentially because the proposed renderer is able to remove some of the foreground artifacts, which are similar in colour to the background. This ensures that background structures are more visible and therefore can be recognised by the LPIPS evaluation.

	Video 1			Video 2		
NeRF Model	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
Nerfacto (baseline)	29.88	0.80	0.17	19.52	0.46	0.37
SeaThru NeRF	31.10	0.79	0.25	21.01	0.60	0.29
Ours (Renderer modification only)	31.79	0.86	0.15	21.09	0.62	0.29

Table 5.4: **Renderer qualitative performance:** The renderer modifications improve on reconstruction quality measured with PSNR, SSIM and LPIPS compared to the nerfacto baseline and the SeaThru NeRF model.

Figure 5.8 demonstrates a reconstruction performed by Nerfacto and the same reconstruction performed by the same model using my renderer. Much of the dark artifacts appearing around the edge of the image are removed, this is a result of the re-weighted density which applies a higher weighting to the lighter coloured background voxels.

(a) Nerfacto reconstruction

(b) Mine (Renderer modification) reconstruction

Figure 5.8: **Nerfacto reconstruction comparison:** In this comparison we can observe that the nerfacto reconstruction contains dark artifacts surrounding the edge, these are eliminated by our renderer.

Figure 5.9 demonstrates a similar comparison for a different dataset. The proposed renderer modifications produce sharper, higher contrast, reconstructions than the corresponding Nerfacto reconstruction image. This is particularly noticeable on the surface of the coral features in the foreground as well as the structure of the watch. This occurs because noise resulting from floaters, which generated by foreground samples are down-weighted before these samples are accumulated to produce the pixel colour. The areas of shadow in the proposed model's reconstruction also appear brighter than those in the Nerfacto reconstruction, this is because the areas in shadow are up-weighted by the proposed renderer. Additionally

much of the dark artifacts are eliminated in the proposed renderer's reconstructions similar to those identified in figure 5.8.

(a) Mine (Renderer modification) reconstruction

(b) Nerfacto reconstruction

Figure 5.9: **Video 2 - Nerfacto reconstruction comparison:** The proposed renderer's reconstruction produces an image with sharper details. It is able to more accurately the high frequency colour variations on the surface of the coral.

5.2.3 Loss Function

Robust Losses

This section analyses the effect of the modifications performed to the loss function, specifically the robust loss algorithm 4.5.1 and the gradient scaled losses algorithm 4.5.2. The robust losses algorithm proposes a solution addressing the problem of 'distractors'. In the RobustNeRF paper [29] they label temporal inconsistencies such as dynamic objects as 'distractors'. Their algorithm detects the effect of 'distractors' in the reconstructed images and assigns a higher loss value if they are detected. In their paper they identify a weaknesses of their implementation was an overall lower reconstruction quality on scenes that do not contain distractors. For our datasets which contain several dynamic components such as lighting due to reflections off the surface of the water as well as moving objects like fish, we observe an increase in reconstruction quality compared to the base Nerfacto model. The main disadvantage is the robust loss function is particularly sensitive to its various threshold hyper parameters. For example, for video 2, modifying the blurred image threshold by 0.1 varied the average evaluation PSNR by 4dB. For each of our experiments we use the best performing value for these hyper-parameters to ascertain an upper bound on reconstruction quality for each dataset.

	Video 1			Video 2		
NeRF Model	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
Nerfacto (baseline)	29.88	0.80	0.17	19.52	0.46	0.37
nerfacto + robust losses	31.32	0.52	0.23	22.06	0.66	0.27

Table 5.5: **Robust Losses results:** This table shows a comparison between the Nerfacto model performance and the Nerfacto model with the robust loss algorithm. The robust loss function generally improves reconstruction quality.

The robust loss algorithm is very effective at eliminating distractors without leaving behind residual colour this can be seen in figure 5.10. The two fish on the left side of the image are removed in the reconstructed version without affecting the colour accuracy of the background static structures. This is due to RobustNeRF's ability to detect observed-low-density regions by thresholding the smoothed $\mathcal{L}2$ loss. This therefore detects contiguous regions of increased MSE loss, which describes a patch of colour inaccuracy consistent with a low density object in the foreground.

(a) Evaluation frame (b) RobustNeRF (c) Proposed model

Figure 5.10: **Robust losses fish removal:** The center image is a reconstructed generated by a model trained with the Robust Losses algorithm. The left image is the corresponding dataset image. It's evident that the reconstructed version does not contain any of the fish. Additionally, the right image is the same reconstruction, generated by the proposed model.

Loss Gradient Scaling

The loss gradient scaling algorithm proposed in the paper 'Floaters No More: Radiance Field Gradient Scaling for Improved Near-Camera Training' [25] aims to address the intrinsic bias towards samples collected from volumes closer to the camera. It does this by up-weighting the loss gradients produced from samples closer to the camera. This integrates with our depth-based re-weighting by the scaling the the loss gradient for each sample by a value proportional to it's distance along the ray. The salient issue with this approach is clear when reconstructing images with near-camera volumes, the loss gradient scaling leads to a noticeable reduction in sharpness. This effect can be observed in figure 5.11, in which the coral formation in the centre of the image retains considerably more detail in the nerfacto baseline reconstruction. To address this issue, we adjust the gradient scaling by activating it only when the average depth of the target reconstruction exceeds a certain threshold. The result of this modification can be seen in figure (5.11 (c)). We refer to this modification as DLGS (Depth Loss Gradient Scaling).

(a) Nerfacto + Loss Gradient Scaling (b) Nerfacto baseline (c) Nerfacto + depth rectified loss gradient scaling (d) Evaluation dataset frame - Video 2

Figure 5.11: **Loss Gradient Scaling on close-up frames:** The three images demonstrate the effect of different loss gradient scaling techniques on the same evaluation frame.

5.3 Final Model Performance

5.3.1 Performance

This section provides an ablative analysis of the model's various components in order to evaluate if they integrate with each-other effectively. We perform this study on 4 separate subsea datasets, the 'shipwreck' datasets particularly highlights the weaknesses of the proposed model; this is discussed in depth in section 5.3.3. For all evaluation datasets we remove frames containing dynamic objects such as fish, this is because our renderer attempts to remove these objects which reduces the colour accuracy. For this reason table 5.6 demonstrates the static reconstruction quality. Additionally each evaluation is performed before the post-process sharpen this is because this process naturally reduces reconstruction accuracy, however it does increase the quality of each reconstruction perceptually.

It's evident from table 5.6 that the increase in reconstruction quality resultant from the proposed renderer has a large variance depending on the dataset. The proposed renderer tends to achieve a lower reconstruction quality on scenes with a large variation in distance between different volumes. Video 2 for example contains many volumes close to the camera as well as large regions with no volumes. Areas without volumes lead to an increase in noise in the reconstruction since the median absorption point occurs in the water. The proposed model attempts to mitigate this effect by thresholding the accumulated density as an indicator for whether the re-weighting should be applied. However, this does introduce the threshold hyper-parameters which require tuning to achieve the best performance. Additionally the proposed method typically had much larger training times than the baseline Nerfacto model. For a dataset of 1000 images with resolution 1920x1080 training for 100,000 iterations the baseline model finished in 2 hours, in contrast the proposed model (Renderer + Robust Loss + Gradient Scaled Loss) finished training after 4 hours. This increase in training times is caused by a number of factors, the most significant being the robust losses algorithm which increased the training times by around 40 minutes. This is because for every training image a convolution and iteration through a grid of image patches occurs, adding significant overhead. An interesting result of the shipwreck dataset demonstrates that analytical performance measures (PSNR and SSIM) do not accurately model the human perceptual difference between two images. The model achieves modest but not poor PSNR and SSIM scores for the shipwreck dataset, however its LPIPS score reports a much worse reconstruction quality.

