Using Neural Radiance Fields and Gaussian Splatting for 3D reconstruction of aircraft inspections

Roos Bottema

Master's thesis
2024:E4

Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

CENTRUM SCIENTIARUM MATHEMATICARUM

Abstract

The rapid evolution of machine learning techniques has revolutionized computer vision, particularly with the introduction of Neural Radiance Fields (NeRF) and the optimization of 3D Gaussians for rendering novel scene views. These methods, such as NeRF and Gaussian Splatting, have demonstrated success in synthetic data scenarios with consistent lighting and well-captured scenes. This research explores the feasibility of applying these techniques to images captured by drones conducting aircraft inspections, aiming to automate and optimize the aviation industry. Motivated by the need for more efficient inspection processes, various models were examined and parameters adjusted to assess the performance of NeRFs and Gaussian Splatting in real-world scenarios. Despite the visual shortcomings observed in both NeRF and Gaussian Splatting, Gaussian Splatting came out as more promising for inspection images, outperforming NeRF visually. Quantitative results are presented, using metrics such as SSIM, PSNR and LPIPS to provide a more concrete understanding of the visual performance of the methods. While Gaussian Splatting exhibits promise, it is essential to acknowledge that the current quality of the output falls short of production standards. Looking ahead, with both methods being relatively new, substantial improvements in performance and broader applications are anticipated in the near future.

Acknowledgements

I would like to thank my supervisor Alessandro Scoppio from Mainblades for great guidance and support during this project. Without fail he taught me skills needed to complete this project that were outside of the scope of my knowledge with regard to computer science. Not only did he contribute to this project on a technical and theoretical level, but he also made it a very fun time at Mainblades in The Hague.

Also, I would like express my appreciation for my supervisors from LTH Viktor Larsson and Erik Tegler for consistent meetings and feedback during the time of the project.

Contents

Chapter 1

Introduction

In this section an introduction to the topic of 3D reconstruction in the area of computer vision is presented in 1.1 together with a review of current research in the field of Neural Radiance Fields in section 1.2. Furthermore, a short overview of the aviation industry with a special focus on the automation of inspections will be given as this is where theory meets real-world implementation in this thesis. The aim and objectives of the thesis are discussed in sections 1.3 and 1.4.

1.1 Background

In the field of computer vision, the problem of 3D reconstruction from 2D images, also known as photogrammetry, is a well-studied one. In fields like virtual reality, gaming, archaeological reconstruction and civil engineering photogrammetry is used with the intention to create, monitor, explore and document real-world 3D models. However, these 3D models are limited to only show the viewpoints of the input. This means that is not possible to see a scene from an angle not captured by the images. Creating these unseen views of a scene is called novel-view synthesis, and will be the overarching topic of this thesis.

Traditional photogrammetry methods like structure from motion (SfM) and multi-view stereo (MVS) were explored in the late 90's as a result of the introduction of epipolar geometry by Hauck in 1883 [7]. In his paper (and follow-up papers), Hauck focuses on the geometric relation between corresponding points in two and three images. These geometric relations are at the foundation of 3D reconstruction techniques. Now with the rise and rapid upcoming of artificial intelligence, and more specifically neural networks, the field of novel view synthesis has gone through substantial development. Neural networks have allowed enhanced efficiency, accuracy and computational capabilities of various processes within the task of novel view synthesis. For example, in order to perform structure from motion, a feature matching algorithm is needed to detect corresponding points in two images. As early as 2008, [21] suggested a machine learning approach to increase the efficiency and repeatability of corner detection. A more recent example from 2018 of the use of machine learning in computer vision is the use of neural networks when it comes to feature matching [15]. In order to implement structure from motion, interest points (points that are stable and repeatable when seen under different lighting condition and from different view points) need to be detected and matched across images. In [5] is suggested the use of a fully convolutional, self-supervised framework to train interest point detectors. The use of this framework resulted in a much larger set of interest points compared to traditional

methods.

The two examples above are only a fraction of the many ways neural networks are implemented in the field of computer vision. In this thesis, the use of Neural Radiance Fields (NeRFs) to represent a scene as a continuous function of 3D location and 2D viewing direction in order to produce novel views will be explored. This technique, first introduced in [18], is a way of representing a scene with a memory efficient multi-layer perceptron (MLP) that takes as input the spatial coordinates of 3D points (obtained by algorithms such as SfM) and the angular coordinates of the camera positions. The output of the MLP is a unique emitted color and volume density for each spatial coordinate and each viewing direction.

In this thesis, the practical use of NeRFs is in the field of aircraft inspections done by unmanned aerial vehicles (UAVs). As shown by [18] and many others afterwards, NeRFs perform very well on synthetic data and specific real-world data sets. The challenge with implementing NeRFs on aerial inspection images, which is the goal with this thesis, is partly the sparse number of input images, but also the fact that the inspection images are close-up images of the surface of aircraft bodies.

1.2 Related work

The publication of the original NeRF paper [18] got a lot of attention and in a short period of time, NeRFs have been improved in various ways. The results of the original paper were promising as the full storage of a scene representation only took about 5 MB, which is less memory than the storage of the input images alone. However, one of the downsides of the implementation was the training time. Originally, it took around 24 hours to fully train the MLP and querying a novel view took 30 seconds. This did not make the original NeRF suitable for real-time applications. In order to address the long training time, the input was encoded [19] using hash-encodings. The use of these hash-encodings comes with a compromise between performance (training time, rendering speed) and quality (visual quality of the renderings), where large hash tables result in higher quality but lower performance, as this will require more memory usage for storage, which cannot be used for training.

Another challenge of the original NeRF is that it requires a large data set of input images to be trained. A way to reduce the number of input images is the introduction of depth-supervision [4]. Depth information about images is obtained in the structure from motion process, which is required as a pre-processing step before training the MLP. Using this depth information has a two-fold advantage: on one hand it increases the performance on data sets with fewer images and on the other it reduces training time. The reduced training time comes from the fact that originally, the MLP that represents an object holds a lot of information on empty space. When using depth information and introducing a loss on density, a training iteration can be terminated early when empty space is detected. An extension of the depth supervision is the use of dense depth priors [20]. Here, a network estimates depth with uncertainty before being incorporated into the NeRF optimisation. The difference between these two approaches, is that [4] directly uses sparse depth information in the optimisation, while [20] learns the depth priors in a separate network to guide the optimisation. Introducing a visibility prior to the MLP [23] has also resulted in state-of-the art performance regarding training time and rendering results on sparse input data.

Other implementations that led to a sped-up training time is the use of a dense voxel grid

[25]. A voxel grid is a three-dimensional grid that represents a volumetric space. A voxel can be seen as a 3D pixel. By directly optimizing the voxel grid representation of the scene, the training time was reduced significantly. Changing from point sampling along a ray for each pixel in the images, using cone casting [1] also reduced training time and resulted in less artifacts in the resulting rendering.

In the original NeRF paper synthetic data was used, meaning that the scene was static and captured under controlled settings. However, modelling real-world scenes with variable illumination, images from different cameras and transient objects was difficult with the original implementation. To address this issue [17] first modelled per-image appearance variations (exposure, lighting, weather) and optimized the appearance embedding for each input image. This was followed by a model that separated the static and transient components in the images (people that walk past a landmark, cars that drive by etc). These two models combined allowed for NeRF to be trained on data sets from famous landmarks with tourist images scraped from the web.

NeRFs are trained on one specific scene, which means that the MLP representing this scene is not generalizable. In order to avoid training new NeRFs for every new scene recent work have explored the use of vision transformers [24]. From the input images a collection of patches are sampled by using epipolar geometry. These patches are projected into feature vectors and processed by a collection of transformers. This is a commonly used method in language models and has even been proven to be useful in the case of NeRFs.

The use of NeRF in practical applications has been limited by the fact that the training of the MLP requires high computational power that is not accessible for most common hardware and that the rendering of the scenes takes too much time. By storing a trained NeRF into a sparse 3D voxel grid structure, where each voxel contains oppacity, color, and a feature vector to represent view-dependent effects [8]. By adjusting the NeRF architecture in order to "bake" the NeRF in the voxel grid, querying the MLP was sped up by three orders of magnitude which enabled real-time rendering. As a continuation on this work, surface textures were stored instead of volumetric textures [3]. This limited the use of memory even more, making it possible to use integrated GPUs allowing for NeRF visualization to run on mobile devices.

Another approach to the rendering of 3D models is gaussian splatting [11]. This method leverages the sparse point cloud generated by SfM algorithms as an initialization of gaussians over the scene. These gaussians are optimized to achieve an accurate represntation over the scene. Thus, this method disregards the use of MLPs altogether and obtains state-of-the-art quality and allows for real-time rendering. A more in-depth theoretical overview of Gaussian Splatting will be given in section 2.3.

