Ole August Støle

From Roadside to Render: An End-to-End NeRF Pipeline for View Synthesis and 3D Reconstruction in the Context of Autonomous Driving

Master's thesis in Computer Science
Supervisor: Gabriel Kiss
Co-supervisor: Frank Lindseth
June 2023

Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Ole August Støle

From Roadside to Render: An End-to-End NeRF Pipeline for View Synthesis and 3D Reconstruction in the Context of Autonomous Driving

Master's thesis in Computer Science
Supervisor: Gabriel Kiss
Co-supervisor: Frank Lindseth
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Abstract

The field of Neural Radiance Fields (NeRF) has experienced a surge in research and development over the past years, with significant enhancements to model performance and an increasing scope of application areas. Amidst this growth, the use of NeRFs in Autonomous Driving (AD) systems has emerged as a promising area of exploration, as NeRFs can enable the generation of photorealistic edge case scenarios and an environment to evaluate systems for AD.

The primary research goal of this thesis is to design and develop an end-to-end pipeline for generating NeRFs, leveraging vehicle-captured video sequences and corresponding camera poses with varying degrees of accuracy.

Initially, a data capture pipeline is created for CARLA, providing synthetic data from a controlled environment. Connecting this data capture pipeline with a NeRF pipeline facilitates the creation of a performance baseline for further experiments. Having established a baseline and an end-to-end pipeline, the thesis explores large-scale NeRF approaches and implements a performant prototype. Finally, the pipeline is extended to enable the input of real data captured by a specialized vehicle with accurate Global Navigation Satellite System (GNSS) and high-resolution cameras.

Most of the findings from the experiments are consistent across synthetic and real data; the configuration of the data-capture significantly affects the data and the resulting NeRF's quality; a large-scale approach where a scene is learned by multiple smaller NeRFs, contrary to a single NeRF, performs better; joint camera pose optimization efficiently reduces the impact of imperfect camera poses, but approximating the poses with Structure from Motion (SfM) a priori demonstrates superior results.

Wrapping up, the application of rendering novel views for generating data from edge case scenarios is investigated. Although the renderings don't match the original images' quality, they are largely successful in producing clear and structurally accurate renderings. This reaffirms NeRFs' effectiveness and potential for AD applications.

Sammendrag

Neural Radiance Fields (NeRF) har opplevd en omfattende vekst i forskning og utvikling de siste årene, med betydelige fremskritt innen modellprestasjon og en økende rekkevidde av bruksområder. En av de lovende anvendelsene er innen autonome kjøretøy, hvor NeRFs blant annet kan benyttes til å generere simulerte miljøer for evaluering av autonome kjøretøy og til å skape fotorealistiske datasett av sjeldne scenarier.

Denne masteroppgaven har som hovedmål å designe og utvikle en ende-til-ende-pipeline for generering av NeRFs, ved å bruke videosekvenser fra kjøretøy og tilhørende kameraposisjoner av varierende nøyaktighetsgrad.

Først etableres en datainnsamlings-pipeline for CARLA, som gir tilgang til syntetiske data fra et kontrollert miljø. Ved å integrere denne pipelinen med en NeRF-pipeline har man en ende-til-ende-pipeline for generering av NeRFs som iterativt kan konfigureres for å danne et utgangspunkt for videre eksperimenter. Etter å ha etablert et utgangspunkt utforskes tilnærminger for stor-skala NeRF og det implementeres en fungererende prototype. Pipelinen utvides deretter til å kunne håndtere ekte data fra et spesialisert kjøretøy utstyrt med nøyaktig GNSS og høyoppløselige kameraer.

Resultatene er stort sett konsistente mellom syntetiske og ekte data; konfigurasjonen av datainnsamlingen har en betydelig innvirkning på kvaliteten av både dataene og den resulterende NeRF-modellen; en tilnærming som involverer flere mindre NeRF-modeller i stedet for én stor NeRF-modell, viser seg å være mer effektiv for å lære en scene i stor skala; parallell optimalisering av kameraposisjonen reduserer effekten av uperfekte kameraposisjoner, men forhåndsprosessering av kameraposisjonene med Structure-from-Motion (SfM) verktøy gir overlegne resultater.

Avslutningsvis undersøkes anvendelsen for å generere bilder fra usette, spesielle scenarier. Til tross for at de genererte bildene ikke gjenspeiler samme kvalitet som de originale bildene er de i stor grad vellykkede i å produsere klare og strukturelt nøyaktige gjengivelser. Dette bekrefter NeRF sitt potensiale i anvendelser for autonome kjøretøy.

Preface

This masters thesis concludes my five years at the Master's degree programme in Computer Science at NTNU. The thesis explores the application of Neural Radiance Fields (NeRF) on data captured from vehicles. The opportunity to work in the dynamic field of NeRF has been particularly interesting, given the constant influx of new research.

I would like to thank my team of supervisors, Frank Lindseth and Gabriel Kiss. The experiments conducted would not have been made possible without your guidance, data capture endeavour and provisioning of computational resources.

Furthermore, I would like to express my gratitude to friends and family for their support and encouragement, with a special thanks extended to Camilla Dybdal.

Lastly, I would like to thank NTNU and the city of Trondheim for five incredible years which I will never forget.

Ole August Støle
Trondheim, June 2023

Contents

Figures

Tables

Acronyms

2AFC	Two Alternative Forced Choice. 14, 15
AD	Autonomous Driving. iii, 1, 12, 22, 54, 62
ANN	Artificial Neural Network. 7, 8, 31
CEP	Circular Error Probability. 36
dB	decibel. 14
ENU	East, North, Up. 37
FOV	Field of View. 28, 32, 55
GNSS	Global Navigation Satellite System. iii, 12, 27, 33, 36, 46, 59, 63
GPS	Global Positioning System. 12, 27, 33, 36, 37, 46, 52, 53, 59, 61, 62
GPU	Graphics Processing Unit. 61
IPE	Integrated Positional Encoding. 16
JND	Just Noticeable Differences. 14, 15
LiDAR	Light Detection and Ranging. 12, 36
LPIPS	Learned Perceptual Image Patch Similarity. ix, 3, 14, 15, 31, 40, 41, 45, 55, 56, 59, 60, 65
MLP	Multilayer Perceptron. 7–10, 15, 17–22, 31, 73, 74
MSE	Mean Squared Error. 14, 15, 57
MVS	Multi-View Stereo. 11, 12

NAPLab NTNU Autonomous Perception Lab. 3, 25, 26, 36–38, 52, 54, 66

NDC Normalized Device Coordinates. 17

NED North, East, Down. 37

NeRF Neural Radiance Fields. 1, 5

NMEA National Marine Electronics Association. 36

PDF Probability Distribution Function. 10, 11, 17

POC Proof of Concept. 3, 34

PSNR Peak Signal-to-Noise Ratio. ix, 3, 14, 15, 31, 40, 41, 45, 56, 57, 59, 65

ReLU Rectified Linear Unit. 18

RTK Real-Time Kinematic Positioning. 36

SBAS Satellite-Based Augmentation System. 36

SfM Structure from Motion. iii, 5, 8, 11, 48

SIFT Scale-Invariant Feature Transform. 11

SSIM Structural Similarity. ix, 3, 14, 15, 31, 40, 41, 45, 55, 56, 59, 65

Chapter 1

Introduction

1.1 Motivation

The field of Neural Radiance Fields (NeRF) has experienced a surge in research and development over the past years, with models becoming exponentially better and more performant [1]. These models have demonstrated impressive, photo-realistic reconstruction and novel view synthesis of 3D scenes, given a set of posed 2D camera images. This surge in research has opened up new frontiers in many applications including the expansion of mapping and navigation capabilities (Google Immersive View[1]), computer graphics and visual effects (Volinga[2]), asset creation (Luma AI[3]), and AD (Wayve[4]).

Looking further into the realm of AD, one area of application that could benefit from these advances stands out. The evaluation of AD systems often involves the re-simulation of scenarios that have been encountered in the past. Yet, the vehicle's path could be altered by even the slightest divergence from the original incident, thus necessitating high-quality rendering of novel views along the altered trajectory [2]. Wayve, a pioneer in the autonomous vehicle industry, has already integrated NeRF technology into their system for the purpose of rendering novel views along altered paths. However, as with most commercial applications, their tools and code remain proprietary leaving them inaccessible for wider research purposes.

In order to explore the use of NeRFs in applications such as AD that could benefit from the generation of high-quality renders of novel views, this thesis seeks to design and develop an end-to-end pipeline capable of capturing data from a scene, and subsequently generating a NeRF representing the same scene.

[1]https://blog.google/products/maps/three-maps-updates-io-2022/
[2]https://volinga.ai/
[3]https://lumalabs.ai/
[4]https://wayve.ai/neural-rendering/

The first part of the project focuses on creating a data capture pipeline for a vehicle in a controllable environment. Having created a pipeline for data capture, it is connected to a pipeline for training, rendering, and evaluating NeRFs. The end-to-end pipeline is then used to configure the data capture and NeRF-settings in order to obtain a performance baseline. Further experiments investigating approaches to deal with noise or increasing scene-size is conducted, before the pipeline is extended to enable input of real data.

1.2 Goal and Research Questions

The primary research goal of this thesis is to design and develop an end-to-end pipeline for generating NeRFs, leveraging vehicle-captured video sequences and corresponding camera poses with varying degrees of accuracy. In order to achieve this goal, four research questions were posed:

RQ 1: What are the critical factors that need to be considered when capturing data for training NeRF models, and how do they impact the performance of the resulting models?

RQ 2: How does the initial camera pose accuracy and segment length impact the final reconstruction? Can rough initial camera poses be optimized to achieve comparable results to those obtained from tools such as COLMAP?

RQ 3: How do different NeRF methods (Instant-ngp [3], Mip-NeRF [4], Nerfacto [5]) perform on unbounded scenes in terms of reconstruction quality and computational efficiency?

RQ 4: What are the technical challenges and considerations for implementing a functional approach for large-scale NeRF within the Nerfstudio API, and how does it compare to approaches not optimized for large-scale in terms of scalability, efficiency, and rendering quality?

1.3 Research Method

The chosen research method for this report is experimental research. The experiments in this report will compare different methods, techniques, and configurations used to capture and process data, and subsequently used to train NeRFs. Observation will be used for the quantitative and qualitative evaluation of the results. The quantitative evaluation will span common image reconstruction quality metrics, whereas the qualitative assessment primarily will consist of analyzing video rendering outputs looking for abnormalities or other interesting findings.

1.4 Contributions

One of the primary contributions of this master's thesis is the establishment of a pipeline from CARLA [6] to Nerfstudio [7]. This pipeline serves as a tool for conducting a variety of experiments, which could further be applied to real-world scenarios. By permitting the manipulation of various experiment settings, such as the camera setup, vehicle speed, route, and image resolution, this pipeline streamlines the process. Furthermore, the pipeline's data output follows standard conventions, thereby enabling the training of NeRFs on synthetic data captured in CARLA. This consequently facilitates the refinement of settings to enhance the resulting image synthesis based on the evaluation of the NeRF. Furthermore, the pipeline is also extended to allow the input of real data captured by the NTNU Autonomous Perception Lab (NAPLab) car [8].

In addition to the pipeline, this thesis introduces a baseline for NeRFs trained on synthetic data captured in CARLA. This baseline can be used to augment both the data capture and the NeRF models on synthetic data. It offers a platform for experimenting with data capture and NeRF settings, and its evaluation metrics (PSNR, SSIM, and LPIPS) align with those widely used in NeRF research, ensuring comparability.

Another contribution lies in the development of a Proof of Concept (POC) for large-scale NeRF approach with the Nerfstudio API. By creating a naive Block-NeRF implementation, it demonstrates how such an approach can dramatically enhance the quality of large-scale NeRFs. This POC provides a testing ground for determining crucial parameters when capturing large-scale data for NeRFs, such as the segment size, the overlap between the blocks, and image merging techniques.

Lastly, the thesis includes the creation of a utility for generating side-by-side view using FFMPEG[9]. This enables the creation of a comparative view of the rendered NeRF and the ground truth. This addition not only facilitates the qualitative assessment of the resulting NeRF but also enhances the overall analysis process.

1.5 Report Outline

Chapter 1 - Introduction: Presents the study's motivation, goal, research method, contributions, and research questions.

Chapter 2 - Background and Related Work: Provides an overview of preliminary methods, techniques, tools, frameworks, and metrics in order to establish a common ground for successive chapters.

Chapter 3 - Methods: Provides details about the multi-stage process necessary for the designed and developed end-to-end pipeline. Sections span in-depth discussion and overview of the virtual environment, the data capture pipeline, the NeRF pipeline, the extension to large-scale scenes, and lastly the extension from synthetic to real data.

Chapter 4 - Experiments and Results: Presents the conducted experiments and subsequent results.

Chapter 5 - Discussion: Presents a detailed discussion of the results and their implications in the context of the research questions.

Chapter 6 - Conclusion and Future Work: Summarizes the main findings of the study and further suggests a direction for future work.

Chapter 2

Background and Related Work

The background chapter of this paper focuses on the use of NeRFs for representing and rendering 3D scenes. The chapter discusses several important NeRF methods, including NeRF [10], Mip-NeRF [4], mip-NeRF 360 [11], Block-NeRF [2], and Instant-ngp [3]. Additionally, the chapter covers several key techniques used in the NeRF pipeline, including positional encoding, hierarchical sampling, stratified sampling, appearance embeddings, learned pose refinement, and visibility prediction.

The chapter also discusses the use of SfM and COLMAP [12] as important tools for the pre-processing step in the NeRF pipeline, and presents several metrics for evaluating NeRFs. By providing an overview of these methods, techniques, tools, and metrics, the background chapter aims to establish a common ground for future chapters.

This thesis assumes a foundational understanding of deep learning. A great resource for individuals looking to acquire such knowledge is the publication by Goodfellow et al. [13].

Please note that, in order to cover the relevant background, some sections of the following chapter have been adapted from my unpublished project work on NeRFs for novel view synthesis and 3D reconstruction, which was completed as part of the course “TDT4501 Computer Science, Specialization Project”.

