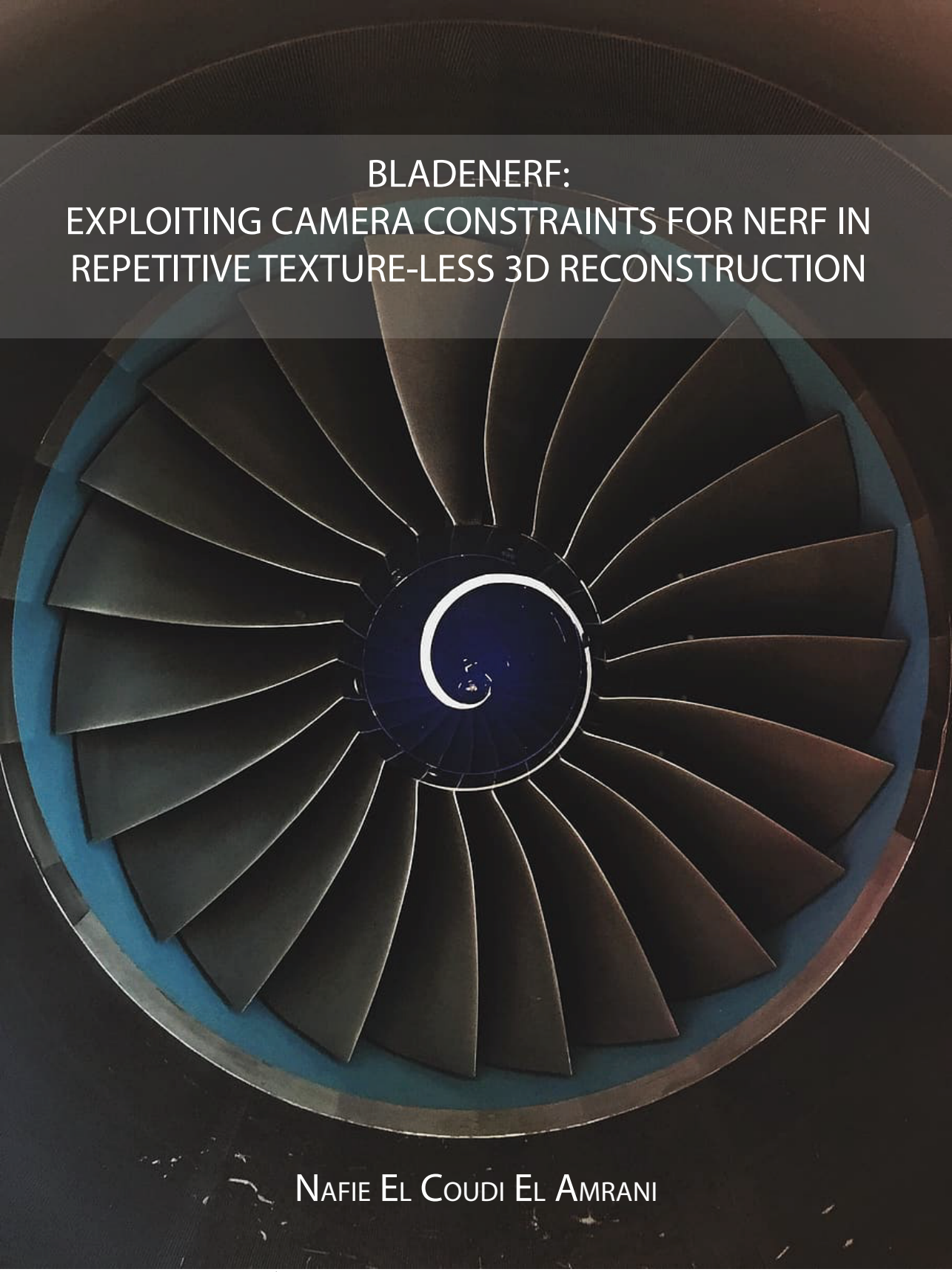
BLADENERF: EXPLOITING CAMERA CONSTRAINTS FOR NERF IN REPETITIVE TEXTURE-LESS 3D RECONSTRUCTION



Nafie El Coudi El Amrani

TU Delft Delft University of Technology

AIIR INNOVATIONS

BLADENERF: EXPLOITING CAMERA CONSTRAINTS FOR NERF IN REPETITIVE TEXTURE-LESS 3D RECONSTRUCTION

Master Thesis

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday 26 th June, 2023 at 14:00 o'clock.

by

Nafie EL COUDI EL AMRANI

Project Duration:	11, 2022 - 06, 2023
Supervisors:	Dr. J.C. van Gemert Dr. Y. Lin
Thesis Committee:	Dr. J.C. van Gemert (Associate Professor) Dr. M. Weinmann (Assistant Professor)

An electronic copy of this dissertation is available at
https://repository.tudelft.nl/.

PREFACE

This report marks the culmination of my Master's thesis project, titled "BladeNeRF: Exploiting camera constraints for NeRF in repetitive texture-less 3D reconstruction", conducted at the Delft University of Technology to obtain the degree of Master of Science. The project was conducted within the Computer Vision Lab at TU Delft in collaboration with Aiir Innovations.

I would like to express my sincere gratitude to Dr. J. C. van Gemert for his outstanding guidance and supervision throughout this project. I am also grateful to Dr. Y. Lin, my daily supervisor, for his constant feedback and support. Both have significantly helped shape the direction and focus of my research. I am also thankful to Miriam Huijser and Steve Nowee from Aiir Innovations for their invaluable guidance and suggestions during the course of my thesis. Furthermore, I want to thank my family and friends for their unwavering support and encouragement. Their belief in me has been a constant motivation throughout my journey. I am deeply grateful to my sister, Assmae, for her unwavering assistance and invaluable support. I also appreciate my friend Max's thorough review of my work and for providing me with invaluable feedback.

Completing this Master's thesis has been a rewarding experience. Thanks to everyone who has accompanied me throughout this journey.

Nafie El Coudi El Amrani
Delft, June 2023

Contents

Preface v

1 Introduction 1

2 Scientific Paper 3

3 Background on Deep Learning 25
- 3.1 Deep Learning . . . 25
- 3.2 Activation Functions . . . 26
- 3.3 Training a Neural Network . . . 26

4 Background on Neural Radiance Fields 29
- 4.1 Neural Fields . . . 29
  - 4.1.1 Advantages of Neural Fields . . . 30
  - 4.1.2 Training Neural Fields . . . 30
- 4.2 NeRF . . . 30
  - 4.2.1 Main Objective of NeRF . . . 30
  - 4.2.2 Training a NeRF model . . . 31
  - 4.2.3 Limitations of NeRF . . . 32
- 4.3 BaRF . . . 33
- 4.4 Mip-NeRF . . . 34
  - 4.4.1 Integrated positional encoding (IPE) . . . 34
  - 4.4.2 Efficient Sampling . . . 35
  - 4.4.3 Loss Function . . . 35

5 Rendering and 3D reconstruction Techniques 37
- 5.1 Volume Rendering . . . 37
  - 5.1.1 Volumetric Data . . . 37
  - 5.1.2 Optical Models . . . 38
  - 5.1.3 The Volumetric Rendering Integral . . . 38
  - 5.1.4 Ray Casting . . . 39
  - 5.1.5 Alpha Blending . . . 40
- 5.2 Marching Cubes Algorithm . . . 41

Bibliography 45

1

INTRODUCTION

The utilization of borescopes has become essential in streamlining the inspection process for aircraft engines [1]. These borescopes consist of monocular cameras mounted on semi-rigid bodies, enabling access to confined areas of the aircraft and capturing valuable footage. One promising application of this footage is the generation of a 3D model of the engine's blades, which holds the potential for automating damage measurement and detection. One notable approach to performing 3D reconstruction of aircraft engine blades using inspection camera footage is Neural Radiance Fields (NeRF) [2]. However, NeRF encounters two significant challenges when attempting to learn the scene representation of engine blades.

The first challenge arises from the unavailability of accurate camera poses, which refer to the location and orientation of the camera. Unfortunately, borescopes lack spatial awareness and cannot provide camera poses, which are crucial for training NeRF models. To overcome this challenge, camera poses can be estimated directly from the borescope images. Traditional methods rely on robust visual features and similarities to estimate camera poses, but they struggle to extract sufficient visual features from the blade images due to their visual similarity. Recently, learning-based approaches proposed to jointly estimate camera poses and scene representation during learning. However, these methods fail to differentiate between the engine blades due to the lack of texture and the presence of significant visual similarities among the blades. The second challenge arises from the limited field of view of the inspection cameras, which primarily capture the frontal view of the blades and lack access to other sides of the blades due to the restricted environment within the aircraft engine. Consequently, NeRF produces artefacts in the mesh reconstruction once the scene representation is learned.

To overcome both challenges, we propose a two-stage method called BladeNeRF. It learns the camera poses of borescope images by incorporating prior knowledge about camera orientation and movement during inspections. Additionally, it guides the model to focus on the regions where the blades are located during training, thus mitigating the artefacts caused by the limited field of view of the input images in NeRF.

1

This thesis consists of two parts. The first part, presented in Section 2, is a CVPR 2023 formatted scientific paper. It covers the motivation, related work, approach, experiments, and results of the research project, targeting computer vision experts. The second part, comprising Sections 3, 4, and 5, provides supplementary material to facilitate understanding of the scientific paper for a more general audience.

2

SCIENTIFIC PAPER

BladeNeRF: Exploiting camera constraints for NeRF in repetitive texture-less 3D reconstruction

Nafie El Coudi El Amrani, Yancong Lin, Jan C. van Gemert
Delft University of Technology

Abstract

Neural Radiance Fields (NeRFs) have demonstrated remarkable capabilities in photo-realistic 3D reconstruction. NeRFs often take as input posed images where the camera poses come from either off-the-shelf SfM or online optimization together with NeRFs. However, we find that both strategies yield suboptimal results in recovering camera poses from images when encountering texture-less and repetitive patterns, particularly in aircraft engine inspection. To reconstruct photo-realistic 3D engine blades from images, we propose BladeNeRF, a new variant of NeRF model that incorporates camera constraints into learning and enables accurate pose learning. In addition, we propose to separate the blades in the foreground from the constant background, eliminating background artefacts and enhancing depth estimation accuracy. Experimental evaluations on synthetic data demonstrate the advantage of our model in precise camera pose estimation and high-fidelity 3D scene reconstruction compared to other NeRF variants.

1. Introduction

The use of borescopes has become crucial in streamlining the inspection process for aircraft engines [47]. These borescopes are equipped with monocular cameras mounted on semi-rigid bodies, allowing access to confined spaces within the engine turbines to capture valuable footage [47]. One promising application of this footage is creating a 3D model of the engine's blades, which offers the potential for automating damage measurement and detection. In the constrained inspection setting, the borescope operates under two limitations. It remains stationary at a specific distance and constantly points towards the blades, which rotate at a constant speed in a single direction. This setting is equivalent to a camera positioned at a fixed distance, capturing RGB images while moving along a circular trajectory at a constant velocity, as depicted in Fig. 1. This restricted movement, coupled with the repetitive and texture-less characteristics of the aircraft engine blades, poses significant challenges in reconstructing an accurate 3D model.

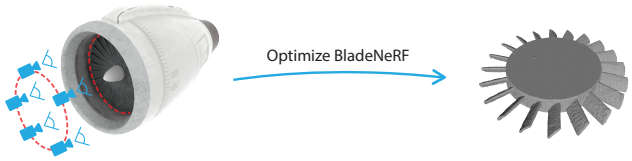
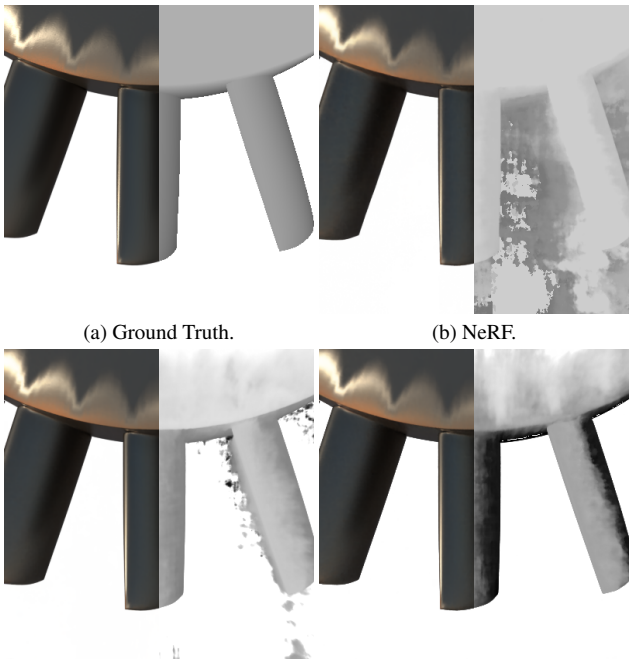


Figure 1. *BladeNeRF enables precise 3D reconstruction of aircraft engines. By capturing RGB images from a borescope (depicted as a camera) following a circular pattern, facing the blades, BladeNeRF's two-stage process learns camera poses and creates a scene representation, providing valuable insight into engine blade analysis.*

The central focus of our work is photo-realistic 3D reconstruction of the aircraft engine blades using the footage obtained from the inspection camera. To this end, we employ Neural Radiance Fields (NeRF) [24], a promising technique for learning scene representations. NeRF utilizes a multi-layer perceptron (MLP) to encode the scene and uses ray tracing techniques to reconstruct pixel colours based on photometric consistency. Within our context, NeRF encounters two notable challenges. The first arises from the unavailability of accurate camera poses, which are crucial for training NeRF models [24]. The second challenge is the limited field of view, as the inspection cameras often face toward the blades. The lack of camera poses and diverse viewpoints poses great challenges for photo-realistic reconstruction in the context of engine blade reconstruction.

A popular option for camera pose estimation is structure from motion (S*f*M) techniques [29, 33, 38], which rely on robust visual features, e.g. SIFT features [21, 22]. However, engine blades are always textureless, making S*f*M unsuitable for our setting [7]. Alternatively, there is a line of research [4, 6, 19, 42, 44], which jointly optimizes camera poses and scene representation during learning. One exemplar model is Bundle-Adjustment Neural Radiance Fields (BARF) [19], which employs Bundle-Adjustment [20, 39] to estimate the six degrees of freedom of camera poses. Although BARF [19] shows promising results on LLFF and Synthetic NeRF datasets [19], it struggles in our specific context due to its inability to differentiate between the blades (see Appendix A). To overcome these challenges, we



(a) Ground Truth. (b) NeRF. (c) Mip-NeRF. (d) BladeNeRF **(ours)**.

Figure 2. *NeRF (b) and Mip-NeRF (c) struggle with depth estimation in the empty regions between the blades. We present **BladeNeRF** (d), a method for learning the camera poses and reconstructing the scene that solves the ambiguities in the background.*

propose an extension to BaRF that leverages prior knowledge of the context, namely, the camera faces toward the blades at a fixed position and follows a circular movement pattern during the inspection. By incorporating these assumptions, we reduce the degrees of freedom to translations along x- and y-axes only.

For high-fidelity 3D reconstruction, we opt for Mip-NeRF [3] as it offers notable improvements in depth estimation compared to NeRF and other NeRF variants. However, we have noticed that Mip-NeRF fails to accurately estimate the depth of the regions in proximity to the blades due to the lack of different perspectives, as shown in Fig. 2. We propose integrating a foreground-background loss term to address this limitation. This loss term selectively excludes the background during training and mitigates the artefacts in depth estimation.

In summary, this paper presents two contributions:

- We propose a method for estimating camera poses in a repetitive, texture-less setting by leveraging the explicit constraints of camera movements during the inspection and the characteristics of engine blades.
- We introduce a fore-ground loss term to eliminate various artefacts in depth estimation, consequently allowing precise 3D reconstruction.

2. Related Works

Neural Radiance Fields. There are several ways to recover a 3D scene from multi-view images, including classical voxel-based representations [32], point clouds [23, 26], polygonal meshes [15] and more recent approaches like layered depth [34, 40], mesh sheets [14], light fields [36], and neural networks [37]. One particularly well-known technique is NeRF [24], which encodes the 3D scene as a continuous function and has achieved excellent performance in multi-view scene reconstruction. However, NeRF models and their derivatives fail to produce high-quality 3D reconstructions when applied in our specific scenario. To this end, our work presents a straightforward yet effective solution by introducing a background regularization technique and leveraging our unique setting-specific knowledge.

NeRF and camera pose estimation. Camera pose estimation is a long-standing task in 3D reconstruction. Structure from Motion (S*f*M) [33, 38] and Simultaneous Localization and Mapping (SLAM) [10, 25] first estimate camera poses based on feature matching [1] or photometric consistency [2], then, they reconstruct the scene's explicit geometry (e.g., triangular meshes or point clouds). However, these methods struggle with texture-less and repetitive patterns due to limitations in the expressiveness of extracted features [31]. Recent approaches leverage neural networks to learn scene representation and camera poses jointly from a collection of unposed images. NeRF$--$ [42] is a pioneer in this line of research but produces poor results in texture-less scenes [44]. The subsequent work SiNeRF [44] employs SIREN layers [35] and a new sampling strategy [44] to efficiently select ray batches, resulting in better joint optimization. GaRF [9] improves joint optimization with Gaussian activation functions. BaRF [19] introduces a fine-to-coarse positional encoding for online camera pose optimization. L2G-NeRF [6] tackles camera misalignment with Local-to-Global registration. NoPe-NeRF [4] integrates mono-depth maps to handle challenging camera trajectories. However, there is limited focus on 3D scene reconstruction from repetitive patterns and texture-less inputs. To fill this gap, our approach, BladeNeRF, incorporates explicit constraints on camera poses to enhance 3D reconstruction.

Neural rendering with sparse views. NeRF struggles in learning scene representation from limited and sparse viewing points [24]. To this end, researchers have explored various strategies. These include leveraging external models for supervising the learning process through perceptual regularization [48], cross-view semantic consistency [16], or sparse depth estimation [11, 43]. Alternative approaches involve training transferable models on well-curated datasets for few-shot learning [5, 8, 46]. Geometry regularization has also been proposed to enhance

few-shot neural rendering [17, 27, 45]. Our work incorporates prior knowledge of the scene in alignment with these geometry-based approaches. In addition, we also introduce a novel depth estimation supervision method using foreground/background segmentation to mitigate background ambiguities.

3. Preliminaries

Fundamentals on NeRF scene representation. NeRF [24] learns the representation of a 3D scene as a volumetric radiance field by optimizing the weights Θ of a multilayer perceptron (MLP) F_Θ. For a 3D point $\boldsymbol{x} \in \mathbb{R}^3$ and a viewing direction vector $\boldsymbol{d} \in \mathbb{R}^3$, F_Θ returns an RGB colour $\boldsymbol{c}$ and a volume density σ: $F_\Theta : (\boldsymbol{x}, \boldsymbol{d}) \mapsto (\boldsymbol{c}, \sigma)$. For a novel view, a camera ray $\boldsymbol{r}(t) = \boldsymbol{o} + t\boldsymbol{d}$ can be cast through the camera center $\boldsymbol{o} \in \mathbb{R}^3$ along the direction $\boldsymbol{d}$. The expected colour $\hat{\boldsymbol{C}}(\boldsymbol{r})$ of camera ray $\boldsymbol{r}$, with near and far bounds t_n and t_f respectively, is:

$$\hat{\boldsymbol{C}}(\boldsymbol{r}) = \int_{t_n}^{t_f} T(t)\sigma(\boldsymbol{r}(t))\boldsymbol{c}(\boldsymbol{r}(t))dt, \tag{1}$$

where $T(t) = exp\left(-\int_{t_n}^{t} \sigma(\boldsymbol{r}(s))ds\right)$ denotes the probability that the ray travels from t_n to t without intersecting with any part of the scene. In practice, the integral in Eq. (1) along $\boldsymbol{r}$ within the predefined scene bounds $[t_n, t_f]$ is approximated using numerical quadrature, achieved by sampling points along each pixel ray. The estimated colour of the pixel $\hat{\boldsymbol{C}}(\boldsymbol{r})$ is compared to the ground truth pixel colour $\boldsymbol{C}(\boldsymbol{r})$ for all camera rays from the novel view. This evaluation determines the final rendering loss for NeRF:

$$\mathcal{L} = \sum_{\boldsymbol{r}\in\mathcal{R}} \left|\left|\hat{\boldsymbol{C}}(\boldsymbol{r}) - \boldsymbol{C}(\boldsymbol{r})\right|\right|_2^2, \tag{2}$$

where $\mathcal{R}$ is the set of camera rays from the novel view.