1.3 Problem statement

1.4 Aim of research

The aim of research is to explore the practical use of NeRF and Gaussian Splatting on the 3D modelling of aircraft inspection images taken by UAVs and to leverage the available sensor information. The approach in this thesis is a combination of quantitative and applied research. Quantitative, as much of the architecture of NeRFs and Gaussian Splatting is provided by open-

source implementation created by the computer vision community where different methods and different parameter values can be evaluated against each other. Applied because the pipeline necessary to process and prepare the inspection data is created from scratch.

1.5 Objectives

To achieve the aim of research as mentioned before, the following objectives are stated:

- Evaluate NeRF model performance: quantify the results of the different NeRF models available and analyze their performance on inspection images to understand the strengths and limitations of the models.
- Use different MLP architectures: Modify the MLP architecture to integrate additional sensor information, such as depth and the underlying 3D models of aircraft. Leveraging supplementary data sources can improve the accuracy and robustness of NeRF-based 3D reconstructions in the context of aircraft inspections.
- Develop a comprehensive pre-processing pipeline: Establish a data processing pipeline to seamlessly integrate raw inspection data into the NeRF and Gaussian Splatting. Developing this pipeline is essential for efficiently incorporating inspection data into the NeRF model, ensuring a smooth workflow from data acquisition to NeRF and Gaussian Splatting output.
- Quantify and analyse results: analyze the outcomes of the algorithmic and data-oriented changes made to the existing structures. Analyzing the implementations is crucial to recommend any suggestion for further development.
- Propose algorithmic and data-oriented enhancements:
  - Recommend algorithmic improvements to elevate the performance and versatility of NeRFs in the context of aircraft inspections. Continuous refinement of the used algorithms is essential to keep improving the model's adaptability and accuracy.
  - Recommend changes and enhancements in current data acquisition such as sensor information, image capturing and other relevant data sources. As the results of any machine learning model depends on the quality of the input, it is essential to keep improving and adapting the data acquisition strategy.

1.6 Scope

The scope for this thesis includes an examination of 3D reconstruction using NeRFs as well as Gaussian Splatting for aircraft inspection in the aviation industry. The study specifically focuses on routine maintenance inspections of commercial aircraft in hangar environments. The images used are captured by high-resolution RGB cameras mounted on UAVs that are equipped with a LiDAR sensor. By transforming inspection information in the correct way to make use of implementation of NeRF in nerfstudio, a developers repository created by the KAIR Lab at Berkely.

Chapter 2

Theory

This chapter covers the theory needed to understand the concept of NeRFs and Gaussian Splatting. First, basic information about neural networks will be provided in order to lay the foundation for the NeRF architecture. Then, in **??** structure from motion is discussed in more detail as it is an important step into creating the input for the MLP. Section **??** explains the multilayer perceptron and in 2.2.2 volume rendering, the final step to produce the NeRF output, is described.

2.1 Fundamentals of neural networks

2.1.1 The neuron

The basic building blocks of neural networks are, as the name suggests, neurons (also referred to as nodes). A neuron receives an input signal x, which can be the weighted sum of multiple inputs, $x = \sum_{j=1}^{n} x_j w_j$. Additionally, a bias can be added to each neuron. To this signal an activation function $f(x)$ is applied in order to produce a final output signal y [2]. See figure 2.1.

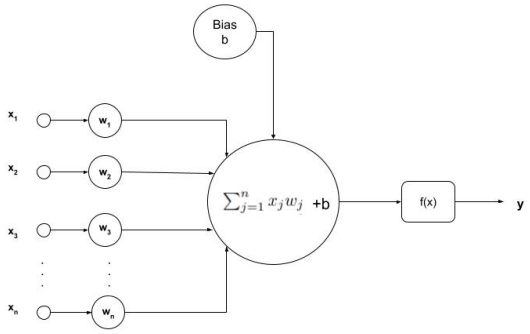


Figure 2.1: Schematic view of a neuron

The activation function in a neuron ensures that non-linearities are introduced into the model.

This non-linearity is essential for neural networks to learn and represent complex relationships within the data. Furthermore, due to the activation functions the output of the neuron will fall within a certain range which facilitates stable and efficient training. Activation functions are non-linear. Examples of activation functions are ReLU $f(x) = \max(0, x)$ or Sigmoid $f(x) = \frac{1}{1+e^{-x}}$.

A neural network consists of multiple layers, and each layer consists of multiple neurons. When a neuron is activated, meaning the output value of a neuron is above a certain threshold, it passes on its output to one or multiple neurons in the next layer. There is an input layer and an output layer in between which there are so called hidden layers.

2.1.2 Multi-layer perceptrons

The simplest type of neural networks are feed-forward networks. In these networks the information flows from the input layer to the output layer through the hidden layer.. Each neuron in one layer is connected to every node in the next layer. Depending on the problem the neural network is supposed to solve, the total number of nodes in the output layer can vary. These networks are also referred to as multi-layer perceptrons.

During training, the predicted output of a network is compared to a target output. This is called supervised learning. The error in the output is than back propagated into the network in order to optimize the weights [14]. Training consists of four steps [2]:

1. Forward pass: the input is passed into the network and flows through the hidden layers to produce an output.

$$z^l = W^l \cdot a^{l-1} + b^l$$
$$a^l = f(z^l)$$

   Where

   - W^l is the weight matrix of layer l
   - a^{l-1} is the output of the previous layer
   - b^l is the bias vector of layer l
   - $f(\cdot)$ is the activation function

2. Loss computation: the output is compared to a target value and losses are computed. An example of a loss function is the mean square error (MSE).

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

   where y_i represents the target value and $\hat{y}_i$ the predicted value.

3. Gradient computation: gradients are calculated for each weight in the network with respect to the loss. The gradients are computed from the output towards the input, which is why this is also referred to as a backward pass. The gradients indicate how much the loss would increase or decrease if the corresponding weight is adjusted. Error δ^L at the output layer L:

$$\delta^L = \nabla_a \text{MSE} \odot f'(z^L)$$

   where

- δ^L is the error at the output layer
- ∇_aMSE s the gradient of the loss with respect to the outputs
- $f'(\cdot)$ is the derivative of the activation function

Partial derivatives with respect to weights and biases:

$$\frac{\partial \text{MSE}}{\partial W_{ij}^L} = a_j^{L-1} \delta_i^L$$
$$\frac{\partial \text{MSE}}{\partial b_i^L} = \delta_i^L$$

Similar computations are made for the hidden layers l.

4. Updating the weights and biases: based on the computed gradients and the learning rate of network (which is a parameter that can be set), the weights are updated in the opposite direction of the gradients.

$$W^l \to W^l - \alpha \frac{\partial \text{MSE}}{\partial W^l}$$
$$b^l \to b^l - \alpha \frac{\partial \text{MSE}}{\partial b^l}$$

where

- α is the learning rate

These five steps are iterated until convergence is reached. This means the system has reached a state where adjustment of the weights will not lead to further significant decrease of the loss.

2.2 Fundamentals of computer vision

Computer vision is a multidisciplinary field that explores models and algorithms to interpret and understand visual information, such as images and videos [13]. Applications vary widely, from image analysis, facial recognition, autonomous systems, medical imaging and augmented reality.

In computer vision one of the main sensors to capture information, and also the one at the core of this thesis, are cameras. Mathematically, a camera can be described by the relationship between a three-dimensional spatial point (X, Y, Z) and the two dimensional projection of this point onto an image plane (u, v). In order to further explain this projection, homogeneous coordinates are introduced. Homogeneous coordinates are a representation of Cartesian coordinates in a projective space. This means that the point $\mathbf{x} = (x, y)$ can be represented by the homogeneous point $\tilde{\mathbf{x}} = (\tilde{x}, \tilde{y}, \tilde{w})$ [26]. In the projective space vectors that only differ by a scale are considered equivalent. Homogeneous coordinates can be converted back into inhomogeneous coordinates by dividing all coordinates with the last element $\tilde{w}$:

$$\tilde{\mathbf{x}} = (\tilde{x}, \tilde{y}, \tilde{w}) = \tilde{w}(x, y, 1)$$

where $(x, y, 1)$ is called the augmented vector.