	Video 1			Video 2		
Model Modifications	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
Nerfacto (baseline)	29.88	0.80	0.17	19.52	0.46	0.37
SeaThru NeRF	31.10	0.79	0.25	21.01	0.60	0.29
RobustNeRF	30.60	0.86	0.15	21.32	0.64	0.28
DLGS	31.74	0.87	0.16	21.49	0.63	0.28
Renderer (mine)	31.79	0.86	0.15	21.09	0.62	0.29
Renderer (mine) + Robust Loss	30.41	0.86	0.16	20.90	0.61	0.29
Renderer (mine) + DLGS	31.35	0.85	0.16	21.12	0.63	0.29
Renderer (mine) + Robust Loss + DLGS	30.38	0.86	0.16	20.82	0.60	0.30
	Video 3			Shipwreck - 70% reduction		
Nerfacto (baseline)	20.29	0.68	0.27	16.78	0.63	0.67
SeaThru NeRF	21.89	0.72	0.25	19.01	0.70	0.59
RobustNeRF	21.74	0.71	0.26	18.12	0.68	0.64
DLGS	21.98	0.71	0.25	17.54	0.66	0.60
Renderer (mine)	22.07	0.71	0.25	18.50	0.68	0.64
Renderer (mine) + Robust Loss	21.69	0.72	0.25	18.13	0.67	0.63
Renderer (mine) + DLGS	21.99	0.72	0.26	17.31	0.66	0.61
Renderer (mine) + Robust Loss + DLGS	21.65	0.72	0.26	17.36	0.66	0.63

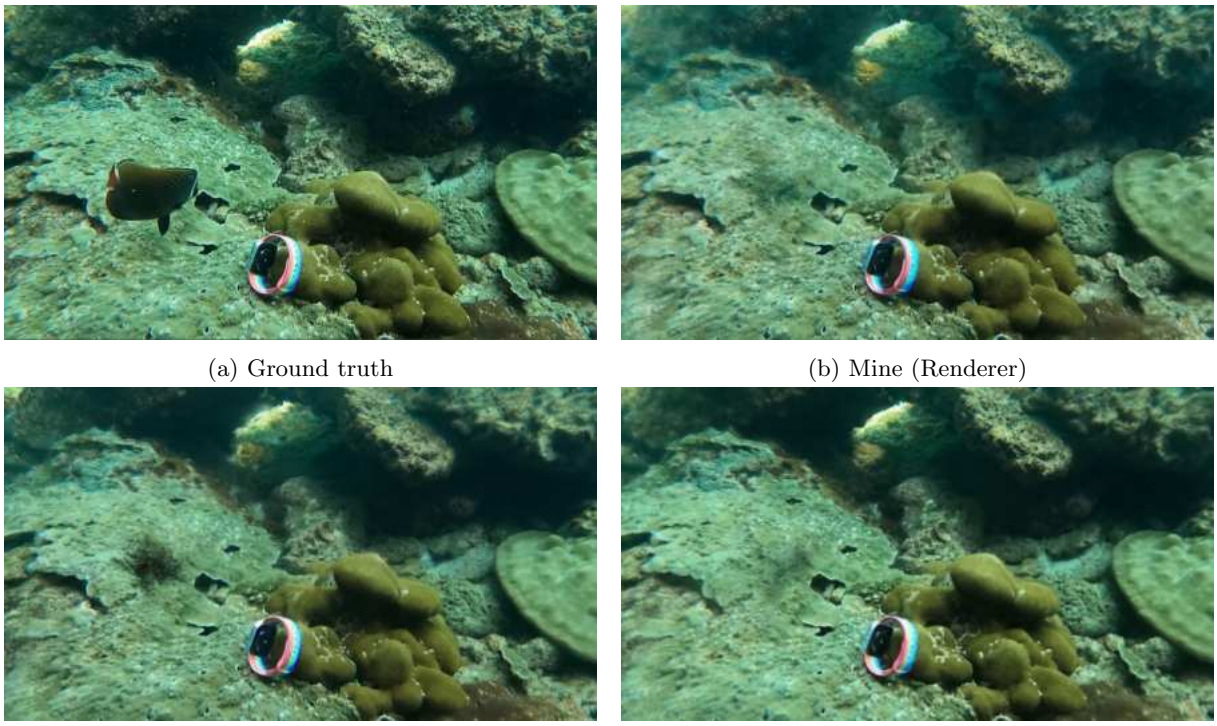
Table 5.6: **Ablative Model Performance:** This table shows the performance scores (PSNR, SSIM and LPIPS) for different iterations of the model in order to identify the effect of each component on the overall reconstruction quality. Images containing dynamic objects (fish) were omitted from the evaluation datasets, preventing the low-density object removal from influencing the reconstruction accuracy metrics.

Table 5.6 demonstrates that on average the proposed renderer and the gradient scaled losses model are able to achieve the highest quality reconstructions, combining these models with the robust loss algorithm tends to reduce the reconstruction quality this is caused by the algorithm mis-classifying static

regions as outliers, this classification is regulated by its various hyper-parameters, however these proved particularly difficult to tune. For this reason our final model omits this component. The increase in reconstruction quality observed for the proposed renderer is due to the fact that it is able to mitigate the effects of scattering caused by the diffuse medium by down-weighting the samples acquired from inside that medium.

5.3.2 Qualitative Performance

This section demonstrates the qualitative performance of the different models, specifically we can analyse each model's ability to remove dynamic objects from the evaluation images.



(a) Ground truth

(b) Mine (Renderer)