NeRF brought two crucial contributions. First, the continuous input coordinates $\boldsymbol{x}$ and $\boldsymbol{d}$ undergo a mapping γ known as "positional encoding" before being fed into F_Θ. This mapping enables F_Θ to approximate higher-frequency functions efficiently. Second, NeRF introduces hierarchical volume sampling, achieved through optimizing two MLPs, to efficiently allocate samples around regions expected to contain visible content [24].

BaRF camera registration. In the first stage of BladeNeRF, we extend BaRF [19] to learn the camera poses from the input RGB images. BaRF learns camera poses by simultaneously optimizing a NeRF model and the camera poses. It introduces a coarse-to-fine camera registration approach by applying a smooth mask on positional encoding across various frequency bands. The k-th frequency component of the positional encodings is calculated as follows:

$$\gamma_k(\boldsymbol{x}, \delta) = w_k(\delta)\big[\cos(2^k\pi\boldsymbol{x}), \sin(2^k\pi\boldsymbol{x})\big], \tag{3}$$

where the weight w_k is:

$$w_k(\delta) = \begin{cases} 0 & \text{if } \delta < 1 \\ \dfrac{1-\cos((\delta-k)\pi)}{2} & \text{if } 0 \le \delta - k \le 1 \\ 1 & \text{if } 1 \le \delta - k, \end{cases} \tag{4}$$

$\delta \in [0, L]$ is a controllable parameter proportional to the optimization progress and L is the size of the higher dimension space. This progressive schedule allows BaRF to first learn camera poses based on smoother signals and then transition to acquiring a high-fidelity scene representation. As a result, the camera poses converge early during training to their optimal positions in space.

Mip-NeRF's extensions to NeRF. The NeRF approach employs a single ray per pixel, which can result in blurred or aliased renderings when training images capture the scene at different resolutions [24]. To this end, Mip-NeRF [3] offers a solution by casting a cone per pixel instead. By using conical frustums for querying, Mip-NeRF provides a continuous and inherently multiscale representation of volume space regions, avoiding the drawbacks of discrete point sampling [3]. We use Mip-NeRF as a backbone for the second stage of BladeNeRF.

4. Approach

To approach the challenging task of estimating the camera poses and learning the scene representation of aircraft engines, BladeNeRF is structured into two stages, as shown in Fig. 3. In the first stage, we estimate the camera poses of the input images by incorporating prior knowledge about the inspection setting into BaRF [19]. Then, we utilize the learned camera poses in the second stage to learn the scene representation by extending Mip-NeRF's approach [3].

4.1. First Stage: Learning Camera Poses

In the first phase of BladeNeRF, our objective is to estimate the set of camera poses $\boldsymbol{\mathcal{P}}$ for the given input images $\boldsymbol{\mathcal{I}}$. Due to the repetitive nature of the engine blades, every few images along the circular pattern showcase high visual similarities as shown in Fig. 4. Based on this observation, we partition the input images $\boldsymbol{\mathcal{I}}$ into N distinct repetitions, where each repetition consists of consecutive images captured between two adjacent blades. This partitioning strategy ensures that visually similar images are not grouped within the same repetition while simultaneously grouping consecutive images with highly overlapping fields of view.

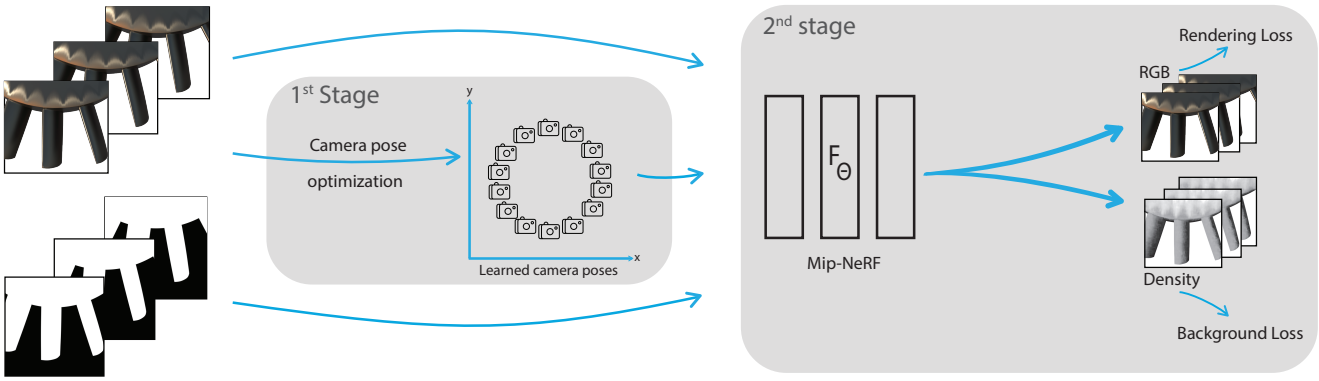


Figure 3. ***Pipeline of BladeNeRF****. Our proposed method, BladeNeRF, learns the scene representation by utilizing RGB images and binary masks to distinguish foreground and background regions. In the first stage, camera poses are learned from the RGB images. The RGB images, binary masks, and learned camera poses are inputted into a Mip-NeRF backbone in the second stage. Our method enables precise depth estimation in the inter-blade regions by incorporating a composite loss comprising a Rendering loss and a newly introduced Background loss.*

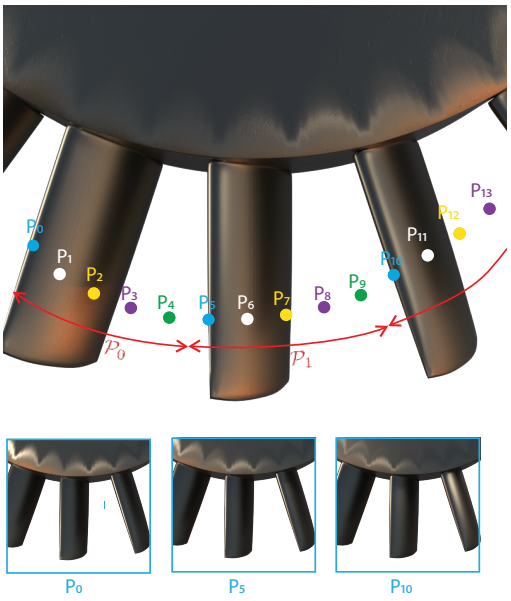


Figure 4. *Visualisation of the relation between one repetition and camera poses. We partition the set of all camera poses into multiple repetitions such as $\boldsymbol{\mathcal{P}}_0$ and $\boldsymbol{\mathcal{P}}_1$ (depicted in red) that contain 5 consecutive camera poses, which are visually different. Due to the repetitive nature of engine blades, images taken from camera poses (shown with the same colours such as $\boldsymbol{P}_0$, $\boldsymbol{P}_5$ and $\boldsymbol{P}_{10}$) are visually similar.*

We denote the i-th image and its camera pose as $\boldsymbol{I}_i$ and $\boldsymbol{P}_i$, respectively. Similarly, we refer to the set of images and camera poses for the n-th repetition as $\boldsymbol{\mathcal{I}}_n$ and $\boldsymbol{\mathcal{P}}_n$.

4.1.1 Parametrization of One Camera Pose

In our work, we represent the camera pose $\boldsymbol{P}$ as a camera-to-world matrix $\boldsymbol{P} = [\boldsymbol{R}|\boldsymbol{t}]$, where $\boldsymbol{R} \in \mathbf{SO(3)}$ and $\boldsymbol{t} \in \mathbb{R}^3$. We assume the blades align with the xy-plane in the world coordinate, and their centre coincides with the origin of the same system, as shown in Fig. 5. We make several assumptions during the inspection:

- The camera maintains a predefined distance to the blades along the z-axis;
- The camera faces the blades from its predefined distance in the negative z-direction;
- The camera orbits around the z-axis with a constant speed;
- The camera follows a circular pattern with a fixed radius from the centre of the world coordinate system.

The first assumption makes the translation along z-axis constant. The second assumption results in zero rotation around the x- and y-axes. The third assumption allows us to manually extrapolate the rotation over the z-axis once the number of blades is known. Specifically, we calculate the degree of rotation between two adjacent frames as $\Delta = \frac{2\pi}{M}$, where M is the total number of frames after a full round inspection. Considering all assumptions, we reduce the number of unconstrained variables from six to two, namely the translations t_x and t_y along the x- and y-axes.

4.1.2 Optimizing Camera Poses of One Repetition

To accurately recover the camera poses from one repetition, we follow the parametrization introduced in Sec. 4.1.1. We initialize the translations t_x and t_y by adding random noise to the ground truth values. Next, we recover the camera poses using BARF [19]. Training continues until convergence, producing optimized camera poses for a particular repetition. The following subsection explains how to recover the camera poses for all repetitions.

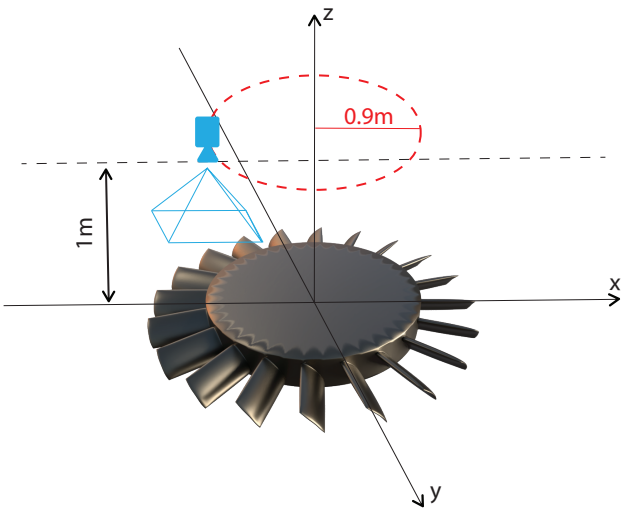


Figure 5. *Visualisation of the experiment setup. The camera is located at a fixed height of 1 meter, it orbits around the z-axis in a circular pattern with a radius of 0.9 meters (depicted in red), and it faces the blades in the negative z-direction.*

4.1.3 Combining Learned Camera Poses of N Repetitions

To obtain the set of camera poses for all frames, we present two strategies by taking advantage of the repetitive nature of the blades.

- **Solution 1 (S1)** optimizes the camera poses for each repetition independently, utilizing the process outlined in Sec. 4.1.2. We then combine the estimated poses from each repetition $\boldsymbol{\mathcal{P}}_n$ into a single set $\boldsymbol{\mathcal{P}}$, where $\boldsymbol{\mathcal{P}} = \bigcup_{n=0}^{n=N} \boldsymbol{\mathcal{P}}_n$ and N is the number of repetitions. This solution is equivalent to training N BARF models in parallel.

- **Solution 2 (S2)** follows the assumption that camera poses within a repetition exhibit comparable relative rotation and translation to camera poses across other repetitions. In essence, the camera poses within each repetition collectively form an arc of the circular trajectory of the camera. As a result, this approach only requires optimizing the camera poses $\boldsymbol{\mathcal{P}}_k$ for a single repetition k, where k can be any repetition. Consequently, we reconstruct the remaining repetitions by transforming $\boldsymbol{\mathcal{P}}_k$ along the circular trajectory using:

$$\boldsymbol{\mathcal{P}} = \bigcup_{n=0}^{n=N} \mathcal{T}(\boldsymbol{\mathcal{P}}_k, \frac{2\pi}{N} n), \tag{5}$$

where $\mathcal{T}(\boldsymbol{\mathcal{P}}_k, \theta)$ is a function that transforms a set of camera poses $\boldsymbol{\mathcal{P}}_k$ around the z-axis by an angle θ along the circular pattern in the counterclockwise direction.

4.2. Second stage: Scene representation

In the second stage of BladeNeRF, we learn the scene representation by leveraging the optimized camera poses $\boldsymbol{\mathcal{P}}$ obtained from the first stage and by training a Mip-NeRF [3] model. In addition, we introduce a background loss term, $\mathcal{L}_{bg}$, that separates the blades in the foreground from the constant background during training, enabling BladeNeRF to eliminate artefacts in the background effectively and thus resulting in more accurate depth estimation of the scene, as illustrated in Fig. 2.

Background loss. To supervise BladeNeRF, we use Blender to extract a collection of binary segmentation masks that accurately separate foreground and background pixels within the input images. These masks are fed into BladeNeRF during training. The background loss is computed for each ray $\boldsymbol{r}$ cast through the scene as:

$$\mathcal{L}_{bg}(\boldsymbol{r}) = \begin{cases} \mid d_{\boldsymbol{r}} \mid, & \text{if } \boldsymbol{r} \text{ is in background} \\ 0, & \text{if } \boldsymbol{r} \text{ is in foreground,} \end{cases} \tag{6}$$

where $d_{\boldsymbol{r}}$ is the density of the scene along the ray $\boldsymbol{r}$. The total loss is a linear combination of the Mip-NeRF [3] losses and our background loss:

$$\begin{aligned} \mathcal{L} = \sum_{\boldsymbol{r} \in \mathcal{R}} & \left|\left| \boldsymbol{C}_c(\boldsymbol{r}) - \hat{\boldsymbol{C}}(\boldsymbol{r}) \right|\right|_2^2 \\ & + \alpha \left|\left| \boldsymbol{C}_f(\boldsymbol{r}) - \hat{\boldsymbol{C}}(\boldsymbol{r}) \right|\right|_2^2 \\ & + \beta \mathcal{L}_{bg}, \end{aligned} \tag{7}$$

where $\boldsymbol{C}_c(\boldsymbol{r})$ and $\boldsymbol{C}_f(\boldsymbol{r})$ represent the estimated colours of the pixel along $\boldsymbol{r}$ using both the "coarse" and "fine" sampling strategies while $\hat{\boldsymbol{C}}(\boldsymbol{r})$ is the ground truth colour along the ray $\boldsymbol{r}$ from the observed input image. α and β are the scaling factors.

5. Experiments

We assess the performance of BladeNeRF from four key aspects: camera pose estimation, novel view synthesis, depth estimation, and learning scene representation through mesh reconstruction.

5.1. Dataset

We manually create a synthetic dataset using Blender, which contains rendered images, depth maps, and associated camera poses. The images in our dataset are rendered at a resolution of 400×400 pixels. The camera faces the blade on the xy-plane from a distance of 1 meter along the z-axis. During the inspection, the camera traces a circular trajectory around the z-axis, following the counterclockwise direction. The radius of the circular path is 0.9 meters.

The synthetic aircraft engine has 20 blades resulting in 20 repetitions. Images are taken at equidistant locations along the circular path. Three sets are created with images captured at different degrees of rotation around the z-axis along the circular pattern. The validation set encompasses 80 images, each obtained at intervals of 4.5 degrees. The first image within this set is captured at an angle of 3 degrees. The training set comprises 100 images, taken at intervals of 3.6 degrees, commencing from 0 degrees. Lastly, the testing set consists of 200 images captured at intervals of 1.8 degrees, with the initial image obtained at an angle of 0.9 degrees. Each set encompasses images from different locations on the circular camera path, ensuring a diverse range of viewpoints for comprehensive evaluation.

5.2. Exp 1: Camera Pose Estimation

We first investigate how well BladeNeRF can estimate camera poses from RGB images of engine blades. We conduct this experiment on our synthetic dataset.

Experimental settings. This experiment aims to recover camera poses for each image. Due to the unique setup, there are only two unknowns - translation along x- and y-axes. We synthetically perturb the ground truth pose with additive noise sampled from $\mathcal{N}(0, 0.15)$, resulting in a translation magnitude of 0.26 meters. Additionally, we assume known camera intrinsics.

We compare BladeNeRF primarily with BaRF [19]. We first report the performance of estimating all six degrees of freedom (DoF). Then we conduct experiments in a controlled setup where only translations along x- and y-axes are considered unknown, and the remaining four DoF are obtained from the ground truth camera poses. Furthermore, we include COLMAP [29], a Structure-from-Motion (SfM) method, in our evaluation to estimate the camera poses of our synthetic dataset.

Implementation details. During the optimisation process, we randomly sample 1024 rays. To ensure accurate numerical integration along each ray, we select 128 samples. Furthermore, we apply the softplus activation function to enhance stability in the volume density output. We use Adam optimiser [18] for 20K iterations. The initial learning rate for the NeRF backbone is set to 5×10^{-4}, which exponentially decays to 1×10^{-4}. The initial learning rate for pose estimation is set to 1×10^{-3}, which decays to 1×10^{-5}. Regarding the scheduler [19], which controls positional encoding at different frequency bands, we linearly adjust the parameter δ between the 5K and 20K iterations and set the value L to be 10. Every run of the first stage of BladeNeRF requires around 30 minutes of computation time on a single NVIDIA 1080 GPU.

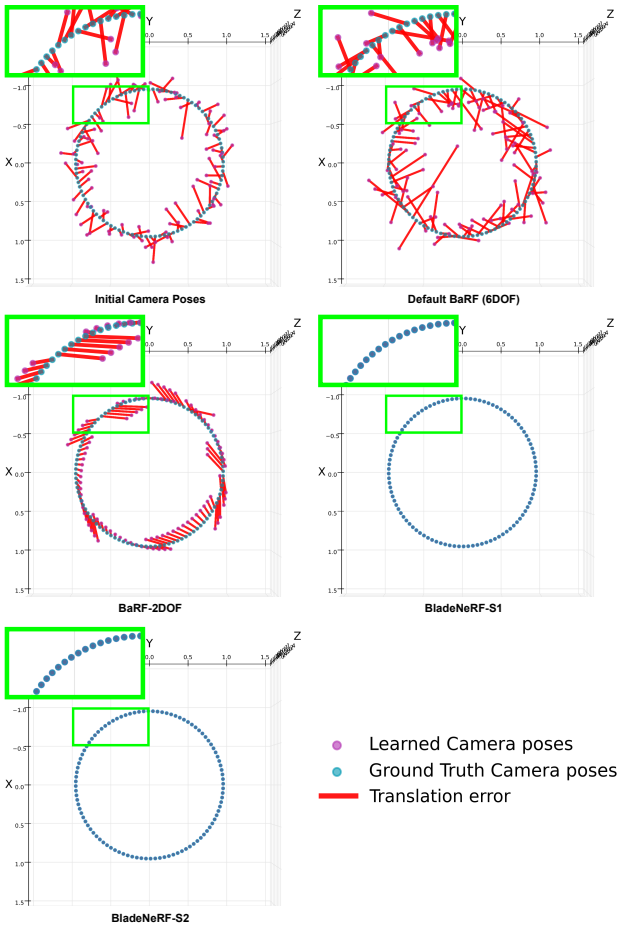


Figure 6. *Top-down visual comparison of the initial and optimized camera poses for our custom dataset. Both BaldeNeRF variants S1 & S2 successfully realign all the camera frames, while both variants of BaRF get stuck at suboptimal solutions, with BaRF-2DOF showing a quasi-periodic pattern. We plot cameras as points instead of frustums to focus solely on the translations along the x- and y-axes as the rotation of the camera is already known in our setting.*

Evaluation criteria. We evaluate camera pose estimation by aligning the optimised poses with ground truth using Procrustes analysis [19, 30], as the poses and the scene are variable up to a 3D similarity transformation. The average translation error resulting from this alignment process, computed across all estimated camera poses, serves as a quantitative measure indicating the accuracy of the registration. Additionally, we report the mean rotation error in degrees when camera rotations are available.