One of the mathematical equations to project 3D points in world coordinates into 2D image coordinates in pixels is called the perspective projection and is given by

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = K \begin{bmatrix} R & T \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

K is called the intrinsic matrix of a camera containing internal parameters. R is and T represent the extrinsic matrix which indicates how the camera is oriented in world coordinates. R is a rotation matrix and T a translation vector. Together, these matrices form the so-called camera matrix. The intrinsic parameters of a camera are the focal length f_x and f_y, the principal point $\mathbf{c} = (c_x, c_y)$, the pixel aspect ratio α and the skew s. The focal length is the distance between the lens and the image sensor of the camera and can be different in the x- and y-direction (meaning the pixels are not square). Mostly, a common focal f is used and an aspect ratio introduced such that $f_y = \alpha f_x$, the principle point is the centre of the image plane in pixel coordinate. Skew can arise between when the sensor and optical axes are not perpendicular. In most scenarios the skew is zero. Together, these parameters form the matrix K [26]:

$$\begin{aligned} K &= \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} f & s & c_x \\ 0 & \alpha f & c_y \\ 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

Figure 2.2 shows how the extrinsic parameters translate and rotate world coordinate and how the intrinsic parameters project onto the image plane. This way of modelling a camera is called the pinhole model.

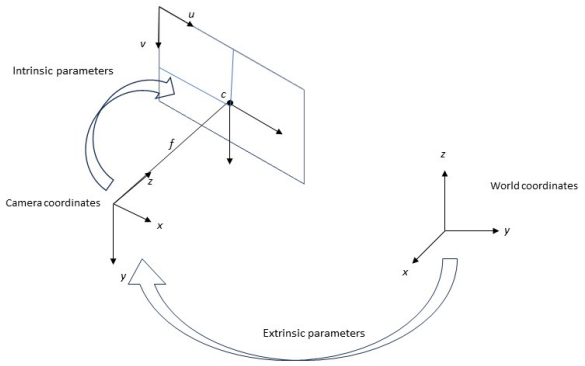


Figure 2.2: From world coordinates to image plane using extrinsic and intrinsic parameters of a camera.

2.2.1 Structure from motion

When a camera takes two photo's from different positions of a scene, it is possible to reconstruct the three-dimensional structure of the scene and estimate the motion of the camera. The biggest challenge in SfM is not the modelling of the problem, but rather the estimation of the parameters in the model [10]. The problem can be stated as follows: given a number of measured points in two images, compute both camera matrices P_1 and P_2 and 3D points X_i such that they project onto the measurements:

$$\lambda_{ij}\mathbf{x_{ij}} = P_i\mathbf{X_j}$$

where λ_{ij} are scaling factors, $\mathbf{x_ij}$ are the image projections, P_i are the camera matrices and $\mathbf{X_j}$ the corresponding 3D points. There are different ways of solving the SfM problem, (for example the 8-point algorithm [16] leveraging geometry between sequential images) but in this thesis the details of these solutions will not be further discussed. The interested reader is referred to [13] and [26] to learn more about this topic.

Important for the understanding of the NeRF architecture is that SfM is used to estimate the camera poses and the camera intrinsics. In the case of using UAVs for image capturing with localization features, it is possible to know the camera poses without the use of SfM. A benefit to using SfM is the fact that a sparse point cloud is produces as a by-product. These points can be used to leverage depth-information, as mentioned in section 1.2.

2.2.2 Volume rendering

Volume rendering is a technique used to visualize and generate images from three-dimensional volumetric data [9]. The 3D scene is represented as a scalar field, meaning that each point is associated with a scalar value that can represent physical quantities or attributes [18]. In the case of NeRFs, these physicial attributes are color and opacity. The data set is visualized by sampling the data at various points. The way this is done in the context of NeRF is by ray casting. Points are sampled along a ray that is cast through the 3D data, and the color and opacity of the points along one ray are combined into a single value. These accumulated values are then used to generate pixels that together form a meaningful 2D image of the scene seen from a certain viewpoint. Ray casting allows for complex light interactions, such as reflection and absorption, to be taken into account when viewing the same spatial point from different angles. See figure 2.3 for a visualization of the rendering of a pixel in a 2D image based on the volumetric data.

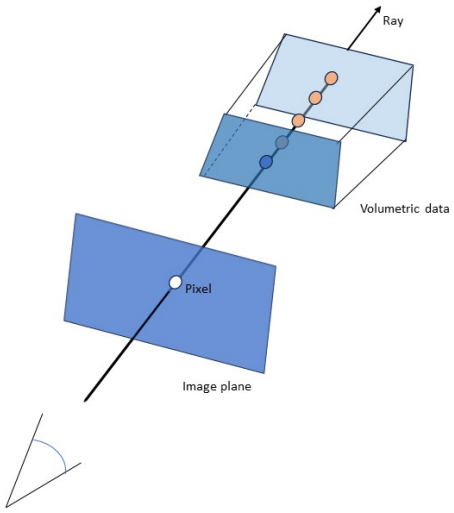


Figure 2.3: Visualization of volume rendering

The mathematical equation for the accumulated color C by volumetric rendering along a ray [11] is given as

$$C = \sum_{i=1}^{N} T_i(1 - \exp(-\sigma_i \delta_i))\mathbf{c_i} \text{ with } T_i = \exp - \sum_{j=1}^{i-1} \sigma_j \delta_j$$

Where for each sample i, the (exponentially decaying) transmittance T_i, density σ_i and rgb color $\mathbf{c_i}$ are taken with intervals δ_i along the ray. Points further along the ray will have less contribution to the final color, as the transmittance for these points is much lower compared to the first few sampled points. Because of this, it can be useful to sample more densely around the point where the ray actually hits the object, compared to further inside the object. This is also called hierarchical volume sampling [18].

2.3 Gaussian Splatting

The most common representations of 3D scenes are meshes and points. Another way of representing volumetric data is by using 3D Gaussians. A 3D Gaussian is defined by a center point (mean) μ, standard deviations in each dimension σ_x, $,\sigma_y$, σ_z, controlling the spread or width of the Gaussian and an opacity α. See figure 2.4. The color of a 3D Gaussian can be represented by another mathematical function, spherical harmonics (SH). These functions are defined on a sphere's surface. They are used in the area of computer graphics to model directional information, such as color. The color on a sphere's surface can vary based on the direction of incident light, which is why SH are suitable. Further information on the topic of spherical harmonics is available in [6, 22, 28].

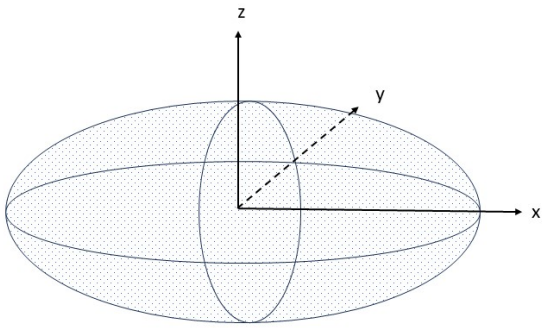


Figure 2.4: A Gaussian shaped as a 3D ellipse centered around it's mean

Built upon the same idea of optimizing a scene representation, Gaussian splatting creates the scene by iteratively optimizing the number of Gaussians and their parameters (position, opacity and covairance) as well as the coefficients of the spherical harmonics (view dependent features, such as color and lighting).

The "splatting" part of Gaussian splatting comes in when points are lifted from the 2D grid of an input view [12] to a 3D grid and then reprojected into the novel view. This results in one Gaussian associated with each pixel to reproject.

These resulting images to the training views are compared to the ground truth images. If the geometry is not accurately represented, the Gaussians are adjusted: small Gaussians in underreconstructed regions can be cloned, large Gaussians in over-reconstructed regions can be split into two Gaussians, (almost fully) transparent Gaussians can be removed [11]. See figure 2.5.

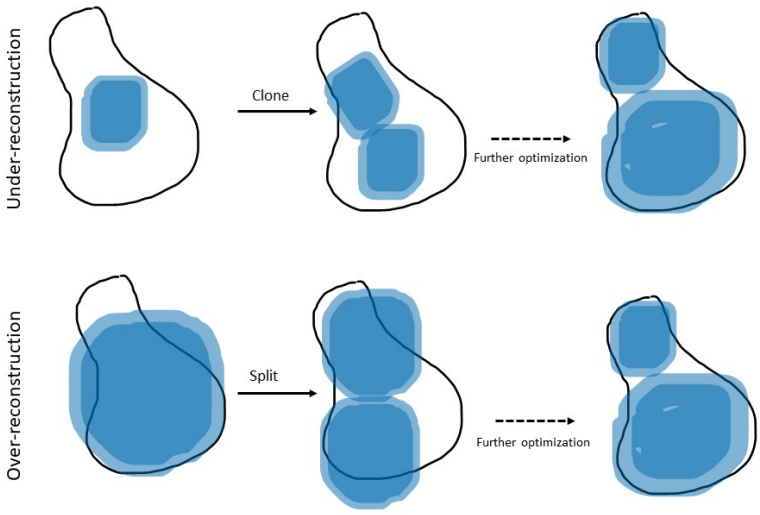


Figure 2.5: Optimization of Gaussians in under- and over-reconstructed regions. Image based of figure 4 in [11].