2.1 Volume Rendering

Volume rendering is a powerful technique for visualizing 3D data sets, which is frequently used in a variety of fields, including medical imaging, scientific visualization, and computer graphics [14]. The method involves the projection of a 3D volume onto a 2D plane, leveraging algorithms where colors and opacities are assigned to each point based on its physical characteristics, such as density or temperature. The resulting 2D image provides a representation of the entire 3D volume, allowing the user to visualize both the external and internal structure of the object. One of the main advantages of volume rendering is that it can simulate complex volumetric effects that are challenging to model using traditional geometric primitives, such as points, lines, triangles, or polygon meshes. Examples of such volumetric effects that may be better simulated with volume rendering include fluids, clouds, flames, smoke, fog, and dust [15].

There are multiple ways of rendering a volume. Well-known techniques include ray casting or raymarching [16], resampling or shear-warp [17], texture slicing [18], and splitting [19]. The basic algorithm, depicted in Figure 2.1, can be broken into four steps:

- **Ray casting:** Cast rays from the image plane into the volume. The volume is often enclosed within a bounding primitive like a cuboid.
- **Sampling:** As the ray enters the volume, sample equally spaced points from the volume. This equidistant sampling builds an intensity profile for the ray; density per volume depth. In the general case where the volume is not aligned with the ray's direction, the sampling point will be positioned in between voxels, and the values must be interpolated from its adjacent voxels.
- **Shading:** Utilizing the generated intensity profile and a transfer function, compute the *RGBα*-color and an illumination-value gradient. The gradient vector points in the direction of the greatest increase, indicating where the most rapid increase in illumination is. The samples are colored with their *RGBα* value and shaded according to the gradient vector and the location of the scene's light source.
- **Compositing:** The final color of the pixel is retrieved by compositing all the shaded samples along the ray.

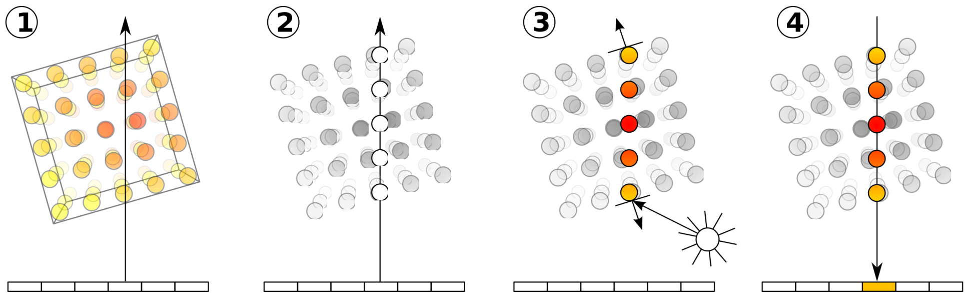

Figure 2.1: The four basic steps of volume rendering [20]. 1) A ray is cast from the image plane into the volume. 2) Points in the volume are sampled. 3) The points are shaded based on their *RGBα*-value and the illumination-value gradients in relationship with the local light source. 4) The final pixel color is obtained by compositing all the shaded samples along the ray.

2.2 Neural Fields

A neural field refers to a field that has been parameterized, either entirely or partially, by an Artificial Neural Network (ANN). The field is a quantity that may be defined for any set of temporal or spatial coordinates. By design, neural fields are both continuous and adaptable. Neural fields are commonly parameterized as Multilayer Perceptrons (MLPs) with gradient-defined activation functions [21].

A typical neural fields algorithm in visual computing would first, across space-time, sample coordinates and feed them into an ANN to produce field quantities. The field quantities are samples from the desired reconstruction domain of the problem, which defines how we represent the world, such as a radiance field or another appropriate representation. Then, we apply a forward map to relate the reconstruction to the sensor domain, which defines how we observe the world, such as through an RGB image. In the sensor domain, supervision is available and we calculate the reconstruction error that guides the ANN optimization process by comparing the reconstructed signal to the sensor measurement. Important parts from the neural field algorithm are visualized in Figure 2.2.

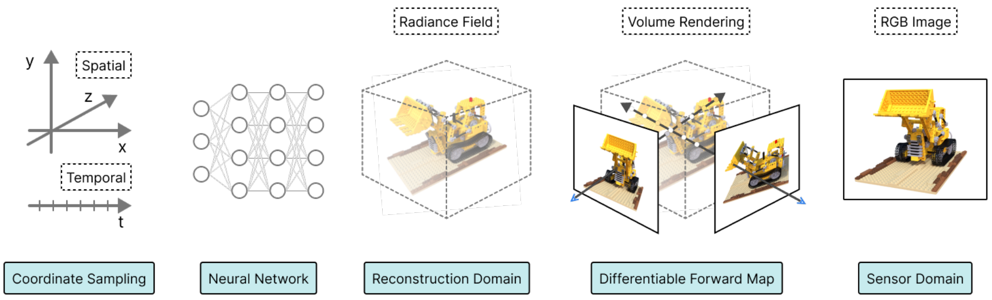

Figure 2.2: A typical neural fields algorithm in visual computing. Inspired by Figure 3 from Neural Fields in Visual Computing [21].

2.2.1 Neural Radiance Fields

The volumetric data we interact with through computer games, movies, and other computer graphic applications, are predominantly represented by meshes. A mesh is a collection of vertices, edges, and faces combined in order to define the shape of objects. Meshes are easy to manipulate and interact with. Another common representation of volume is voxels, where a 3D point in space is represented by a value, for example a color. Both of these modeling techniques are *explicit* representations. As we want to increase the resolution of a scene, we have to model increasingly smaller regions of space, that is, increase the number of voxels/triangle-meshes in the volume. This does not scale well as the memory requirements increase as we increase the number of voxels/triangle-meshes. Due to memory constraints, we have to find another way to represent the scene, instead of representing it as explicit blocks or meshes in space. NeRF provides an *implicit* representation of the scene by utilizing an MLP.

NeRF is a neural volumetric representation. The name, neural radiance fields, provides a clue as to what it is. It is a field, a space full of particles, where each particle has a given radiance, a color emitted by the particle in a certain direction, and the field is represented with an ANN. NeRF parameterizes 3D scenes as 3D neural fields, mapping 3D coordinates to radiance and density. The 3D scene can subsequently be rendered via volume rendering, as discussed in section 2.1.

2.2.2 Differentiable Rendering

NeRFs are made possible due to differentiable rendering, a major breakthrough in 3D reconstruction. Differentiable rendering enables the rendering process to be implemented in a way that is amenable to gradient-based optimization. This in turn allows for the use of techniques such as backpropagation to optimize the parameters of a scene in order to produce a desired visual result. It allows reconstruction of 3D neural fields representing shape and/or appearance given only 2D images, instead of 3D supervision. This is particularly valuable as 3D data is often expensive to obtain, while 2D images are ubiquitous. As a result, non-experts can become 3D content creators without the barrier of specialized hardware, which has important social implications [21].

2.3 NeRF - Representing Scenes as Neural Radiance Fields for View Synthesis

The first paper to present NeRFs was *NeRF - Representing Scenes as Neural Radiance Fields for View Synthesis* [10]. NeRFs provide a method for reconstructing 3D scenes and synthesizing novel views. Given multiple 2D images and their corresponding camera poses, NeRF builds a dataset by sampling points in the volume. The points in the dataset are passed through an MLP in order to predict the given points' density and color. The predicted density and color values are then composited into a final color which is compared to the reference image pixel's color. The MLP is optimized to minimize the difference between the predicted and reference pixel color.

Given multiple 2D images and their corresponding camera poses, which can be retrieved from calibrated camera rigs or approximated with SfM techniques as will be discussed in section 2.4, NeRF builds a dataset by sampling points in the volume. Points are sampled along a ray $\boldsymbol{r}(t)$ with origin $\mathbf{o}$, the camera's center of projection, and viewing direction $\mathbf{d}$. The sampling rate of significant parts of the volumetric scene is increased with a technique called *hierarchical sampling*, which will be discussed in section 2.3.

$$\boldsymbol{r}(t) = \boldsymbol{o} + t\boldsymbol{d} \tag{2.1}$$

Points (3D-coordinates) $\boldsymbol{x}_k = \boldsymbol{r}(t_k)$, where $t_k \in t$, in conjunction with their viewing direction $\boldsymbol{d}_k$, make up the dataset in which the MLP is trained on. It has been shown that MLPs have a hard time learning high-frequency signals given low-dimensional input [22]. In order to remedy this, the points are normalized to lie in the interval of $[-1,1]$ and the signals' dimensionality is increased by applying positional encoding. The dimensionality of the points and their corresponding viewing directions are increased by applying $\gamma(\cdot)$, shown in Equation 2.2. Positional encoding will be further explained in section 2.3.

$$\gamma(p) = [\sin(\pi p), \cos(\pi p), ..., \sin(2^{L-1}\pi p), \cos(2^{L-1}\pi p)]^T \tag{2.2}$$

A predicted color can be obtained by passing the encoded 3D coordinate and its corresponding encoded camera pose through the MLP denoted F_θ, where θ represents the network parameters. The output of F_θ is a 4D vector containing a color *RGB* and a density σ. In order to retain multiview consistency, NeRF first predicts the volume density σ as a function of only the location $\mathbf{x}$. Subsequently, the RGB color $\boldsymbol{c}$ is predicted as a function of both the location and viewing direction.

Using the points' density σ and color $\boldsymbol{c}$ we can approximate the volume rendering, as discussed in section 2.1, to synthesize a novel view of the scene. The expected color $C(\boldsymbol{r})$ can be derived by Equation 2.3, which further can be approximated with numerical quadrature as shown in Equation 2.4.

$$C(\boldsymbol{r}) = \int_{t_n}^{t_f} T(t)\sigma(\boldsymbol{r}(t))\boldsymbol{c}(\boldsymbol{r}(t),\boldsymbol{d})dt \quad T(t) = \exp\left(-\int_{t_n}^{t_f} \sigma(\boldsymbol{r}(s))ds\right) \tag{2.3}$$

$$\hat{C}(\boldsymbol{r}) = \sum_{i=1} T_i \alpha_i \boldsymbol{c}_k, \qquad \text{where } T_i = \exp(-\sum_{j=1}^{i-1} \sigma_j \delta_j), \tag{2.4}$$

$$\alpha_i = (1 - e^{-\sigma_i}), \tag{2.5}$$

$$\delta_i = t_{i+1} - t_i \tag{2.6}$$

The transmittance $T(t)$ represents the probability that the ray will not intersect any objects up to point t. $\sigma(\boldsymbol{r}(t))$ and $c(\boldsymbol{r}(t),\boldsymbol{d})$ represents the density and color of point $\boldsymbol{r}(t)$, respectively.

Since volume rendering is differentiable, we optimize the loss between the synthesized and ground truth observed image. This is done by calculating the total squared error between the ground truth pixel colors $C(\boldsymbol{r})$ and the synthesized pixel color $\hat{C}(\boldsymbol{r})$ over all the rays $\boldsymbol{r} \in \mathcal{R}$. This loss is called the *photometric loss*. Both the coarse and fine networks, subscripted with c and f respectively and later elaborated upon in section 2.3, are optimized over.

$$L = \sum_{\boldsymbol{r}\in\mathcal{R}} \left[\left\| \hat{C}_c(\boldsymbol{r}) - C(\boldsymbol{r}) \right\|_2^2 + \left\| \hat{C}_f(\boldsymbol{r}) - C(\boldsymbol{r}) \right\|_2^2 \right] \tag{2.7}$$

Positional Encoding

Positional encoding is one of the many proposed encoding schemes, first introduced in the original NeRF paper. It is a method used to increase the dimensionality of an input vector, as it is shown that deep networks are biased toward learning lower-frequency functions. It is important to note that positional encoding in the context of NeRF is distinct from the positional encoding employed in transformers.

Stratified sampling

Stratified sampling, illustrated in Figure 2.3, is a sampling method that has proven benefits in machine learning as it helps prevent overfitting. The sampling method can be broken into three steps:

- **Partition the dataset into strata**: In the case of NeRF, the rays are partitioned into equally sized bins where the points contained within a bin $\boldsymbol{x} \in [t_i, t_i + 1]$ along the ray $\boldsymbol{r}(t)$, comprises the respective stratum.
- **Determine the number of samples to take from each stratum**: The sample size from a stratum can vary based on predefined functions. Importance sampling can be achieved by increasing the sample size in accordance with a PDF.
- **For each stratum, apply simple random sampling**: Simple random sampling is applied to each stratum to select a predefined number of points randomly and with equal probability.

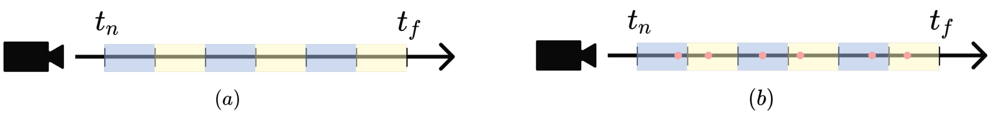

Figure 2.3: Illustration of stratified sampling. a) The ray is uniformly binned from the near bound t_n to the far bound t_f of a ray $r(t)$, defining the strata. b) A single random point is randomly sampled from each stratum.

Hierarchical sampling

When training a NeRF, different sampling strategies can be utilized. The sampling strategy is an important choice as it is core to how the dataset for the MLP is constructed. The original NeRF paper [10] proposed the use of a sampling approach called hierarchical sampling. This method involves training two MLPs: a *coarse* MLP and a *fine* MLP. During training, the coarse MLP employs stratified sampling which involves uniform interval sampling within each bin along the ray. The coarse MLP outputs a Probability Distribution Function (PDF) that highlights the samples that significantly contribute to the final predicted RGB value. This PDF is then passed to the fine MLP which sample points along the ray in accordance with the PDF. This strategy of having a coarse network predict the important areas to sample, that is the regions containing volume, facilitates the fine network in generating improved predictions.