Results. We present our quantitative results on camera estimation in Tab. 1 and visualise the learned camera poses alongside their ground truth values in Fig. 6. In our setting, we find that COLMAP fails to produce any camera poses due to its inability to extract and match robust features across the input images. Moreover, the default BaRF

Method	Translation error (m) ↓	Rotation error (°) ↓
Initial Camera poses	0.1623	N/A
BaRF (default)	0.5066	0.3818
BaRF-2DOF	0.0352	N/A
BaldeNeRF-S1	**0.0027**	N/A
BaldeNeRF-S2	0.0086	N/A

Table 1. *Quantitative results of camera pose estimation. BladeNeRF outperforms BaRF [19] in estimating x- and y-translations. BladeNeRF-S1 demonstrates slightly better performance than BladeNeRF-S2. BaRF-2DOF reduces the translation error compared to BaRF. Default BaRF estimates camera poses with a higher translation error compared to the initial camera poses. Values with N/A represent methods where no rotations of the camera poses are estimated.*

method yields a suboptimal solution, exhibiting high translation and rotation errors. Consequently, the estimated camera poses deviate even further from the ground truth than the initial ones. To address this limitation, we introduce BaRF-2DOF, a variant that simplifies the camera optimisation problem by focusing solely on optimising the x- and y-translations, similar to BladeNeRF. This variant significantly improves over the default BaRF, reducing the error by a factor of 15. However, BaRF-2DOF still struggles to estimate accurate camera poses and shows a quasi-periodic pattern where multiple subsets of learned camera poses deviate collectively from the ground truth.

In contrast, both variants of BladeNeRF successfully estimate precise camera poses. Notably, the BladeNeRF-S1 variant of BladeNeRF outperforms BladeNeRF-S2 in terms of the translation error. This performance disparity arises because BladeNeRF-S1 optimises camera poses for each repetition independently, while BladeNeRF-S2 assumes that the estimated camera poses from the selected repetition produce a sufficiently low translation error. Otherwise, the error propagates to subsequent repetitions after rotation. However, it is essential to note that the computation cost for BladeNeRF-S1 is 20 times higher than that of BladeNeRF-S2 as our dataset has 20 blades (repetitions).

5.3. Exp 2: Learning Scene Representation

We study 3D neural scene representation learning using optimised camera poses from the first stage. We evaluate two BladeNeRF variants using NeRF [24] and Mip-NeRF [3] backbones on our synthetic dataset. We refer to Sec. 5.1 for details about the configuration of the synthetic dataset.

Experimental settings. We conduct a comparative analysis between two popular backbone choices, NeRF [24] and Mip-NeRF [3]. In addition to the vanilla NeRF and Mip-NeRF models, we test the impact of the proposed background loss by integrating this loss term into both models. We compare not only photometric reconstruction but also depth estimation, as well as reconstructed meshes.

Implementation details. For models using NeRF [24] as a backbone, we employ a batch size of 2048 rays, with 64 coordinates sampled in the coarse MLP and an additional 128 coordinates in the fine MLP. The optimisation process converges after approximately 200k iterations, equivalent to around 20 hours of computation time. For models using Mip-NeRF as a backbone, including BladeNeRF, we train for 200k iterations with the scaling factors α and β, introduced in Eq. (7), equal to 1×10^{-1} and 1×10^{-3}, respectively. The batch size is set to 2048. All experiments are conducted on a single NVIDIA 1080 GPU and take approximately 4 hours to train. We use the Adam optimiser [18] for all models with an initial learning rate of 5×10^{-4}. The learning rate decays exponentially to 5×10^{-5} during optimisation. The remaining hyperparameters of the Adam optimiser are set to the default values. To reconstruct the scene meshes, we use an off-the-shelf marching cubes implementation [28] with a threshold of 5 and a resolution of $256 \times 256 \times 64$ to generate the 3D meshes.

Evaluation criteria. To evaluate the quality of RGB reconstructions, we report the three error metrics used by NeRF [24]: PSNR, SSIM [41], and LPIPS [49]. For the depth estimation quality, we report quantitative metrics, including the mean absolute error (MAE), the root mean squared error (RMSE), and the relative absolute difference (AbsRel) between the predicted and the ground truth depth.

Results. We report quantitative results in Tab. 2 and visualise qualitative results in Fig. 7. We first compare the performance of all models on novel view synthesis. Fig. 7 shows that all models can synthesise high-fidelity novel views, as the visual reconstructions resemble the ground truth images. Tab. 2 shows that models with a Mip-NeRF backbone perform marginally better than the NeRF-based models when using the optimised camera poses from the first stage. Comparing BladeNeRF-S1 and BladeNeRF-S2, we find that BladeNeRF-S1 yields substantially better results in terms of PSNR and LPIPS. This coincides with our findings in Sec. 5.2 where BladeNeRF-S1 produces lower translation errors than BladeNeRF-S2. Additionally, we report the performance using the ground truth camera poses. Notably, the Mip-NeRF backbones output high-quality results when the ground truth poses are available, implying that precise camera poses are a key to neural scene representation. However, it is yet unclear why NeRF backbones benefit less from accurate camera poses. Regarding the proposed background loss, we find minor benefits in terms of photometric reconstructions.

Second, we compare the performance of all models on depth estimation and mesh reconstruction. Tab. 2 demonstrates that models utilising a Mip-NeRF backbone outper-

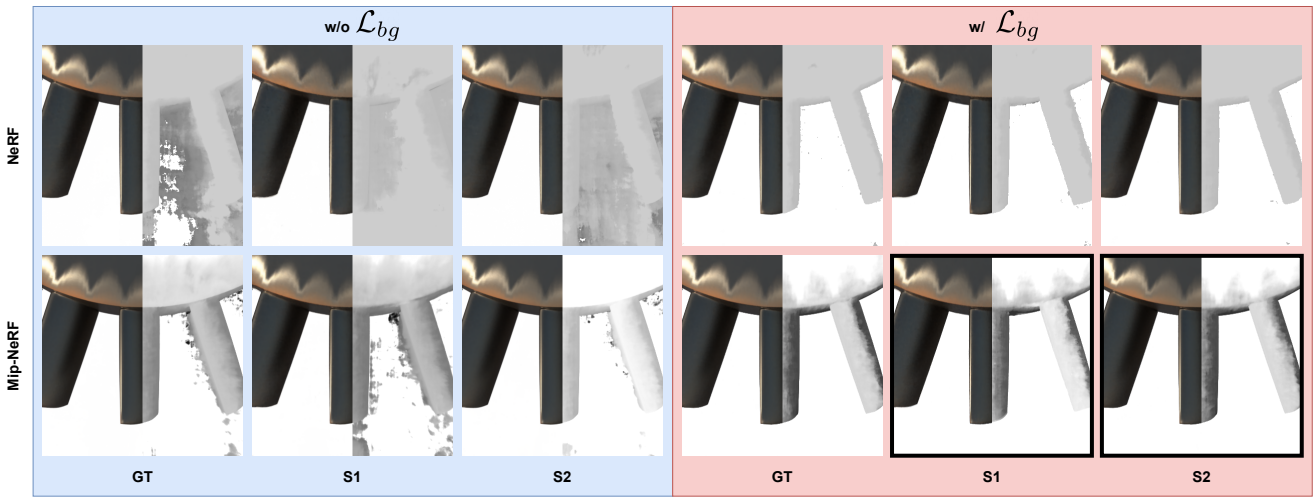


Figure 7. *Qualitative results of BladeNeRF with S1 and S2 camera poses with NeRF and Mip-NeRF as backbones on our synthetic dataset. We use ground truth camera poses for comparison. We visualise the RGB reconstructions (left) and estimated depths (right). The RGB reconstructions for all variants visually look good. Variants with NeRF as backbone struggle heavily with depth estimation in the regions between the blades, while Mip-NeRF gets rid of most of the artefacts. With the background loss $\mathcal{L}_{bg}$, the artefacts in depth estimation between the blades disappear. BladeNeRF S1 and S2 variants are highlighted with black borders.*

Method	RGB quality metrics									Depth quality metrics								
	PSNR↑			SSIM↑			LPIPS↓			MAE↓			RMSE↓			AbsRel↓		
	GT	S1	S2	GT	S1	S2	GT	S1	S2	GT	S1	S2	GT	S1	S2	GT	S1	S2
NeRF w/o $\mathcal{L}_{bg}$	39.99	38.76	34.09	99.62	99.58	99.36	0.163	0.173	0.540	0.687	0.693	0.906	0.961	0.976	1.232	0.812	0.825	0.879
NeRF w/ $\mathcal{L}_{bg}$	40.03	38.80	34.25	99.99	99.66	99.62	0.150	0.152	0.510	0.475	0.675	0.691	0.834	0.912	0.979	0.615	0.764	0.829
Mip-NeRF w/o $\mathcal{L}_{bg}$	46.03	39.85	34.72	99.99	99.99	99.99	0.037	0.134	0.486	0.465	0.549	0.552	0.738	0.899	0.912	0.545	0.625	0.628
Mip-NeRF w/ $\mathcal{L}_{bg}$ (BladeNeRF)	**47.08**	**39.96**	**35.03**	**99.99**	**99.99**	**99.99**	**0.032**	**0.129**	**0.430**	**0.371**	**0.373**	**0.481**	**0.722**	**0.725**	**0.727**	**0.447**	**0.447**	**0.449**

Table 2. *Quantitative results of scene representation. BladeNeRF achieves high-quality RGB and depth estimation with GT, **S1**, and **S2** camera poses. SSIM and LPIPS values are scaled by 100. Mae, RMSE and AbsRel values are scaled by 10.*

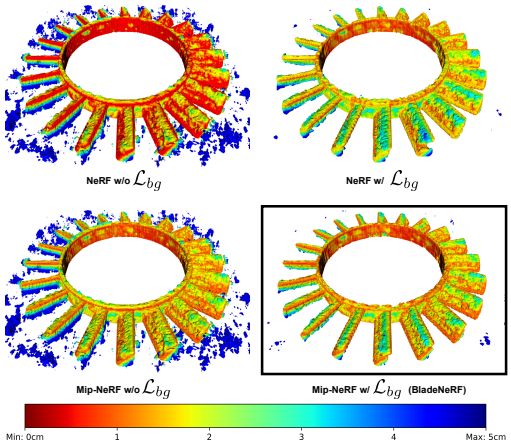


Figure 8. *Qualitative results of mesh reconstruction. Both BladeNeRF (bold borders) and NeRF with the background loss $\mathcal{L}_{\text{bg}}$ reconstruct a sharp mesh reconstruction of the scene. S1 camera poses are used to reconstruct these meshes. To colourise the meshes, we use the Hausdorff distance [12] between reconstructed and ground truth meshes.*

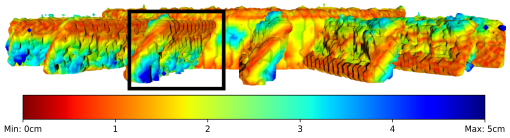


Figure 9. *Mesh reconstruction of the scene using BladeNeRF-S1 viewed from the side. In the black rectangle, an empty region behind the blades is observed, and the blades are reconstructed with an angle even though BladeNeRF does not use any RGB input images of the regions behind the blades. To colourise the mesh, we use the Hausdorff distance [12] between reconstructed and ground truth meshes.*

form NeRF-based models. This difference in performance is also evident in Fig. 7, where Mip-NeRF models exhibit significantly fewer artefacts than NeRF models. The artefacts displayed by Mip-NeRF are primarily confined to the regions surrounding the blades, whereas NeRF models exhibit artefacts spanning most of the background. We hypothesise that this disparity stems from Mip-NeRF's ability to encode information from a larger volume into the colour of a single pixel. In contrast, NeRF encodes the colour

of one pixel based on an infinitely small ray traversing the scene. However, further investigation is needed to confirm this hypothesis.

Moreover, we compare the depth estimation result using optimised camera poses from the first stage. Tab. 2 indicates that BladeNeRF-S1 yields slightly better results than BladeNeRF-S2 camera poses in terms of MAE, RMSE, and AbsRel. However, both produce slightly higher errors than models with ground truth camera poses.

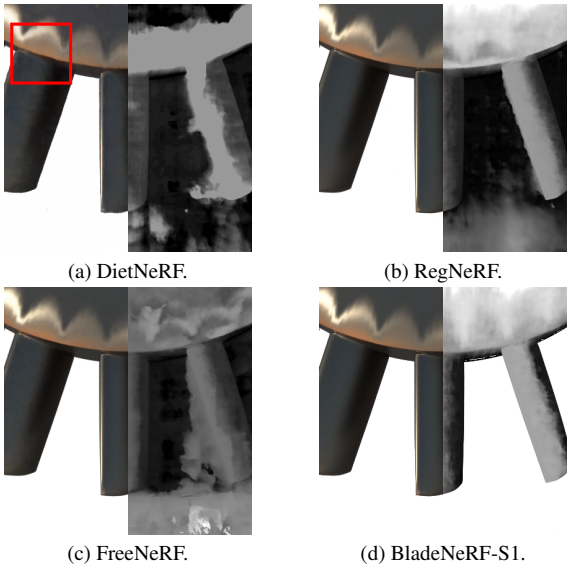
Furthermore, we explore the impact of incorporating background loss in the models. Regardless of their backbones, models with background loss demonstrate significantly improved depth estimation results. Visually, as depicted in Fig. 7 for depth estimation and Fig. 8 for mesh reconstruction, the inclusion of background loss enables the models to eliminate all artefacts in the background for Mip-NeRF models. Nevertheless, some minimal artefacts remain visible for NeRF models. We attribute the elimination of these artefacts to the direct influence of the background loss, which effectively guides the backbone models to learn that the background region has zero density, indicating the absence of any scene components between the blades.

Lastly, our findings highlight BladeNeRF's ability to infer the presence of empty space behind the blades, as depicted in Fig. 9. This is particularly noteworthy because our synthetic dataset solely consists of input images captured from the front of the blades without providing explicit information about the scene behind them. However, BladeNeRF's inferred shape of the blades differs from the actual shape of the blades in the ground truth mesh. This difference is visible in Fig. 9 where parts of the reconstructed mesh close to the actual mesh are coloured in red. We believe that the angle of inclination of the reconstructed blades by BladeNeRF is similar to the angle of the cones cast through the pixels at the edges of the observed images, which can represent parts of the space behind the blades.

5.4. Exp 3: Comparison to Methods For Sparse Views

In our setting, only front-view images are available. In this experiment, we study whether sparse-view NeRF models can produce better results than our model, which is not specifically designed for sparse-view reconstruction. We compare BladeNeRF with other NeRF derivatives designed for sparse views.

Experimental setup and evaluation criteria. We conduct a comparative analysis between our approach and DietNeRF [16], RegNeRF [27], and FreeNeRF [45]. We compare their performance in terms of photometric reconstruction and depth estimation metrics. We use the same metrics for Exp. 2 in Sec. 5.3. All three methods are implemented using their publicly available code bases. We keep all the default parameter settings and train them for 200K iterations on a single NVIDIA 1080 GPU.



(a) DietNeRF. (b) RegNeRF. (c) FreeNeRF. (d) BladeNeRF-S1.

Figure 10. *DietNeRF exhibits suboptimal results with blurry blades (in the red square). The other methods produce high-quality RGB reconstructions. For depth estimation, FreeNeRF, RegNeRF, and DietNeRF produce a background between the blades and cloudy artefacts near the blades, while BladeNeRF produces a clean background and a better depth estimation.*

Method	RGB Quality Metrics			Depth Quality Metrics		
	PSNR↑	SSIM↑	LPIPS↓	MAE↓	RMSE↓	AbsRel↓
DietNeRF [16]	27.59	86.53	0.989	0.734	1.858	0.993
FreeNeRF [45]	31.01	98.83	0.433	0.616	1.680	0.953
RegNeRF [27]	34.04	99.69	0.331	0.452	1.511	0.918
BladeNeRF-S1	**39.96**	**99.99**	**0.129**	**0.373**	**0.725**	**0.447**

Table 3. *Quantitative results of the comparison between DietNeRF, FreeNeRF, RegNeRF and BladeNeRF-S1. FreeNeRF, RegNeRF and BladeNeRF-S1 are able to reconstruct high-fidelity RGB reconstructions while only DietNeRF struggles heavily with RGB reconstruction in our setting. For depth estimation, DietNeRF, FreeNeRF and RegNeRF produce substantially higher errors compared to BladeNeRF-S1. SSIM and LPIPS values are scaled by 100. Mae, RMSE and AbsRel values are scaled by 10.*

Results. We visualise the results of novel view synthesis and depth estimation in Fig. 10. In addition, we show the mesh reconstruction in Fig. 11. RegNeRF, FreeNeRF, and BladeNeRF can reconstruct high-fidelity RGB images. However, DietNeRF exhibits noticeable blurriness in RGB reconstruction. This blurriness is particularly evident in the red square region depicted in Fig. 10. In terms of depth estimation and mesh reconstruction, RegNeRF, DietNeRF, and FreeNeRF exhibit noticeable artefacts, leading to high quantitative evaluation errors as reported in Tab. 3. An interesting observation is that RegNeRF hallucinates a

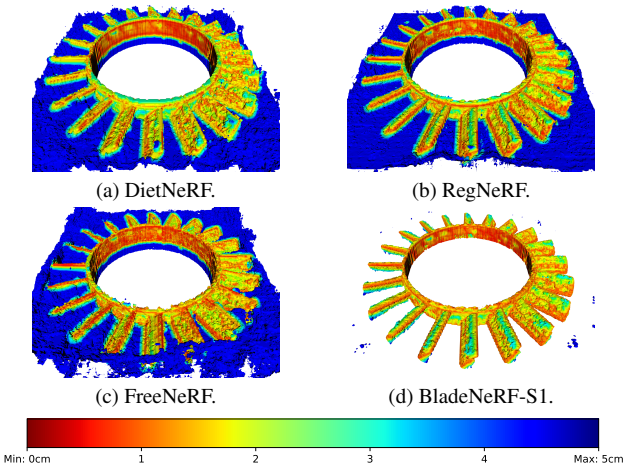


Figure 11. *Comparison of mesh reconstruction quality. All methods, except BladeNeRF, hallucinate a plane-like structure in the background, making the blades blend into the background. However, the background generated by RegNeRF is smoother than the background generated by DietNeRF and FreeNeRF. To colourise the meshes, we use the Hausdorff distance [12] between reconstructed and ground truth meshes.*

smooth plane-like structure in the background while accurately capturing the blade shapes. In contrast, FreeNeRF and DietNeRF struggle to reconstruct the shapes of the blades in our specific scenario reliably. RegNeRF, DietNeRF, and FreeNeRF are primarily designed to learn high-quality scene representations from multiple-angle images, which makes them less effective when working with single-sided scene images like the ones in our case.