2.4 NeRF specifications

With the introduction to neural networks, multi-layer perceptrons and computer vision techniques like SfM and volume rendering, a closer look can be taken at the underlying structure of NeRFs. As mentioned before, the goal of representing a 3D scene with a multi-layer perceptron is to render novel views of the scene, not captured by the input images.

The first step is to obtain the camera positions using SfM. As mentioned in section 1.2, the sparse point cloud of generated 3D points can be used as depth supervision. The 5D vector obtained as a result of SfM containing the spatial coordinates x, y and z together with the viewing direction θ and ϕ for each camera are the input to MLP. However, the position and direction do not enter the MLP at the same time [29]. First, an encoded version of the position is the input the first 8 fully connected layers (each layer consisting of 256 neurons). The output of these first layers is σ (the density, or opacity) and a feature vector. This feature vector is concatenated with the viewing directions and passed into the final fully-connected layer of 128 channels to output the view-dependent RGB color.

In order to optimize the results, the input coordinates are positionally encoded. Without these positional encodings, the renderings perform poorly at representing high-frequency variation (meaning details) in color and geometry. Encoding the positions means that each coordinate gets mapped to sine waves with different frequencies. For sine waves with higher frequencies, coordinates with values close to each other will have a larger difference in value compared to sine waves with lower frequencies. This means that highly detailled spatial features in an image will be contained when using high frequency encoding. Figure 2.6 shows this principle.

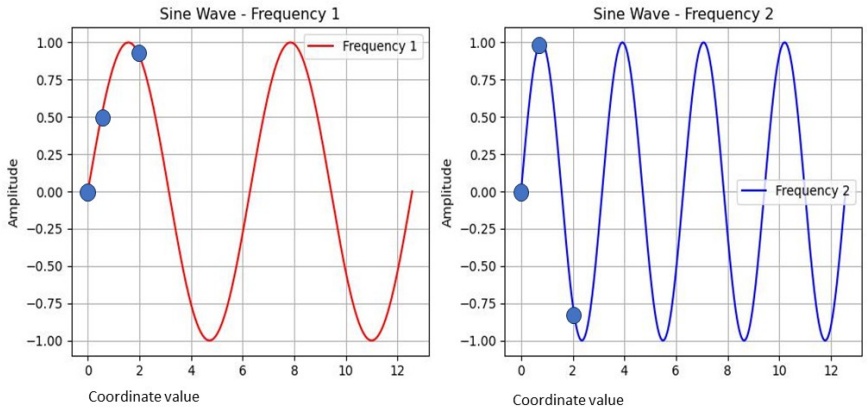


Figure 2.6: Positional encoding: the difference in amplitude for the three points in the left figure is smaller compared to the difference in amplitude for the same three points in the right figure.

The components of the three coordinate values are mapped into a 10-dimensional space (L = 10) and the viewing direction unit vector is mapped into a 4-dimensional space (L = 4): $\gamma(p) = (sin(2^0\pi p), cos(2^0\pi p), ..., sin(2^{L-1}\pi p), cos(2^{L-1}\pi p))$ [29, 18]. See figure 2.7 for a schematic view of the MLP.

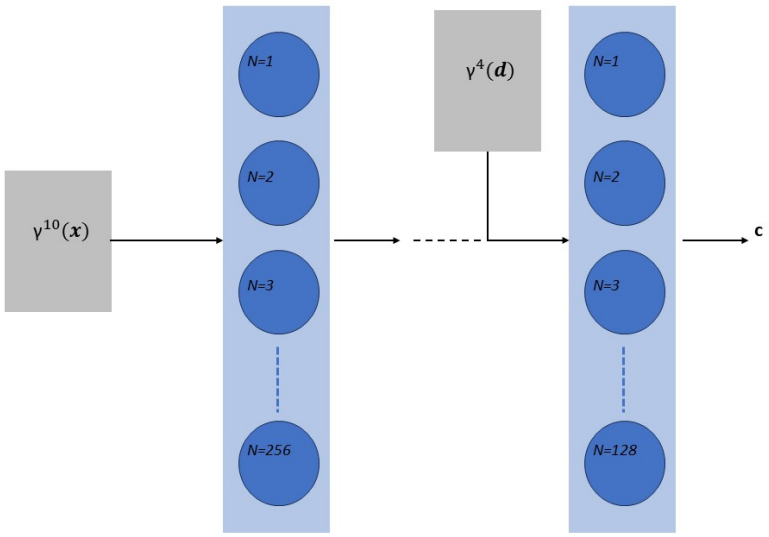


Figure 2.7: Structure of the NeRF MLP where the encoded positions are the input to the first fully connected layer with N=256 neurons, and where the encoded viewing directions enter the MLP at the last fully connected layer of N=1218 neurons. Inspired by figure 3 in [29].

The output of the MLP is used for volume rendering and recreating the scene from the viewpoints of the input images. These resulting 2D images are then compared to the ground truth and a mean square error based on pixel value is backpropagated into the MLP for optimization.

In order to evaluate the results, three metrics are used:

- PSNR (peak-signal-to-noise-ratio): a measure for the ratio between the maximum possible amplitude of the original image and the power of corrupting noise that affects the quality of the image. Higher PSNR values indicate better image quality.
- SSIM (structural similarity index measure): assessment of the perceived quality of an image by considering luminance, contrast and structure. SSIM values range from -1 to 1, where 1 indicates perfect similarity between the original image and the NeRF produced image.
- LPIPS (learned perceptual image patch similarity): a neural network analyzes two images on feature level and assesses the perceptual similarity. Values range from 0 to 1, where 0 indicates identical images.

Chapter 3

Data Processing

NeRF performs best with input images that capture a scene from many different angles, with large overlap and consistent lighting. The large overlap is needed for SfM to detect interest points for 3D reconstruction and determination of camera positions. As SfM is not needed due to known camera positions during inspection, this section will describe the steps needed to go from raw inspection data to NeRF input.

3.1 Change of coordinate systems

During aircraft inspections, a UAV flies around the plane following a pre-determined path. During the flight the drone takes photos of the aircraft at locations defines as "points of interest". In reality, the location of the center of the image is always a bit off compared to the point of interest. All information about an inspection is saved to a .json file. The relevant properties in the inspection file for NeRF are the translation and rotation matrix of the imaging center pose. These matrices define the positioning of the camera with regard to the origin of the world coordinate system, which is defined at the nose of the aircraft. The x-axis points along the body, the y-axis to the left and the z-axis upward. Figure 3.1 shows the origin of the coordinate system at the nose, as well as the camera positions of a wing inspection. The red axis is the x-axis, the green axis the y-axis and the blue axis the z-axis. As seen in the figure, the x-axis for the camera points down towards the wing of the aircraft.

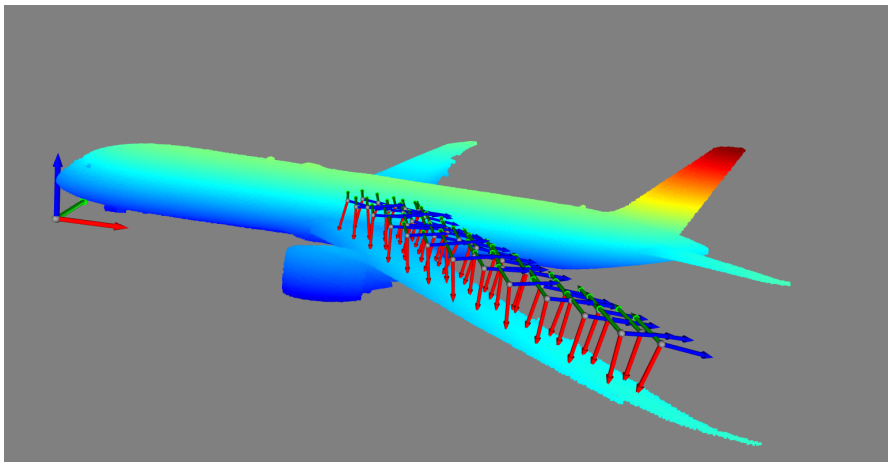


Figure 3.1: World coordinate system is defined at the nose of the plane. On top of the wing the position and orientation of the camera can be seen.