An overview of the NeRF pipeline can be viewed in Figure 2.4

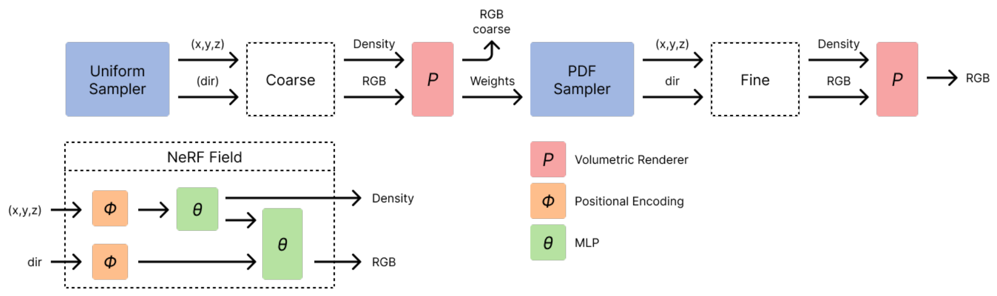

Figure 2.4: Overview of the NeRF pipeline

2.4 COLMAP (Structure from Motion)

SfM is a technique for estimating 3D structures from sequences of 2D input images. Although some NeRF approaches have endeavored to eliminate the necessity for pose supervision [23][24], accurate camera poses are typically a strict requirement in most NeRF methods. As a result, SfM techniques may be employed as a preprocessing mechanism for retrieving the camera poses of the input images.

COLMAP is a general-purpose SfM [12] and Multi-View Stereo (MVS) [25] pipeline. The general overview of the pipeline contains the following steps:

1. Feature detection and extraction
2. Feature matching and geometric verification
3. Structure and motion reconstruction

During the first step, Scale-Invariant Feature Transform (SIFT) [26] is used to extract features in the images. This algorithm finds sparse feature points in the image and describes their appearances using numerical descriptors.

During the second step, features are matched. Correspondences between the feature points are matched across different images, leveraging feature matching and geometric verification. There are different options for matching algorithms, some presented in Table 2.1. *Exhaustive matching* would match every image against every other image, leading to the best reconstruction results. The time complexity is not an issue if the number of images is relatively low (several hundreds). Another option is *sequential matching* which is useful if the captured images are in sequential order, for example, if they are sampled from a video.

During the third step, structure and motion is reconstructed. First, COLMAP will generate a sparse reconstruction of the scene, extracting the camera poses. The sparse output serves as input to the MVS to recover a dense representation of the scene. The dense representation estimates the dense surfaces. In NeRF-applications, COLMAP is primarily used as a preprocessing step for retrieving the camera poses of input images. Because of this, the pipeline is usually ceased after the sparse reconstruction.

Table 2.1: An overview of the time and memory complexities of a selection of COLMAP matching algorithms. For sequential matching k is the number of adjacent images each image n is matched against. For Vocabulary tree-based matching, k is the number of top-retrieved images that each image n is matched against.

Matching algorithm	Time complexity	Space complexity
Exhaustive	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
Sequential	$\mathcal{O}(nk)$	$\mathcal{O}(nk)$
Vocabulary Tree	$\mathcal{O}(n^2)$	$\mathcal{O}(nk)$

2.5 Nerfstudio

Since the publication of the original NeRF paper [10], a multitude of different methods regarding NeRF has been published. With the magnitude of published methods, some with corresponding source code and some not, it is not trivial to compare them on self-captured data. Nerfstudio [7] is an open-source framework/API that streamlines the process, training, evaluation, and rendering of NeRFs. The components that make up NeRFs are modularized in a way that allows interpretable implementation of different NeRF methods. In addition, Nerfstudio ships with implemented versions of some of the most important published methods to date for real-world captures. The core concepts within Nerfstudio are presented in Figure 2.5.

2.6 CARLA

CARLA [6] is an open-source simulator for AD research which enables the capture of synthetic data in a controllable and programmable environment. Its highly customizable simulation environment allows for the creation and testing of a wide range of driving scenarios and conditions. CARLA includes a variety of sensors, such as cameras, Light Detection and Ranging (LiDAR), and Global Positioning System (GPS)/GNSS, which can be used to collect both visual and non-visual data. We can interact with the CARLA simulator via a Python API, which allows for the control of vehicle position, orientation, and most other behaviors within the simulation. This flexibility enables the generation of diverse and realistic data, making CARLA, among many other things, a powerful tool for the development

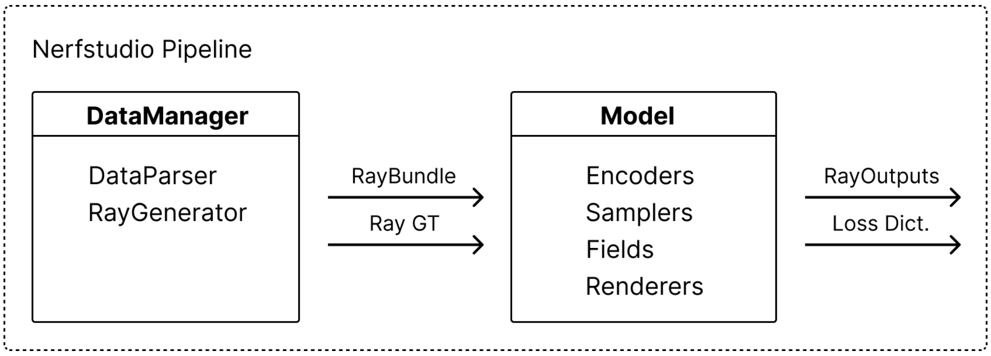

Figure 2.5: The components of the Nerfstudio pipeline. `DataManagers` process input images into bundles of rays (`RayBundles`) that are rendered by the `Model` to produce a set of NeRF outputs (`RayOutputs`). A dictionary of losses supervises the pipeline. Figure and caption adapted from Figure 2 in *Nerfstudio: A Modular Framework for Neural Radiance Field Development* [5].

and testing of data capture pipelines. Data captured from CARLA can be seen in Figure 2.6.

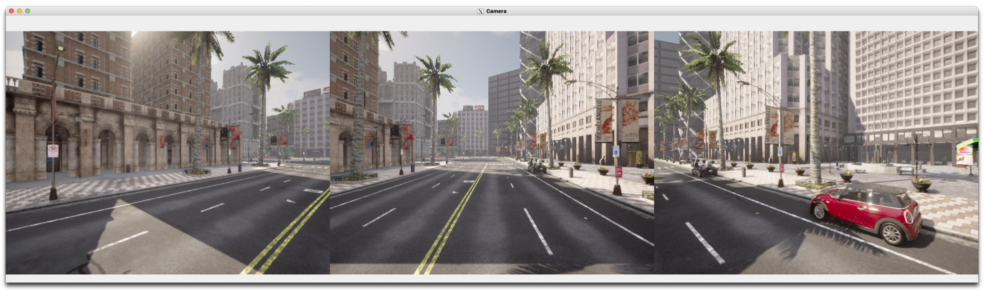

Figure 2.6: A screenshot from a simulation in the CARLA environment where a car has been equipped with three cameras rotated at different angles. The output from the mounted cameras have been horizontally stacked and rendered with OpenCV [27].

2.7 Evaluating NeRFs

Evaluating the quality of NeRFs is a difficult task due to the visual nature of the modality. Once a NeRF is trained, it is typically employed to render an image, making image similarity metrics the most critical measure for evaluating NeRF quality. The evaluation of image similarity has been a persistent challenge in computer graphics, but the following metrics are commonly used throughout most papers comparing NeRFs [28].

2.7.1 Peak Signal-to-Noise Ratio (PSNR)

PSNR is a common measure for quantifying reconstruction quality for images and videos. It builds upon Mean Squared Error (MSE) which measures the absolute difference between each pixel in two images I_1 and I_2.

$$\text{PSNR} = 20 \cdot \log_{10}\left(\frac{MAX_I}{\sqrt{\text{MSE}}}\right), \text{ where} \tag{2.8}$$

$$\text{MSE} = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I_1(i,j) - I_2(i,j)]^2, \tag{2.9}$$

and MAX_I represents the dynamic range of an image; the largest range of pixel values that the picture may contain. This is 255 when pixels are represented with 8 bits per sample. In more general terms, MAX_I is $2^B - 1$ when samples are recorded with B bits per sample. Due to the high dynamic range of many signals, PSNR is typically expressed as a logarithmic number using the decibel (dB) scale.

2.7.2 Structural Similarity (SSIM)

While MSE and PSNR are effective measures of reconstruction error, they do not evaluate an image in the same manner as humans. Rather than comparing pixel values in order to conclude if two images are similar, humans assess images holistically. SSIM attempts to replicate this approach by comparing an image's luminance l, contrast c, and structure s between two windows $\boldsymbol{x}$ and $\boldsymbol{y}$ of a common size $N \times N$.

$$\text{SSIM}(\boldsymbol{x},\boldsymbol{y}) = l(\boldsymbol{x},\boldsymbol{y})^{\alpha} \cdot c(\boldsymbol{x},\boldsymbol{y})^{\beta} \cdot s(\boldsymbol{x},\boldsymbol{y})^{\gamma} \qquad \text{with } \alpha,\beta,\gamma = 1 \tag{2.10}$$

$$= \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{2.11}$$

SSIM is bounded by $-1 \leq \text{SSIM} \leq 1$ where 1 implies perfect correlation, 0 implies no similarity, and -1 implies perfect anti-correlation.

2.7.3 Learned Perceptual Image Patch Similarity (LPIPS)

LPIPS [29] is a perceptual similarity metric based on deep network activations. Unlike MSE, PSNR and SSIM which uses relatively shallow functions in order to determine similarity across images, LPIPS leverages deep neural networks to obtain a metric that perceives similarity in a way that is similar to humans.

The dataset used to train LPIPS includes two types of perceptual judgments; Two Alternative Forced Choice (2AFC) and Just Noticeable Differences (JND). In 2AFC, people were asked to select which of two distorted images was “closer” to the reference, while in JND, people were presented with two image patches, one reference and one distorted, and asked if they were the same. Some samples from the dataset is depicted in Figure 2.7. By training on this dataset, LPIPS is able to capture higher-level features of images that are more relevant to human perception, such as object shapes and textures. This makes it a more effective similarity metric than traditional metrics like MSE, PSNR, and SSIM. Image patches with a low LPIPS score are perceptually similar.

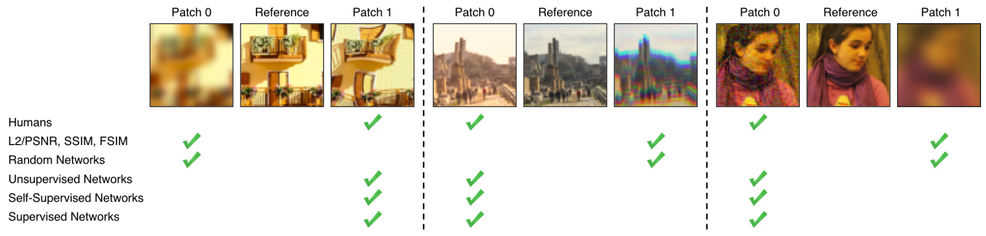

Figure 2.7: LPIPS is trained to perceive image similarity the same way humans do. The dataset used to train the similarity metric contains two types of perceptual judgments; 2AFC and JND. Figure 1 from LPIPS [29].

2.8 Related Work

Since the publication of the original NeRF paper, there has been a surge of related papers proposing new methods, techniques, and applications. This section covers some of the significant NeRF methods, including more specific work on the application of NeRF on vehicle-captured data.

2.8.1 Mip-NeRF: A Multiscale Representation for Anti-Aliasing Neural Radiance Fields

NeRF demonstrates impressive performance when the training and evaluation images are captured at similar distances from the scene, without the need to account for scale or aliasing effects. When you add more cameras, pulled away from the scene, NeRF starts to deteriorate because it is a single-scale model now trying to solve a multi-scale problem. As a consequence, renderings from the NeRF exhibit aliasing artifacts in distant views and excessive blur in close-up views.

A solution to this problem would be to adopt a technique that is used in offline ray tracing, *supersampling*. With supersampling, we would march multiple rays through the same pixel’s footprint. This technique does not resolve the issue of aliasing effects, but it does result in improved visual quality of the rendered image. However, doing so would be very computationally expensive and it would further increase the lengthy training times of NeRF, which already relies on querying an underlying MLP hundreds of times for a single ray.

Another sampling strategy apart from casting rays through each pixels' footprint would be to cast a cone, as seen in Figure 2.8. The cone's radius is determined by the size of the pixel's footprint on the image plane, thereby enabling the cone to model the whole volume of space visible by the pixel. When the pixel's cone is rendered, all the content within that visible volume will be averaged out, instead of rendering out whatever intersects with the infinitely narrow ray cast by NeRF. The cone is divided into conical *frustums*. The frustums are approximated with a multivariate Gaussian since they are easier to manipulate and have a closed-form solution.

Instead of positionally encoding a single point along the ray, we compute the expected positional encoding with respect to the constructed Gaussian. With this encoding, Mip-NeRF is able to reason about the scale of its input by analyzing the scale of the encodings. This allows the model to understand the difference between small and large volumes. This encoding scheme is called Integrated Positional Encoding (IPE), and has a simple closed-form solution that can be computed quickly.

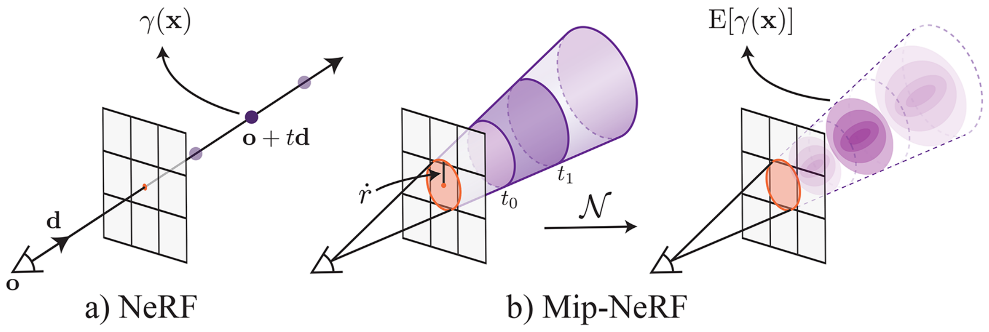

Figure 2.8: A comparison of sampling strategies. NeRF (a) samples points along rays traced through each pixel before positionally encoding the points. Mip-NeRF (b) cast cones through the pixels' footprint and into the volume before applying IPE. Figure 1 from Mip-NeRF [4].