5.5. Exp 4: BladeNeRF on Damaged Blades

We test BladeNeRF's ability to reconstruct damaged engine blades. We run our experiment on a synthetic dataset with damaged blades. We use the same experimental setups and evaluation criteria used in Exp. 1 and Exp. 2 (see Secs. 5.2 and 5.3 for details).

Synthetic damaged dataset. We generate a dataset similar to the synthetic dataset introduced in Sec. 5.1. The only difference is the addition of damages to the blades. To generate the damages on the blades, we use BladeSynth [13], a Blender tool that generates synthetic datasets with different types of damages on engine blades. We introduce two types of damages. First, we add 50 dents and deformations on the leading edges of the blades. Second, we chop off parts of two separate blades as shown in Fig. 12.

Camera pose estimation. We report the translation errors between the estimated and ground truth camera poses in Tab. 4. Both variants of BladeNeRF estimate accurate camera poses of the damaged dataset, with BladeNeRF-S1 slightly outperforming BladeNeRF-S2. However, BladeNeRF-S1 exhibits a slight increase in

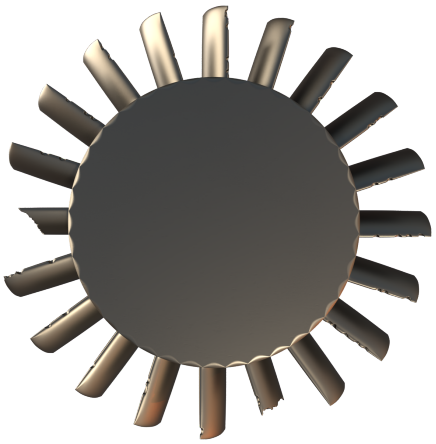


Figure 12. *3D model of the custom damaged dataset. We introduce two types of damages. We add dents of different sizes on the leading edges of the blades, and two blades are missing parts of different sizes.*

Dataset	Method	Translation error (m) ↓
Damaged	BladeNeRF-S1	**0.0053**
	BladeNeRF-S2	0.0084
Synthetic	BladeNeRF-S1	**0.0027**
	BladeNeRF-S2	0.0086

Table 4. *Quantitative results of camera pose estimation of BladeNeRF on the damaged and synthetic datasets. BladeNeRF-S1 demonstrates slightly better performance than BladeNeRF-S2 on both datasets. BladeNeRF-S1 produced a marginally higher translation error on the damaged dataset than the synthetic dataset, while BladeNeRF-S2 exhibits comparable performance on both datasets.*

translation error on the damaged dataset compared to the synthetic dataset. This disparity comes from the repetitions where a part of the blades is chopped off, such as repetition #7. For instance, in the synthetic dataset, repetition #7 displayed a translation error of 0.0023m, whereas, in the damaged dataset, the error increased to 0.0105m.

In contrast, BladeNeRF-S2 demonstrates reduced sensitivity to the damages compared to BladeNeRF-S1 by using a repetition with minimal translation error to estimate camera poses of other repetitions. It is important to note that the repetition selected by BladeNeRF-S2 to estimate the other repetitions has only dents a few dents and no instances of chopped-off blades. The results of BladeNeRF-S2 would have been worse if the selected repetition had a chopped-off blade. In conclusion, dents have a negligible impact on camera pose estimation, while the presence of chopped-off blades significantly affects the accuracy of BladeNeRF, particularly for BladeNeRF-S1. BladeNeRF-S2's strategy allows us to avoid the impact of the chopped-off blades.

Scene representation and 3D reconstruction. We conduct experiments using ground truth camera poses and

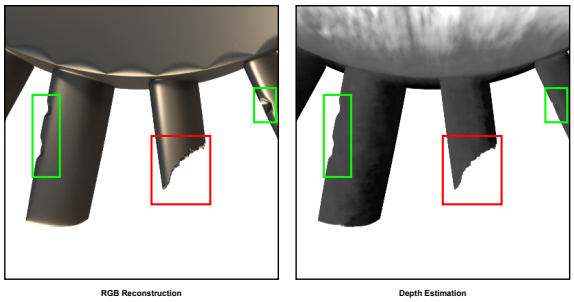


Figure 13. *RGB reconstruction and Depth Estimation from BladeNeRF-S1 on the damaged dataset. The dents (depicted in green) are visible on the RGB reconstruction but are less sharp in the depth estimation. The chopped-off blade (depicted in red) is clearly visible in both RGB and depth estimation.*

Method	RGB Quality Metrics			Depth Quality Metrics		
	PSNR↑	SSIM↑	LPIPS↓	MAE↓	RMSE↓	AbsRel↓
BladeNeRF-GT	46.87	99.99	0.046	0.370	0.724	0.446
BladeNeRF-S1	37.42	99.99	0.348	0.412	0.727	0.449
BladeNeRF-S2	34.88	99.99	0.450	0.438	0.730	0.455

Table 5. *Quantitative results of BladeNeRF on the damaged dataset. BladeNeRF-GT, utilizing ground truth camera poses in our second stage, achieves the highest performance, while BladeNeRF-S1 and BladeNeRF-S2 yield inferior results. Both BladeNeRF-S1 and BladeNeRF-S2 perform at a comparable level. LPIPS values are scaled by 100. Mae, RMSE and AbsRel values are scaled by 10.*

learned camera poses from both variants. The quantitative results are presented in Tab. 5. Our findings indicate that both variants of BladeNeRF yield comparable results in terms of RGB and depth estimation metrics. However, BladeNeRF-S1 exhibits a slight advantage over BladeNeRF-S2, consistent with the outcomes reported in Exp. 2 in Sec. 5.3. BladeNeRF achieves high-fidelity reconstructions of dents and chopped-off blades in RGB and depth estimations, as shown in Fig. 13. Nevertheless, BladeNeRF only successfully reconstructs big-size defects such as chopped-off blades when reconstructing the 3D mesh, as shown in Fig. 14. Meanwhile, the smaller dents on the leading edges remain imperceptible in the resulting mesh. Given that the damages are clearly visible in RGB and depth estimations, we attribute BladeNeRF's inability to reconstruct dents on the 3D mesh to the low resolution used on the marching cube algorithm. We believe that using a higher resolution could potentially enable the recovery of dents and more minor defects on the blades.

6. Conclusion

We present BladeNeRF, a new NeRF variant specifically designed for the photo-realistic reconstruction of aircraft engines from unposed RGB images. Our model benefits from two designs: (1) we incorporate prior knowledge of

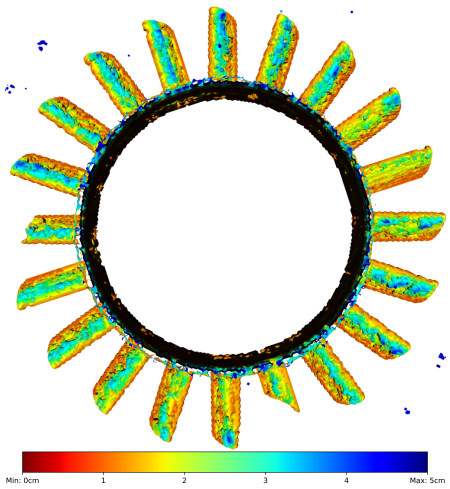


Figure 14. *Mesh reconstruction of BladeNeRF-S1 on the damaged dataset. The dents on the leading edges of the blades are not discernible due to the limitations of the low-resolution marching cube algorithm. However, BladeNeRF-S1 demonstrates accurate reconstruction of the chopped-off blades. To colourise the mesh, we use the Hausdorff distance [12] between reconstructed and ground truth meshes.*

camera constraints into the BaRF method to enhance camera pose estimation; (2) we extend Mip-NeRF by integrating binary masks that separate the foreground and background regions to guide the model to learn high-fidelity scene representation in both foreground and background.

One limitation of our method is its dependence on the specific settings employed in our study. The assumptions made during the first stage restrict BladeNeRF's generalizability to diverse inspection scenarios. For example, the method is not adaptable to images captured from varying distances or trajectories beyond a circular path. Relaxing these assumptions in future research would be valuable, enabling 3D mesh reconstruction beyond the scenario considered in our study.

Furthermore, our evaluation has only been conducted on synthetic data, as we encountered challenges in testing the approach on real data due to the lack of suitable datasets that conform to the specific assumptions imposed in camera pose estimation. To overcome this limitation, we propose two potential avenues for future work. One approach involves collecting real-world data that adheres to the assumptions made by our method, thereby enabling the validation of BladeNeRF in a realistic context. Alternatively, relaxing the assumptions would allow the utilization of available real data, broadening the applicability of BladeNeRF and facilitating evaluation in practical scenarios. It is important to note that extracting binary masks from real images is a non-trivial task and poses an additional challenge when applying BladeNeRF to real datasets.

Moreover, BladeNeRF exhibits a notable limitation in terms of its long training time. In time-sensitive aircraft inspection scenarios, efficiency is of utmost importance. To address this limitation, integrating faster NeRF models into the BladeNeRF framework represents a promising avenue for future research. Leveraging these models, known for their shorter training times, would reduce the computational burden and enhance the practicality of BladeNeRF in the domain of aircraft inspection, particularly in near-real-time analysis requirements.

Additionally, BladeNeRF could benefit from an end-to-end approach to simultaneously estimate the camera poses and learn the scene representation. Using an end-to-end pipeline, BladeNeRF can correct the camera poses if it produces high reconstruction errors. This approach can potentially reduce the second stage's sensitivity to the accuracy of the camera poses produced by the first stage.

Overall, our method demonstrates the successful utilization of neural radiance fields (NeRF) for accurate 3D reconstruction in the domain of aircraft engine inspections. Through empirical experiments, BladeNeRF showcases its capabilities, encompassing accurate camera pose estimation and learning high-fidelity scene representations.

References

[1] Sameer Agarwal, Noah Snavely, Ian Simon, Steven M. Seitz, and Richard Szeliski. Building rome in a day. In *2009 IEEE 12th International Conference on Computer Vision*, pages 72–79, 2009. 2

[2] Hatem Alismail, Brett Browning, and Simon Lucey. Photometric bundle adjustment for vision-based slam. In Shang-Hong Lai, Vincent Lepetit, Ko Nishino, and Yoichi Sato, editors, *Computer Vision – ACCV 2016*, pages 324–341, Cham, 2017. Springer International Publishing. 2

[3] Jonathan T. Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P. Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 5855–5864, October 2021. 2, 3, 5, 7

[4] Wenjing Bian, Zirui Wang, Kejie Li, Jiawang Bian, and Victor Adrian Prisacariu. Nope-nerf: Optimising neural radiance field with no pose prior. *CVPR*, 2023. 1, 2

[5] Anpei Chen, Zexiang Xu, Fuqiang Zhao, Xiaoshuai Zhang, Fanbo Xiang, Jingyi Yu, and Hao Su. Mvsnerf: Fast generalizable radiance field reconstruction from multi-view stereo. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14124–14133, 2021. 2

[6] Yue Chen, Xingyu Chen, Xuan Wang, Qi Zhang, Yu Guo, Ying Shan, and Fei Wang. Local-to-global registration for bundle-adjusting neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8264–8273, 2023. 1, 2

[7] Ziang Cheng, Hongdong Li, Yuta Asano, Yinqiang Zheng, and Imari Sato. Multi-view 3d reconstruction of a texture-less smooth surface of unknown generic reflectance. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16226–16235, 2021. 1

[8] Julian Chibane, Aayush Bansal, Verica Lazova, and Gerard Pons-Moll. Stereo radiance fields (srf): Learning view synthesis from sparse views of novel scenes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2021. 2

[9] Shin-Fang Chng, Sameera Ramasinghe, Jamie Sherrah, and Simon Lucey. Garf: Gaussian activated radiance fields for high fidelity reconstruction and pose estimation, 2022. 2

[10] Andrew J. Davison, Ian D. Reid, Nicholas D. Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):1052–1067, 2007. 2

[11] Kangle Deng, Andrew Liu, Jun-Yan Zhu, and Deva Ramanan. Depth-supervised NeRF: Fewer views and faster training for free. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022. 2

[12] David A. Edwards. The structure of superspace. In Nick M. Stavrakas and Keith R. Allen, editors, *Studies in Topology*, pages 121–133. Academic Press, 1975. 8, 10, 11, 16, 18, 19, 20

[13] Chengming Feng. Bladesynth: Damage detection and assessment in aircraft engines with synthetic data. Master's thesis, Delft University of Technology, The Netherlands, Aug 2022. 10

[14] Ronghang Hu, Nikhila Ravi, Alexander C. Berg, and Deepak Pathak. Worldsheet: Wrapping the world in a 3d sheet for view synthesis from a single image. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2021. 2

[15] Tao Hu, Liwei Wang, Xiaogang Xu, Shu Liu, and Jiaya Jia. Self-supervised 3d mesh reconstruction from single images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6002–6011, June 2021. 2

[16] Ajay Jain, Matthew Tancik, and Pieter Abbeel. Putting nerf on a diet: Semantically consistent few-shot view synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 5885–5894, October 2021. 2, 9

[17] M. M. Johari, Y. Lepoittevin, and F. Fleuret. Geonerf: Generalizing nerf with geometry priors. *Proceedings of the IEEE international conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. 3

[18] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR*, 2015. 6, 7

[19] Chen-Hsuan Lin, Wei-Chiu Ma, Antonio Torralba, and Simon Lucey. Barf: Bundle-adjusting neural radiance fields. In *IEEE International Conference on Computer Vision (ICCV)*, 2021. 1, 2, 3, 4, 6, 7, 14

[20] Manolis I. A. Lourakis and Antonis A. Argyros. Sba: A software package for generic sparse bundle adjustment. *ACM Trans. Math. Softw.*, 36(1), mar 2009. 1

[21] David G Lowe. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1150–1157. Ieee, 1999. 1

[22] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–110, 2004. 1

[23] Priyanka Mandikal and R. Venkatesh Babu. Dense 3d point cloud reconstruction using a deep pyramid network. *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1052–1060, 2019. 2

[24] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020. 1, 2, 3, 7

[25] Raúl Mur-Artal, J. M. M. Montiel, and Juan D. Tardós. Orb-slam: A versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015. 2

[26] K. L. Navaneet, A. Mathew, S. Kashyap, W. Hung, V. Jampani, and R. Venkatesh Babu. From image collections to point clouds with self-supervised shape and pose networks. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1129–1137, Los Alamitos, CA, USA, jun 2020. IEEE Computer Society. 2

[27] Michael Niemeyer, Jonathan T. Barron, Ben Mildenhall, Mehdi S. M. Sajjadi, Andreas Geiger, and Noha Radwan. Regnerf: Regularizing neural radiance fields for view synthesis from sparse inputs. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022. 3, 9

[28] Pablo Márquez Neila. PyMCubes: marching cubes for Python, 2023. `https://github.com/pmneila/PyMCubes`, version 0.1.4. 7

[29] Johannes L. Schonberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. 1, 6

[30] Peter H Schönemann. A generalized solution of the orthogonal procrustes problem. *Psychometrika*, 31(1):1–10, 1966. 6

[31] Johannes L. Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4104–4113, 2016. 2

[32] S.M. Seitz and C.R. Dyer. Photorealistic scene reconstruction by voxel coloring. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1067–1073, 1997. 2

[33] Rajvi Shah, Aditya Deshpande, and P. J. Narayanan. Multi-stage SFM: A coarse-to-fine approach for 3d reconstruction. *CoRR*, abs/1512.06235, 2015. 1, 2

[34] Meng-Li Shih, Shih-Yang Su, Johannes Kopf, and Jia-Bin Huang. 3d photography using context-aware layered depth inpainting. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. 2

[35] Vincent Sitzmann, Julien N.P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. In *Proc. NeurIPS*, 2020. 2

[36] Vincent Sitzmann, Semon Rezchikov, William T. Freeman, Joshua B. Tenenbaum, and Fredo Durand. Light field networks: Neural scene representations with single-evaluation rendering. In *Proc. NeurIPS*, 2021. 2

[37] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. In *Advances in Neural Information Processing Systems*, 2019. 2

[38] Noah Snavely, Steven M. Seitz, and Richard Szeliski. Photo tourism: Exploring photo collections in 3d. *ACM Trans. Graph.*, 25(3):835–846, jul 2006. 1, 2

[39] Bill Triggs, Philip F. McLauchlan, Richard I. Hartley, and Andrew William Fitzgibbon. Bundle adjustment - a modern synthesis. In *Workshop on Vision Algorithms*, 1999. 1

[40] Shubham Tulsiani, Richard Tucker, and Noah Snavely. Layer-structured 3d scene inference via view synthesis. In *ECCV*, 2018. 2

[41] Zhou 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, 2004. 7

[42] Zirui Wang, Shangzhe Wu, Weidi Xie, Min Chen, and Victor Adrian Prisacariu. NeRF−−: Neural radiance fields without known camera parameters. *arXiv preprint arXiv:2102.07064*, 2021. 1, 2

[43] Yi Wei, Shaohui Liu, Yongming Rao, Wang Zhao, Jiwen Lu, and Jie Zhou. Nerfingmvs: Guided optimization of neural radiance fields for indoor multi-view stereo. In *ICCV*, 2021. 2

[44] Yitong Xia, Hao Tang, Radu Timofte, and Luc Van Gool. Sinerf: Sinusoidal neural radiance fields for joint pose estimation and scene reconstruction. *arXiv preprint arXiv:2210.04553*, 2022. 1, 2

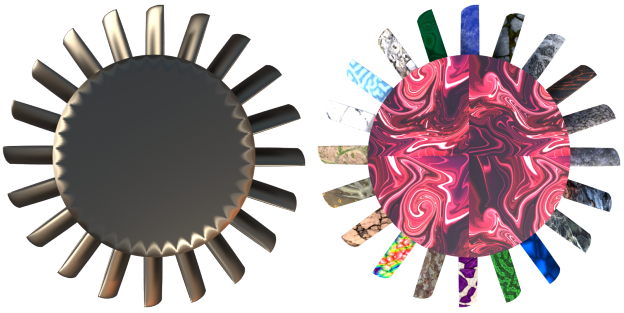
[45] Jiawei Yang, Marco Pavone, and Yue Wang. Freenerf: Improving few-shot neural rendering with free frequency regularization. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2023. 3, 9

[46] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelNeRF: Neural radiance fields from one or few images. In *CVPR*, 2021. 2

[47] Zhongda Yuan and Mingguang Liu. Specification for engine borescope inspection report. *IOP Conference Series: Earth and Environmental Science*, 186(5):012001, sep 2018. 1

[48] Jason Y. Zhang, Gengshan Yang, Shubham Tulsiani, and Deva Ramanan. NeRS: Neural reflectance surfaces for sparse-view 3d reconstruction in the wild. In *Conference on Neural Information Processing Systems*, 2021. 2

[49] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018. 7

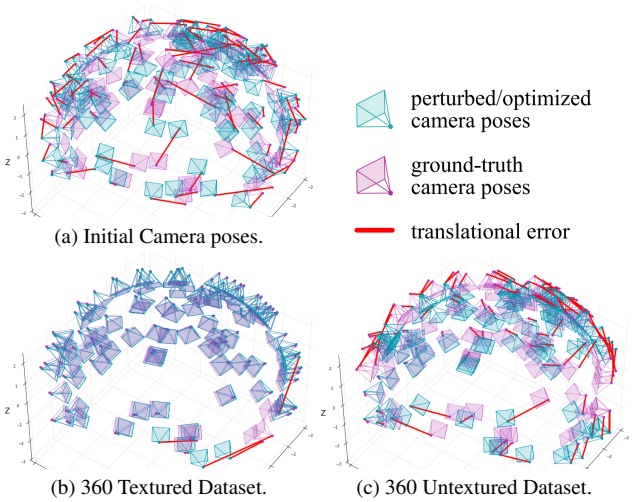


(a) 360 Untextured Dataset.. (b) 360 Textured Dataset.