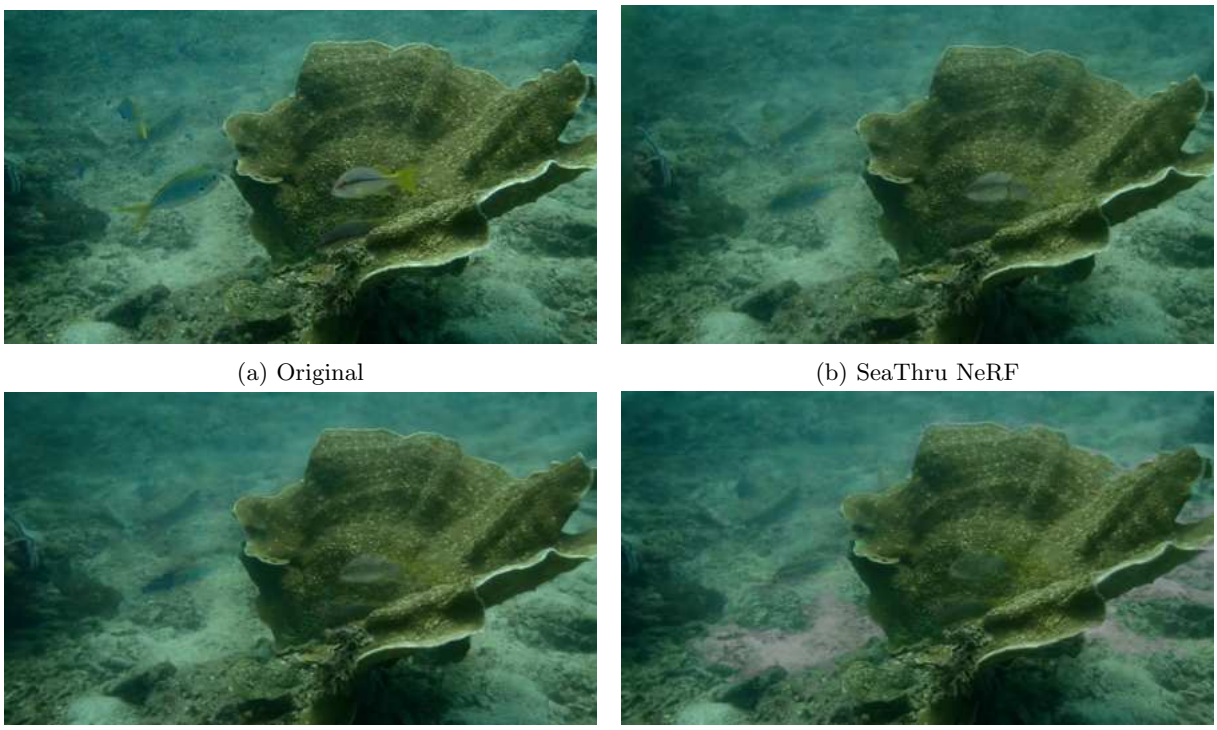
(c) Nerfacto

(d) Robust Losses

Figure 5.12: **Fish Removal Comparison (Video 2):** This comparison demonstrates the removal of the fish on the left hand side of the evaluation frame, while each model has some dark artifacting in the place of the fish, the proposed renderer (mine) and the robust losses reconstructions produce noticeably less artifacts.

Additionally we observe that the model's ability to remove dynamic objects is highly dependant on the value we select for the standard deviation of the Gaussian that we use to re-weight the density observations. This effect can be identified in figure 5.13 where the model using a standard deviation of 0.1 is able to completely remove the dynamic objects including the fish in the center however it results in a loss of colour accuracy on the sea bed. This occurs because the volumes in close proximity (fish and coral) are more accurately segmented with a higher frequency function. However this exacerbates the effect of errors in depth estimation, which in turn centres the Gaussian either in front or behind the front face of the target volume. We therefore lose colour accuracy. A future extension to this project could include a method to optimise the standard deviation for each ray, rather than assuming a single distribution will represent every ray.

Figure 5.14 highlights the differences in reconstruction accuracy between the proposed model, RobustNeRF and the Nerfacto baseline. Nerfacto retains the best colour accuracy in comparison to the original dataset frame. The proposed model tends to produce more vibrant colours particularly in the low-light background regions, this is because it constrains the transmittance at the surface volume, reducing the effects of diffuse transmittance caused by the water in between. RobustNeRF produces the worst reconstruction accuracy potentially due to erroneously detecting 'floaters' and therefore down-weighting the RGB loss in these regions.



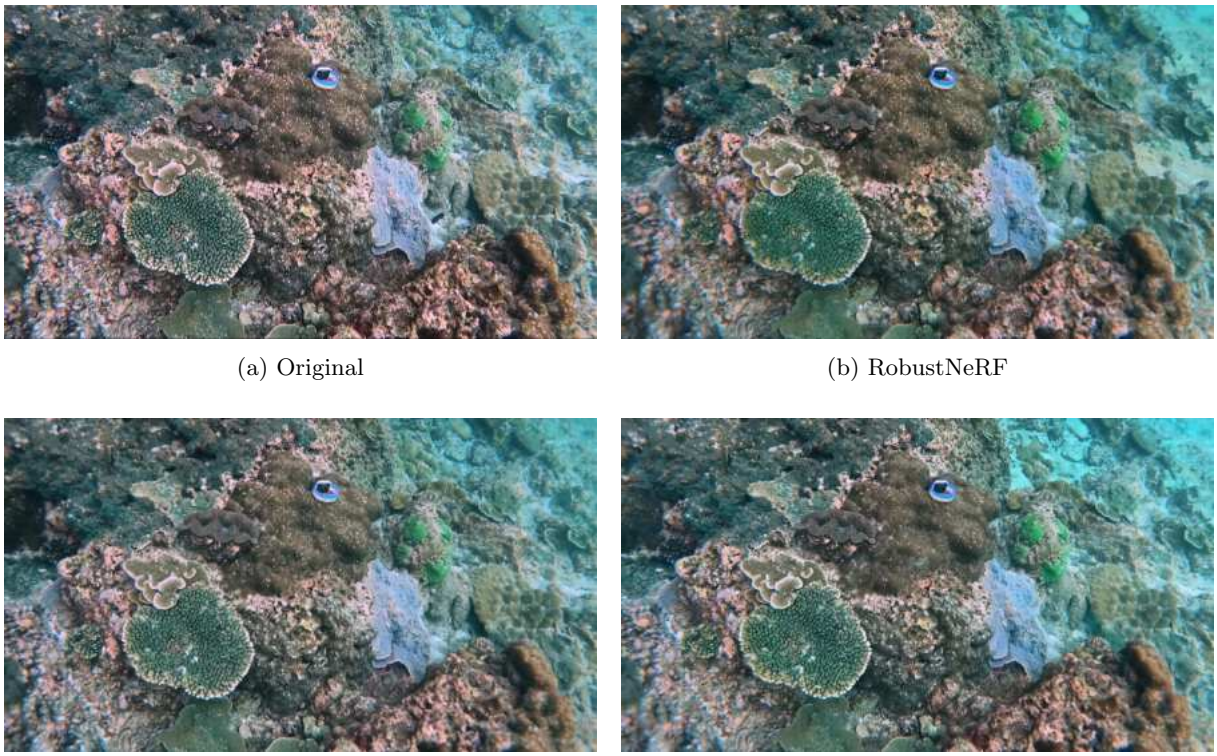
(a) Original

(b) SeaThru NeRF

(c) Mine (std= 0.3)

(d) Mine (std= 0.1)

Figure 5.13: **Fish Removal Comparison (Video 1):** This grid shows a comparison of the fish removal of the central fish. SeaThru NeRF retains the most colour from the fish. The proposed method is able to remove the fish completely at the cost of a reduction in colour accuracy.



(a) Original

(b) RobustNeRF

(c) Nerfacto

(d) Mine

Figure 5.14: **Reconstruction accuracy comparison (Video 3):** This comparison demonstrates the reconstruction accuracy for a scene with many high frequency details.