To use the camera positions in NeRF studio, a transformation is required. The model assumes that camera coordinates align with the x-axis pointing right, the y-axis pointing upward, and the z-axis extending back and away from the camera [27]. Achieving this alignment involves a negative rotation of 90 degrees around the y-axis, followed by a negative rotation of 90 degrees around the z-axis. Figure **??** shows the new orientation of the cameras.

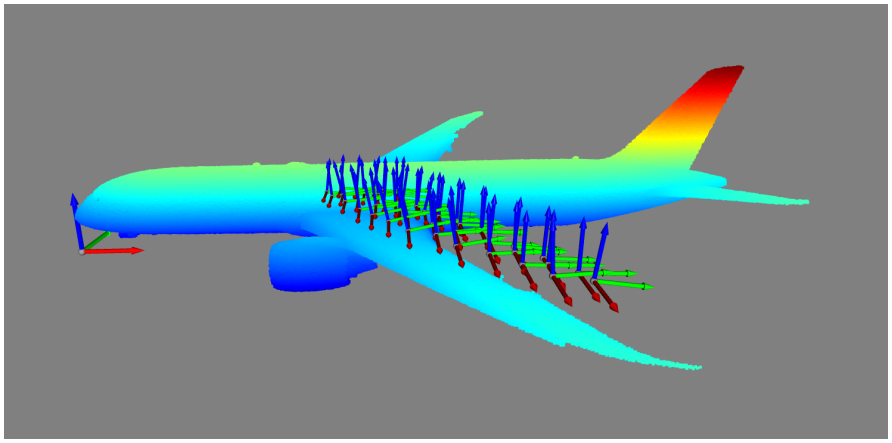


Figure 3.2: Rotated coordinate system to match with the correct definition of the axes for NeRF.

3.2 Depth supervision

Adding depth supervision to the training of a NeRF requires depth images to be available. These depth images are generated from the known path together with the given mesh of the aircraft model. The known path contains the positions where the images are taken and based on the mesh structure a depth image is formed. Figure 3.3 shows such a depth image.

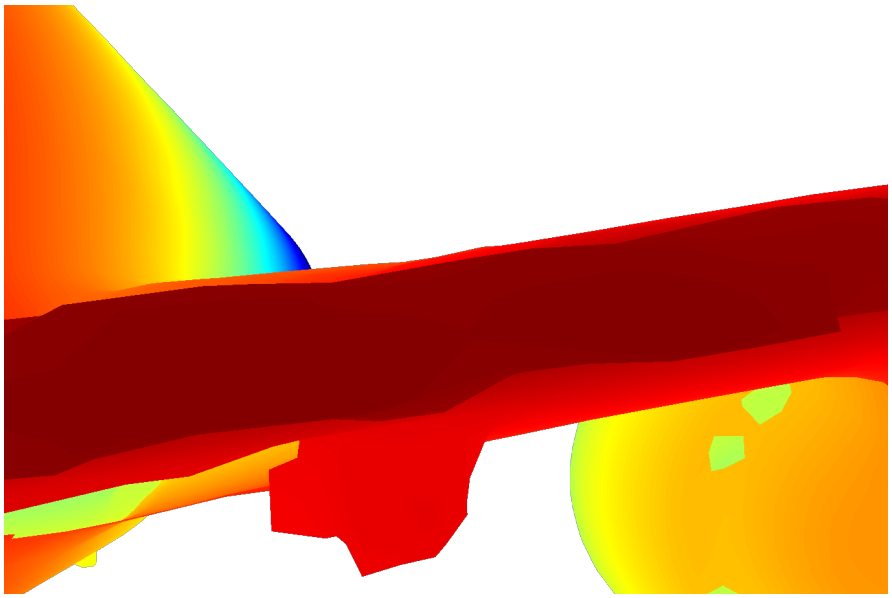
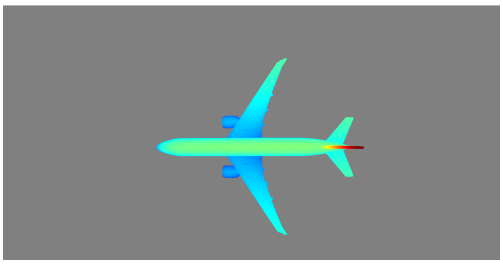


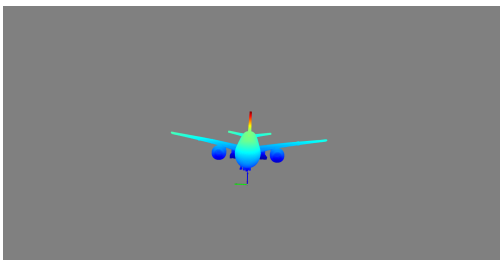
Figure 3.3: Depth image generated from viewpoints and underlying mesh.

3.3 Gaussian Splatting

Optimizing Gaussian splats to represent the scene requires an initialization of the gaussians. This can be done at random, or using the sparse point cloud which is a by-product of Colmaps SfM. As SfM is not used on inspection images, due to too little overlap, the initialization of the point cloud can be done by using the aircraft model. For every aircraft that is being inspected, an accurate point cloud of the model is available. By trimming this point cloud to only contain the points of the inspected area of the aircraft, the gaussians can be initialized relatively densely and with accurate underlying geometry. The central point of the gaussians can now be initialized and pruned/densified from the trimmed point cloud. The values of the spherical harmonics are initialized at random. Figure 3.4 shows the full model point cloud together with the trimmed point cloud from two different views.

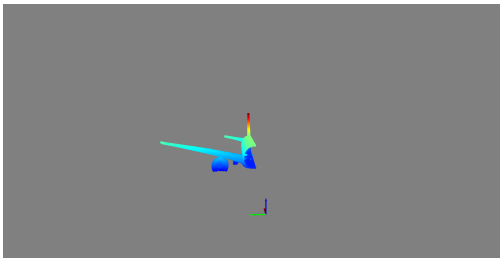


(a) Full model - view from above



(b) Full model- frontal view

(c) Trimmed model - view from above



(d) Trimmed model - frontal view

Figure 3.4: Aircraft point cloud full model and trimmed model showing only inspected parts of the aircraft. The trimmed model can be used as initialization for the gaussians.

3.4 Creating camera files

Both for NeRF and Gaussian Splatting the inspection information must be put together in a transforms.json file. The file contains the intrinsic parameters of the camera as well as the transformation matrix for each viewpoint.

Chapter 4

Method

4.1 Data capturing

The data was acquired during two flights at the maintenance hangars at KLM. Previous to the flight, the path was generated to cover the right fuselage, tail fin and part of the wing. Both flights executed the same path, with the same overlap between images. The horizontal overlap as well as the vertical overlap between the images was chosen to be 40%. The first flight used a 50 mm lens and the second flight a 35 mm lens. Standard procedure for inspections is using the 50 mm lens with a horizonal overlap of 30% and a vertical overlap of 20%. To explore the use of NeRFs in the context of aircraft inspections, a series of systemic tests were performed:

- Comparative analysis of models
- Investigation of parameters within models
- Testing the effect of lens variability.

The results were analyzed using quantitative metrics as well as qualitative visual assessment. The quantitative evaluation was done using established metrics such as PSNR, SSIM and LPIPS. The qualitative assessment was performed through image comparison by rendering views that were withheld from the training set and comparing these rendered images to ground truth.

4.2 Vanilla NeRF

Vanilla NeRF refers to the implementation of the original NeRF paper [18]. It is the baseline model to which the other models will be compared. The model consists of two MLPs, one directly taking encoded positional information as input and producing density, the other one taking density together with encoded direction as input to produce rgb values. Together, density and rgb values are rendered to represent the full scene by ray tracing the volume densities. Figure 4.1 shows a schematic view of the model.

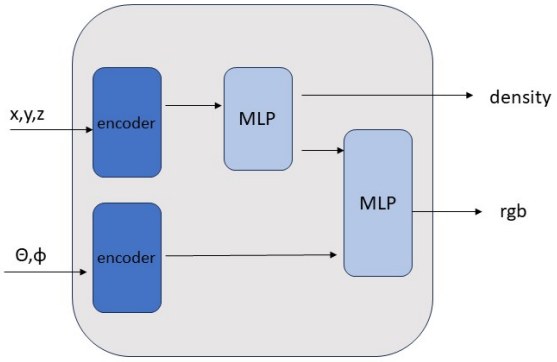


Figure 4.1: Structure of a NeRF field containging encoders for both the position and direction, as well as the MLPs to produce density and rgb value

4.3 Nerfacto

The model consists of two NeRF fields, one taking x,y,z and direction as input, sampled between predefined distances from the camera. The renderer produces rgb values for the coarse scene as well as weights that are higher for areas that are of higher importance for the final scene, such as surfaces. Areas with higher weights will then be sampled more densely by the second sampler. The x,y,z values as well as the direction from the finely sampled input data will generate the final rgb values for the scene. Figure 4.2 shows a schematic view of the structure. Both the coarse field and the fine field are of the same structure as shown in figure 4.1.