Figure 15. *Top down view of the blades used to create 360 Textured and 360 Untextured Dataset.*

Dataset	Translation error (m) ↓	Rotation error (°) ↓
360 Untextured	0.42210	13.236
360 Textured	**0.05995**	**1.978**

Table 6. *Quantitative results of camera pose estimation. BaRF is able to correctly estimate the camera poses of the textured dataset compared to the untextured dataset.*



(a) Initial Camera poses.

(b) 360 Textured Dataset. (c) 360 Untextured Dataset.

Figure 16. *Visual comparison of the initial and optimized camera poses for the 360 Textured and 360 Untextured datasets. BaRF is able to correctly estimate most of the camera poses on the textured dataset but finds a suboptimal solution for the untextured dataset.*

A. BaRF on texture-less and repetitive blades

We explore BaRF's ability [19] to estimate camera poses from images that display repetitive visual patterns and lack textures when captured from various angles.

Datasets. To perform this exploratory experiment, we create two variants of our datasets (360 Textured Dataset and 360 Untextured Dataset). Both datasets employ the same 3D object of the blades. The creation process of these datasets mirrors that of the synthetic dataset introduced in Sec. 5.1. The only distinction is that the camera is situated on a hemisphere with a radius of 3 meters and oriented towards the centre of the blades. We refer to Fig. 15 for a visual representation.

Experimental setup. We use the same experimental setup as BaRF [19] and run the experiment for 200k iterations to estimate the camera poses.

Evaluation criteria. We report the same quantitative metrics to evaluate the quality of the estimated camera poses as reported in BaRF: the translation error in meters and the rotation errors in degrees.

Results. We present the camera pose estimation results of BaRF for both the textured and untextured datasets in Fig. 16. Tab. 6 compares the estimated camera poses to the ground truth camera poses by measuring translation and rotation errors. BaRF struggles to accurately estimate camera poses on the untextured dataset but performs well on the textured one. However, it faces difficulties with the side-view images of the blades on the textured dataset due to their visual similarity from all sides.

B. Relaxing the assumptions for pose estimation

This experiment studies the performance of BladeNeRF in two more challenging settings. Firstly, we aim to remove the restriction on the rotation around the z-axis. By doing so, we eliminate the assumption of a constant speed of rotation for the camera, and the requirement for equidistant image captures along the circular pattern in the first stage of BladeNeRF. Secondly, we aim to recover all six degrees of freedom (DoF) in BladeNeRF, thereby removing all assumptions regarding camera poses.

Experimental setup. We compare BaRF and BladeNeRF models to evaluate the impact of relaxing camera assumptions. To ensure a comprehensive evaluation, we test three versions of each model that estimate different DoF. The first variant estimates 2 DoF, allowing for translation along the x- and y-axes. The second version estimates 3 DoF, which includes rotation around the z-axis, in addition to x- and y-translations. The third version estimates all 6 DoF, accounting for rotation and translation across the x-, y-, and z-axes. This comparison analysis helps us understand how relaxing camera assumptions impact the performance and capabilities of BladeNeRF.

Evaluation and implementation details. We use a similar setup to Exp. 1 in Sec. 5.2 and report the rotation

Method	DOF	Translation Error (m) ↓	Rotation Error (°) ↓
BaRF	2	0.0352	-
	3	0.0578	**0.2055**
	6	0.5066	0.3818
BladeNeRF-S1	2	**0.0027**	-
	3	0.5855	0.7122
	6	1.996	2.1748
BladeNeRF-S2	2	0.0086	-
	3	0.8442	1.3478
	6	2.9788	1.7921

Table 7. *Quantitative results of comparison between BladeNeRF and BaRF. We compare three variants of each model, namely estimating two, three and six degrees of freedom (DoF). BladeNeRF outperforms BaRF in the case of two DoF. However, both BladeNeRF's variants produce substantially higher translation and rotation errors in the case of three and six DoF. BladeNeRF-S1 outperforms BladeNeRF-S2 across all DoF.*

and translation errors between the optimized and the ground truth camera poses.

Results. In Tab. 7, we present the quantitative results obtained from our evaluation. As outlined in Sec. 5.2, both variants of BladeNeRF substantially outperform the default BaRF method in the case of 2 DoF. However, for the case of 3 DoF, the default BaRF method outperforms both BladeNeRF variants. This discrepancy arises from BladeNeRF's inability to effectively learn the rotation around the z-axis from the input images in each repetition.

Moreover, in the case of 6 DoF, the default BaRF method consistently outperforms both BladeNeRF variants. Merely relaxing the assumptions imposed on BladeNeRF yields suboptimal results, necessitating further exploration of alternative approaches for precise camera pose estimation.

Additionally, it is interesting to note that BladeNeRF-S1 consistently outperforms BladeNeRF-S2 across all degrees of freedom, which aligns with the outcomes of Exp. 1 in Sec. 5.2. This suggests that independent learning of camera poses for each repetition, as implemented in BladeNeRF-S1, yields better performance than assuming uniformity across all repetitions, as employed in BladeNeRF-S2.

C. Different Camera Trajectories

We test BladeNeRF's ability to learn the camera poses and scene representation of datasets with camera trajectories other than the circular trajectory. To this end, we synthesize two custom datasets. The camera follows a sinusoidal trajectory in the first dataset, referred to as the sinusoidal dataset. It oscillates from a distance of 0.75 meters from the centre of the blades to a distance of 1.1 meters every ten images. Half a period of the camera trajectory represents one repetition in this dataset. In the second dataset,

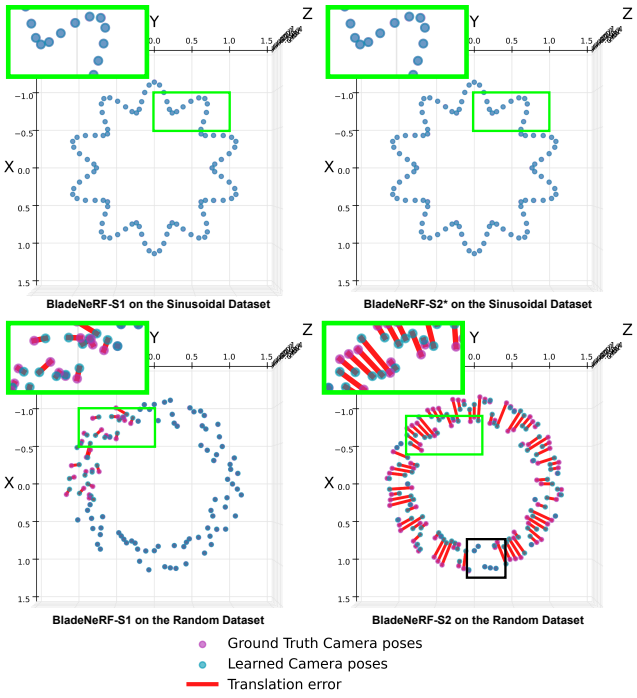


Figure 17. *Top-down visual comparison of the optimized and ground truth camera poses of the sinusoidal and random datasets using BladeNeRF's first stage. Both variants of BladeNeRF accurately estimated camera poses on the sinusoidal dataset. However, we use BladeNeRF-S2*, a variant of BladeNeRF-S2, that reasons about the camera pose in terms of periods on the camera trajectory instead of repetitions. On the other hand, BladeNeRF-S1 found suboptimal camera poses for a few repetitions on the random dataset, while BladeNeRF-S2 estimated suboptimal camera poses for all repetitions except the one (in the black rectangle) used to generalize to the rest of the repetitions.*

referred to as the random dataset, the camera is randomly put at a distance between 0.75 meters and 1.1 meters from the centre of the blades. The camera rotates in the counterclockwise direction around the z-axis in both datasets at a constant speed.

Camera pose estimation. We present the quantitative results of camera estimation using BladeNeRF on the random, sinusoidal, and synthetic datasets in Tab. 8. We also visualise all three datasets' estimated and ground truth camera poses in Fig. 17.

In the sinusoidal dataset, each repetition covers only half a period of the sinusoidal camera trajectory. Therefore, generalizing the estimated camera poses from one repetition to the rest leads to incorrect camera pose estimations for half of the repetitions. To address this, we introduce BladeNeRF-S2*, a modified camera optimization strategy of BladeNeRF-S2 that estimates camera poses for two consecutive repetitions, covering one complete period of the sinusoidal trajectory. The learned camera poses from one

Dataset	BladeNeRF Variant	Translation Error (m) ↓
Sinusoidal	S1	0.0121
	S2*	0.0149
Random	S1	0.3134
	S2	0.4270
Synthetic	S1	**0.0027**
	S2	0.0086

Table 8. *Quantitative results of camera pose estimation on the sinusoidal, random and synthetic datasets. The translation errors of both variants of BladeNeRF on the random dataset are higher by approximately a factor of 50 compared to the synthetic dataset. BladeNeRF-S1 and BladeNeRF-S2* are able to estimate the camera poses but produce higher translation errors on the sinusoidal dataset compared to the synthetic dataset. BladeNeRF-S2* is a variant of BladeNeRF-S2 that reasons about camera poses in terms of periods on the camera trajectory instead of repetitions.*

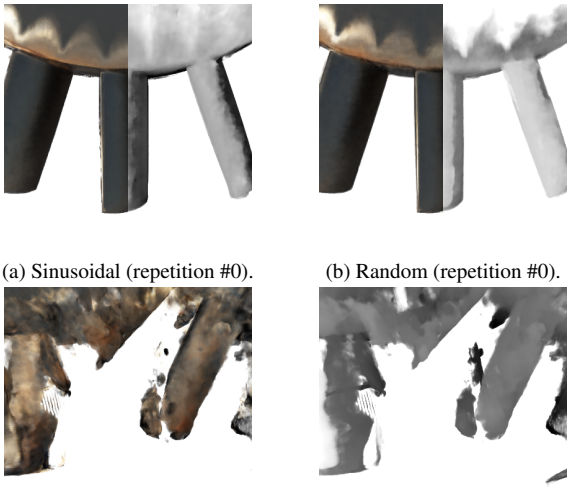
period are then generalized to the remaining periods, similar to BladeNeRF-S2's approach with repetitions.

Both BladeNeRF-S1 and BladeNeRF-S2* successfully estimate camera poses on the sinusoidal dataset, although they have slightly higher translation errors compared to the synthetic dataset. We believe that BladeNeRF can learn camera poses on datasets with periodic camera trajectories, but adjustments may be required to accommodate the specific setting of the dataset.

On the random dataset, BladeNeRF-S1 accurately estimates camera poses for most repetitions but has found suboptimal solutions for a few consecutive repetitions. These repetitions are in a region with slightly brighter lighting conditions. We hypothesize that the bright lighting influenced BladeNeRF-S1's ability to learn accurate camera poses. In the case of BladeNeRF-S2, it predictably produces higher translation errors as it assumes uniform patterns across all repetitions, which does not hold in the random dataset.

Scene representation and 3D reconstruction. The quantitative results of RGB reconstructions and depth estimations on the sinusoidal and random datasets are presented in Tab. 9. Visualisations of the RGB reconstructions and depth estimations for both datasets are depicted in Fig. 18. Additionally, mesh reconstructions of both datasets using BladeNeRF-S1 are shown in Fig. 19.

BladeNeRF-S1 can learn the scene representation on the sinusoidal dataset. However, it produces noticeable blurriness in the RGB and depth estimations, leading to significant artefacts in the mesh reconstructions, particularly in the regions in front of the mesh, as seen in Fig. 19. This suboptimal scene representation on the sinusoidal dataset can be attributed to the marginally inaccurate camera poses learned during the first stage of BladeNeRF.



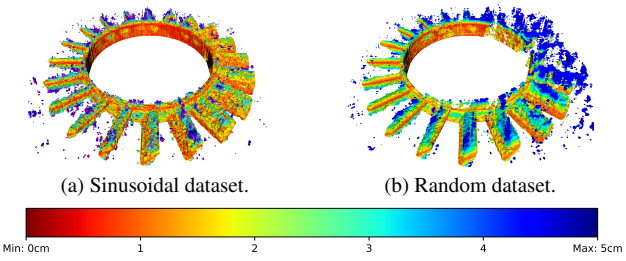
(a) Sinusoidal (repetition #0). (b) Random (repetition #0).

(c) Random (repetition #12) RGB Reconstruction. (d) Random (repetition #12) Depth Estimation.

Figure 18. *RGB and Depth estimation of the random and sinusoidal datasets using BladeNeRF-S1. At repetition #0, BladeNeRF-S1 produces blurry RGB and depth estimation on random and sinusoidal dataset. At repetition #12, BladeNeRF-S1 fails to learn the scene representation.*

Dataset	RGB Quality Metrics			Depth Quality Metrics		
	PSNR↑	SSIM↑	LPIPS↓	MAE↓	RMSE↓	AbsRel↓
Sinusoidal	**24.93**	**92.61**	**3.4429**	**3.768**	**7.336**	**0.453**
Random	19.84	82.15	13.253	3.949	7.644	0.474

Table 9. *Qualitative results of BladeNeRF-S1 on the sinusoidal and random datasets. BladeNeRF-S1 achieves low-quality RGB and depth estimation on both datasets with notably poorer results observed on the random dataset. LPIPS and SSIM values are scaled by 100. Mae, RMSE and AbsRel values are scaled by 10*



(a) Sinusoidal dataset. (b) Random dataset.

Figure 19. *Mesh reconstruction of the sinusoidal and random datasets using BladeNeRF-S1. On the random dataset, BladeNeRF-S1 fails to reconstruct accurately the mesh in the regions where the camera poses have high translation errors. On the sinusoidal dataset, BladeNeRF-S1 reconstructs a complete mesh, albeit with significant artefacts on the top side of the blades. To colourise the mesh, we use the Hausdorff distance [12] between reconstructed and ground truth meshes.*

Dataset	BladeNeRF Variant	Translation Error (m) ↓
40 Views	S1	0.0069
	S2	0.0119
60 Views	S1	0.0034
	S2	0.0094
100 Views	S1	0.0027
	S2	0.0086
200 Views	S1	**0.0025**
	S2	0.0087

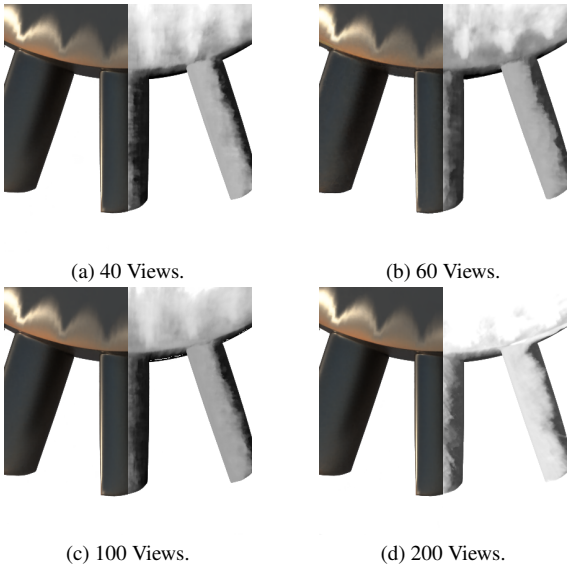
Table 10. *Quantitative results of comparison of camera pose estimation between the datasets with 40, 60, 100 and 200 views. Both variants of BladeNeRF perform similarly on the 100 and 200 views datasets, with BladeNeRF-S1 achieving the lowest translation error on the 200 views dataset. In contrast, both variants of BladeNeRF produce high errors with the 60 and 40 views datasets. The translation error in estimating camera poses increases as the number of views fed to BladeNeRF decreases.*

Regarding the random dataset, BladeNeRF-S1 successfully recovers RGB and depth estimation for repetition #0, where the camera poses are accurately estimated by the first stage of BladeNeRF. However, for repetition #12, where the first stage produces inaccurate camera poses, BladeNeRF-S1 fails to learn the scene representation. This leads to the mesh reconstruction exhibiting significant artefacts in the regions surrounding repetitions with the inaccurately estimated camera poses. These observations highlight the sensitivity of the second stage of BladeNeRF to the inaccuracies introduced by the first stage of BladeNeRF.

D. Effect of Number of Views

We investigate the effect of the number of input images on the performance of BladeNeRF. To this end, we synthesize three custom datasets using the same configuration as the synthetic dataset introduced in Sec. 5.1. We run BladeNeRF on the three custom datasets and compare their performance to the synthetic dataset. The camera on all four datasets follows a circular trajectory with a radius of 0.9 meters from a distance of 1 meter along the z-axis in the counterclockwise direction as depicted in Fig. 5. The only difference between the four datasets is the number of images taken along the circular trajectory of the camera. For simplicity, we refer to the synthetic dataset as the 100 views dataset since it has 100 views as input. The input images are captured every 9, 6, 3.6 and 1.8 degrees of rotation around the z-axis for the 40, 60, 100 and 200 views datasets.

Camera pose estimation. We present the quantitative results of camera pose estimation using both variants of BladeNeRF on the 40, 60, 100 and 200 views datasets in Tab. 10. For the 100 and 200 views dataset, both variants of BladeNeRF produce comparable translation errors. This is likely due to the presence of sufficient visual information in both datasets, where each repetition contains five or ten input views, enabling the underlying BaRF model to learn accurate camera poses. Doubling the input views results in a high overlap of the field of view of the input images and does not produce better results in terms of translation errors. In contrast, when applied to the 40 and 60 views datasets, both variants of BladeNeRF exhibit higher translation errors, with the lowest performance observed on the 40 views dataset. The increase in translation error arises from the limited overlap in the field of view among the input images in these datasets, hindering BaRF's ability to learn accurate camera poses.



(a) 40 Views. (b) 60 Views. (c) 100 Views. (d) 200 Views.