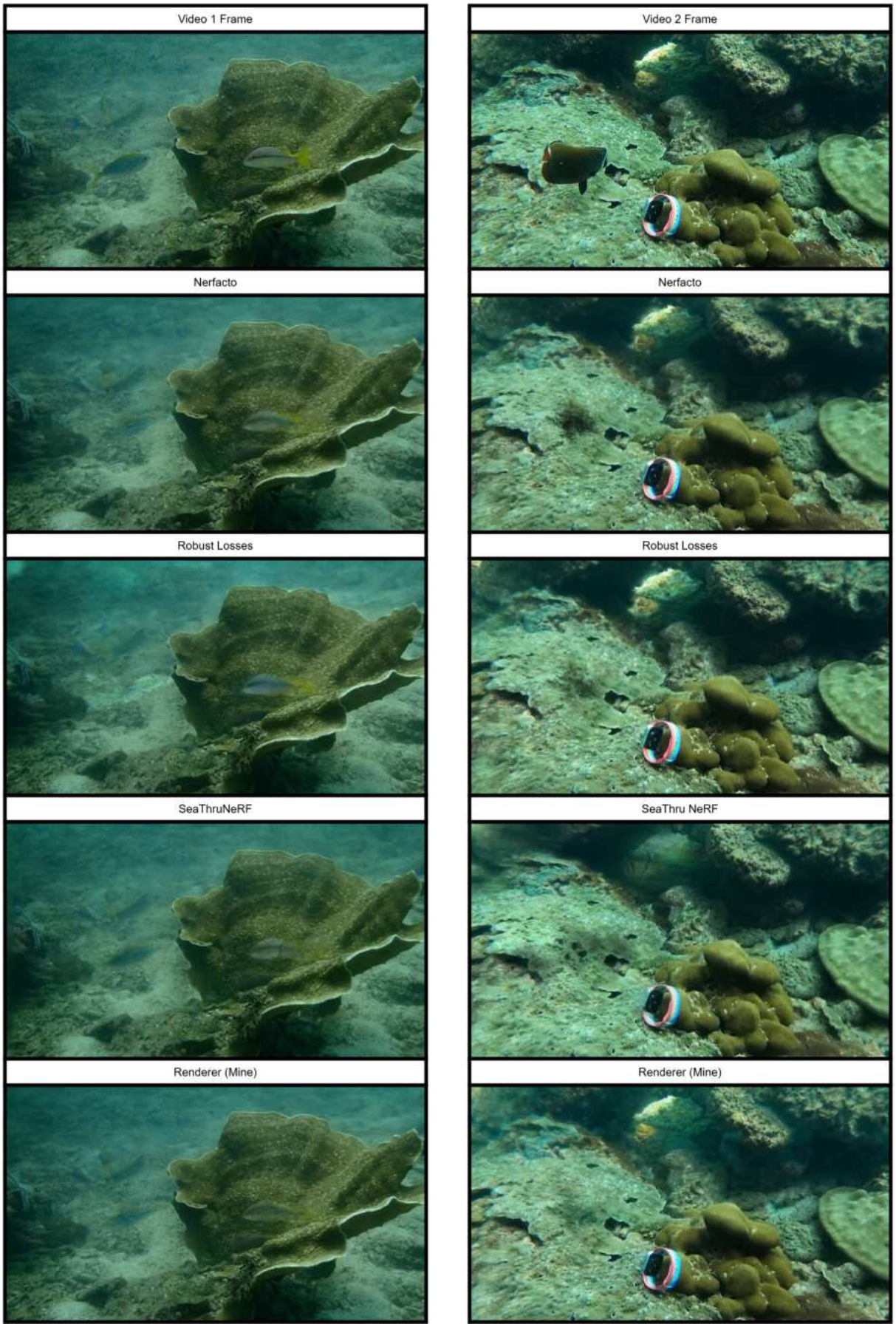


Figure 5.15: **Fish removal all models:** Removal of dynamic objects with 4 separate models on two different videos.

Figure 5.15 demonstrates a comparison of the model's ability to remove small dynamic objects compared with the existing methods explored in this study. The proposed model is able to eliminate most of the colour attributed to the fish in each scene. In both cases some colour is projected onto the surface in the background. Nerfacto, SeaThru and RobustNeRF all assign a larger density to the fish in the foreground and therefore the colour contributed by the fish is more prevalent in their reconstructions.

Post Process Sharpen

Quantitative analysis shows that the unmask sharpen generally resulted in a lower PSNR and SSIM scores, this is evident from table 5.7. This demonstrates how the sharpen reduces the colour accuracy of the target reconstructions. The LPIPS score generally remains consistent because most of the structural and contextual information in the image remains the same before and after the sharpen. This effect is further demonstrated in image 5.16, in which we can observe the squared difference between the sharpened and non-sharpened reconstructions. It's clear that the largest areas of difference occur in the high frequency detailing on the surface of the coral/rock formations. This leads to a reduction in reconstruction quality due to an over-saturation of the colours along edges. Additionally this process can amplify any higher frequency noise in the image.

	Video 1			Video 2		
Scalar	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
1.0	30.87	0.85	0.15	20.65	0.60	0.29
1.5	29.64	0.83	0.15	20.21	0.59	0.29
2.0	28.31	0.82	0.16	19.98	0.57	0.31

Table 5.7: **Post Process Sharpen Performance:** Quantitative performance of the post process sharpen on two different datasets. The 'scalar' value scales the quantity of high frequency signal that is added back to the image.

(a) Dataset evaluation frame

(b) Proposed model reconstruction

(c) Sharpened reconstruction

(d) Squared difference of sharpening

Figure 5.16: **Post Process Sharpen Qualitative Performance:** This grid of images demonstrates an example of the post process sharpen applied to a generated evaluation frame. Additionally, we highlight the squared difference between the sharpened and non-sharpened reconstructions.

Dynamic Object Removal - Non-subsea

The salient unique feature of the proposed rendering approach is its ability to reduce the contribution of small dynamic objects in the target reconstructions. This feature has potential applications for all scenes, not exclusively subsea datasets. This is demonstrated in figure 5.17. In the dataset video, the spool of wire is initially rolling horizontally across the table. Since this is a much larger object than the fish our model is only able to partially remove parts of the spool compared to the nerfacto reconstruction. This is because a large object moving at the same speed as smaller object will contribute a density to a given voxel for an increased number of frames (which corresponds to a longer amount of time). This will result in the MLP encoding a higher density for this set of voxels as the model observes those voxels containing a volume for a longer duration of time. This causes the depth to be estimated before the background volume as the model accumulates an increased amount of density before reaching this volume.

(a) Dataset evaluation frame

(b) Nerfacto reconstruction

(c) Proposed model reconstruction

Figure 5.17: **Non-subsea dynamic object removal:** These images highlight the model's ability to reduce the contribution of dynamic objects on the reconstructed images. In the dataset video the green wire spool is rolling horizontally across the table. In the nerfacto reconstruction we can observe a large low density structure in the place of the wire spool. In the proposed model reconstruction the spool is still visible however in some areas we can observe the table behind.

Figure 5.17 indicates that the proposed model is much more effective at mitigating the influence of smaller, faster-moving dynamic objects. A potential approach for increasing the efficacy of the model could include modifying the proportion of density accumulation used to perform depth estimation. However we can also observe that the benefits of this rendering approach are not specific to subsea datasets and that it has potential applications in all scenes with dynamic objects.

5.3.3 Model Weaknesses

Dataset imperfections

In order to accurately reconstruct images of the target scene the model must be trained on a suitable dataset, there are many reasons why a dataset may lead to an imperfect 3D representation. A major issue for any NeRF model becomes apparent when casting rays from each frame, specifically the geometry of the camera lens causes a deviation from the rectilinear projection assumption. This generally occurs as a radial distortion (figure 5.18) where the image plane is projected onto the surface of a sphere.

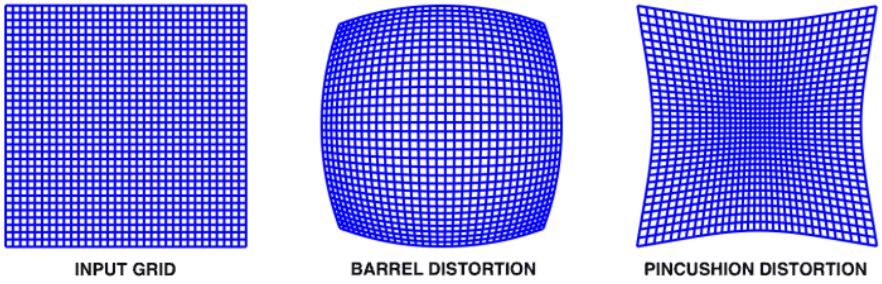
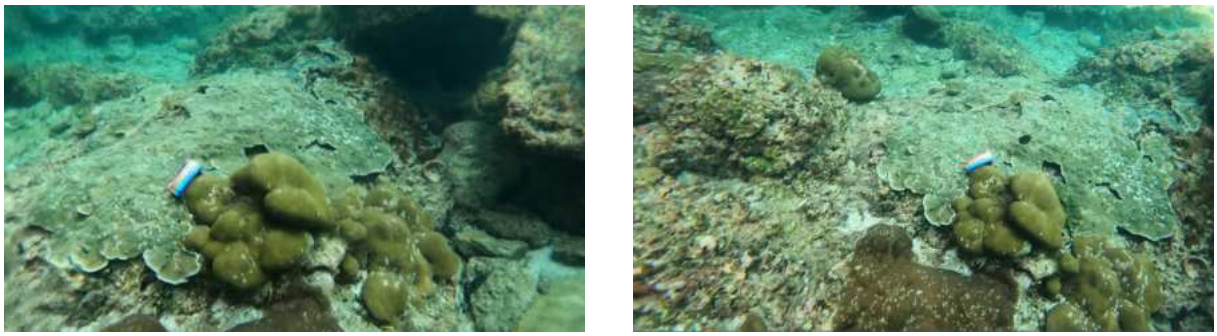