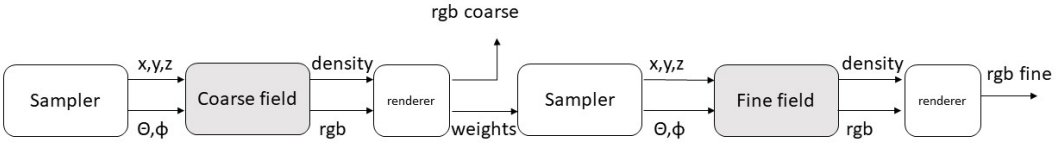


Figure 4.2: Schematic view of the nerfacto method

The different parameters that will be varied are:

- near and far field values with default values of 0.05 and 1000.0 respectively
- field optimizer learning rate with a default value of 0.01
- camera optimizer learning rate with a default value of 0.001
- orientation loss with a default value of 0.0001
- distortion loss with a default value of 0.002
- rgb coarse loss with a default value of 1.0
- rgb fine loss with a default value of 1.0

4.4 Depth Nerfacto

Depth nerfacto uses the same structure as nerfacto with the difference that there is a loss introduced on rendered depth and ground truth depth. The depth that are used are either generated

4.5 Gaussian Splatting

For Gaussian Splatting to work on the inspection images, a new data parser was written in python and integrated in the code to handle the specifics of the inspection. The data parser reads the transformation files to create camera instances and initializes the gaussians with the trimmed point cloud of the aircraft and random colors. The data parser returns an instance of

a scene class, which is optimized during the iterations. Within the model, the parameters that were varied were

- percentage of the scene that could be covered by one gaussian **percent_dense**. Default value 0.01.
- the iteration until which densification happened **dens_until_iter**. Default value 25000.
- the densification interval **dens_interval**. Default value 100.
- threshold for the densification gradient **dens_grad_threshold** . Default value 0.0002
- the influence of the ssim value to the total loss **dssim**. Devault value 0.2.
- opacity learning rate **opacity_lr**. Default value 0.05.
- feature learning rate **feature_lr**. Default value 0.0025.
- positional learning rates initial and final value, **position_lr_init** and **position_lr_final**. Default values 0.00016 and 0.0000016 respectively.

Chapter 5

Results

For results of every training iteration performed see Appendix A. This section will highlight the best metrics for different sets of parameters for and for the different models.

5.1 NeRF

5.1.1 Vanilla NeRF

Vanilla NeRF was trained at 100000 iterations.

Table 5.1: Results for default values.

	SSIM	**PSNR**	**LPIPS**
Vanilla NeRF	0.7069	14.7053	0.6518

5.1.2 Nerfacto

A Nerfacto model was trained with 30000 iterations. Table 5.2 shows the metrics for a default run.

Table 5.2: Results for default values.

	SSIM	**PSNR**	**LPIPS**
default	0.4679	16.1756	0.7113

Figure 5.1 show the loss for the default run. The loss increases during training after 10000 iterations, which is unwanted behaviour.

Figure 5.1: Loss for a default run

After running the default nerfacto, the parameters mentioned before were varied. Table 5.3 shows the metrics of the best runs and the values that corresponded to these runs for each parameter.

	Value	**SSIM**	**PSNR**	**LPIPS**
near-plane far-plane	0.05 10.0	0.4603	16.1814	0.7191
field opt. lr	0.001	0.4863	15.7510	0.6973
camera opt. lr	0.0001	0.4098	15.8281	0.7602
rgb loss fine rgb loss coarse	0.001 1.0	0.4944	16.2891	0.6866
orientation loss	0.1	0.4619	16.3939	0.7086
distortion loss	0.0002	0.4406	16.0487	0.6967

Table 5.3: Metrics for the best results for varying the different parameters.

Looking at the losses in figure **??**, the same problem is apparent as in the default run: the losses seem to increase over time instead of decreasing. For every case except the run where the camera optimizing learning rate was changed the loss did end up to be lower than the default run.

5.1.3 Depth nerfacto

In order to improve the results depth supervision was added. The same parameter values were used that resulted in the best metrics for nerfacto. Table 5.4 shows the metrics for a default run and figure 5.2 shows the loss. The same problem becomes visible for nerfacto: the loss goes up instead of down.

Table 5.4: Results for default values.

	SSIM	**PSNR**	**LPIPS**
default	0.6957	16.2424	0.6248

Figure 5.2: Loss for a default run

The results for using the same parameter values as the improved runs for nerfacto are shown in table 5.5.

	Value	**SSIM**	**PSNR**	**LPIPS**
near-plane far-plane	0.05 10.0	0.6901	16.0631	0.6391
field opt. lr	0.001	0.7636	17.0179	0.5778
camera opt. lr	0.0001	0.6704	15.6230	0.6688
rgb loss fine rgb loss coarse	0.001 1.0	0.6951	15.8343	0.6357
orientation loss	0.1	0.6890	15.9221	0.6411
distortion loss	0.0002	0.7109	16.5220	0.6192

Table 5.5: Metrics for the best results for varying the different parameters.

Figure 5.3 shows the loss for the runs from table 5.5. Depth0 represents default, depth1 to the first row in the table, depth2 to the second row etc. The loss does become less for the adjusted value of the field optimizer learning rate and for the adjusted value of the orientation loss multiplier, but in all cases the loss increases.

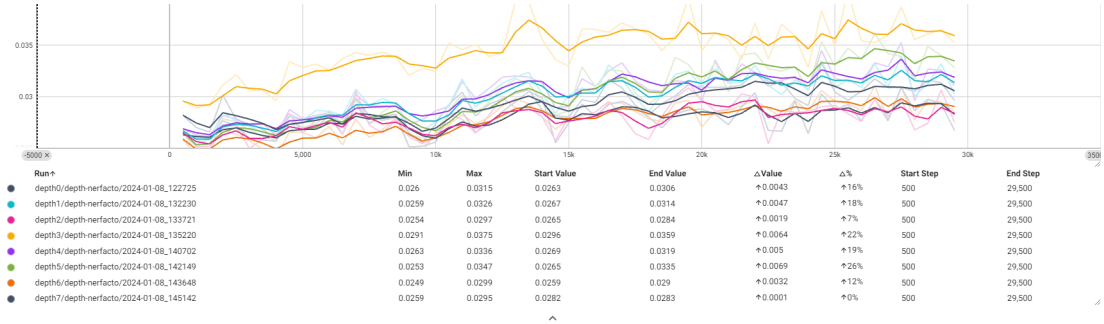


Figure 5.3: Losses for varying parameters run

5.1.4 35 mm lens - depth nerfacto

Due to the visual improvement of the novel views when introducing depth supervision, the influence of the lens is tested using depth-nerfacto.

Table 5.6: Results for default values.

	SSIM	PSNR	LPIPS
default	0.5713	14.2270	0.7267

Figure **??** show the loss for the default run. The loss increases during training after 10000 iterations, which is unwanted behaviour.
The results for using the same parameter values as the improved runs for nerfacto are shown in table 5.7.

	Value	SSIM	PSNR	LPIPS
near-plane far-plane	0.05 10.0	0.5623	13.9307	0.7317
field opt. lr	0.001	0.5503	14.4477	0.7297
camera opt. lr	0.0001	0.5470	13.9423	0.7580
rgb loss fine rgb loss coarse	0.001 1.0	0.5586	14.0179	0.7270
orientation loss	0.1	0.5688	14.0336	0.7206
distortion loss	0.0002	0.5855	14.4415	0.7044

Table 5.7: Metrics for the best results for varying the different parameters.

Figure 5.4 shows the loss for the runs from table 5.7. Depth0 represents default, depth1 to the first row in the table, depth2 to the second row etc. The loss does become less for the adjusted value of the field optimizer learning rate and for the adjusted value of the orientation loss multiplier, but in all cases the loss increases.