This method draws inspiration from mipmapping [30], a technique traditionally used in computer graphics pipelines to mitigate aliasing artifacts. Mipmapping involves generating a pre-filtered set of discretely downsampled signals, typically images, which accelerates rendering by shifting the responsibility of anti-aliasing to a pre-computation phase. Mip-NeRF extends NeRF to simultaneously represent the pre-filtered radiance field for a continuous space of scales, thereof "Mip-NeRF".

Figure 2.9 provides an overview of the Mip-NeRF pipeline. The primary distinction from the NeRF pipeline, as depicted in Figure 2.4, is the incorporation of the Mip-NeRF field, which utilizes IPE.

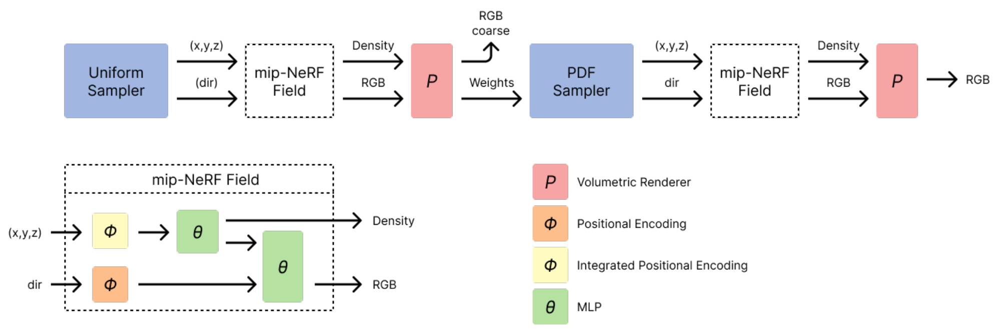

Figure 2.9: Overview of the Mip-NeRF [4] pipeline.

2.8.2 Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields

Mip-NeRF introduces several valuable techniques for advancing the capabilities of NeRFs. However, the techniques are primarily focused on forward-facing scenes [31] as opposed to unbounded scenes that are object-centric, featuring elaborate backgrounds and images captured from 360° around the object. To extend the capabilities of Mip-NeRF to unbounded scenes, Mip-NeRF 360 [11] proposes three primary techniques, which are covered in this subsection.

Representation

The first challenge with extending Mip-NeRF to unbounded scenes is that such scenes are large, but Mip-NeRF requires a bounded domain. Mip-NeRF handles scenes unbounded in one direction by warping the space into *projective space*, Normalized Device Coordinates (NDC), but the challenge arises when the scene is unbounded in all directions. A solution is to apply a Kalman-like warp to Mip-NeRF Gaussians in order to warp the Mip-NeRF into non-Euclidean space. All the Gaussians outside a sphere of radius one will smoothly be warped into a non-euclidean space within a sphere of radius two. This non-Euclidean space is used to represent the input to the MLP.

Efficiency

Expanding a bounded scene to an unbounded scene results in larger scenes that require a more substantial network capacity. However, using a large MLP is too expensive given that it has to be queried hundreds of times for a single ray. A solution is to distill scene geometry from a large NeRF MLP into a small *proposal MLP* while training. The proposal MLP will only output a set of weights, no colors. By feeding a set of location points through the proposal MLP, the outputted weights can then be used as a PDF to resample the ray, similar to how the coarse network in hierarchical sampling guides the fine network's sampling. The resampled points are then used to render a color, which as normal is supervised with photometric loss. Rather than supervising the proposal MLP to accurately reconstruct the image, which is done for both the coarse and fine MLPs in Mip-NeRF, the output weights are supervised to be consistent with the output weights from the NeRF MLP. This is accomplished using a loss function that encourages the outputted weight histograms to be consistent with one another. This is made possible with some strong assertions on the relation between the two distributions, which ultimately summarizes the same underlying and true distribution. This new approach to training accelerates training speeds by 300%.

Ambiguity

Reconstructing 3D content from 2D photos is inherently ambiguous since the content of unbounded scenes can be everywhere and will only be seen by a tiny number of rays. This problem becomes more pronounced as scene size increases. The original NeRF-paper partially addressed this behavior by introducing random Gaussian noise to the output σ values, before passing them through the Rectified Linear Unit (ReLU) [32]. This stimulated densities to drift toward either zero or infinity, which slightly enhanced visual performance. However this regularization is insufficient for the more challenging task that Mip-NeRF 360 tackles. Instead, Mip-NeRF 360 introduces a novel regularizer, specifically designed for Mip-NeRF ray intervals, that encourages each ray's histogram to be as close to a delta function as possible. This regularizer reduces the occurrences of "floaters", which are semi-transparent objects that appear to be floating in space.

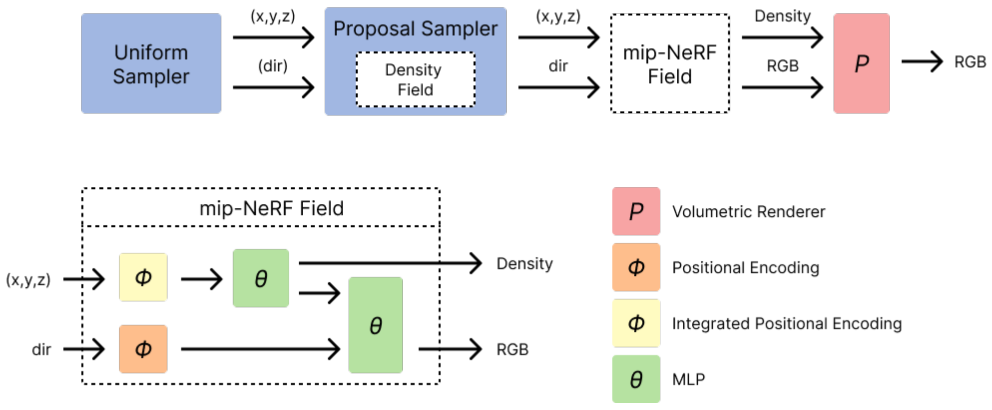

Figure 2.10: Overview of the Mip-NeRF 360 [11] pipeline

Figure 2.10 provides an overview of the Mip-NeRF 360 pipeline, which differs in several respects from the previous Mip-NeRF pipeline depicted in Figure 2.9. Notably, the pipeline does not render a redundant RGB value, and instead features two distinct fields: a density field and a Mip-NeRF field.

2.8.3 Block-NeRF: Scalable Large Scene Neural View Synthesis

Block-NeRF [2] is a paper that demonstrates a method for reconstructing large-scale scenes using NeRFs. This is achieved by splitting large areas into multiple blocks of a certain radius, with each block having a connected NeRF referred to as a Block-NeRF. The different Block-NeRFs are trained on images within their respective radius of responsibility. At inference time, only the Block-NeRFs with a radius that spans the requested location are kept. These have all been trained on image data from the requested location and can render an output. Some of the remaining Block-NeRFs might still lack a direct line of sight to the requested location, which results in low "visibility", which will be discussed further in section 2.8.3. These Block-NeRFs are also filtered out, resulting in a pool of Block-NeRFs with good visibility. The remaining Block-NeRFs render the given location, and their outputs are merged to render the final image output.

Block-NeRF leverages multiple techniques in order to enable the reconstruction of large scenes. These include *appearance embeddings*, *learned pose refinement* and *visibility prediction*.

Appearance embeddings

Per-image appearance conditioning is a technique that was first proposed for NeRFs in NeRF-W [33] and has since been employed in multiple other methods. The appearance embeddings help reduce artifacts in the scene, especially "ghosting" artifacts which present themselves as fog in the final render. The appearance embedding is a vector in a low-dimensional space, unique for every input image, that is optimized jointly with the NeRF in order to allow the NeRF to process and represent 3D scenes with variable lighting, exposures, weather, and post-processing effects. To account for these variations, the final part of the MLP is conditioned by passing the viewing direction concatenated with the appearance embedding.

Learned pose refinement

Camera pose refinement is a technique that has been proposed to alleviate the strict requirement for accurate camera poses in NeRF. This is accomplished by treating the camera poses and intrinsics as learnable parameters and jointly optimizing them with the 3D scene representation, that is, optimizing both the photometric loss and the corresponding camera poses. It was proposed for forward-facing scenes in *NeRF--* [24] and later built upon in *BaRF* [23] to support the use on object-centric scenes. Block-NeRF leverages this technique on a per-driving-segment basis. Although it is a technique optimized for forward-facing and object-centric scenes, Block-NeRF demonstrates the technique's efficacy in reducing cloudy artifacts and increasing the sharpness and overall quality of the resulting 3D representation.

Visibility prediction

Visibility prediction is performed to predict whether a given point is within the line of sight of a specific Block-NeRF. The prediction is made by a secondary MLP F_v that is trained to learn an approximation of the visibility of a sampled point. Given a location and a viewing direction, F_v outputs an approximation of that location's transmittance (T in Equation 2.3). The transmittance of a location will be close to 1 if it is visible, that is, if it is located in free space or on the surface of the first intersected object. Objects inside or behind the first intersected object will have a transmittance close to 0. If the point is visible from multiple viewing directions, the resulting transmittance will be the average of these observations. F_v is supervised by the Block-NeRF's main MLP F_σ.

2.8.4 Instant-ngp: Instant Neural Graphics Primitives with a Multiresolution Hash Encoding

Instant-ngp [3] is a paper that introduces a method for accelerating the training- and inference of NeRFs. Previous NeRF methods have required hours or even days of training to learn a scene, but the team at Nvidia was able to significantly reduce both the training- and inference time while maintaining the same level of quality. This achievement represented a significant advancement in the field of NeRFs, and has greatly improved the efficiency and effectiveness of NeRF-based applications.

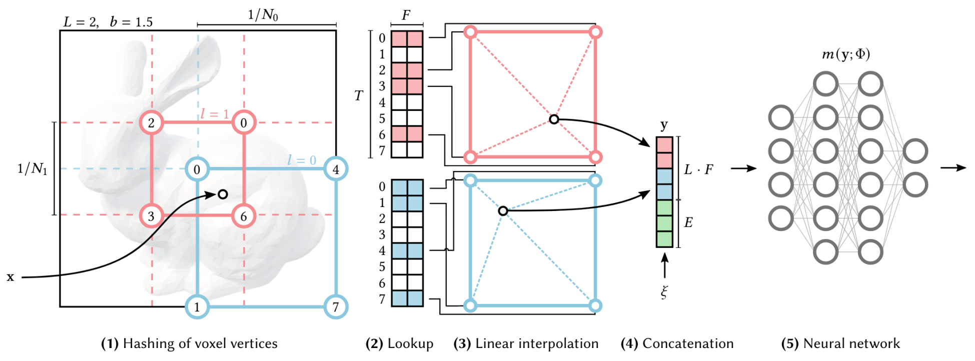

Figure 2.11: Illustration of the multiresolution hash encoding in 2D. Figure 3 from Instant-ngp [3].

The main technique proposed by Instant-NeRF is a new parametric encoding for the scene's spatial data, coined *Multi-resolution hash encoding*. In the original NeRF paper, the location $\mathbf{x}$ is represented by a positional encoding as described in section 2.3. With the multiresolution hash encoding, the location $\mathbf{x}$ is represented in a hash table by a linear interpolation of its closest vertices at multiple resolutions. This parametric encoding has several advantages in terms of computational effectiveness, resulting in several magnitudes of increased training and inference speed. Although a larger memory cost is imposed by allocating several hash tables, the number of required parameter-updates per backpropagation is significantly reduced. An overview of the multiresolution hash encoding can be seen in Figure 2.11.

2.8.5 Nerfacto

Nerfstudio, discussed in section 2.5, also provides its own method dubbed Nerfacto [5]. Nerfacto leverages techniques from several other published methods that have proved to work well for real data capture. The combination of techniques, partially depicted in Figure 2.12, results in a method that strikes a great balance between quality and speed. The primary techniques implemented in Nerfacto have already been discussed. However, they will be briefly listed and summarized in this section:

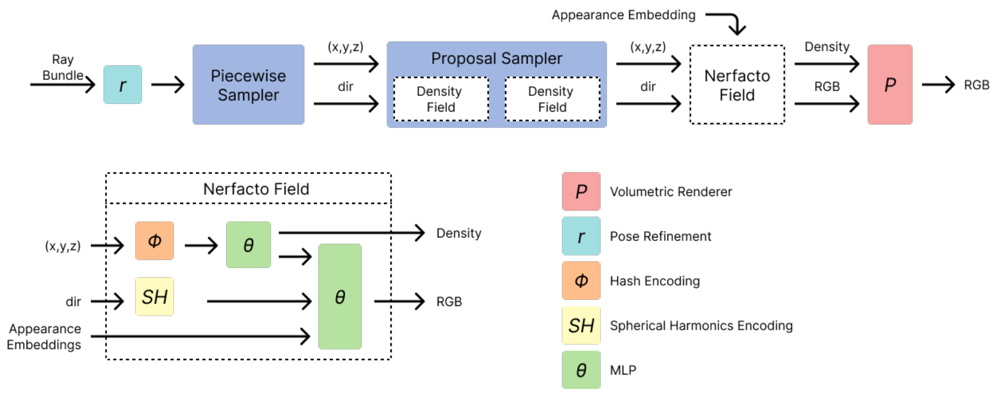

Figure 2.12: Overview of the Nerfacto pipeline

Camera pose refinement, as described in section 2.8.3 is a technique proposed to reduce the impact of imperfect camera poses. It is an effective measure to reduce cloudy artifacts and increase the sharpness and overall quality of the resulting 3D representation.

Per image appearance conditioning, as described in section 2.8.3, is a technique that allows the NeRF to process and represent 3D scenes with variable lighting, exposures, weather, and post-processing effects. In Nerfacto, the appearance embedding is a vector of size 32, which is concatenated with the viewing direction before it is passed through the MLP.

Hash encodings, as described in subsection 2.8.4, is an effective encoding scheme used to severely decrease training- and inference time. In Nerfacto, 16 hash tables with 2^{19} rows, each storing a feature vector of size 2, are allocated. The subsequent MLP has a very low capacity, with only one hidden layer containing 64 neurons.

Proposal sampling, as described in subsection 2.8.2, is a sampling technique used to increase the sampling density of areas that contribute most to the final render. Nerfacto extends the proposal sampler used in Mip-NeRF 360 [11] by utilizing two density functions implemented as small fused-MLP with hash encodings [3]. This provides accurate and fast density estimations.