Figure 5.18: **Lens Distortion Diagram:** Image plane radial distortions caused by the lens geometry. Credit: https://learnopencv.com/understanding-lens-distortion/

If we know the intrinsic parameters of the camera used to acquire the video dataset then we can easily account for the lens distortion when casting our own rays. However, none of our datasets retain their original metadata and therefore we cannot determine the lens distortion parameters. Recent advancements in Deep Convectional Neural Networks has enabled machine learning models to estimate [19] the camera's intrinsic parameters. Future iterations of this project could potentially learn these parameters as a part of the existing machine learning pipeline, this would inevitably increase training times due to an increase in model complexity but eliminate the need to guess the camera's intrinsic parameters. Figure 5.18 demonstrates the effect of the lens distortion on the reconstruction, the colours at the edge of the frame are effectively stretched outward due to the adjacent rays acquiring the same colours because the field was trained using frames with a different lens distortion.



(a) Lens Distortion (b) Motion Blur

Figure 5.19: **Dataset Imperfection: (a)** Lens distortions blur the image around the edge of the reconstructed image. **(b)** Motion blur greatly affects the areas of higher frequency colour variations such as the surface of the coral structures.

Another potential issue with the input video is motion blur. Since the camera is generally moving during the time it takes to expose a frame, the captured image is blurred in the direction of travel. If a dataset does contain motion blurred images, these blurry effects will appear in their corresponding reconstructions and potentially in reconstructions from similar poses. Additionally, if a given voxel is motion blurred in one frame and not in other frames the field may learn an erroneous view dependant colour for that voxel. The effects of this can be mitigated by applying a pre-processing step to deblur the video such as a Wiener deconvolution. These methods typically require a model for how the video was

initially blurred. For example, consider a situation where either the camera is in motion while the scene remains static, or conversely, where the camera is stationary while the scene is dynamic. The motion blur can be estimated with a simple linear point spread function, describing the optical flow of each pixel.

Partial Occlusions

Our NeRF model suffers from the inability to interpolate colour and density in between partially occluded regions of the scene, this becomes particularly apparent for unbounded outdoor scenes. Unfortunately the inaccuracies resulting from occluded regions in the scenes is not quantifiable if those regions are also occluded in our evaluation dataset. An obvious example of this is the surface of the water from underneath, intuitively we understand that this should be a relatively constant blue region. However, since all of our datasets focus on objects on the sea floor the model cannot learn the colour of the surface voxels. This issue is suffered by every NeRF approach as they cannot accurately perform inference on regions for which they have no data. An in-painting stable diffusion model could potentially rectify the reconstruction in these regions since they are able to infer information about the corrupted areas using the structural and textural information in the successfully reconstructed regions, however this is out of the scope of this project.

Light Intensity Attenuation

One of the major problems encountered by this model that remains unaddressed occurs when attempting to produce a 3D representation of a very large underwater scene. Specifically, the water will attenuate the intensity of light according to the following relation (known as the Beer-Lambert law [20]), where I represents the intensity at distance d along a light ray with initial intensity I_0 (c and ϵ model properties of the medium).

$$I = I_0 e^{-\epsilon c d} \tag{5.1}$$

This implies that as light travels through the water it's intensity decays exponentially, meaning that in order to capture any detail the camera has to be in close proximity to the volume. This problem is exacerbated when two frames viewing the same voxels at varying distances from those voxels, rays projected from the camera further away may be fully attenuated before they reach the voxels on the surface of the volume. With the current rendering approach our model will assume that a volume exists as the ray terminates, leading to an inconsistent representation of the structures in the scene from different viewing angles. Furthermore our model assumes that density is consistent from different viewing angles. This assumption leads to the computation of voxel density as the mean density observed across all frames, essentially introducing a low density volume in the place of the water. In turn the depth renderer computes the median absorption point before the actual surface of the structures. The shipwreck dataset is an example of a dataset that exhibits these properties, example evaluation images generated from this dataset can be seen in figure 5.21. The reconstructed example images exhibit reduced colour accuracy and contain significant blur, however they do retain most of the original structures visible in the corresponding dataset images. These effects are consistent with our hypothesis above. The colour accuracy issues stem from the fact that the colour of the water is a dark blue colour, since the water is assigned a non-zero density this colour is blended with the background colour. Similarly the blur results from the haze effect generated by this non-zero density region in the foreground.

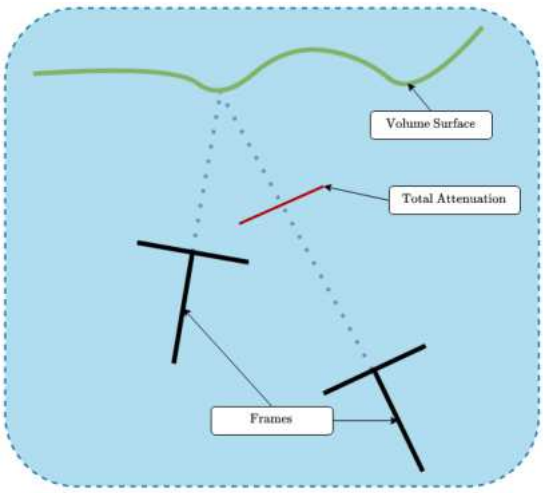


Figure 5.20: **Light attenuation between different frames:** Both cameras view the same voxels however the rays projected from the rightmost camera are attenuated before they hit the surface voxels.

(a)

(b)

Figure 5.21: **Shipwreck dataset evaluation images:** The left images show the original dataset images, the right side demonstrates the images generated by our model. It is evident that the colour accuracy is severely limited in both examples, additionally both images are notably blurrier than their corresponding dataset images.

The problem of light attenuation creates another major issue in the pre-processing pose estimation stage. The first steps of COLMAP's [31, 32] pose estimation algorithm are feature extraction and feature matching. In order to perform this step effectively every frame in the dataset must have overlapping features with at least one other frame such that from any single frame there must be a path to any other frame where neighboring frames have overlapping features. If voxels appear with different colours and densities from varying camera angles, features are no longer consistent and therefore COLMAP cannot identify poses for certain frames. Additionally for frames it *can* compute relative poses for, it may not be able to link those groups of frames relative to each-other. Therefore any frames whose pose cannot be computed relative to the largest group of identified frames must be discarded from our dataset, further reducing reconstruction quality and potentially omitting large areas of a scene from the usable dataset (the shipwreck dataset was reduced by 50%).