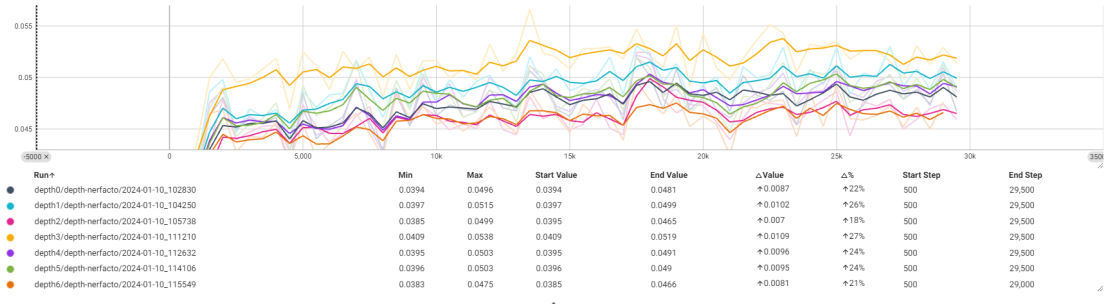


Figure 5.4: Losses for varying parameters run

5.2 Gaussian splatting

5.2.1 50 mm lens

The gaussian splatting was trained with 30000 iterations. Table 5.8 shows the metrics for a default run.

Table 5.8: Results for default values

SSIM	PSNR	LPIPS
0.7744	16.0250	0.3950

As visible in figure 5.5, the loss goes up during training which is unwanted behaviour in an optimization problem.

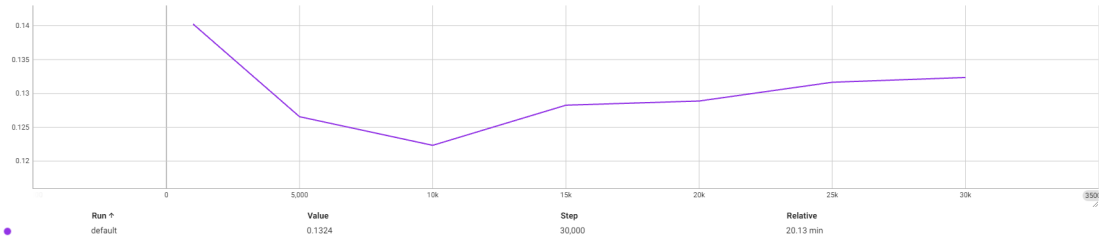


Figure 5.5: L1 loss for a default run

Table 5.9 shows the metrics of the best runs for varying the values of the different learning rates and the density parameters. Only the parameters in the table shown were varied, all the others were kept at default.

Table 5.9: Best results for default values with only varying one parameter at the time

	Value	SSIM	PSNR	LPIPS
scaling_lr	0.001	0.78121	16.3267	0.3817
opacity_lr	0.0001	0.8066	17.6044	0.3708
feature_lr	0.001	0.7846	16.8363	0.3872
position_lr_init **position_lr_final**	0.000016 0.0000016	0.7823	16.6445	0.3920
percent_dens **dens_grad_tresh** **dens_iter**	0.01 0.02 100	0.8011	17.5468	0.3698

Figure 5.6 shows the L1 loss for these runs. The purple curve is the default run. All runs result in a lower loss compared to the default run. However, in the case of the feature and scaling learning rate, the loss goes up with more iterations.

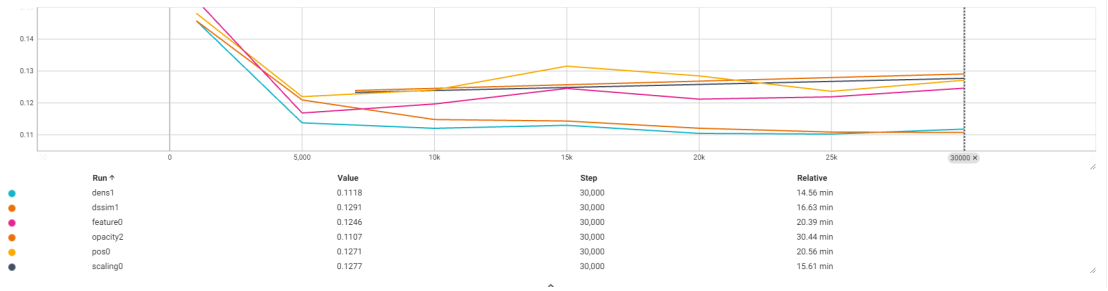


Figure 5.6: L1 loss for the runs with the best metrics

Opacity learning rate and the density parameters seem to lead to smallest value of the loss after 30000 iterations. This also matches the values of the ssim, psnr and lpips metrics as they are the best for these two paramters. Upon visual inspection it also becomes clear that these parameters have resulted in the highest quality novel view renderings.

5.3 35 mm lens

The gaussian splatting was trained with 30000 iterations. Table 5.10 shows the metrics for a default run.

Table 5.10: Results for default values

SSIM	PSNR	LPIPS
0.6768	14.4505	0.4596

As visible in figure 5.7, the loss goes up during training which is unwanted behaviour in an optimization problem.

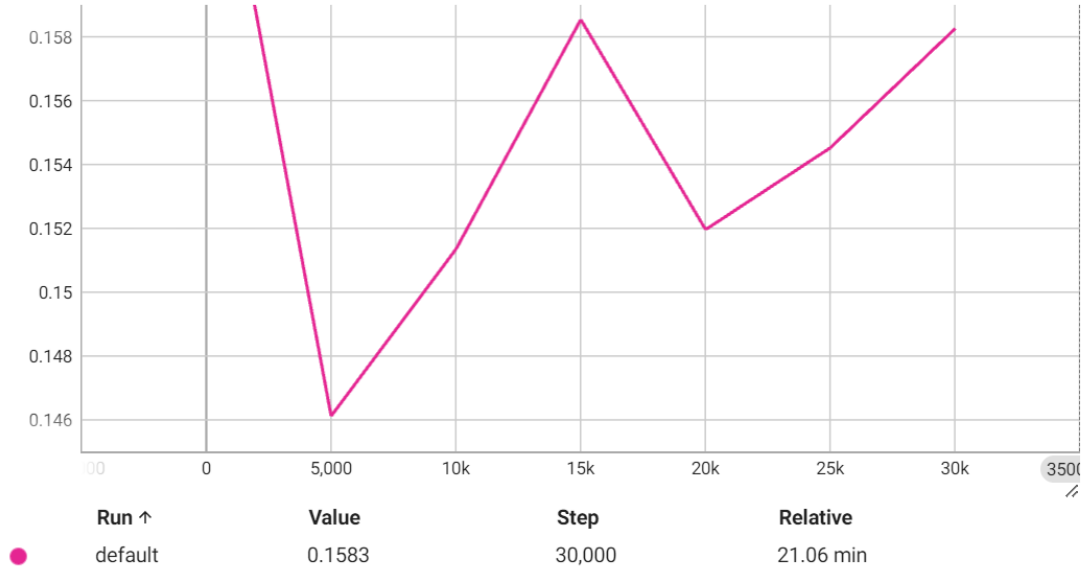


Figure 5.7: L1 loss for a default run

Table 5.11 shows the metrics of the best runs for varying the values of the different learning rates and the density parameters. Only the parameters in the table shown were varied, all the others were kept at default.

Table 5.11: Best results for default values with only varying one parameter at the time

	Value	**SSIM**	**PSNR**	**LPIPS**
scaling_lr	0.05	0.7124	15.0370	0.4513
opacity_lr	0.001	0.7090	16.4628	0.4262
feature_lr	0.001	0.6868	14.9470	0.4490
position_lr_init **position_lr_final**	0.000016 0.0000016	0.6797	14.7494	0.4514
percent_dens **dens_grad_tresh** **dens_iter**	0.01 0.02 100	0.7210	16.6319	0.4273

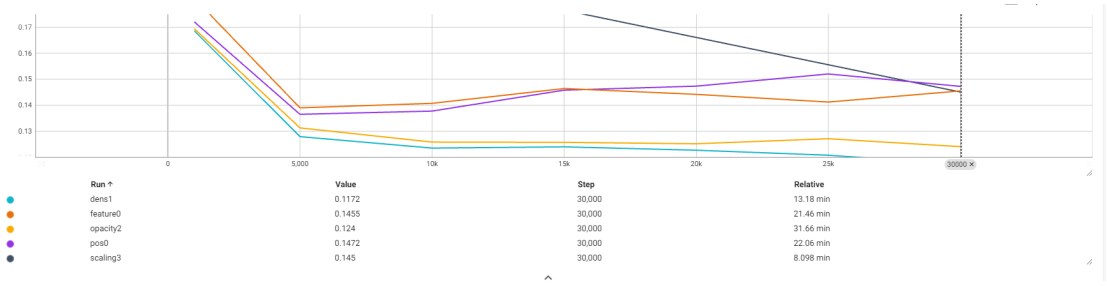


Figure 5.8: L1 loss for a default run

5.4 Summary of results

In short it becomes apparent that Gaussian Splatting has led to the best visual results.