Scene contraction, as described in subsection 2.8.2, is a technique proposed in Mip-NeRF 360 [11] to extend Mip-NeRF to support unbounded scenes.

2.8.6 Wayve - Building City-Scale Neural Radiance Fields for Autonomous Driving

Wayve[1], a London-based company, is a leading innovator in AD technology. At Nvidia GTC, they presented a talk on how they leverage NeRFs to extend their pipeline used to train autonomous vehicles. Their initial need was a simulator that was indistinguishable from reality, but building such a simulator is very time-consuming, challenging, and expensive. Their solution was to use NeRFs to automatically "build the simulator with data."

The pipeline, partially depicted in Figure 2.13, begins by creating a large corpus of data, captured by a specialized Wayve-vehicle fitted with high-quality cameras. The corpus is then split into data segments spanning roughly 100 meters and containing a small overlap between the previous and successive segments. In order to mask out transient objects and separate foreground and background, a segmentation model is applied to each segment. After this, each segment utilizes COLMAP to approximate the images' corresponding camera poses. Once the camera poses are acquired, each of the NeRFs are trained in parallel with an undisclosed method. Using the capture trajectory, a camera path is created and rendered by swapping NeRFs on-demand when the current camera within the path is close to another NeRF's boundary.

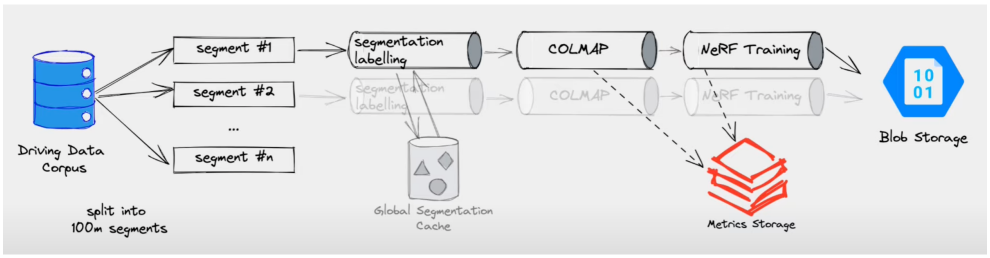

Figure 2.13: An overview of Wayve's pipeline for large-scale NeRF. Screenshot from Wayve's talk at Nvidia GTC [34].

Wayve's pipeline is simpler and more straightforward than that of Block-NeRF because their goal is to stitch together scenes that were recorded one after the other, rather than creating a scene that can be traversed in all directions. This differ-

[1]`https://wayve.ai`

ence in objective allows Wayve to sidestep many of the technicalities involved in Block-NeRF making their pipeline more streamlined.

Chapter 3

Methods

This chapter is split into multiple sections, all essential for the end-to-end pipeline, depicted in Figure 3.1, to allow the capture of data and subsequent training, evaluation, and render of a NeRF. The following sections thoroughly discuss the separate components that comprise the end-to-end pipeline. The code developed and used throughout the project can be found in the attached GitHub repositories[1][2][3].

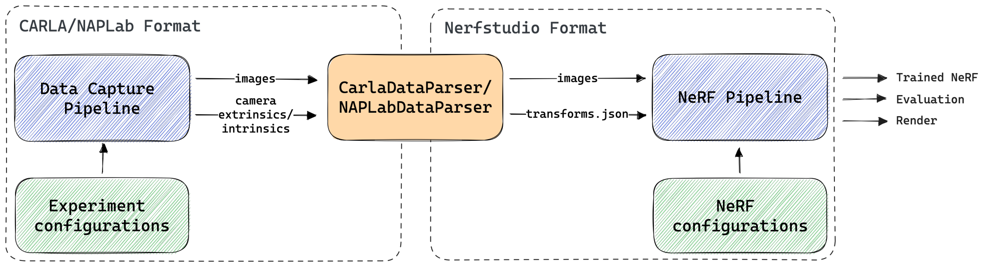

Figure 3.1: Overview of the end-to-end pipeline that captures data from CARLA or the NAPLab car, converts the data into Nerfstudio format, and utilizes the resulting data to train, evaluate, and render a camera path for the NeRF.

3.1 Implementing the CARLA Data Capture Pipeline

This section will explain how the CARLA-simulator, introduced in section 2.6, is utilized to capture synthetic data. It'll cover basic CARLA-concepts and give a thorough explanation of how the different concepts are combined and utilized in order to create a data capture pipeline.

[1]https://github.com/olestole/nerfstudio
[2]https://github.com/olestole/carlo
[3]https://github.com/olestole/naplab-data-capture

Creating a CARLA Actor

To capture data in the virtual CARLA-environment, a vehicle that is both controllable and programmable is required. This vehicle is commonly referred to as the "ego" vehicle (`carla.Vehicle`) and is a special instance of the most basic CARLA instance, the `carla.Actor`. It can be spawned by defining a spawn point, a choice of vehicle, and whether the vehicle should operate on autopilot or not. Although an arbitrary number of autopilot vehicles can be spawned to simulate a complex traffic environment, the objective is to capture synthetic data with minimal transient objects; therefore, only the ego vehicle is spawned.

Setting up the environment with the traffic manager

In order to define the environment in which the spawned ego vehicle will operate, we leverage the `carla.TrafficManager` available on the `carla.Client` that is used to connect to the simulator. The traffic manager enables the definition of some important aspects of the environment:

- **Define the ego vehicle's route:** The traffic manager allows us to set the route instructions for a specific actor. We can define arbitrary routes by creating an array of route instructions, for example, `['Left', 'Left', 'Right', 'Straight', 'Right']`.
- **Select to ignore the traffic lights:** Since the only moving vehicle in the environment is the ego vehicle, we do not have to abide by the rules of traffic. In order to speed up the data capture we choose to ignore the traffic lights.
- **Set the vehicle speed:** The traffic manager provides the ability to configure a vehicle's speed as a percentage of the default speed of 30 km/h. A setting of 100% adheres to the default speed, while a setting of 50% corresponds to a speed of 15 km/h, and so on.

Experiment configuration

In order to evaluate numerous environment and vehicle setups, along with their corresponding outcomes, an `ExperimentConfig`-class was created. This class enables the definition of the following configurations:

- **Camera rig:** Mounts a list of RGB-cameras with configurable camera settings, location, and rotation to the ego vehicle. A "rig"-file, a special type of JSON file exported from the NAPLab car discussed in section 3.7, can optionally be parsed into a camera rig for the ego vehicle.
- **Data capture frequency**: Sets the frequency of data capture, namely images and camera poses.
- **Stop-criteria:** Specifies the ego vehicle's stop condition either by a stop distance or a number of completed turns.

- **Camera noise:** Enables the simulation of noisy GPS/GNSS sensor readings by adding Gaussian noise to the location output from the mounted cameras.
- **Spawn transform:** Enables specifying the spawn-location/-rotation of the ego vehicle.
- **Speed:** Sets the target speed for the ego vehicle.
- **Route:** Sets the route instructions for the ego vehicle.

CARLA data parser

To capture and store data during the vehicle's drive, a custom `CarlaDataParser`-class is created. This class provides methods for capturing an image with its corresponding camera pose, transforming the camera poses into specified formats, and exporting the captured data. The format and data conventions will be thoroughly described in section 3.2.

CARLA data capture loop

The CARLA Run-loop includes the following steps:

- Read and apply the configurations from `ExperimentConfig`.
- Spawn an ego vehicle.
- While the ego vehicle has not met a stop-criteria, such as having driven the set amount of distance/turns, keep driving and capture data.
  - Get the image from all the cameras mounted to the car: Each sensor in CARLA has a `listen` method, triggered whenever data is retrieved by the sensor. The Carlo-repository[4] simplifies the process of obtaining image data from the corresponding camera, a process which traditionally would involve managing the listen-callback, decoding the `carla.Image` data, and converting raw data bytes into `np.ndarrays`. Using the Carlo abstraction, the steps are reduced to creating a list of cameras, retrieving the `np.ndarray` containing the image data, stacking the camera outputs, and presenting the resulting image using a library like `OpenCV` [27].
  - Pass the captured image and corresponding camera pose to the `CarlaDataParser`'s `append_frame` method.
- Export the parsed CARLA-data with `CarlaDataParser`'s `export_transforms` method.
- Terminate the simulation and perform necessary clean-up activities.

[4] `https://github.com/olestole/carlo/`

3.2 Implementing the CARLA Data Parser for Nerfstudio

Having obtained a pipeline that enables the creation of a controllable environment for a vehicle with an arbitrary setup of sensors, and subsequent data-capture, the captured data have to be exported in a format readable by Nefstudio. This section will elaborate on the creation of `CarlaDataParser` and the conversion from CARLA to Nerfstudio

In order to train a NeRF, images and corresponding camera poses are needed. As discussed in section 2.3, the camera poses are $4x4$ homogeneous transformation matrices, containing both the translation and rotation in reference to a coordinate system. Nerfstudio expects the data in a file structure as shown in Figure 3.2. In order to handle the data parsing from CARLA to Nerfstudio we create a helper-class `CarlaDataParser` which has methods for saving images as PNG files, calculating the camera's intrinsic parameters, transforming the extrinsic parameters, and appending parsed frames to a `transforms.json`-file. The `transforms.json`-file contains the camera's intrinsics and a list of frames where each frame holds an image path and the respective image's camera pose. An example `transforms.json`-file can be seen in Figure 3.3.

```
your_nerf_data/
├── images/
│   ├── 0001.png
│   ├── 0002.png
│   ├── 0003.png
│   └── ...
└── transforms.json
```

Figure 3.2: File structure expected by Nerfstudio

CARLA provides access to basic camera attributes that can be used to approximate the intrinsic parameters. Given the image's width w and height h, in accordance to the camera's Field of View (FOV), we can calculate the focal length f and subsequently the x- and y-component of the focal length f_x and f_y:

$$f = \tan\left(\frac{\text{FOV}}{2}\right) \qquad f_x = \frac{0.5 \times w}{f} \qquad f_y = \frac{0.5 \times h}{f}$$

The camera's principal points c_x and c_y are assumed to be the center of the image plane, and obtained with:

```
{
    "camera_model": "OPENCV",
    "fl_x": 200.0,
    "fl_y": 150.0,
    "cx": 200.0,
    "cy": 150.0,
    "w": 400,
    "h": 300,
    "k1": 0,
    "k2": 0,
    "p1": 0,
    "p2": 0,
    "frames": [...]
}
```

Figure 3.3: Example of a `transforms.json`-file with the intrinsic parameters and a collapsed list of frames containing the extrinsic parameters of the camera.

$$c_x = \frac{w}{2} \qquad\qquad c_y = \frac{h}{2}$$

Once calculated, the intrinsic values are added to the `transforms.json`-file.

CARLA is built with Unreal Engine and subsequently uses its coordinate convention. The Unreal Engine coordinate convention, illustrated in Figure 3.4, is a left-handed system where +X is forward, +Y is right, and +Z is up. Nerfstudio uses the OpenGL/Blender coordinate convention for cameras, which is a right-handed system, and its world space is oriented such that +X is right, +Y is forward, and +Z is up. The disparity in coordinate conventions between CARLA and Nerfstudio would make the NeRF-renderings non-interpretable, if we trained a NeRF directly on the transformation matrices exported from CARLA. In order to convert the transformation matrices that contain both rotation and translation we apply `CarlaDataParser`'s `carla_to_nerf`-transformation. The transformation from CARLA's left-handed coordinate system to NerfStudio's right-handed coordinate system can be represented as a matrix multiplication.

Let M_{carla} be the transformation matrix in CARLA convention, and M_{nerf} be the transformation matrix in Nerfstudio convention. The transformation can be expressed as:

$$M_{nerf} = R_z(90) \cdot R_x(-90) \cdot T \cdot M_{carla}$$

where T is a translation matrix to swap the y and z coordinates, and R_x, R_z are rotation matrices around the x and z axes, respectively. The order of multiplication is from right to left.

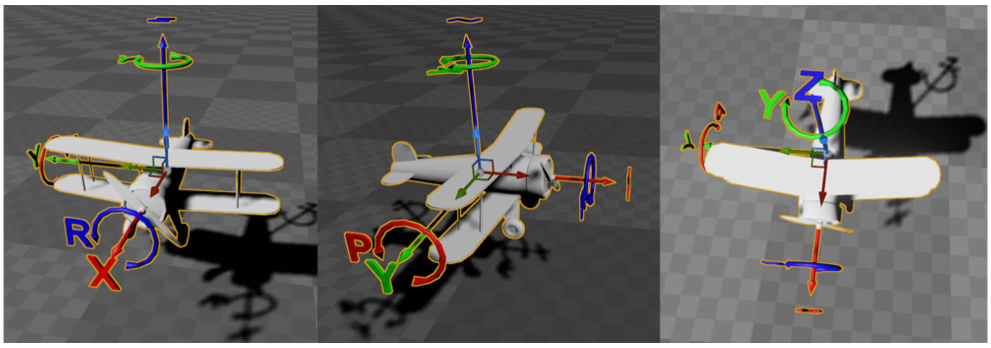

Figure 3.4: CARLA uses the Unreal Engine's coordinates system, which is a Z-up left-handed system [35].

3.3 Implementing the NeRF Pipeline

Having obtained the synthetic data captured from CARLA in a format suitable for training NeRFs in Nerfstudio, the next step is to develop and utilize a pipeline that automates the processing, training, evaluation, and rendering of various experiments using this data and the Nerfstudio API. This section will elaborate on the pipeline and its components depicted in Figure 3.5.

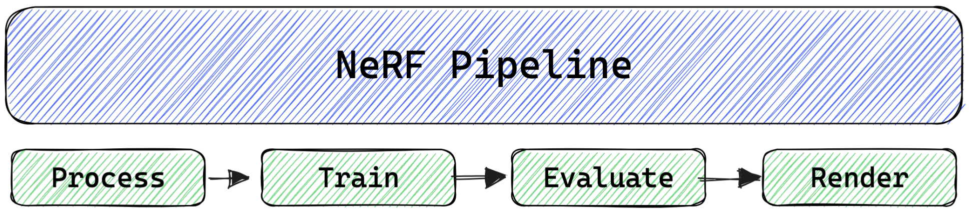

Figure 3.5: The components of the NeRF pipeline implemented in Nerfstudio. The quantitative results are obtained through the evaluation step, while the qualitative results are obtained through the render step.