Chapter 6

Conclusion

The major contribution provided by this project is the novel rendering algorithm, which addresses the complications of performing novel view synthesis in diffuse media. Specifically it is able to reduce the affect of floating artifacts generated by the intrinsic bias towards near camera samples thereby improving reconstruction quality of the generated images. Additionally in some cases it is able to re-weight the samples to effectively remove small dynamic objects (such as fish) from the target reconstructions. This project was able to investigate integrating this method with the Robust losses algorithm and the loss gradient scaling algorithm, we determined quantitatively that the best performing combination is the proposed renderer and the loss gradient scaling. This project also provides a tool optimising a dataset reduction minimising loss of reconstruction accuracy whilst decreasing training times and VRAM requirements. This tool enabled the study to experiment with representing much larger scenes such as the shipwreck dataset. Finally we apply a post-process sharpen to each image which attempts to recover some high frequency detailing which the MLP was unable to accurately encode. We observe that this qualitatively improves the sharpness of the target reconstructions while reducing the PSNR and SSIM scores due to a reduction of colour accuracy.

The initial goals for this project consisted of: Evaluating existing methods to determine a suitable technique to act as the basis for this project, developing a novel approach addressing the issues associated with performing NeRF on underwater scenes and evaluate that methods performance. Nerfstudio's Nerfacto model was chosen for the basis of the proposed model the decision for which is evaluated in sections 5.1.1 and 4.3. This project does introduce a novel rendering technique which attempts to solve the dynamic volumes and floating artifact issues however fails to account for the light intensity attenuation as discussed in section 5.3.3. Additionally, our method's efficacy is reduced by common photography effects such as motion blur and lens distortions. However, each of these issues could potentially be mitigated by integrating other state-of-the-art methods with our rendering algorithm. Finally we present a comprehensive analysis of this method compared to existing techniques which focus on remedying similar problems. In most cases the proposed model can achieve similar or higher reconstruction accuracy, measured with PSNR, SSIM and LPIPS and qualitatively it is generally more effective at reducing the contribution of dynamic objects on the reconstructions.

Future work on this project could focus on increasing the efficacy of the dynamic object removal as this has potential applications in all scenes (not exclusively underwater) as demonstrated in section 5.3.2. Specifically a method could dynamically optimise the standard deviation of the reweighting-Gaussian utilised by our renderer to better represent the volume at each pixel as in some cases the current model leaves residual colour projected onto the background volumes from these dynamic objects. Additionally, our rendering approach could potentially be combined with SeaThru NeRF's medium parameter optimisation in order to model the light intensity attenuation described in section 5.3.3.

Bibliography

[1] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5460–5469, 2022. doi:10.1109/CVPR52688.2022.00539. 8

[2] Sunitha Basodi, Chunyan Ji, Haiping Zhang, and Yi Pan. Gradient amplification: An efficient way to train deep neural networks. *Big Data Mining and Analytics*, 3(3):196–207, 2020. doi:10.26599/BDMA.2020.9020004. 20

[3] Léon Bottou. Large-scale machine learning with stochastic gradient descent. *Proc. of COMPSTAT*, 01 2010. doi:10.1007/978-3-7908-2604-3_16. 4

[4] Sara Fridovich-Keil, Giacomo Meanti, Frederik Rahbæk Warburg, Benjamin Recht, and Angjoo Kanazawa. K-planes: Explicit radiance fields in space, time, and appearance. In *CVPR*, 2023. 25

[5] Kunihiko Fukushima. Visual feature extraction by a multilayered network of analog threshold elements. *IEEE Transactions on Systems Science and Cybernetics*, 5(4):322–333, 1969. doi:10.1109/TSSC.1969.300225. 2

[6] Estevão S. Gedraite and Murielle Hadad. Investigation on the effect of a gaussian blur in image filtering and segmentation. In *Proceedings ELMAR-2011*, pages 393–396, 2011. 22

[7] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 1996. URL: https://api.semanticscholar.org/CorpusID:2036193. 2

[8] Hwan Heo, Taekyung Kim, Jiyoung Lee, Jaewon Lee, Soohyun Kim, Hyunwoo J. Kim, and Jin-Hwa Kim. Robust camera pose refinement for multi-resolution hash encoding. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023. 9

[9] Geoffrey Hinton. Papers with code - rmsprop explained. URL: https://paperswithcode.com/method/rmsprop. 3

[10] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In S. C. Kremer and J. F. Kolen, editors, *A Field Guide to Dynamical Recurrent Neural Networks*. IEEE Press, 2001. 20

[11] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. URL: https://www.sciencedirect.com/science/article/pii/0893608089900208, doi:10.1016/0893-6080(89)90020-8. 2

[12] Yoonwoo Jeong, Seokjun Ahn, Christopher B. Choy, Animashree Anandkumar, Minsu Cho, and Jaesik Park. Self-calibrating neural radiance fields. *CoRR*, abs/2108.13826, 2021. URL: https://arxiv.org/abs/2108.13826, arXiv:2108.13826. 9

[13] James T. Kajiya and Brian P Von Herzen. Ray tracing volume densities. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '84, page 165–174, New York, NY, USA, 1984. Association for Computing Machinery. doi:10.1145/800031.808594. 5

[14] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics (SIGGRAPH Conference Proceedings)*, 42(4), July 2023. URL: http://www-sop.inria.fr/reves/Basilic/2023/KKLD23. v, 24

[15] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 3, 16

[16] John Lazzari and Xiuwen Liu. Understanding the Spectral Bias of Coordinate Based MLPs Via Training Dynamics. *arXiv e-prints*, page arXiv:2301.05816, January 2023. arXiv:2301.05816, doi:10.48550/arXiv.2301.05816. 22

[17] D. Levy, A. Peleg, N. Pearl, D. Rosenbaum, D. Akkaynak, S. Korman, and T. Treibitz. Seathru-nerf: Neural radiance fields in scattering media. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 56–65, Los Alamitos, CA, USA, jun 2023. IEEE Computer Society. URL: https://doi.ieeecomputersociety.org/10.1109/CVPR52729.2023.00014, doi:10.1109/CVPR52729.2023.00014. 11, 30

[18] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. *CoRR*, abs/1908.03265, 2019. URL: http://arxiv.org/abs/1908.03265, arXiv:1908.03265. 3

[19] Sebastian Lutz, Mark K. Davey, and Aljosa Smolic. Deep convolutional neural networks for estimating lens distortion parameters. IMVIP 2019: Irish Machine Vision & Image Processing, Technological University Dublin, 2019. URL: https://api.semanticscholar.org/CorpusID:203646369. 39

[20] Thomas G Mayerhöfer, Susanne Pahlow, and Jürgen Popp. The bouguer-Beer-Lambert law: Shining light on the obscure. *Chemphyschem*, 21(18):2029–2046, September 2020. 40

[21] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *CoRR*, abs/2003.08934, 2020. URL: https://arxiv.org/abs/2003.08934, arXiv:2003.08934. iv, 1, 2, 5, 6, 8, 9, 22

[22] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *CoRR*, abs/2201.05989, 2022. URL: https://arxiv.org/abs/2201.05989, arXiv:2201.05989. iv, 8

[23] Thomas Müller, Fabrice Rousselle, Jan Novák, and Alexander Keller. Real-time neural radiance caching for path tracing. *ACM Transactions on Graphics*, 40(4):1–16, July 2021. URL: http://dx.doi.org/10.1145/3450626.3459812, doi:10.1145/3450626.3459812. 8, 15, 16, 24, 25

[24] Nerfstudio. Nerfacto. URL: https://docs.nerf.studio/nerfology/methods/nerfacto.html. 24, 25, 30

[25] Julien Philip and Valentin Deschaintre. Radiance field gradient scaling for unbiased near-camera training. *arXiv preprint arXiv:2305.02756*, 2023. arXiv:2305.02756. 12, 21, 23, 32