Figure 20. *Visual comparison of BladeNeRF on the synthetic dataset with different numbers of input images. BladeNeRF can estimate high-quality RGB reconstructions (right) and depth estimations (left).*

Dataset	RGB Quality Metrics			Depth Quality Metrics		
	PSNR↑	SSIM↑	LPIPS↓	MAE↓	RMSE↓	AbsRel↓
40 Views	24.05	82.99	1.135	0.981	1.754	1.284
60 Views	25.66	85.23	1.067	0.917	1.719	1.153
100 Views	**39.96**	**99.99**	0.129	**0.373**	**0.725**	0.447
200 Views	37.89	**99.99**	**0.091**	0.379	0.714	**0.449**

Table 11. *Quantitative results of scene representation on the 40, 60, 100 and 200 views datasets. BladeNeRF demonstrates high-quality RGB and depth estimation results on the 100 and 200 views datasets. However, its performance is comparatively poorer on the 40 and 60 views datasets on RGB and depth estimation. LPIPS and SSIM values are scaled by 100. Mae, RMSE and AbsRel values are scaled by 10.*

Scene representation and 3D reconstruction. The RGB and depth estimation results of BladeNeRF-S1 on the 40, 60, 100, and 200 views datasets are presented in Fig. 20. Additionally, mesh reconstructions for all four datasets are

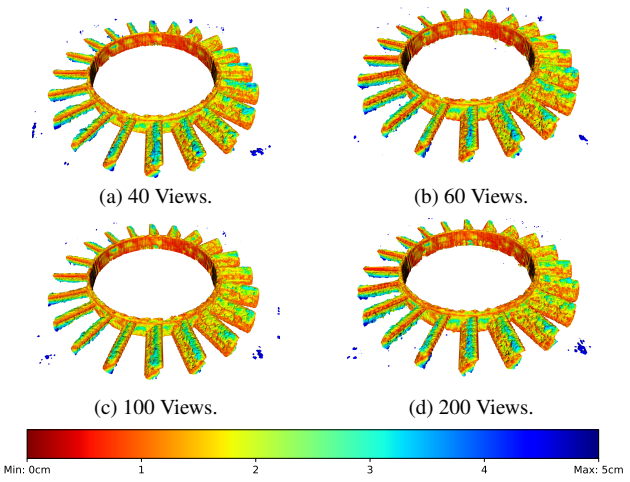


Figure 21. *Mesh reconstruction of the 40, 60, 100 and 200 Datasets using BladeNeRF-S1. BladeNeRF-S1 can estimate accurate camera poses on different numbers of views. To colourise the mesh, we use the Hausdorff distance [12] between reconstructed and ground truth meshes.*

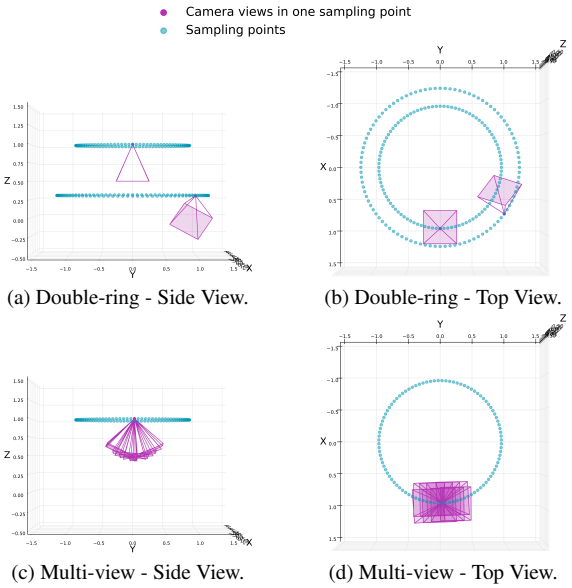


Figure 22. *The visualisation presents the camera's spatial positioning and orientation in the multi-view and double-ring datasets. In the multi-view dataset, a circular pattern is employed, encompassing multiple sampling points (blue). At each sampling point, 8 RGB images are generated by rendering the scene from distinct camera orientations (pink). The double-ring dataset involves the utilization of two circular patterns. In the upper pattern, an individual RGB image is rendered at each sampling point (blue), with the camera facing the blades directly (similar to the synthetic dataset). Conversely, the lower circular pattern entails the rendering of one RGB image per sampling point (pink), while the camera is rotated by 25 degrees around the x-axis and -35 degrees around the y-axis.*

displayed in Fig. 21. Detailed quantitative results can be found in Tab. 11. BladeNeRF-S1 can effectively learn scene representations on all four datasets since no significant discrepancies are observed upon qualitative examination of the RGB reconstructions, depth estimations, and mesh reconstructions across all four datasets. However, in terms of quantitative evaluation, the 40 and 60 views datasets exhibit slightly worse results than the 100 and 200 views datasets. The 40 views dataset produces the least optimal results among the four datasets. It is worth noting that the quantitative results of the 200 views dataset closely resemble those of the 100 views dataset. This experiment suggests that adding more views does not necessarily improve the scene representation while reducing the number of input views results in reduced performance for BladeNeRF.

E. Multiple viewing angles during inspection

In this section, we present two alternative experimental setups to evaluate whether the setup in BladeNeRF produces the best results in learning the scene representation. We create unique custom datasets for both setups, distinct from the synthetic dataset introduced in Sec. 5.1.

E.1. Additional custom datasets

Multi-view dataset. In this experimental setup, we simulate an inspection scenario where the camera follows a circular pattern in the counterclockwise direction with a radius of 0.9 meters and a height of 1 meter. RGB images are captured at different angles around the x- and y-axes at each sampling point along this circular pattern, with a maximum rotation of 30 degrees. Fig. 22 visualises eight camera poses sampled at the same location on the circular pattern.

Dataset	# of sampled points on the circle	# of viewing directions	# of input RGB images
Multi-view #1	200	4	800
Multi-view #2	100	8	800
Multi-view #3	50	16	800
Multi-view #4	25	32	800
Synthetic	200	1	200

Table 12. *The characteristics of all variants of the multi-view dataset and the synthetic dataset introduced in Sec. 5.1.*

To evaluate the effectiveness of this experimental setup, we create four distinct variants of the multi-view dataset for the training set, as described in Tab. 12. The testing and validation sets remain consistent with the synthetic dataset, ensuring a standardized evaluation process.

Double-ring dataset. In this dataset, the camera moves along two circular patterns in the counterclockwise direction. The first circle has a radius of 0.9 meters, and the camera is positioned 1 meter high along the z-axis, facing the blades, similar to the synthetic dataset. The second circle is bigger, with a radius of 1.2 meters, and the camera is positioned at a height of 0.3 meters. The camera is also

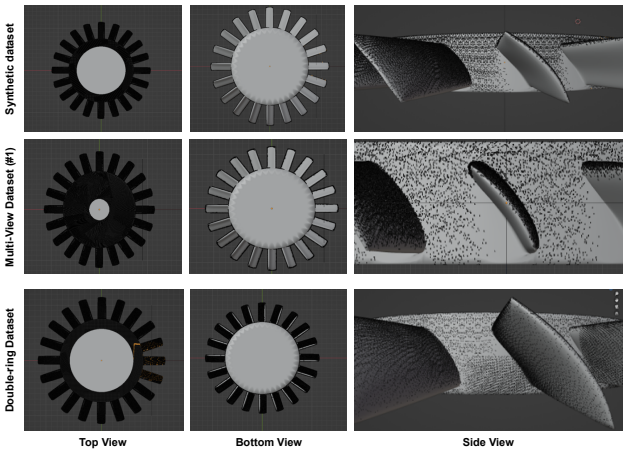


Figure 23. *Coverage of the multi-view and double-ring datasets. Both the multi-view (Variant #2) and synthetic datasets have full coverage of the top side of the blades. However, neither has any visual data of the regions located behind the blades or the backside of the blades. In contrast, the double-ring dataset captures RGB images of both the regions behind the blades and the backside of the blades.*

tilted 25 degrees around the x-axis and -35 degrees around the y-axis. This setup allows BladeNeRF to capture views of the blades from above and at an angle, which provides important information about the space behind the blades.

Coverage of the datasets. The primary distinction between the newly created custom datasets and the synthetic dataset is the visible regions to the cameras within the training set. We develop a Blender tool that projects 400×400 rays from the camera's origin through the input RGB images to show this difference. When a ray intersects any part of the blades, we mark the point of intersection with a black dot. For convenience, we refer to the collection of all intersection points as the *coverage* of a training set. This coverage represents the visible portions of the blades in at least one RGB image from the training set. Fig. 23 shows the coverage of the synthetic dataset, the multi-view dataset (variant #2), and the double-ring dataset.

E.2. Results

We evaluate the effectiveness of BladeNeRF on both the multi-view and double-ring datasets in improving scene representation. We use ground truth cameras and only execute the second stage of BladeNeRF. The qualitative results in Fig. 25 show that both setups produce high-quality RGB reconstructions. However, depth estimation and mesh reconstruction exhibit noisy artefacts without background loss. BladeNeRF on the multi-view and the double-ring datasets outperforms BladeNeRF on the synthetic dataset, particularly in mesh reconstruction. They accurately capture the shape and reduce ambiguities behind them, as shown in Fig. 24. BladeNeRF on the multi-view dataset

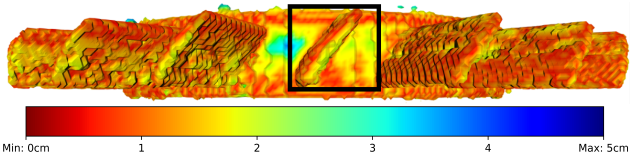


Figure 24. *Side view of mesh reconstruction of the multi-view dataset (Variant #1). In the black rectangle, the convex shape of the mesh is visible due to the multiple viewing angles available during training. This is the primary advantage of the multi-view and double-ring over the synthetic dataset used in the main paper. To colourise the mesh, we use the Hausdorff distance [12] between reconstructed and ground truth meshes.*

achieves better results than the double-ring dataset in depth estimation and mesh reconstruction due to its use of multiple random views compared to the double-ring dataset's limited viewing directions. No significant performance differences are observed among the four variants of the multi-view dataset. While both custom datasets provide better scene representations, adapting the first stage of BladeNeRF to these setups is not feasible. Further research is to explore alternative methods for learning camera poses in these new experimental setups.

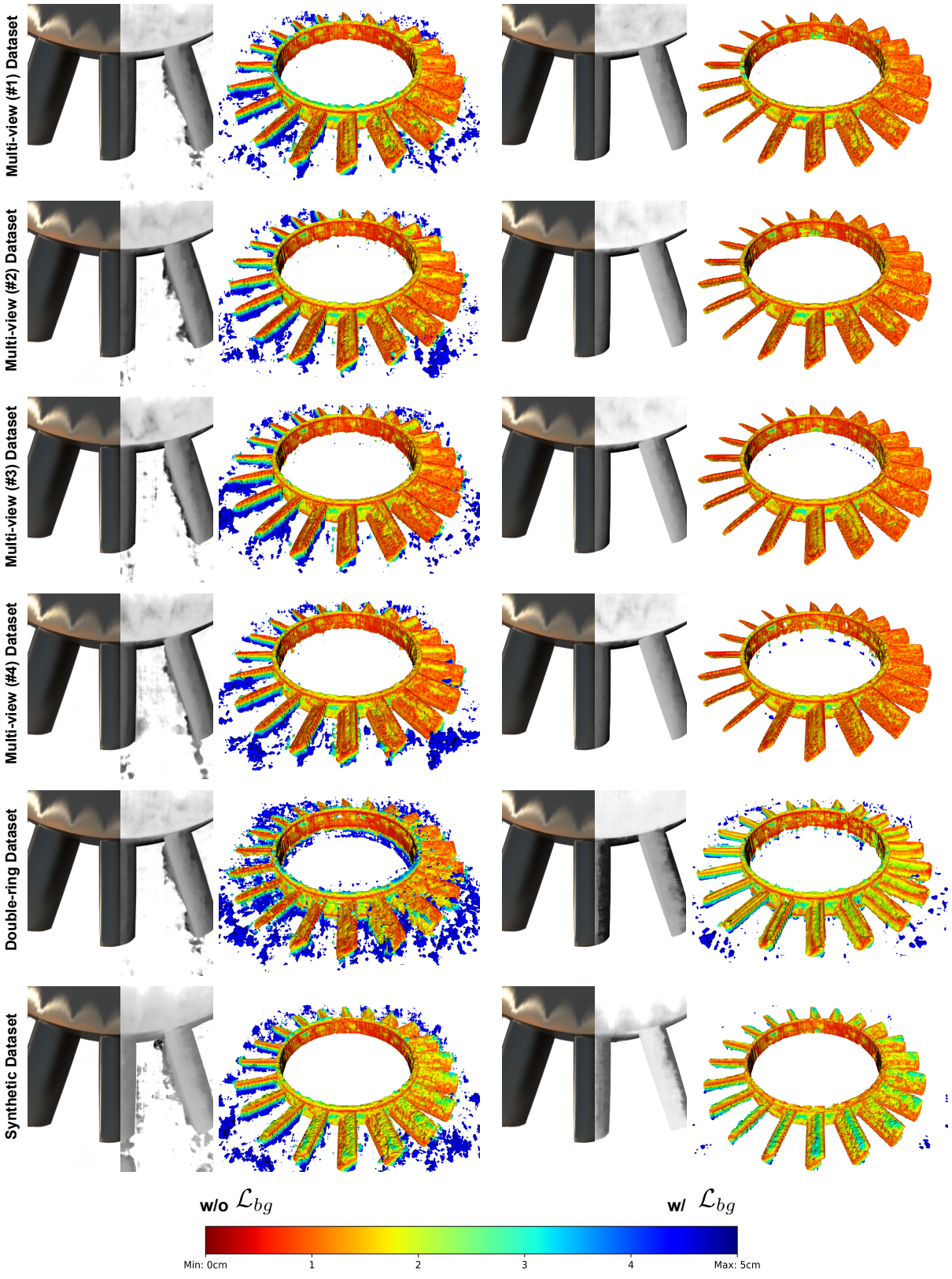


Figure 25. *BladeNeRF's second stage reconstructed RGB, depth, and mesh using ground truth camera poses from the multi-view, double-ring and synthetic datasets. BladeNeRF successfully estimates the RGB values for all datasets but faces challenges in depth estimation when $\mathcal{L}_{\text{bg}}$ is not utilized. It is worth noting that using $\mathcal{L}_{\text{bg}}$ results in sharper blade details for both the Multi-view and Double-ring datasets compared to the Synthetic dataset used in the main paper. To colourise the meshes, we use the Hausdorff distance [12] between reconstructed and ground truth meshes.*

3

BACKGROUND ON DEEP LEARNING

3.1. DEEP LEARNING

Deep learning is a sub-field of machine learning. It imitates the way humans think using neural networks. The latter extracts information from data by forming a relationship between input and output. A neural network is a connected directed graph structure where each node, also called a neuron, performs some simple computation [3]. Each neuron gets an input x and performs an affine transformation as depicted in Equation 3.1. Then, it applies an activation function f shown in Equation 3.2 to produce the output y. A mathematical model of such a neuron is shown in Fig. 3.1.

$$u = \sum_{i=1}^{n} w_i x_i + b \tag{3.1}$$

$$y = f(u) \tag{3.2}$$

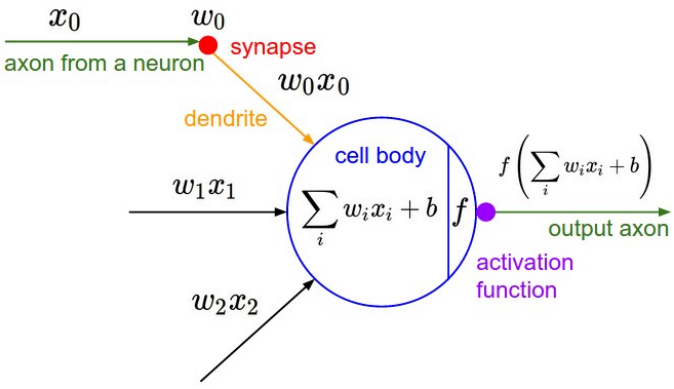


Figure 3.1: *Mathematical model of a single neuron in an artificial neural network [4].*

A deep neural network is composed of multiple neural networks stacked on each other, also known as an artificial neural network. Each layer is composed of multiple neurons.

In general, deep neural networks have an input layer, one or multiple hidden layers and an output layer as shown in Fig. 3.2. Neurons in one layer are connected to neurons in the previous and next layers. Within the same layer, neurons are generally not connected. This general architecture is known as a Multi-Layer Perceptron (MLP). Each connection between neurons shares information using the weights and biases as parameters. These parameters are trained and updated using back-propagation, also known as training the network.

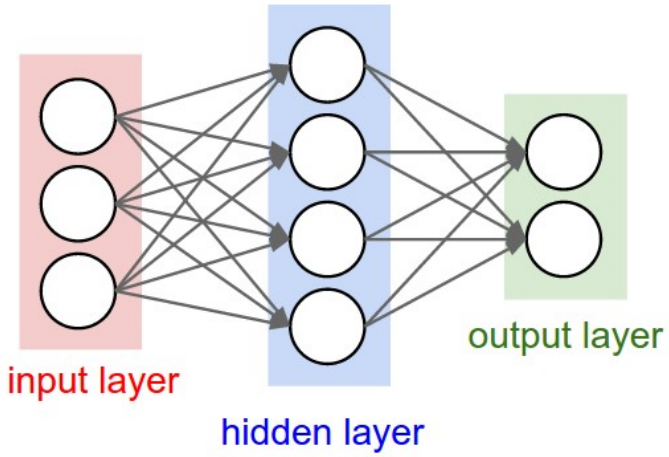


Figure 3.2: *A 3-layer neural network with one input layer of 3 neurons, one hidden layer of 4 neurons and one output layer with 2 neurons [4].*

3.2. ACTIVATION FUNCTIONS

The goal of deep neural networks is to learn relations between input and output that can be non-linear. As shown in Section 3.1, a linear transformation is the first operation a neuron applies on the input. Therefore, the neural network needs to have a source of non-linearity. Activations functions (also known as non-linearities) are applied to the neurons' output to introduce a non-linear element to the neural network. The most common activation functions are Sigmoid, ReLU and Tanh functions [5] shown in Fig. 3.3. Since activation functions directly affect a neural network's performance, they must be carefully chosen. For example, the Sigmoid function suffers from slow convergence and vanishing gradients [6]. Therefore, the activation function needs to be picked according to the architecture of the network and the task at hand.

3.3. TRAINING A NEURAL NETWORK

An objective, also known as a loss function, is set to train a deep neural network. The neural network updates its trainable parameters, i.e., the weights and biases (generally initialized randomly), to minimize the loss function. This function is computed by calcu-

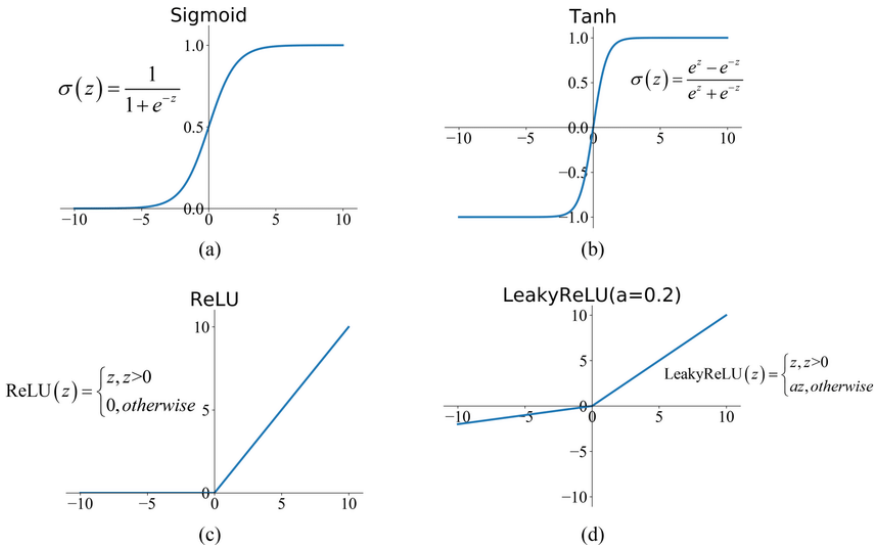


Figure 3.3: *Most commonly used activation functions [5].*

lating the difference between the predicted value by the network and the ground truth. The mean squared loss is a commonly used loss function [7]. To optimize the neural network's parameters, gradient descent can be used. However, the training data used in deep learning is huge, which makes naively using gradient descent computationally inefficient. Fortunately, we can use back-propagation on Stochastic Gradient Descent [8]. An important aspect of using gradient descent methods is deciding the size of the steps to take while optimizing the objective function. Several techniques are used in deep learning, such as Adam optimizer [9], RmsProp [10], and Momentum [11].

4

Background on Neural Radiance Fields

Neural Radiance Fields (NeRFs) [2] have gained significant attention in computer vision, sparking a surge in research publications since the release of the original paper in 2020. This chapter explains how Neural Radiance Fields and two notable follow-up publications work.

4.1. Neural Fields

The concept of a neural field was popularized by Xie et al. [12], referring to a neural network that serves as a parameterization for a signal. This signal can be a single 3D scene or object but can also represent other discrete or continuous signals, such as audio or images. Neural fields are commonly used in computer graphics and computer vision, specifically in image synthesis and 3D reconstruction, which is the main topic of this thesis.