Process

In most scenarios when working with real data, such as images of an object captured with a handheld camera, we do not have accurate camera poses. In those scenarios we have to pre-process the data in order to approximate camera poses for the captured images. It can be accomplished by leveraging SfM-algorithms as discussed in section 2.4. In the scenario where the data is captured from CARLA, the poses are as accurate as they can be as they've been calculated in a deterministic environment. Assuming the CARLA-data is parsed according to the method discussed in section 3.2, no further preprocessing is necessary.

Train

During training, the parameters in the model are optimized to represent the 3D scene. A batch of pixels is created for each training iteration. The default batch of 4096 pixels is obtained by randomly sampling the training images which defaults to 90% of the total dataset, leaving 10% as an evaluation dataset. Rays are marched through the sampled pixels and the model's networks predict RGB values and densities for the points sampled along the rays. The points' values are composited and the photometric loss, in combination with more complex losses, is computed. The losses are backpropagated through the model's networks and the ADAM optimizer[36] is used to update the network parameters.

For the Nerfacto-model, the primary model used throughout this thesis and previously discussed in subsection 2.8.5, the training process entails backpropagating the loss and updating both the NeRF MLP and the proposal MLP. The Nerfacto-model leverages three different losses to guide the training; *photometric loss*, *interlevel loss*, and *distortion loss*. Photometric loss is the standard NeRF-loss explained in Equation 2.7. Interlevel- and distortion-loss are both from the Mip-NeRF 360 [11] implementation explained in subsection 2.8.2. The interlevel loss encourages the model to generate consistent predictions across different levels of the multiscale hierarchy, while the distortion loss encourages the model to generate smooth and continuous predictions.

Nerfstudio provides an API to configure all aspects of the pipeline, including dataset-split, number of pixels to sample, which optimizer to use, the configuration of the optimizer's exponential decay, the ANN's dimensionality, and much more. The implementation details and configurations used for the models in this thesis are attached in section A.1.

Evaluate

To quantitatively evaluate the quality of the NeRF models, we utilize the metrics PSNR, SSIM, and LPIPS, as elaborated upon in section 2.7. During the evaluation, the camera poses from the evaluation dataset are fed into the trained model which subsequently renders the novel views. The rendered images are then compared against the corresponding ground truth images from the evaluation dataset according to the aforementioned metrics.

The Nerfstudio API provides a script to load a model-checkpoint and compute these metrics, providing a quantitative assessment of the trained NeRF. To further facilitate the comparison of NeRF models across multiple experiments, an additional script was developed. This script load the evaluation outputs of multiple experiments, compares the resulting metrics, and produces a formatted LaTeX table that highlights the best and worst metrics.

Render

Although the quantitative metrics offer valuable insights into the quality of the model, it is crucial to assess the qualitative output. Consequently, all trained NeRF models are rendered in order to visually compare the results across different methods, configurations, and datasets.

The Nerfstudio API provides a script for rendering trained NeRF models given a camera path; a sequence of camera poses, each with a corresponding FOV and aspect ratio. The Nerfstudio viewer, depicted in Figure 3.6, enables the creation and editing of camera paths, which can be exported and used for rendering. In addition to the possibility to create a custom camera path, a script to render the NeRF based on the input data's camera path was created. This option enables the creation of a side-by-side render, a useful and intuitive way to evaluate how well the NeRF has learned the input scene, that is, qualitatively assessing the NeRF's performance.

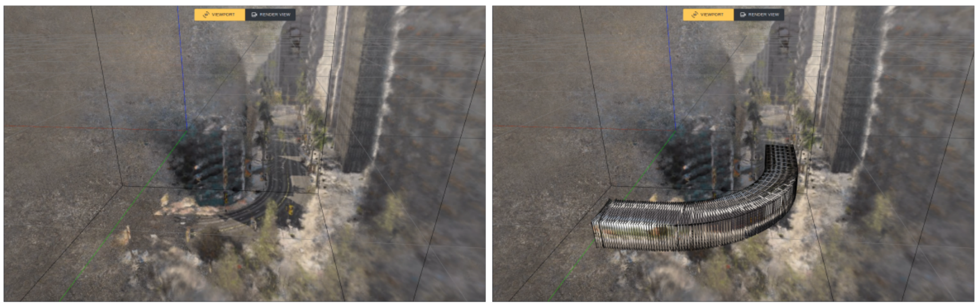

Figure 3.6: An overview of the Nerfstudio viewer. The scene seen on the left has been trained on the set of images and camera poses revealed within the viewer in the image to the right.

3.4 Establishing the CARLA-Baseline

A pipeline to evaluate numerous CARLA configurations to identify a configuration that generates high-quality data for NeRF training has been established. In order to facilitate further experiments, a baseline need to be created.

The experiments chosen to define a suitable baseline are mostly based on heuristics and knowledge of what is important for good NeRF results. When capturing video or images for a NeRF it is important that the scene is well-lit, that the captured images are not blurry, and that there are no transient objects present. If the camera poses of the captured images are to be approximated with the use of SfM-methods like COLMAP, it is also very important that the images have an overlap in order to secure feature-matching across the images. The five experiments chosen to define the baseline, which will be elaborated upon in chapter 4, were:

- **Camera setup:** How many cameras should be mounted to the ego vehicle, and in what translation and rotation?
- **Capacity:** How long should the segments used to capture data be?
- **Number of frames:** How many frames should be captured?
- **Image size:** What resolution should the mounted cameras capture at?
- **Vehicle speed:** At what speed should the ego vehicle drive while capturing data?

The best results from each of the experiments, based on qualitative results and quantitative metrics discussed in section 2.7, were used to iteratively build a baseline used for further experiments.

3.5 Comparative Experiments with the CARLA-Baseline

Having established a baseline, the impact of different NeRF- and capture settings on the quality of the resulting models can be evaluated. This provides a starting point for conducting further experiments, which can be systematically varied to explore the factors that impact the quality of resulting trained NeRF. Additionally, we can simulate real-world capture scenarios using the synthetic data generated by the baseline. The specific experiments ran with the established baseline, elaborated upon in chapter 4, were:

- **Simulated noise conditions:** Noise in the camera poses due to inaccurate readings from the GPS/GNSS. Additionally, the COLMAP's effectiveness in approximating the camera poses is compared to the joint camera optimization.
- **Comparing NeRF-models:** How do the different NeRF models perform and compare against each other?
- **Large-scale NeRF**: How can we train NeRFs on increasingly larger scenes?
- **Real data**: How does the end-to-end pipeline extend to real data?
- **Novel views along altered trajectory**: How is the quality of renders from altered trajectories?

3.6 Extending the Pipeline to Support Large-Scale Scenes

When we change the route of the CARLA-baseline and create a larger dataset, spanning kilometers of road data, the NeRF models evidently have a hard time generating high-quality image synthesis. That result is expected as the models' underlying MLPs only have a certain capacity. We could increase the capacity by increasing the number of hidden layers and neurons per layer, but this would lead to linearly increasing training -and rendering times. Rendering is already an expensive operation which further supports the claim for another solution.

As discussed in subsection 2.8.3 it is an open research field within the NeRF community to expand the capability of NeRFs to enable the representation of large scenes. Compared to the other fields within NeRF research, there are not a lot of papers exploring large-scale NeRFs, which might be due to the amount of data needed and the corresponding data capture endeavor. Luckily, we have constructed a pipeline that automates synthetic data capture and the subsequent processing, training, evaluation and rendering of the NeRF. Due to this we expand the pipeline to enable the evaluation of a large-scale NeRF approach, based on a naive implementation of Block-NeRF [2].

One of the main challenges in implementing Block-NeRF is obtaining the camera poses for the captured images. Traditional SfM methods, such as COLMAP, become computationally expensive and slow when dealing with large datasets, as is demonstrated by the feature matching complexity overview presented in Table 2.1. However, being in possession of the image's corresponding camera poses simplifies the process. As we are in possession of both, a naive Block-NeRF POC was created. The steps in the algorithm can be summarized as follows:

- **Split the dataset into multiple smaller datasets:**
  The *split_transforms* function takes an original `transforms.json` file and a sequence of images and splits them into *n* roughly equal-sized new datasets, which are formatted according to the structure shown in Figure 3.2, resulting in a file structure shown in Figure 3.7.
- **Train separate NeRFs on the split dataset:**
  Run a standard training loop on each of the *n* datasets created in the previous step.
- **Create a camera path spanning the segments contained in the complete dataset:**
  There are multiple options for creating the camera path: A custom camera path can be created in the Nerfstudio viewer, a previously exported camera path can be used, or a helper function could be utilized to extract the camera path of the input images. The latter option will later be used to generate the side-by-side render. No matter which option is chosen, an important aspect in this naive implementation is that the camera-to-world matrices in the camera path are in the same scale and coordinate system as the Block-NeRF's `transforms.json`-files.
- **Create a lookup table for which Block to render which camera pose:**
  In order to know which NeRF to render given a specific 3D point and direction, expressed by the $4x4$ camera-to-world matrix in the camera path, we create a naive lookup table. The lookup table is indexed on the camera path's location index and returns the Block-NeRF with the minimum Euclidean distance.
- **Modify the camera path to account for the offset, transformations, and scales of each NeRF:**
  Before a NeRF is trained, the input camera poses are scaled to fit a $[-1, 1]$ bounding box, and transformed so that the average up vector is aligned

with the Z-axis. The respective transformation which is carried out to make the training easier is stored in a `dataparser_transforms.json`-file. The respective file contains the applied transform matrix and scale. Each of the Block-NeRF's `dataparser_transforms.json`-file is different, and in order to render the desired camera path seamlessly across the different Block-NeRFs according to the lookup table, we have to augment it to account for the transformations. We achieve this by creating a new, transformed camera path where the segments which are to be rendered by Block-NeRF$_i$ have their camera-to-world transformed by applying Block-NeRF$_i$'s scale s and transformation matrix t.

- **Render the created camera path:**
  Building upon Nerfstudio's render script, the lookup table is passed as a parameter and used to conditionally select which model to render. As all the changes to the camera path have been done a priori, the resulting render is a seamless video through the scene leveraging multiple Block-NeRFs.

```
block_nerf
├── block_0
│   ├── images/
│   │   ├── 0001.png
│   │   ├── 0002.png
│   │   ├── 0003.png
│   │   └── ...
│   └── transforms.json
├── ...
└── block_n
    └── ...
```

Figure 3.7: Block-NeRF file structure after having split a single dataset into multiple smaller datasets.

3.7 Extending the Pipeline to Support Real Data

The pipeline from data capture in CARLA to image synthesis with a trained NeRF demonstrates efficacy. This final section of the method chapter discusses how the pipeline is extended to enable the input of real data, not captured in a virtual environment like the CARLA simulator.

The traditional approach for generating NeRFs involves the use of real data. The real data is usually captured from handheld cameras. With handheld cameras it is uncomplicated to capture well-lit, non-blurry, object-centric images that does not contain transient objects. Until now we've used a virtual environment to capture data as it has provided a fully controllable environment in which we could test different setups and ensure optimal conditions. The following challenge in this project is capturing high-quality data from sensors, mounted to a moving vehicle, that is suitable for training NeRF models.

To capture data, we utilized the NAPLab car [8]. The NAPLab car is fitted with multiple sensors, including cameras, GPS/GNSS, and LiDAR. The cameras capture data with a resolution of 1920×1080. The GPS/GNSS-system consists of two Swift Navigation Duro Ruggedized Receivers [37], which offer superior positioning accuracy compared to regular GPS/GNSS systems. According to the manufacturer's documentation, each Duro module has a horizontal position accuracy of 0.75 meters (Circular Error Probability [CEP] of 50 in Satellite-Based Augmentation System [SBAS] mode) without Real-Time Kinematic Positioning (RTK), and can achieve centimeter-level accuracy with RTK enabled. This high level of accuracy enables the capture of rough camera poses alongside the images.

After having captured data with the NAPLab car, the data has to be parsed and processed before it can be transformed into the expected Nerfstudio-format. A repository[5] that streamlines the reading and parsing of NAPLab's data into a manageable format was leveraged. Firstly, in order to read the video data, FFMPEG is utilized to transform .h264-files into sequences of `np.ndarray`s that are subsequently served by a generator function. Secondly, the GPS data is read and parsed using regex and subsequently served by a generator function. The GPS data is formatted as National Marine Electronics Association (NMEA) sentences, a message standard used by GPS receivers to communicate with other devices.

A key aspect of the data parsing is the synchronization of the camera with the GPS timestamp. This is achieved by leveraging functionality from the same repository[5] that aligns the camera with the frame closest in time to the GPS timestamp. Following synchronization, the sensor data can be accessed and further processed using a regular loop.

The synchronized data need to be converted into Nerfstudio's data format in order to be used in the predefined pipeline. In order to achieve this, a `NAPLabDataParser` that implements the same methods as the `CarlaDataParser`, is created. The main difference is the implementation of the transformation function that converts the translation and rotation data from one coordinate convention to another. The transformation matrix construction process involves several steps:

[5]`https://github.com/olestole/naplab-data-capture`

1. The initial GPS latitude, longitude, and altitude readings are stored as the reference point.
2. The `geodetic2ned` function from the `pymap3d` library [38] is used to transform GPS data into North, East, Down (NED) coordinates.
3. The NED coordinates are then converted to East, North, Up (ENU) coordinates and then to Blender coordinate conventions.
4. The cameras' rotation is estimated by comparing adjacent GPS data points using trigonometry.
5. The resulting translation and rotation data are combined into a single transformation matrix.

The computation of intrinsic values follows the same methodology as discussed in section 3.2. These intrinsics, in combination with the transformation matrix, are assembled into a `transforms.json` file. The output from the `NAPLabDataParser`, that is the `transforms.json` file and images, is in the same format as that from the `CarlaDataParser`. Therefore, the subsequent stages of the pipeline remain unaffected, ensuring the smooth extension from synthetic to real data inputs.

The dataset captured from the NAPLab car is presented in Figure 3.8.