[26] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-NeRF: Neural Radiance Fields for Dynamic Scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020. 25

[27] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred A. Hamprecht, Yoshua Bengio, and Aaron Courville. On the Spectral Bias of Neural Networks. *arXiv e-prints*, page arXiv:1806.08734, June 2018. arXiv:1806.08734, doi:10.48550/arXiv.1806.08734. 8, 22

[28] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986. URL: https://api.semanticscholar.org/CorpusID:205001834. 2

[29] Sara Sabour, Suhani Vora, Daniel Duckworth, Ivan Krasin, David J. Fleet, and Andrea Tagliasacchi. RobustNeRF: Ignoring Distractors with Robust Losses. *arXiv e-prints*, page arXiv:2302.00833, February 2023. arXiv:2302.00833, doi:10.48550/arXiv.2302.00833. 12, 20, 23, 31

[30] Robin M. Schmidt, Frank Schneider, and Philipp Hennig. Descending through a crowded valley - benchmarking deep learning optimizers. *CoRR*, abs/2007.01547, 2020. URL: https://arxiv.org/abs/2007.01547, arXiv:2007.01547. 25

[31] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 10, 41

[32] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise view selection for unstructured multi-view stereo. In *European Conference on Computer Vision (ECCV)*, 2016. 10, 41

[33] Shilei Sun, Ming Liu, Zhongyi Fan, Yuxue Liu, Chengwei Lv, Liquan Dong, and Lingqin Kong. Efficient Ray Sampling for Radiance Fields Reconstruction. *arXiv e-prints*, page arXiv:2308.15547, August 2023. arXiv:2308.15547, doi:10.48550/arXiv.2308.15547. 22

[34] Jacob Søgaard, Lukáš Krasula, Muhammad Shahid, Dogancan Temel, Kjell Brunnström, and Manzoor Razaak. Applicability of existing objective metrics of perceptual quality for adaptive video streaming. *Electronic Imaging*, 28(13):1–1, 2016. URL: https://library.imaging.org/ei/articles/28/13/art00010, doi:10.2352/ISSN.2470-1173.2016.13.IQSP-206. 4

[35] Matthew Tancik, Ethan Weber, Evonne Ng, Ruilong Li, Brent Yi, Terrance Wang, Alexander Kristoffersen, Jake Austin, Kamyar Salahi, Abhik Ahuja, David Mcallister, Justin Kerr, and Angjoo Kanazawa. Nerfstudio: A modular framework for neural radiance field development. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Proceedings*, SIGGRAPH '23. ACM, July 2023. URL: http://dx.doi.org/10.1145/3588432.3591516, doi:10.1145/3588432.3591516. 24, 30

[36] S. Taskinen and K. Nordhausen. *Iterative Weighted Least Squares*, pages 1–4. Springer International Publishing, Cham, 2020. doi:10.1007/978-3-030-26050-7_169-1. 20

[37] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image Quality Assessment: From Error Visibility to Structural Similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, April 2004. doi:10.1109/TIP.2003.819861. 4

[38] Zirui Wang, Shangzhe Wu, Weidi Xie, Min Chen, and Victor Adrian Prisacariu. Nerf-: Neural radiance fields without known camera parameters. *CoRR*, abs/2102.07064, 2021. URL: https://arxiv.org/abs/2102.07064, arXiv:2102.07064. 9

[39] Chiyuan Zhang, Qianli Liao, Alexander Rakhlin, Brando Miranda, Noah Golowich, and Tomaso A. Poggio. Memo no . 067 june 27 , 2017 theory of deep learning iii : Generalization properties of sgd. 2017. URL: https://api.semanticscholar.org/CorpusID:9939024. 7

[40] Lin Zhang, Lei Zhang, Xuanqin Mou, and David Zhang. A comprehensive evaluation of full reference image quality assessment algorithms. In *2012 19th IEEE International Conference on Image Processing*, pages 1477–1480, 2012. doi:10.1109/ICIP.2012.6467150. 4

[41] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. *CoRR*, abs/1801.03924, 2018. URL: http://arxiv.org/abs/1801.03924, arXiv:1801.03924. 5

Appendix A

Hyperparameters

```
optimiser                              = RAdam
learning_rate                          = 0.01
near_plane                             = 0.05
far_plane                              = 3000
hashmap_levels                         = 16
hashmap_min_resolution                 = 16
hashmap_max_resolution                 = 8192
log2_hashmap_size                      = 21
features_per_hashmap_level             = 2
hidden_layer_dim                       = 256
num_layers_bas                         = 2
num_layers_colour                      = 3
hidden_dim_colour                      = 256
num_proposal_samples_per_ray           = (512, 512)
num_nerf_samples_per_ray               = 64
proposal_update_every                  = 5
proposal_warmup                        = 5000
num_proposal_iteration                 = 2
use_proposal_weight_anneal             = True
proposal_weight_anneal_slope           = 10.0
proposal_weight_anneal_max_num_iters = 5000
use_single_jitter                      = True
start_depth_renderer_iters             = 5000
appearance_embed_dim                   = 32
use_appearance_embedding               = True
use_average_appearance_embedding       = True
average_init_density                   = 0.1
distortion_loss_mult                   = 0.002
depth_renderer_distribution_base       = 0.1
depth_renderer_standard_deviation      = 0.2
train_num_rays_per_batch               = 16384
eval_num_rays_per_batch                = 4096
network_decay_final_lr                 = 0.0001
network_decay_max_steps                = 50000
network_decay_warmup                   = 1024

proposal_nets = [
    {
        hidden_dim        = 16
        log2_hashmap_size = 17
        num_levels        = 5
        max_res           = 512
        use_linear        = False
    },
    {
        hidden_dim        = 16
        log2_hashmap_size = 17
        num_levels        = 7
        max_res           = 2048
        use_linear        = False
    }
]
```

Appendix B

Model Code

Project GitHub: https://github.com/Luca-G17/AquaNeRF

Appendix C

Unmask Sharpen Code

```
import cv2
import numpy as np
import os
import argparse

RENDERS='renders/'

# https://gist.github.com/soroushj/1217c523f6fe3cd3eccfd67337007e02#file-unsharpmask-py
def unmask_sharpen(image, scalar=1.5, threshold=0):
    blurred = cv2.GaussianBlur(image, (3, 3), 1)
    sharpend = float(scalar + 1) * image - float(scalar) * blurred
    sharpend = np.maximum(sharpend, np.zeros(sharpend.shape))
    sharpend = np.minimum(sharpend, np.ones(sharpend.shape) * 255)
    sharpend = sharpend.round().astype(np.uint8)
    if threshold > 0:
        low_constrast_mask = np.absolute(image - blurred) < threshold
        np.copyto(sharpend, image, where=low_constrast_mask)
    return sharpend

def sharpen_folder(folder, scalar=1.5):
    img_ext = [".png", ".jpg"]
    files = os.listdir(folder)
    files = [f for f in files if any(f.endswith(ext) for ext in img_ext)]
    sharpend_folder = f'{folder}/sharpend_{str(scalar).replace(".", "")}/'
    if not os.path.exists(sharpend_folder):
        os.mkdir(sharpend_folder)

    for i, image_name in enumerate(files):
        image = cv2.imread(f'{folder}/{image_name}')
        sharpend = unmask_sharpen(image, scalar)
        cv2.imwrite(f'{sharpend_folder}/{image_name}', sharpend)
        print(f'sharpend images: {i + 1:03}/{len(files)}\r', end="")

parser = argparse.ArgumentParser()
parser.add_argument('--frame_dir', type=str, default='')
parser.add_argument('--scalar', type=int, default=15)
args = parser.parse_args()
if args.frame_dir == '':
    print("Command line argument 'frame_dir' missing")
else:
    sharpen_folder(RENDERS + args.frame_dir, args.scalar / float(10))
```