Most neural field variants use fully connected neural networks to encode the features of objects or scenes. It is important to note that training involves a single network that captures a specific single scene. Unlike traditional machine learning methods, the goal is intentionally overfitting the neural network to the scene. This means that neural fields capture the unique characteristics of the scene within the network weights, resulting in a representation tailored specifically to that scene.

Interestingly, "neural fields" derives from the "field" concept in physics, which describes a quantity defined across spatial and/or temporal coordinates. In this context, a field is commonly represented as a mapping from a coordinate, denoted as x, to a corresponding quantity, denoted as y, which can be a scalar, vector, or tensor. Examples of fields encompass gravitational fields and electromagnetic fields, among others.

4.1.1. ADVANTAGES OF NEURAL FIELDS

Traditional approaches for storing 3D scenes often rely on voxel grids or polygon meshes. However, both methods have inherent limitations. Voxel grids tend to be resource-intensive, resulting in significant storage costs. On the other hand, polygon meshes are restricted to representing rigid surfaces. In contrast, neural fields provide efficient and compact 3D representations of objects due to their differentiable and continuous nature. Additionally, neural fields offer the advantage of arbitrary dimensions and resolutions, accommodating diverse requirements. Moreover, they exhibit domain-agnostic characteristics and do not rely on specific input, further enhancing their versatility.

4.1.2. TRAINING NEURAL FIELDS

A general framework is commonly followed to obtain a representation of a scene in neural fields. This involves sampling the coordinates of the scene and inputting them into a neural network, which generates corresponding field quantities. Next, the produced field quantities, such as 2D RGB images, are mapped to the sensor's domain. Finally, a reconstruction error is calculated, and the neural network is optimized accordingly. This iterative process allows neural fields to learn and refine their scene representations effectively.

4.2. NERF

4.2.1. MAIN OBJECTIVE OF NERF

Neural Radiance Fields **(NeRF)**, proposed by Mildenhall et al. [2], is a continuous representation of scenes with complex geometry and materials from a set of RGB input images and their camera poses [1] taken from multiple angles. The main idea is to encode inside one multi-layer perceptron (MLP) an instance-specific implicit representation of a scene. Using this representation, novel views can be synthesized consistently with the training set of views, as shown in Fig. 4.1.

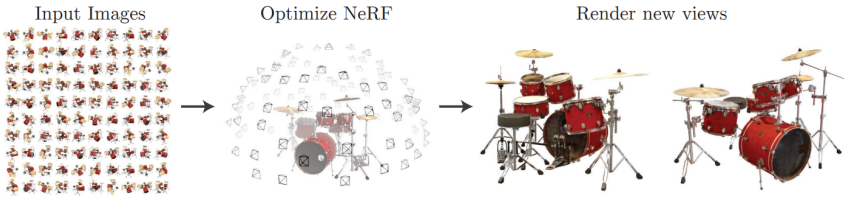


Figure 4.1: *The main idea behind NeRF. From a set of RGB images taken from multiple viewing points, NeRF trains a neural network to learn the scene representation, which allows to synthesize novel views. Source: [2]*

[1] A camera pose is a representation of the camera's location and its orientation.

4.2.2. Training a NeRF model

NeRF learns the representation of a 3D scene by optimizing the weights Θ of a multi-layer perceptron F_Θ. It renders the colour of each pixel of a camera by projecting a ray $\boldsymbol{r}(t) = \boldsymbol{o} + t\boldsymbol{d}$ from the camera's centre of projection $\boldsymbol{o} \in \mathbb{R}^3$ along the direction $\boldsymbol{d} \in \mathbb{R}^3$ to pass through the centre of the pixel. $\boldsymbol{N}$ samples are sampled between the near and far planes of the camera t_n and t_f, respectively, to form a vector of sorted distances $\boldsymbol{t}$, as shown in Fig. 4.2.

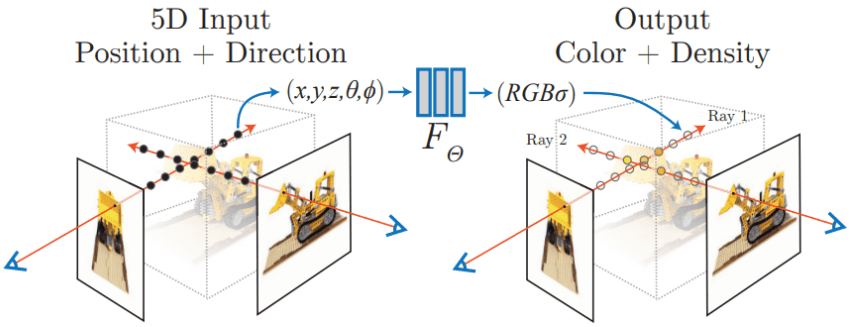


Figure 4.2: *An overview of our NeRF's scene representation and differentiable rendering procedure. NeRF synthesizes images by sampling 5D coordinates (location and viewing direction) along camera rays (left diagram), feeding those locations into an MLP to produce a colour and volume density (right diagram). Source: [2]*

For each distance $t_k \in \boldsymbol{t}$, its position in the scene $\boldsymbol{x} = \boldsymbol{r}(t_k)$ is computed then transformed to a positional encoding (PE) where the i-th frequency encoding $\gamma_i(\boldsymbol{x})$ is:

$$\gamma_i(\boldsymbol{x}) = \left[\sin(2^i \boldsymbol{x}), \cos(2^i \boldsymbol{x})\right] \tag{4.1}$$

where L is a hyperparameter.

The positional encoding allows the MLP to behave as an interpolation function, which critically influences the performance of NeRF [2]. The positional encodings of each ray are fed as input to the MLP F_Θ to output a colour $c \in \mathbb{R}^3$ and a density $\tau \in \mathbb{R}$:

$$\forall\, t_k \in \boldsymbol{t}, \quad F_\Theta : \gamma(r(t_k)) \rightarrow (c_k, \tau_k). \tag{4.2}$$

The viewing direction is also an input of F_Θ, omitted for simplicity. Using numerical quadrature [13] and the output values of F_Θ, the final predicted color of the pixel $C(\boldsymbol{r}, \boldsymbol{t})$ is approximated using the following equation:

$$C(\boldsymbol{r}, \boldsymbol{t}) = \sum_{k=1}^{N} T_k (1 - exp(-\tau_i \delta_k)) \boldsymbol{c_k}, \tag{4.3}$$

where:

$$T_k = exp(-\sum_{j=1}^{j-1} \tau_j \delta_j) \tag{4.4}$$

represents the probability that the ray travels from t_n to t_f without hitting any other part of the scene, and $\delta_k = t_{k+1} - t_k$ is the distance between consecutive samples. More details on how to render the pixels' colours are discussed in Chapter 5.

To optimize the parameters of NeRF, the mean squared error differences between the predicted pixels and the pixels of a set of observed images and their camera poses are minimized using gradient descent. To efficiently sample distances $\boldsymbol{t}$, NeRF trains a "coarse" and a "fine" MLP simultaneously. First, 64 evenly spaced distances $\boldsymbol{t}$ with stratified sampling form the vector $\boldsymbol{t^c}$ used for the "coarse" MLP. Next, the weights

$$w_k = \sum_{k=1}^{N} T_k(1 - exp(-\tau_i \delta_k) \tag{4.5}$$

produced by the "coarse" model are taken as a piecewise constant PDF describing the distribution of the visible scene content, and 128 new $\boldsymbol{t}$ values are drawn from that PDF using inverse transform sampling to produce $\boldsymbol{t^f}$. The union of both $\boldsymbol{t}$ values are sorted and used for the "fine" MLP. The loss of NeRF then becomes:

$$\begin{aligned} L = \sum_{\boldsymbol{r} \in R} \Big(& \| C(\boldsymbol{r}, \boldsymbol{t^c}) - C^*(\boldsymbol{r}) \|^2 \\ & + \| C(\boldsymbol{r}, sort(\boldsymbol{t^f} \cup \boldsymbol{t^c}) - C^*(\boldsymbol{r}) \|^2 \Big) \end{aligned} \tag{4.6}$$

where R is the set of all rays across all ground truth images and $C^*(\boldsymbol{r})$ is the ground truth pixel colour.

4.2.3. LIMITATIONS OF NERF

NeRF achieves high-fidelity results as presented in the original paper [2]. However, it also has multiple limitations. We present some of NeRF's limitations and a few proposed solutions to each limitation.

- **Reliance on accurate camera poses**: Accurate camera poses for all input RGB images are a prerequisite for NeRF. Unfortunately, obtaining these camera poses is not feasible in many scenarios where NeRF could be highly beneficial. Therefore, multiple papers proposed methods to jointly optimize the camera poses and a NeRF model, such as Bundle-Adjusting Neural Radiance Fields [14].

- **Inability to learn scene representation on few images**: NeRF relies heavily on a substantial quantity of posed images as input data. However, specific scenarios may only offer a sparse collection of images. When training a NeRF model with limited input, the presence of artefacts becomes more pronounced. These artefacts stem from errors in estimating scene geometry and the occurrence of divergent behaviour during the initial training phases. Multiple methods were proposed to solve this issue, such as RegNeRF [15], DietNeRF [16] and FreeNeRF [17]. These methods use external signals to improve the learning process of NeRF, making it more robust and improving the accuracy of scene representation even with sparse input data.

- **NeRF cannot handle multi-resolution input images**: Training NeRF on input images with varying resolutions often leads to pronounced artefacts [18]. The straightforward approach of supersampling NeRF by employing multiple rays per pixel proves impractical due to the considerable computational overhead. Each ray requires querying a multilayer perceptron multiple times, rendering this solution inefficient. Mip-NeRF [18] introduced an alternative approach to address this challenge. It proposes using a cone, rather than a single ray, to render the colour of each pixel. This approach enables the integration of multiple resolutions within a single representation, mitigating the artefacts associated with training NeRF on images with different resolutions.

- **NeRF works only on static scenes**: NeRF, in its original form, encounters limitations when attempting to learn the scene representation of dynamic scenes. However, subsequent research efforts have addressed this challenge. One of these efforts is D-NeRF [19], which incorporates temporal and motion factors into the scene representation. D-NeRF uses a deformable volumetric function to model and represent dynamic scenes, overcoming the limitations of the original NeRF framework.

- **NeRF's training takes a long time**: The original NeRF implementation requires an extensive computation time of over 15 hours to learn the representation of a single scene. Recognizing this limitation, subsequent research efforts have been dedicated to addressing this issue. Notably, derivative works of NeRF have emerged, offering significantly accelerated learning times, allowing the representation of a scene to be acquired within seconds, such as Instant NGP [20].

4.3. BaRF

BaRF [14] extends NeRF to solve NeRF's reliance on accurate camera poses. It allows the recovery of the camera poses of the input images by simultaneously optimizing for registration and reconstruction, shown in Fig. 4.3.

BaRF can recover all six degrees of freedom [2] of the camera pose. It proposes a simple strategy for coarse-to-fine registration by applying a smooth mask on the positional encoding from low to high-frequency bands. The changed positional encodings are calculated as follows:

$$\gamma(\boldsymbol{x}, \alpha) = w_i . \left[\sin(2^i \boldsymbol{x}), \cos(2^i \boldsymbol{x})\right] \tag{4.7}$$

where the weight w_i is:

$$w_i(\alpha) = \begin{cases} 0 & \text{if } \alpha < 1 \\ \dfrac{1 - \cos((\alpha - i)\pi)}{2} & \text{if } 0 \leq \alpha - i \leq 1 \\ 1 & \text{if } 1 \leq \alpha - i \end{cases} \tag{4.8}$$

[2] BaRF learns three translations and three rotations around the x, y and z-axes

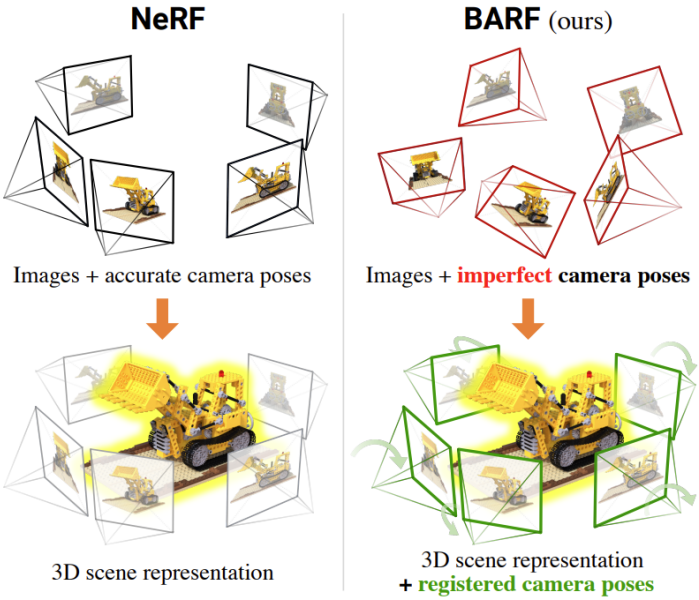


Figure 4.3: *Training NeRF requires accurate camera poses for all images. While BARF can learn 3D scene representations from imperfect (or even unknown) camera poses by jointly optimizing for registration and reconstruction Source: [14]*

and $\alpha \in [0, L]$ is a parameter proportional to the optimization progress. This schedule allows BaRF first to learn the camera poses on smooth signals (when $\alpha = 0$) and then move to learn a high-fidelity scene representation (when $\alpha = L$). The schedule in Equation 4.8 allows the camera poses to converge early during training to their optimal places in space.

4.4. MIP-NERF

Mip-NeRF [18] proposes an approach to train a NeRF model on multi-resolution images. Unlike NeRF, Mip-NeRF uses volumetric frustums along a cone or cylinder to overcome NeRF's aliasing and sampling problems, as shown in Fig. 4.4. Mip-NeRF proposes three main changes to NeRF: Integrated positional encodings, the usage of only one MLP with a slightly changed sampling strategy, and a tweaked loss function.

4.4.1. INTEGRATED POSITIONAL ENCODING (IPE)

A sorted vector of distances $\boldsymbol{t}$ is defined to split the ray into a set of intervals $T_i = [t_i, t_{i+1})$. For each interval, the mean and covariance of a Gaussian approximation $\boldsymbol{(\mu, \Sigma) = r(T_i)}$ of the conical frustum corresponding to the interval are calculated. Their radii are based on the pixel size on the image plane and the ray's focal length. The mean and covariance values are used to create a feature vector using an integrated positional en-

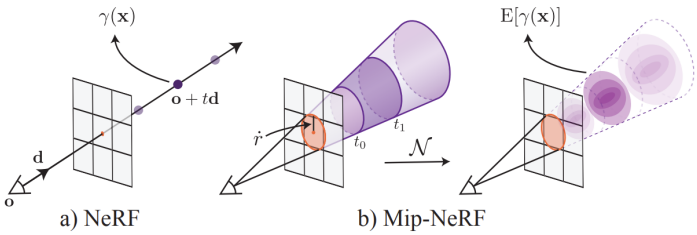


Figure 4.4: *NeRF (a) samples points along rays cast from the camera centre of projection through each pixel, then encodes those points with a positional encoding (PE) γ. Mip-NeRF (b) reasons about the 3D conical frustum defined by a camera pixel. These conical frustums are featurized with the integrated positional encoding (IPE), which works by approximating the frustum with a multivariate Gaussian and then computing the (closed form) integral $E[\gamma(x)]$ over the positional encodings of the coordinates within the Gaussian. Source: [18]*

coding:

$$\gamma(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \left\{ \begin{bmatrix} \sin(2^l \boldsymbol{\mu}) \exp(-2^{2l-1} diag(\boldsymbol{\Sigma})) \\ \cos(2^l \boldsymbol{\mu}) \exp(-2^{2l-1} diag(\boldsymbol{\Sigma})) \end{bmatrix} \right\}_{l=0}^{L-1} \tag{4.9}$$

where L is a hyperparameter. Similarly to NeRF, the integrated positional encodings are fed into the MLP to render the pixels' colours of all rays.

4.4.2. EFFICIENT SAMPLING

Mip-NeRF uses only one MLP F_Θ but does query it twice in a hierarchical sampling strategy. First, the ray is rendered using sorted evenly-spaced "coarse" distances $\boldsymbol{t^c}$ that are sampled from a uniform distribution spanning the distance between the camera's near and far planes t_n and t_f:

$$\boldsymbol{t^c} \sim U[t_n, t_f], \quad \boldsymbol{t^c} = \text{sort}(\{t^c\}) \tag{4.10}$$

Once the "coarse" weights $\boldsymbol{w^c}$ are calculated, "fine" distances $\boldsymbol{t^f}$ are sampled using inverse transform sampling from the histogram defined by $\boldsymbol{t^c}$ and $\boldsymbol{w^c}$:

$$t^f \sim \text{hist}(\boldsymbol{t^c}, \boldsymbol{w^c}), \quad \boldsymbol{t^f} = \text{sort}(\{t^f\}) \tag{4.11}$$

This slightly changed strategy improves the sampling efficiency and allows the MLP to concentrate on scene contents.

4.4.3. LOSS FUNCTION

Mip-NeRF's loss function is a weighted combination between the "fine" and "coarse" reconstruction losses:

$$L = \sum_{\boldsymbol{r} \in R} \left(\|C(\boldsymbol{r}, \boldsymbol{t^c}) - C^*(\boldsymbol{r})\|^2 + \frac{1}{10} \|C(\boldsymbol{r}, \boldsymbol{t^f} - C^*(\boldsymbol{r})\|^2 \right) \tag{4.12}$$

where R is the set of all rays across all ground truth images and $C^*(\boldsymbol{r})$ is the ground truth pixel colour.

5

RENDERING AND 3D RECONSTRUCTION TECHNIQUES

This chapter provides an introduction to volume rendering techniques. It is important to have this background information to understand how Neural Radiance Fields (NeRFs) can generate new views by casting rays through the scene and recovering RGB colours and densities from different perspectives. Additionally, we discuss the marching cubes algorithm, which allows us to create a polygonal mesh of the scene once we have learned its representation using NeRF models.

5.1. VOLUME RENDERING

Direct volume rendering techniques enable the generation of visual representations of three-dimensional volumetric data sets without the need for explicit extraction of geometric surfaces in the data [21]. These methods rely on applying an optical model, which facilitates the translation of data values into corresponding optical properties, encompassing attributes such as colour and opacity [22].

5.1.1. VOLUMETRIC DATA

Volumetric data represents information in a three-dimensional format, capturing spatially varying attributes, complex structures, and continuous functions within a volume. It finds applications in diverse domains such as medical imaging, scientific simulations, computer graphics, and computational fluid dynamics.

While volumetric data has the capability to represent continuous functions, practical implementations often involve the use of a uniform 3D array of samples. This data is organized into individual volume elements known as "voxels," analogous to pixels in a 2D image, each containing location coordinates and associated data values. It can be represented as a signal f as follows:

$$f(x) \in \mathbb{R} \text{ , with: } x \in \mathbb{R}^3 \tag{5.1}$$

Given the discrete nature of volumetric data, interpolation methods such as tri-linear, Spline and Radial Basis Function (RBF) Interpolation are employed to estimate values at intermediate locations from neighbouring voxels. This process is called *reconstruction*.