Vanilla NeRF, despite training for 100000 iterations, showed the poorest results both metric wise and visually for a default run.

Nerfacto and depth nerfacto performed better than vanilla NeRF. Changing the different learning rates and loss parameters led to decreases in the final loss values, but overall the losses increased during training. Despite metrics that showed improvement, nerfacto's visual results were very poor and the scene was hardly distinguishable on the novel view images. Adding depth supervision to nerfacto led to an increase in visual quality, however many artifacts and blurred areas remainedin the novel views. As for nerfacto, the loss increased during evaluation for the depth supervision.

Gaussian splatting showed the most clear renderings of novel views. Despite artifacts and blurred areas being present, it was possible to see the scene it was reconstructing. The loss function behaved in a more expected manner for an optimization problem: it decreased. By varying certain paramters it was possible to improve the metrics of the results, but the visual quality remained rather poor.

When it comes to comparing the use of the 50 mm and 35 mm lens, the 35 mm lens resulted in better visual results both for depth supervised NeRF and for Gaussian Splatting.

Chapter 6

Discussion

6.1 Data

The original inspection images were downsized to speed up the training process both in the case of Gaussian Splatting and NeRF. During a standard inspection the vertical overlap is 20% and the horizontal overlap 30%. In order to improve the results, the path the drone executed was chosen to take photos with 40% overlap in both directions. Even with the horizontal and vertical overlap of 40%, Colmap performed badly for SfM. Because the camera positions are known during an inspection, this did not interfere with the ability to train a NeRF and Gaussian Splatting. However, for Gaussian Splatting not only the camera positions are needed, but also a sparse point cloud for initialization of the gaussians. Using the (trimmed) underlying aircraft model already forced the gaussians in an accurate position regarding the geometry of the scene.

One difficulty with the inspection images was the drone reflection on the surface of the aircraft. Neither NeRF nor Gaussian Splatting was able to render these reflections. Another challenge with inspection images was the fact that in certain images the hangar became visible, leading to artifacts in the final rendering. The inconsistency of the background can have led to poor rendering results. As inspection hardly ever regard the full body of an aircraft, because this is very time intensive and often times just not necessary, there was a lack of data to train on the full scene of an aircraft. Giving NeRF and Gaussian Splatting the full aircraft model as input might have rendered a more coherent result compared to the partial inspection images used that do not show a 360 degree view of the scene to render.

6.2 Results

The overall results are too low in quality to be used in production. Due to artifacts and lack of detail, the models can not be implemented for inspection.

For NeRF, despite changing the different learning rates and loss multipliers, the loss continued to increase during training. This can be caused by a variety of different reasons. One of them is overfitting to the training data. Upon inspection of the loss during training, the loss did go down. The overfitting can be a cause of insufficient training data. The scene to reconstruct is rather big, but the number of input images was relatively small, meaning that parts of the scene are only visible in very few images. As NeRF produces a continous function representing

the volumetric data of the scene, it becomes very difficult to produce values for the a 360 degree scene when the input data only covers one side of scene. In this case only the surface of one side of the aircraft, instead of images being captured 360 degrees around the body of the plane. Another reason for overfitting can be a lack of regularization. Regularization penalizes large weights in a model, reducing the risk of assigning too much importance to certain areas or features in the images. Introducing L1 (adding a penalty term as the absolute values of the coefficients of the loss function) or L2 (adding a penalty term as the squared values of the coefficients of the loss function) regularization could change the behaviour of the loss function and lead to decrease, instead of increase, of the function during training.

Besides overfitting to the training data, other reasons for the poor performance of NeRF can be faulty hyperparameters. Despite the fact that the learning rates and loss multipliers were varied to study their influence on the novel renderings, the parameters can not have been optimal for the model. As tuning by hand takes a lot of time, there was not a lot of room for varying other hyperparameters like batch size, regularization techniques or testing different optimizers. A more efficient approach for future work would be to implement cross-validation to reduce the randomness of the tuning and minimizing the manual labor. Moreover, the changing of the these parameters manually led to randomness being introduced in the training which can also be reduces by using cross validation or other more structured methods of tuning hyper parameters. However, when taking a look at the loss during training instead of during evaluation, it did go down. This points to the network actually learning and improving with each iteration.

The expectation was for depth nerfacto to outperform nerfacto, which visually was the case. However, the results were not optimal and the novel renders showed a lot of artifacts and even areas that seemed to appear doubled. One explanation can be the fact that the depth images are generated based on the underlying mesh of the predicted position for the camera. In reality, however, the UAV will always have a slight off-set from the desired position, meaning it will not be at the exact position from where the depth images were created. This, in turn, leads to a supervision on inaccurate values, decreasing the final results.

Another interesting observation between nerfacto and depth-nerfacto is the fact that despite very similar (in some cases even lower) metrics, depth-nerfacto did result in better visual results. The metrics used (SSIM, PSNR and LPIPS) are tools that can be used to compare different runs within the same model, but they are less useful for comparing between models. The visual results are more telling for making those types of comparison.

When comparing NeRF to Gaussian Splatting, Gaussian Splatting produced the most promising results. With Gaussian Splatting there seems to be an overarching image of the scene, but on a detailed level there are a lot of artifacts and too smooth areas. This can be due to the fact that the amount of input images is low, which leads to, as mentioned before, most views not being visible in more than just a few pictures. As the Gaussians are initialized using the point cloud of the underlying model, this can also have led to overfitting to the training images. As some images depict parts of the hangar too, Gaussians had to be culled and moved to represent these parts of the scene, despite the fact that those are not the parts of interest for the final results.

Capturing the images with a 35 mm lens improved the rendered results for both NeRF and Gaussian Splatting. Despite the fact that the inspection taken with the 35 mm lens had fewer images, the scene was captured in more "zoomed out" manner. This resulted in both NeRF and Gaussian Splatting being able to learn the scene in a more accurate way.

The training of both the NeRFs and the Gaussian Splatting was very time intensive. This was a limitation to the testing of different models. A major improvement was made by introducing the nerfacto method, combining some of the best practices when it comes to NeRF. Where vanilla NeRF took over 24 hours to train, a nerfacto model was trained within an hour. It took around the same time to train the Gaussian Splatting.

Chapter 7

Conclusion

The conclusion of this work is that the currently available methods for training NeRFs and Gaussian Splatting are not yet at a production level. The results indicate that up until a certain extend it is possible to render a rough representation of an inspection scene, but not accurately enough to replace the 3D models generated using classic computer vision methods. However, as the use of neural networks in the area of volume rendering is relatively new, it does hold a lot of potential. Extending this research and adjusting the models proposed in this work to be tailored to the specifications of inspections can in the future contribute to a more optimized work flow of automated aircraft inspections.

Chapter 8

Future Work

Due to the nature of the inspection images, the network struggled to represent the scene in a geometrically correct manner. Therefore, one recommendation for future work would be to initialize the MLPs with a pre-trained model on images showing the full aircraft body. This way, the model has a better understanding of the scene previously to being trained on relevant inspection images. This way, the inspection images can be used to texture the pre-trained model. A downside with regard to the practical implementation of this strategy is that a different pre-trained model is needed for different aircraft models. This will inevitably result in extended processing times per inspection, essentially hindering the potential for optimizing automated inspections.

Another recommendation is to make use of masking the input images. The purpose of using masks to remove the background is to reduce the floating artifacts this has resulted in. Especially in the case of Gaussian Splatting, where the underlying geometry was accurately initialized by the point cloud this can make a difference.

As there is a LiDAR mounted on the UAV, collecting depth information during the inspection would facilitate a more accurate depth supervision compared to the depth created by nerfstudio. This requires the LiDAR scans to be matched with the correct image, projected onto the 2D image plane and interpolated into a full depth image. Using these generated depth images might potentially lead to better results with regard to NeRF.

Finally, a rise in the use of vision transformers can be a potential for improving the quality of NeRFs and Gaussian splatting on scenes that are not captured under perfect conditions, such as inspection scenes. The use of vision transformers in combination with convolutional neural networks can facilitate the distinction between global and local features of a scene.

Using the current methods available, it would be interesting to see if NeRF would be a good tool to texture an existing mesh. Because there is a geometrically accurate mesh for all airplane models that are inspected, this could be a good start to implement NeRF in the automation of aircraft inspections.

Master's Theses in Mathematical Sciences 2024:E4
ISSN 1404-6342

LUTFMA-3523-2024

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/