Appendix D

Metric Evaluation Code

```python
import cv2
import numpy as np
import os
import argparse
import re
from math import log10, sqrt
from skimage.metrics import structural_similarity
from torchmetrics.image.lpip import LearnedPerceptualImagePatchSimilarity
from torch import from_numpy, clamp

RENDERS='renders/'

def psnr(original, reconstruction):
    if (original.shape != reconstruction.shape):
        h, w, _ = reconstruction.shape
        original = cv2.resize(original, (w, h))
        print(f"PSNR: Input image shapes do not match, resizing to {(w, h)}")
    mse = np.mean((original - reconstruction) ** 2)
    if (mse == 0):
        psnr = 100
    else:
        max_pixel = 255
        psnr = 20 * log10(max_pixel / sqrt(mse))
    return psnr

def ssim(original, reconstruction):
    score = structural_similarity(original, reconstruction, channel_axis=-1)
    return score

def lpips(original, reconstruction, model):
    t_original = clamp(from_numpy(np.transpose([original], (0, 3, 1, 2))) / 255.0, 0, 1)
    t_reconstruction = clamp(from_numpy(np.transpose([reconstruction], (0, 3, 1, 2))) /
    255.0, 0, 1)
    return model(t_original, t_reconstruction)

def get_images_in_dir(dir):
    image_exts = ['.png', '.jpg']
    files = os.listdir(dir)
    files = [f for f in files if any(f.endswith(ext) for ext in image_exts)]
    return files

def average_metrics(original_dir, reconstruction_dir):
    recons = get_images_in_dir(reconstruction_dir)
    originals = get_images_in_dir(original_dir)

    image_dict = {}
    for image in originals:
        image_no = re.findall(r"[0-9]+", image)[0]
        image_dict[image_no] = [image]
    for image in recons:
        image_no = re.findall(r"[0-9]+", image)[0]
        image_dict[image_no].append(image)

    total_psnr = 0
```

```
    total_ssim = 0
    total_lpips = 0
    lpips_model = LearnedPerceptualImagePatchSimilarity(normalize=True)
    for i, image_pair in enumerate(image_dict.values()):
        original = cv2.imread(f'{original_dir}/{image_pair[0]}')
        reconstruction = cv2.imread(f'{reconstruction_dir}/{image_pair[1]}')
        total_psnr += psnr(original, reconstruction)
        total_ssim += ssim(original, reconstruction)
        total_lpips += lpips(original, reconstruction, lpips_model)
        print(f"Computing metrics (PSNR+SSIM+LPIPS) for images: {i + 1:03}/{len(
    image_dict.values())}\r", end="")

    average_psnr = total_psnr / float(len(image_dict.values()))
    average_ssim = total_ssim / float(len(image_dict.values()))
    average_lpips = total_lpips / float(len(image_dict.values()))
    print()
    print(f"Average PSNR: {average_psnr:.2f}dB")
    print(f"Average SSIM: {average_ssim:.2f}")
    print(f"Average LPIPS: {average_lpips:.2f}")

parser = argparse.ArgumentParser()
parser.add_argument('--gt_path', default='')
parser.add_argument('--recon_path', default='')
args = parser.parse_args()
if args.gt_path != '' and args.recon_path != '':
    average_metrics(RENDERS + args.gt_path, RENDERS + args.recon_path)
```

Appendix E

Dataset Reduction code

```
import os
import cv2
import argparse
import json
import shutil
from utils.utils import psnr

RENDERS='renders/'

def copy_images(source_dir, destination, frame_names, total_copied):
    for im in frame_names:
        total_copied += 1
        shutil.copyfile(f'{source_dir}/{im}', f'{destination}/{im}')
        print(f'Copying images: {total_copied + 1}/{4 * int(len(frame_names))}\r', end=""
    )

    return total_copied

def filter_images(dataset_dir, filter_regime='proportion', keep_proportion=0.7,
    psnr_threshold=28.0):
    if filter_regime == 'proportion':
        filtered_dir = f'{dataset_dir}/filtered_prop_{int(keep_proportion * 10)}'
    elif filter_regime == 'psnr':
        filtered_dir = f'{dataset_dir}/filtered_psnr_{int(psnr_threshold * 10)}'

    if not os.path.exists(filtered_dir):
        os.mkdir(filtered_dir)
        os.mkdir(f'{filtered_dir}/images')
        os.mkdir(f'{filtered_dir}/images_2')
        os.mkdir(f'{filtered_dir}/images_4')
        os.mkdir(f'{filtered_dir}/images_8')

    with open(f'{dataset_dir}/transforms.json', 'r') as t_file:
        transforms = json.load(t_file)
        remove_every = int(1 / (1 - keep_proportion))
        kept_frames = []
        frame_count = len(transforms['frames'])
        frames = transforms['frames']
        frames = sorted(frames, key=lambda f : int(f['file_path'].replace('images/frame_'
    , '').replace('.jpg', '').replace('.png', '')))

        if filter_regime == 'proportion':
            for i in range(frame_count):
                if (i + 1) % remove_every != 0:
                    kept_frames.append(frames[i])
        elif filter_regime == 'psnr':
            i = 0
            while i < frame_count:
                kept_frames.append(frames[i])
                offset = 0
                f = i
                psnr_value = 100
                frame1 = cv2.imread(f'{dataset_dir}/{frames[i]["file_path"]}')
                while psnr_value > psnr_threshold and f < frame_count - 1:
```

```
                    offset += 1
                    f = i + offset
                    frame2 = cv2.imread(f'{dataset_dir}/{frames[f]["file_path"]}')
                    psnr_value = psnr(frame1, frame2)
                    print(f'Processing PSNR comparison: {f}/{frame_count}\r', end='')
                if offset == 0: i += 1
                else: i += offset

        transforms['frames'] = kept_frames

    with open(f'{filtered_dir}/transforms.json', 'w', encoding='utf-8') as f:
        json.dump(transforms, f, ensure_ascii=False, indent=4)

    print(f'Finished writing {filtered_dir}/transforms.json')
    kept_frame_names = [frame['file_path'].replace('images/', '') for frame in
    kept_frames]
    total_copied = 0
    total_copied = copy_images(f'{dataset_dir}/images', f'{filtered_dir}/images',
    kept_frame_names, total_copied)
    total_copied = copy_images(f'{dataset_dir}/images_2', f'{filtered_dir}/images_2',
    kept_frame_names, total_copied)
    total_copied = copy_images(f'{dataset_dir}/images_4', f'{filtered_dir}/images_4',
    kept_frame_names, total_copied)
    total_copied = copy_images(f'{dataset_dir}/images_8', f'{filtered_dir}/images_8',
    kept_frame_names, total_copied)
    print(f'Kept {100 * len(kept_frame_names) / frame_count:.2f}% of the original dataset
     -- ({len(kept_frame_names)}/{frame_count})')

def main():
    parser = argparse.ArgumentParser()
    parser.add_argument("--path")
    parser.add_argument("--filter_regime", default='proportion')
    parser.add_argument("--psnr_threshold", default=280, type=int)
    args = parser.parse_args()
    if args.path != '':
        filter_images(args.path, filter_regime=args.filter_regime, psnr_threshold=args.
    psnr_threshold / 10.0)

if __name__ == '__main__':
    main()
```