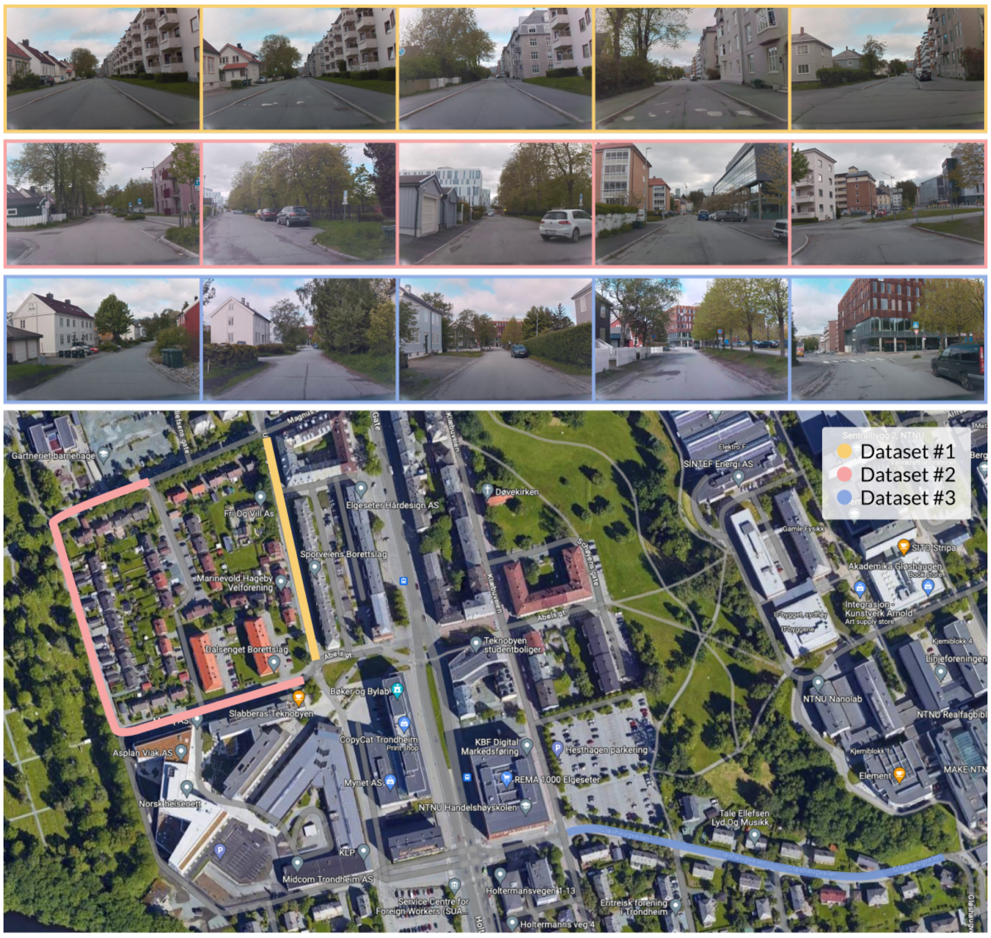

Figure 3.8: Overview of the datasets of real data captured from the NAPLab car. 1) contains 235 images over a straight road segment of approximately 170m, 2) contains 199 images over a segment of approximately 380m, 3) contains 160 images over a segment of approximately 280m.

Chapter 4

Experiments and Results

This chapter provides a detailed account of the experiments conducted and results obtained, in order to establish a baseline for NeRFs trained on synthetic data captured from the CARLA simulator. The defined baseline is then utilized to conduct further experiments investigating the impact of various settings and methods for training NeRFs, including pre-processing, different NeRF-models, and camera optimization. Furthermore, the experiments' insights are applied to evaluate the proposed pipeline's transferability from synthetic to real data.

The results obtained from the experiments are evaluated both quantitatively, using the metrics discussed in section 2.7, and qualitatively. The included qualitative results are frames extracted from video renders of the respective NeRFs. Although the frames convey important results, the video renders provide a much clearer understanding of correspondences between frames, potential artifacts, learned geometry, and other relevant information. To showcase these results, a companion page is available where different renders from this chapter's experiments can be browsed and compared.

Quantitative results are highlighted in the tables, with green signifying the best results, and red indicating the worst. Blue is employed to denote a configuration selected for subsequent experiments.

Link to companion page with video renders[1]

[1]https://nerf.olestole.com/

4.1 Experiment 1: Defining a Baseline

To facilitate comparison between experiments, it is crucial to be in possession of a baseline. However, no existing baselines for NeRF models trained on synthetic data captured in CARLA currently exist. Therefore, this study established a baseline by optimizing settings across various categories in CARLA, including camera setup, capacity, camera settings, vehicle speed, and number of frames. Both a quantitative and qualitative assessment was conducted for the results of each experiment. Based on the assessment, one of the configurations was selected and kept for further experiments.

By establishing this baseline, it can be used as a reference point for improving both the data capture and NeRF models on synthetic data. The metrics used to evaluate the baseline (PSNR, SSIM, and LPIPS) are widely used in NeRF research, ensuring the comparability of results with future experiments and research.

The CARLA configurations held constant for the respective experiments are depicted in Table 4.1.

Table 4.1: Overview of the configurations that were kept constant for each experiment conducted to define the baseline. The configurations that were selected to be part of the baseline data capture configuration, based on the results from the experiments, are highlighted in blue

Experiment	Camera setup	Distance	Image freq.	Image res.	Speed
1.1	–	125m	3.3 FPS	600 × 450	100%
1.2	$[-10°, 10°]$ yaw	–	3.3 FPS	600 × 450	100%
1.3	$[-10°, 10°]$ yaw	4 turns	–	600 × 450	100%
1.4	$[-10°, 10°]$ yaw	4 turns	5 FPS	–	100%
1.5	$[-10°, 10°]$ yaw	4 turns	5 FPS	400 × 300	–
1.6	$[-10°, 10°]$ yaw	4 turns	5 FPS	400 × 300	50%

4.1.1 Experiment 1.1: Camera Setup

The CARLA simulator provides a convenient way to attach multiple cameras with varying settings to a vehicle. To optimize the performance of NeRFs on RGB images, a series of experiments were conducted to determine the optimal camera setup. All cameras were positioned at the same base translation, at the roof of the ego vehicle approximately 3 meters above ground level. While this camera placement may be considered unrealistically elevated, it facilitates an unobstructed capture of the scene without interference from the ego vehicle. The camera setups tested in this study were:

- Single forward-facing camera.
- Two forward-facing cameras, with a counterrotated yaw.
- Three cameras, one with zero yaw and two with counterrotated yaw.

Table 4.2: Comparison of different camera setups' impact on the NeRF's performance.

Description	**PSNR ↑**	**SSIM ↑**	**LPIPS ↓**
Single camera, $[0°]$ yaw	22.76	0.742	0.150
Two cameras, $[-10°, 10°]$ yaw	23.90	0.782	0.136
Two cameras, $[-30°, 30°]$ yaw	23.24	0.756	0.155
Two cameras, $[-50°, 50°]$ yaw	23.73	0.739	0.174
Two cameras, $[-70°, 70°]$ yaw	24.32	0.740	0.173
Three cameras, $[-50°, 0°, 50°]$ yaw	23.79	0.764	0.165
Three cameras, $[-70°, 0°, 70°]$ yaw	23.68	0.755	0.177

From Table 4.2, we can see that the camera setups produce relatively similar results across the three metrics, with only small differences between them. The camera setup with two cameras at -70° and 70° yaw achieves the highest PSNR score, indicating that it produces the most accurate images. On the other hand, the configuration with two cameras at $-10°$ and $10°$ yaw achieves the highest SSIM and lowest LPIPS scores, indicating that it produces the most visually similar and perceptually pleasing images. Due to the configuration's high SSIM and low LPIPS, it was chosen as the camera setup baseline for further experiments.

4.1.2 Experiment 1.2: Capacity

As discussed in subsection 2.8.3 and section 3.6, the capacity of a NeRF is limited. In order to quantitatively assess this capacity, an experiment was designed involving five increasingly longer routes for a CARLA vehicle to capture data. The routes' length ranged from 50m to approximately 450m, as depicted in Figure 4.1. The longest route corresponds to a full lap around the block.

The results, presented in Table 4.3, show that the quality of the rendered images degrades as the segments' length increases. The longest segment, the full lap around the block which includes four turns, achieves the worst results across all three metrics. Despite achieving the worst results, we selected the longest run as the baseline for further experiments. This is because conducting experiments on a full lap around the block provides a more realistic scenario for evaluating the performance of the NeRF on data captured by a vehicle. The full lap encompasses a variety of different scenes, including straight roads, curves, intersections, and

Table 4.3: Comparison of different segment lengths' impact on the NeRF's performance.

Description	PSNR ↑	SSIM ↑	LPIPS ↓
50 meters	23.54	0.773	0.109
100 meters	23.59	0.763	0.141
2 turns	23.60	0.757	0.182
3 turns	22.63	0.720	0.211
4 turns	22.50	0.696	0.241

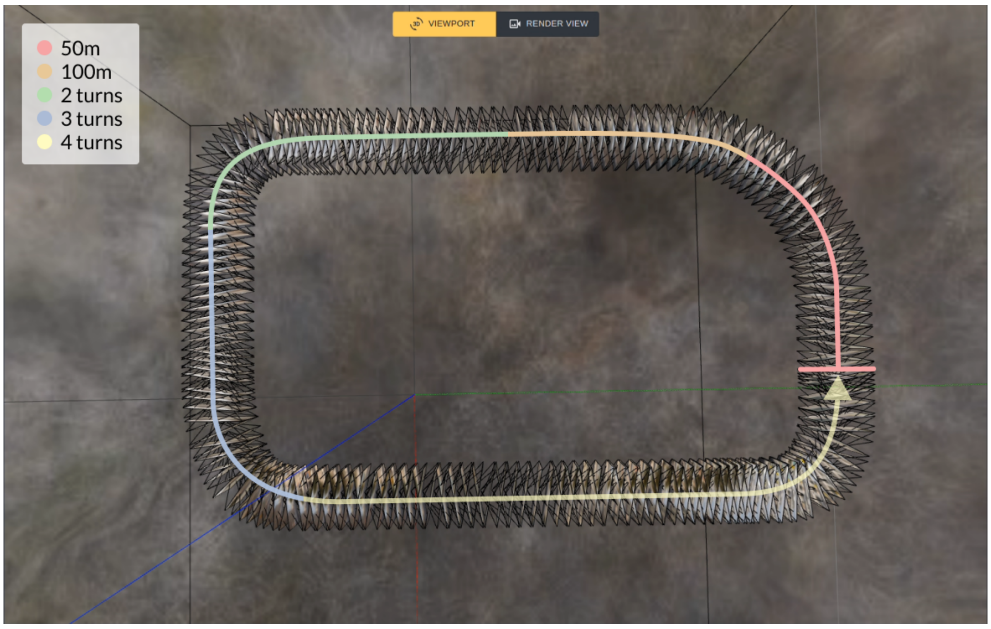

Figure 4.1: Overview of the increasingly longer routes used to capture data in CARLA, during Experiment 1.2.

varying lighting conditions. This diverse range of environments can help to test the NeRF's ability to learn a varying scene and provide a more comprehensive evaluation of its performance.

4.1.3 Experiment 1.3: Number of Frames

A NeRF's dataset is comprised of images and corresponding camera poses. The size of this dataset determines the amount of information available for the NeRF to learn the underlying 3D scene's structure and appearance. A larger dataset can provide more diverse and detailed information about the scene, which can help the NeRF to capture fine-grained details and generalize better to novel views.

Given that the NeRF is set to sample batches across all input images, a larger set of input images should result in higher-quality image synthesis. To test this hypothesis, we conducted an experiment in which we varied the number of frames captured from a CARLA run, resulting in different-sized datasets. The results of this experiment are shown in Table 4.4.

Table 4.4: Comparison of different data capture frequencies' impact on the NeRF's performance. The simulator operates at 10 FPS.

Description	**PSNR ↑**	**SSIM ↑**	**LPIPS ↓**
Capture data every frame (10 FPS)	23.26	0.724	0.228
Capture data every 2nd frame (5 FPS)	23.25	0.727	0.221
Capture data every 3rd frame (3.3 FPS)	22.56	0.697	0.240
Capture data every 4th frame (2.5 FPS)	22.22	0.685	0.250
Capture data every 5th frame (2 FPS)	21.92	0.678	0.259

The results indicate that the trained NeRF performs better when trained on a larger dataset. Furthermore, the difference in performance between capturing data every frame or every second frame is negligible. As a result, we select the latter option for the baseline, as the decreased capture frequency enhances the performance of the CARLA pipeline.

4.1.4 Experiment 1.4: Image Resolution

CARLA allows the configurations of image resolution outputted by mounted cameras. To evaluate the impact of input image resolution on the output image synthesis, data was captured from CARLA at five increasingly higher image resolutions. The obtained results are presented in Table 4.5.

Table 4.5: Comparison of different image resolutions' impact on the NeRF's performance.

Description	**PSNR ↑**	**SSIM ↑**	**LPIPS ↓**
Image resolution of 200×150	23.35	0.749	0.082
Image resolution of 400×300	23.61	0.776	0.104
Image resolution of 800×600	23.24	0.763	0.169
Image resolution of 1200×900	23.07	0.732	0.233
Image resolution of 1600×1200	22.82	0.727	0.267

Figure 4.2 depicts the resulting image synthesis of the same frame across the mod-

els trained on low-, medium-, and high-resolution data. Although the quantitative results indicate that the 200 × 150-dataset performs best, the qualitative results demonstrate that NeRF trained on higher-resolution data is able to represent fine-grained details and produce more visually pleasing images. Based on both the qualitative and quantitative assessments, the configuration with an image resolution of 400 × 300 was chosen as the baseline for further experiments.