5.1.2. OPTICAL MODELS

Most direct volume rendering algorithms conceptualize the volumetric data as a distribution of light-emitting particles characterized by a specific density. This density information is subsequently mapped to RGBA [1] quadruplets for the purpose of compositing them along viewing rays. We present only three models used in the subsequent subsections:

- **Absorption only**: The volume is characterized by cold, perfectly black particles that absorb all incident light. These particles neither emit nor scatter light.
- **Emission only**: The volume is characterized by light-emitting particles with negligible absorption capabilities. Thus, they do not absorb any incident light.
- **Absorption + Emission**: This is the most commonly used optical model in direct volume rendering. It involves particles emitting light and occluding (absorbing) incoming light. However, it does not account for scattering or indirect illumination.

In the following subsections, we assume the simple emission-absorption optical model.

5.1.3. THE VOLUMETRIC RENDERING INTEGRAL

Let $\boldsymbol{x}(t)$ represent a ray cast into the volume, parametrized by the distance t from the eye. The scalar value corresponding to a position along the ray is denoted as $s(\boldsymbol{x}(t))$. The volume rendering integral equation integrates the absorption coefficients $K(s)$ and emissive colours $c(s)$ along the ray, considering the emission-absorption optical model. To simplify the equations, we express the emission c and absorption coefficients K as functions of the eye distance t instead of the scalar value s:

$$K(t) := K(s(\boldsymbol{x}(t))) \tag{5.2}$$

$$c(t) := c(s(\boldsymbol{x}(t))) \tag{5.3}$$

In Fig. 5.1, radiant energy emitted at a distance $t = d$ along the viewing ray undergoes continuous absorption along the distance d until it reaches the eye. Therefore, only a fraction c' of the original radiant energy c emitted at $t = d$ will ultimately reach the eye at $t = 0$.

The computation of the amount of radiant energy c' reaching the eye involves integrating the absorption coefficient along the distance d:

$$c' = ce^{-\int_0^d K(t)dt} \tag{5.4}$$

[1] "A" represents the opacity.

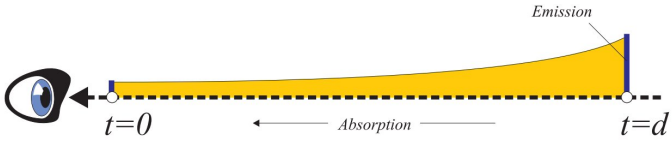


Figure 5.1: *Radiant energy emitted at time $t = d$ undergoes partial absorption as it propagates along the ray to reach the eye at $t = 0$.*

The integral in the exponent is called the *optical depth* [2] and can be expressed as follows:

$$\tau(d_1, d_2) = \int_{d_1}^{d_2} K(t)\,dt \tag{5.5}$$

Equation 5.4 calculates the amount of radiant energy c', assuming that the light was emitted only at a single point along the ray, namely at $t = d$. To calculate the total amount of radiant energy C reaching the eye from the direction of the ray, radiant energy emitted at all positions t along the ray should be considered. This is achieved by the *The Volumetric Integral*:

$$C = \int_0^{\infty} c(t)\,e^{-\tau(0,t)\,dt} \tag{5.6}$$

Since volumetric data is usually discrete, the volumetric integral 5.6 is approximated by alpha compositing the samples along the ray. We discuss this in Subsection 5.1.4.

5.1.4. RAY CASTING

One approach to render a 2D image from volumetric data is to use Ray Casting. It involves simulating the path of rays cast from a virtual camera into a scene. These rays are traced from each pixel on the image plane into the scene. At equispaced intersection points, the volumetric data is mapped to optical properties via a lookup table [3], yielding an RGBA quadrupled for this location. The solution of the volume rendering integral equation 5.6 is, then, approximated via alpha blending to create the colour of all pixels to form the 2D projection onto the screen as shown in Fig. 5.2.

NUMERICAL APPROXIMATION OF THE VOLUME RENDERING INTEGRAL

The optical depth τ in Equation 5.5, which quantifies the accumulated absorption along the ray up to a specific position $\boldsymbol{x}(t)$, can be estimated using a Riemann sum approximation:

$$\tau(0, t) \approx \tilde{\tau}(0, t) = \sum_{i=0}^{\lfloor t/\Delta t \rfloor} K(i\Delta t)\Delta t \tag{5.7}$$

with Δt denoting the distance between successive resampling locations.

[2] Optical depth allows Neural Radiance fields to estimate depth.
[3] The lookup table maps volumetric data values to optical properties, i.e, RGBA values. Transfer Functions are usually used as a lookup table in ray casting.

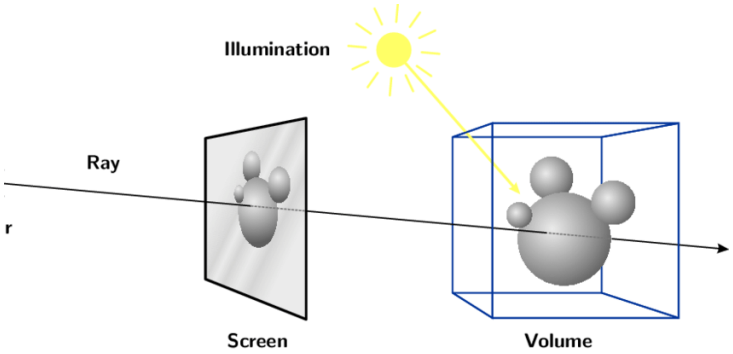


Figure 5.2: *From the camera position in space, a ray* $\mathbf{r}$ *is cast through the centre of a pixel on the screen. As it progresses through the volume, it accumulates contributions from coloured samples, which are shaded based on the illumination model employed. The resulting information is then stored in a pixel, and this iterative process is repeated for all pixels in the image. Source of image [23]*

Replacing $\tau(0,t)$ by $\tilde{\tau}(0,t)$ in Equation 5.6, results in a summation in the exponent term. It can be replaced directly by multiplying the exponential terms:

$$e^{-\tilde{\tau}(0,t)} = \prod_{i=0}^{\lfloor t/\Delta t \rfloor} e^{-K(i\Delta t)\Delta t} \tag{5.8}$$

By defining the *opacity* A as:

$$A_i = 1 - e^{-K(i\Delta t)\Delta t} \tag{5.9}$$

Equation 5.8 can be re-written into:

$$e^{-\tilde{\tau}(0,t)} = \prod_{i=0}^{\lfloor t/\Delta t \rfloor} (1 - A_i) \tag{5.10}$$

This way, the absorption of the i-th ray segment can be approximated using the opacity A_i instead of the absorption at a single point. Following the same logic, the emitted colour of the i-th ray segment can be approximated by:

$$C_i = c(i\Delta t)\Delta t \tag{5.11}$$

This allows the approximation of the volume rendering integral 5.6 numerically as follows:

$$\tilde{C} = \sum_{i=0}^{n} C_i \prod_{j=0}^{i-1} (1 - A_j) \tag{5.12}$$

where j represents the intervals from the eye to the interval emitting radiant energy represented by i.

5.1.5. ALPHA BLENDING

The volume rendering integral's numerical approximation 5.12 can be evaluated using *alpha blending* in either front-to-back or back-to-front order. In all equations in this

subsection, *associated colours* are used. They represent colours pre-multiplied by their associated opacity [24].

FRONT-TO-BACK ORDER

Equation 5.12 can be evaluated iteratively in fron-to-back by stepping i from 1 to n:

$$C'_i = C_{i-1} + (1 - A'_{i-1})C_i \tag{5.13}$$

$$A'_i = A_{i-1} + (1 - A'_{i-1})A_i \tag{5.14}$$

The values C'_i and A'_i are computed based on the color C_i and opacity A_i at the current position i, along with the composited color $C'_{i-1}$and opacity A'_{i-1} from the previous position $i-1$. The initial conditions are $C'_0 = 0$ and $A'_0 = 0$.

BACK-TO-FRONT ORDER

Equation 5.12 can be evaluated iteratively in front-to-back by stepping i from 1 to n:

$$C'_i = C_i + (1 - A'_i)C'_{i+1} \tag{5.15}$$

The value C'_i is computed based on the color C_i and opacity A_i at the current position i, along with the composited color C'_{i+1} from the previous position $i+1$. The initial condition are $C'_n = 0$.

COMPARISON OF BOTH ORDERS

Front-to-back compositing involves tracking alpha values, while back-to-front compositing does not require it. Previously, this posed a challenge for hardware implementations, but with modern GPU, it is no longer an issue. The primary advantage of front-to-back compositing is the early ray termination. This optimization trick allows the termination of ray progression as soon as the cumulative alpha value reaches 1.0 or a sufficiently close value.

5.2. MARCHING CUBES ALGORITHM

Marching Cubes Algorithm is a computer graphics algorithm for extracting a polygonal mesh representation of a 3D discrete scalar field. It is commonly employed for visualizing isosurfaces, which are surfaces that represent a particular value or threshold within the scalar field in volumetric data.

Marching Cubes employs a divide-and-conquer strategy to identify the surface within a *logical cube* created from eight voxels. This cube is formed by selecting four voxels from each of two adjacent slices, as depicted in Fig. 5.3.

The algorithm determines how the surface intersects with this cube and then "marches" to the next one. To find the surface intersection within a cube, a value of one is assigned to a vertex if the data value at that vertex is greater than or equal to the value of the surface (also known as the isovalue) being constructed. These vertices are considered to be

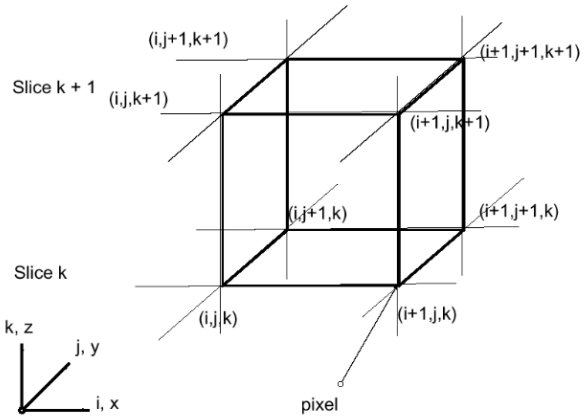


Figure 5.3: *Marching cubes divides the volumetric dataset into logical cubes, created from 8 voxels from two adjacent slices of the 3D dataset.*

inside or on the surface. Vertices of the cube with values below the surface's value are assigned a zero and are considered outside the surface. The surface intersects with the cube's edges where one vertex is outside the surface (given one), and the other is inside the surface (given zero). This criterion helps determine the surface's topology within the cube.

A cube can intersect with a surface in $2^8 = 256$ ways, considering its eight vertices and two states (inside and outside). To make this process easier, a lookup table is created to provide information about which edges are intersected for each case based on the labelling of the cube's vertices.

It is possible to triangulate all 256 cases, but it can be time-consuming and error-prone. The problem is simplified by two symmetries of the cube, reducing the number of cases to 14 patterns. The first symmetry arises from the fact that the topology of the triangulated surface remains unchanged if the relationship between the surface values and the cube is reversed. Complementary cases, where vertices greater than the surface value are swapped with those less than the value, are considered equivalent. Therefore, only cases with zero to four vertices greater than the surface value need to be considered, reducing the number of cases to 128. The second symmetry, known as rotational symmetry, further reduces the problem to 14 patterns through visual inspection. The algorithm can handle the triangulation process more efficiently by leveraging these symmetries. Fig. 5.4 shows the resulting triangulation for each of the 14 patterns.

To determine where surfaces intersect, an index is created for each case showing each vertex's state. This index has eight bits, with each bit corresponding to a vertex. Using the vertex numbering shown in Figure 5.5, we can refer to an edge table that provides

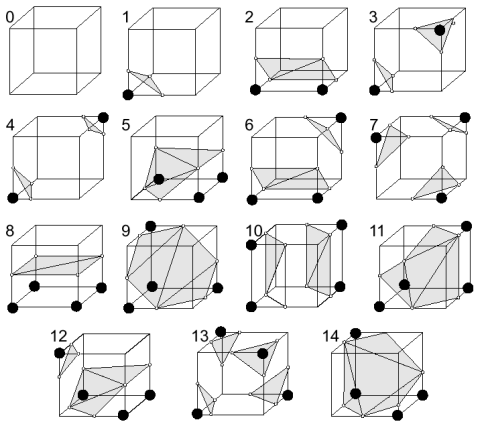


Figure 5.4: *All possible 14 cases of triangulated cubes. The simplest pattern labelled 0, occurs when all vertex values are consistently above or below the chosen threshold, resulting in no triangles. Pattern 1 emerges when the surface divides one vertex from the remaining seven, forming a single triangle defined by the three edge intersections. The remaining patterns generate multiple triangles. By applying complementary and rotational symmetry to these 14 basic patterns, the full set of 256 cases can be obtained.*

information about all edge intersections for a given cube configuration.

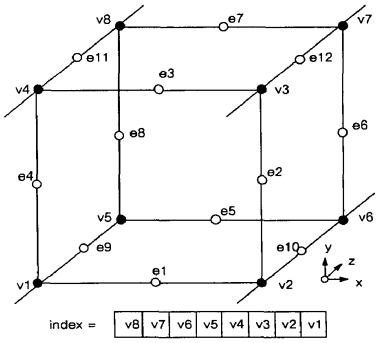


Figure 5.5: *Each cube is indexed by an 8 bits digit. Each bit corresponds to a vertex and represents whether the vertex is inside or outside the surface.*

With the help of the index, the edges that intersect the surface are identified, and the exact intersection location along that edge can be determined using linear interpolation. Although higher-degree interpolations can be used, they provide minimal improvements over linear interpolation as the algorithm generally generates only one to four triangles per cube [25].

Once all cubes have been processed, the marching cubes algorithm combines the gen-

erated triangles from all the voxels to form a polygonal mesh approximating the scalar field's isosurface. This mesh can then be rendered and visualized in 3D graphics applications. Finally, the Marching Cubbes Algorithm comes down to the following steps:

Algorithm 1 Marching Cubes Algorithm

Input: N, the volumetric data
Output: P, polygonal mesh

1: **for each** $cube \in N$ **do**
2: Classify each vertex v
3: Build an index
4: Get intersected edge list
5: Find exact intersection by interpolation
6: Connect intersections to polygon
7: Triangulate polygon to iso-surface
8: **end for**

BIBLIOGRAPHY

[1] Z. Yuan and M. Liu. "Specification for engine borescope inspection report". In: *IOP Conference Series: Earth and Environmental Science* 186.5 (2018), p. 012001. DOI: 10.1088/1755-1315/186/5/012001. URL: https://dx.doi.org/10.1088/1755-1315/186/5/012001.

[2] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis". In: *ECCV*. 2020.

[3] A. Rosebrock. *Introduction to neural networks*. May 2021. URL: https://pyimagesearch.com/2021/05/06/introduction-to-neural-networks/.

[4] A. Karpathy. *CS231N convolutional neural networks for visual recognition*. URL: https://cs231n.github.io/neural-networks-1/.

[5] V. Nair and G. E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines". In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML'10. Haifa, Israel: Omnipress, 2010, pp. 807–814. ISBN: 9781605589077.

[6] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri. "Activation functions in deep learning: A comprehensive survey and benchmark". In: *Neurocomputing* 503 (2022), pp. 92–108. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2022.06.111. URL: https://www.sciencedirect.com/science/article/pii/S0925231222008426.

[7] Q. Wang, Y. Ma, K. Zhao, and Y. Tian. "A Comprehensive Survey of Loss Functions in Machine Learning". In: *Annals of Data Science* 9.2 (Apr. 2022), pp. 187–212. DOI: 10.1007/s40745-020-00253-. URL: https://ideas.repec.org/a/spr/aodasc/v9y2022i2d10.1007_s40745-020-00253-5.html.

[8] L. Bottou, F. E. Curtis, and J. Nocedal. "Optimization Methods for Large-Scale Machine Learning". In: *SIAM Review* 60.2 (2018), pp. 223–311. DOI: 10.1137/16M1080173. eprint: https://doi.org/10.1137/16M1080173. URL: https://doi.org/10.1137/16M1080173.

[9] D. Kingma and J. Ba. "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations* (Dec. 2014).

[10] H. Geoffrey. "Lecture 6.5 - rmsprop: Divide the gradient by a running average of its recent magnitude". University Lecture. 2012.

[11] S. Ruder. "An overview of gradient descent optimization algorithms". In: *CoRR* abs/1609.04747 (2016). arXiv: 1609.04747. URL: http://arxiv.org/abs/1609.04747.

[12] Y. Xie, T. Takikawa, S. Saito, O. Litany, S. Yan, N. Khan, F. Tombari, J. Tompkin, V. Sitzmann, and S. Sridhar. “Neural Fields in Visual Computing and Beyond”. In: *Computer Graphics Forum* (2022). ISSN: 1467-8659. DOI: 10.1111/cgf.14505.

[13] J. T. Kajiya and B. P. Von Herzen. “Ray Tracing Volume Densities”. In: *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’84. New York, NY, USA: Association for Computing Machinery, 1984, pp. 165–174. ISBN: 0897911385. DOI: 10.1145/800031.808594. URL: https://doi.org/10.1145/800031.808594.

[14] C.-H. Lin, W.-C. Ma, A. Torralba, and S. Lucey. “BARF: Bundle-Adjusting Neural Radiance Fields”. In: *IEEE International Conference on Computer Vision (ICCV)*. 2021.

[15] M. Niemeyer, J. T. Barron, B. Mildenhall, M. S. M. Sajjadi, A. Geiger, and N. Radwan. “RegNeRF: Regularizing Neural Radiance Fields for View Synthesis from Sparse Inputs”. In: *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*. 2022.

[16] A. Jain, M. Tancik, and P. Abbeel. “Putting NeRF on a Diet: Semantically Consistent Few-Shot View Synthesis”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. 2021, pp. 5885–5894.

[17] J. Yang, M. Pavone, and Y. Wang. “FreeNeRF: Improving Few-shot Neural Rendering with Free Frequency Regularization”. In: *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*. 2023.

[18] J. T. Barron, B. Mildenhall, M. Tancik, P. Hedman, R. Martin-Brualla, and P. P. Srinivasan. “Mip-NeRF: A Multiscale Representation for Anti-Aliasing Neural Radiance Fields”. In: *ICCV* (2021).

[19] A. Pumarola, E. Corona, G. Pons-Moll, and F. Moreno-Noguer. “D-NeRF: Neural Radiance Fields for Dynamic Scenes”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021.

[20] T. Müller, A. Evans, C. Schied, and A. Keller. “Instant Neural Graphics Primitives with a Multiresolution Hash Encoding”. In: *ACM Trans. Graph.* 41.4 (July 2022), 102:1–102:15. DOI: 10.1145/3528223.3530127. URL: https://doi.org/10.1145/3528223.3530127.

[21] M. Levoy. “Display of surfaces from volume data”. In: *IEEE Computer Graphics and Applications* 8.3 (1988), pp. 29–37. DOI: 10.1109/38.511.

[22] N. Max. “Optical models for direct volume rendering”. In: *IEEE Transactions on Visualization and Computer Graphics* 1.2 (1995), pp. 99–108. DOI: 10.1109/2945.468400.

[23] P. Ljung. “Efficient Methods for Direct Volume Rendering of Large Data Sets”. PhD dissertation. Institutionen för teknik och naturvetenskap, 2006.

[24] J. Blinn. “Compositing. 1. Theory”. In: *IEEE Computer Graphics and Applications* 14.5 (1994), pp. 83–87. DOI: 10.1109/38.310740.

[25] W. E. Lorensen and H. E. Cline. “Marching Cubes: A High Resolution 3D Surface Construction Algorithm”. In: SIGGRAPH ’87. New York, NY, USA: Association for Computing Machinery, 1987, pp. 163–169. ISBN: 0897912276. DOI: 10.1145/37401.37422. URL: https://doi.org/10.1145/37401.37422.