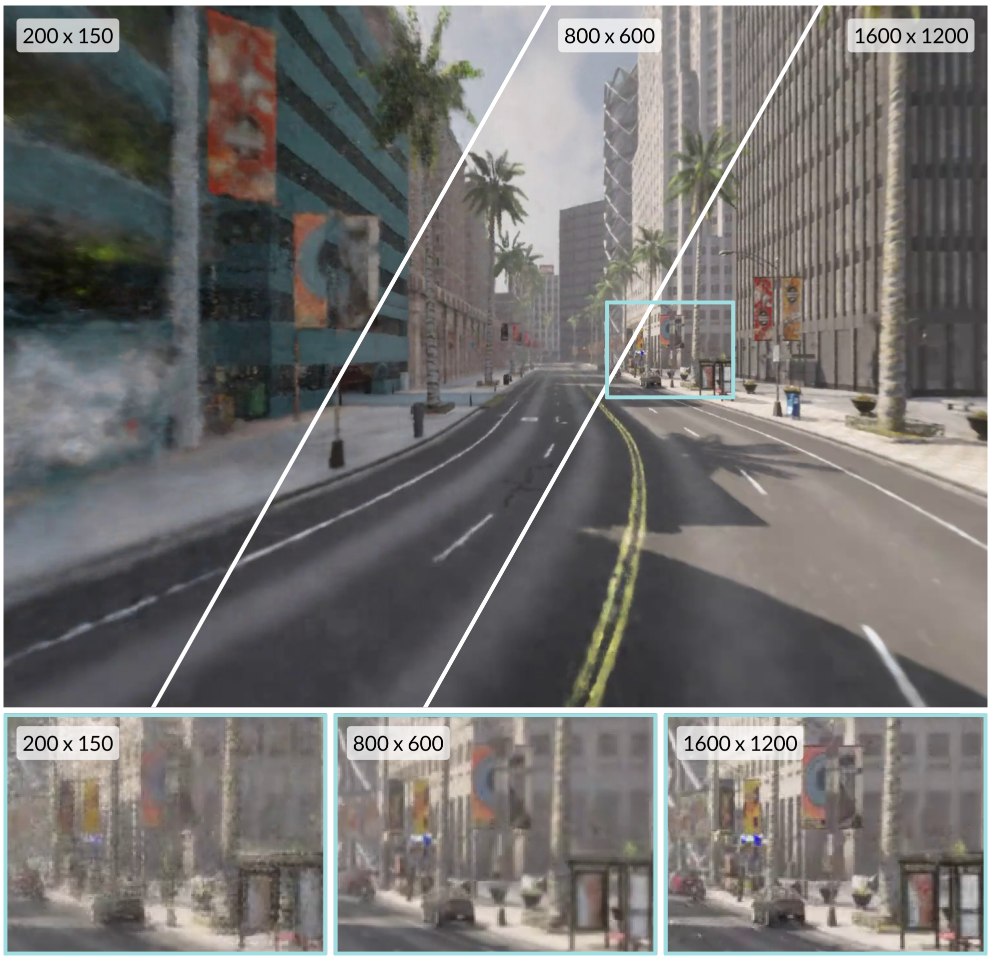

Figure 4.2: Comparison of frames rendered from models trained with training-data of different image resolution during Experiment 1.4. The bottom row provides a detailed view of the same cropped section, across the three model renders.

4.1.5 Experiment 1.5: Vehicle Speed

Higher vehicle speeds result in the capture of larger portions of the scene in a shorter amount of time. However, the motion of the vehicle during capture can lead to image blurring and distortion, potentially decreasing the performance of the resulting NeRF. To investigate the impact of vehicle speed on the quality of the captured data and the resulting NeRF, we conducted a series of experiments where data was captured at varying speeds.

Table 4.6: Comparison of different vehicle speeds' impact on the NeRF's performance. CARLA convention conveys speed as a percentage of the default speed of 30 kmh.

Description	**PSNR ↑**	**SSIM ↑**	**LPIPS ↓**
50% speed	24.06	0.775	0.181
100% speed	23.50	0.755	0.184
150% speed	23.41	0.742	0.190
200% speed	22.72	0.714	0.200

The results of the experiments, presented in Table 4.6, indicate that the speed of the vehicle significantly impacts the quality of the NeRF trained on the captured data. Specifically, we observe that the NeRF trained on the data captured at 50% speed achieves the highest scores on all three metrics (PSNR, SSIM, and LPIPS), while the NeRF trained on the data captured at 200% speed achieves the lowest scores on all three metrics. These findings suggest that slower vehicle speeds can lead to higher-quality data capture, while faster vehicle speeds can lead to lower-quality data capture. As a result, the configuration of 50% reduced vehicle speed was selected for further experiments.

4.1.6 Experiment 1.6: Assessing the Combined Baseline

The configurations selected for the baseline used in further experiments are a combination of the configurations that produced the best results across the previous experiments. The baseline's distance is the only setting that deviates from the configurations that produced the best results, and it will remain constant with a full lap around the block. As a result, the baseline's configurations, depicted in Table 4.8, consist of two forward-facing RGB-cameras, counterrotated with $-10°$- and $10°$ yaw, capturing data every second frame, along a city-block that spans ~ 450 meters in distance, with an image size of 400×300, and a vehicle speed that is 50% slower than the default of 30 km/h. With this configuration, the baseline achieves a PSNR, SSIM, and LPIPS of 24.20, 0.767, and 0.167 respectively, as presented in Table 4.7.

The baseline's high scores across all three metrics demonstrate that this configuration yields high-quality data capture. As a result, the combined baseline can serve as a useful starting point for future research and applications.

Table 4.7: The baseline metrics for a Nerfacto model trained on synthetic data captured from CARLA, with the configurations depicted in Table 4.8.

Description	**PSNR ↑**	**SSIM ↑**	**LPIPS ↓**
Combined baseline	24.20	0.767	0.169

Table 4.8: Overview of the configurations that were selected to be part of the baseline data capture configuration, based on the results from the previous experiments.

Parameter	**Value**
Camera setup	Two cameras, $[-10°, 10°]$ yaw
Distance	4 turns
Image capture freq.	Capture data every 2$^{\text{nd}}$ frame
Image resolution	Image resolution of 400×300
Speed	50% speed

4.2 Experiment 2: Simulated Noise Conditions

During the capture of images and corresponding camera poses from a vehicle in a real-world scenario, the accuracy of camera poses is often impaired, primarily due to imperfections in GPS/GNSS-readings. To investigate the impact of noisy camera poses and the effectiveness of camera pose optimization, an experiment was conducted in which Gaussian noise was introduced to the translational component of the camera poses obtained from the CARLA pipeline. The addition of Gaussian noise to the camera poses simulates the effects of imperfect camera poses in real-world data capture. The magnitude of the added noise is determined by the standard deviation of the Gaussian distribution, which increases progressively throughout the experiments. The results for the runs with and without camera optimization, either treating the camera parameters as joint learnable parameters or not, are presented in Table 4.9 and Figure 4.3, with a full table of additional increments in subsection A.2.1.

As seen from the results in Table 4.9, the NeRF's quality degrade across all metrics as the noise increases. While the difference in quantitative results between the runs with and without camera pose optimization may appear small, the qualitative comparison depicted in Figure 4.3 provides clear evidence of the optimization's efficacy. In addition, it seems to have a more substantial impact on shorter seg-

Table 4.9: Results for Gaussian Noise experiment on both the baseline and shorter segments. The shorter segment is 10% the size of the baseline segment, approximately 50m in length.

Description	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
	Baseline segment					
	With camera optimizer			**Without camera optimizer**		
$\mathcal{N}(0, 0.0)$	23.41	0.714	0.321	24.70	0.793	0.179
$\mathcal{N}(0, 0.1^2)$	22.51	0.674	0.321	22.46	0.677	0.281
$\mathcal{N}(0, 0.5^2)$	19.08	0.501	0.372	19.37	0.499	0.474
$\mathcal{N}(0, 1.0^2)$	17.67	0.434	0.433	18.21	0.444	0.560
$\mathcal{N}(0, 3.0^2)$	16.66	0.408	0.637	16.32	0.386	0.648
	Shorter segment					
	With camera optimizer			**Without camera optimizer**		
$\mathcal{N}(0, 0.0)$	24.83	0.825	0.102	25.68	0.862	0.077
$\mathcal{N}(0, 0.1^2)$	23.03	0.753	0.115	23.32	0.755	0.147
$\mathcal{N}(0, 0.5^2)$	19.06	0.483	0.197	20.07	0.513	0.299
$\mathcal{N}(0, 1.0^2)$	18.09	0.407	0.338	18.29	0.418	0.405
$\mathcal{N}(0, 3.0^2)$	15.71	0.345	0.613	15.08	0.363	0.699

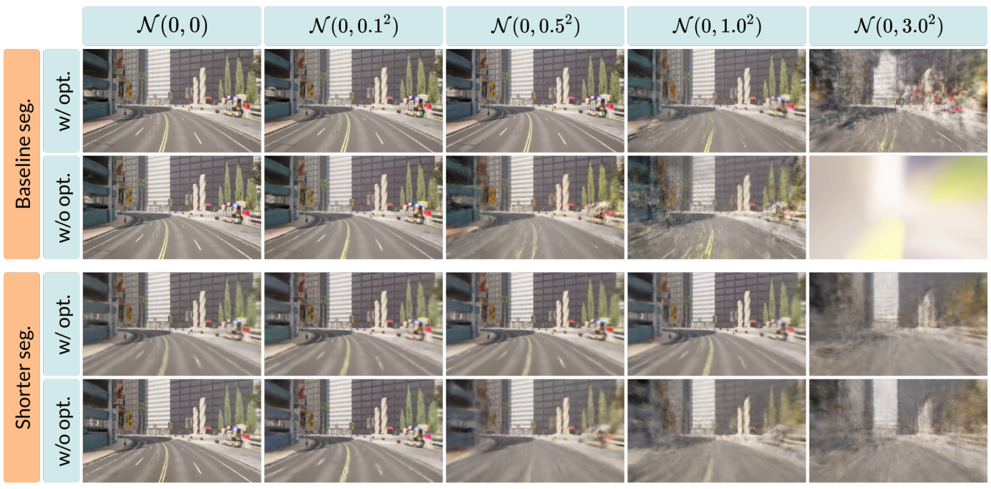

Figure 4.3: Comparison of the effect of simulated noise conditions for the trained NeRF.

ments, than larger ones.

4.2.1 Experiment 2.1: COLMAP versus Camera Pose Optimization

The camera optimization in the previous experiment produced stable, qualitative results. In order to further investigate the effect of direct camera optimization on vehicle-captured data, a comparison to COLMAP's result on the same dataset is conducted. Table 4.10 presents the results from the NeRF trained on the same images as in Experiment 2, but with camera poses approximated with the SfM tool, COLMAP.

Table 4.10: Results for approximating camera poses with COLMAP for the baseline and short segment.

Description	PSNR ↑	SSIM ↑	LPIPS ↓	Time processing
COLMAP - Baseline segment	24.19	0.759	0.160	01:12:15
COLMAP - Shorter segment	25.23	0.827	0.093	00:02:00

First, looking back at the results in Table 4.9, we observe that small amounts of noise severely degrade the quality of the NeRF even when the camera poses are optimized throughout the NeRF's training. Comparing those results with COLMAP's results on the same datasets as depicted in Table 4.10, it becomes apparent that although direct pose optimization offers the benefit of avoiding pre-processing, the NeRF trained with camera poses approximated by COLMAP consistently delivers superior performance across all metrics when the camera poses are noisy, regardless of segment size. The results thus suggest that employing COLMAP for initial pose approximation could significantly improve model performance when the initial camera poses are inaccurate, despite the longer processing time required.

4.3 Experiment 3: Comparing NeRF-models

Nerfstudio has implemented multiple well-known NeRF-models including Instant-ngp [3] and Mip-NeRF [4]. In this experiment, we investigate the model's impact on the output. The comparison of the different models is presented in Table 4.11 and Figure 4.4.

Table 4.11: The result of training different models implemented in the Nerfstudio framework on the combined baseline dataset.

Description	PSNR ↑	SSIM ↑	LPIPS ↓	Iterations
Nerfacto	24.20	0.767	0.169	15'000
Instant-ngp	24.25	0.756	0.232	15'000
Nerfacto-big	23.56	0.741	0.283	100'000
Mip-NeRF	9.49	0.165	0.775	300'000

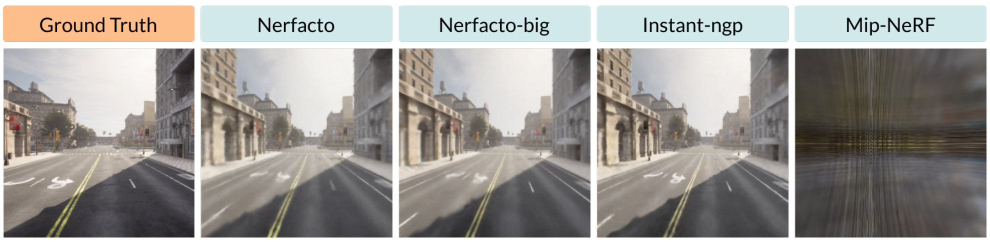

Figure 4.4: Qualitative comparison of different NeRF-models trained on the same dataset.

The quantitative results indicate that there are no significant differences between the fast methods, Nerfacto, Nerfacto-big, or Instant-ngp. However, both the quantitative and qualitative results reveal that the Mip-NeRF model is unable to learn the unbounded scene, resulting in the worst scores across all metrics.

4.4 Experiment 4: Large-Scale NeRF

The naive Block-NeRF implementation allows for the captured scene to be split into an arbitrary number of segments. In this experiment, we would like to explore the impact of splitting the scene on the overall quality of the generated results. In particular, we compare the performance of Block-NeRF to that of a single NeRF trained on the entire scene. Table 4.12 shows the result of splitting the baseline-scene into 4 segments, while Table 4.13 shows the result of splitting a larger scene, a trajectory of approximately 1200 meters, into 12 segments. The qualitative results of the 12-segment run can be seen in Figure 4.5.

The experimental findings indicate that the naive Block-NeRF implementation

Table 4.12: The average metrics across the four segments compared to the metrics for a single NeRF trained on the same scene. The full overview of each segment's metrics can be viewed in the appendix in Table A.6.

Description	PSNR ↑	SSIM ↑	LPIPS ↓
Average across 4 Block-NeRFs	24.47	0.790	0.184
Single NeRF	24.20	0.767	0.169

Table 4.13: The average metrics across the twelve segments compared to the metrics for a single NeRF trained on the same scene. The full overview of each segment's metrics can be viewed in the appendix in Table A.7.

Description	PSNR ↑	SSIM ↑	LPIPS ↓
Average across 12 Block-NeRFs	24.77	0.801	0.159
Single NeRF	22.52	0.654	0.424

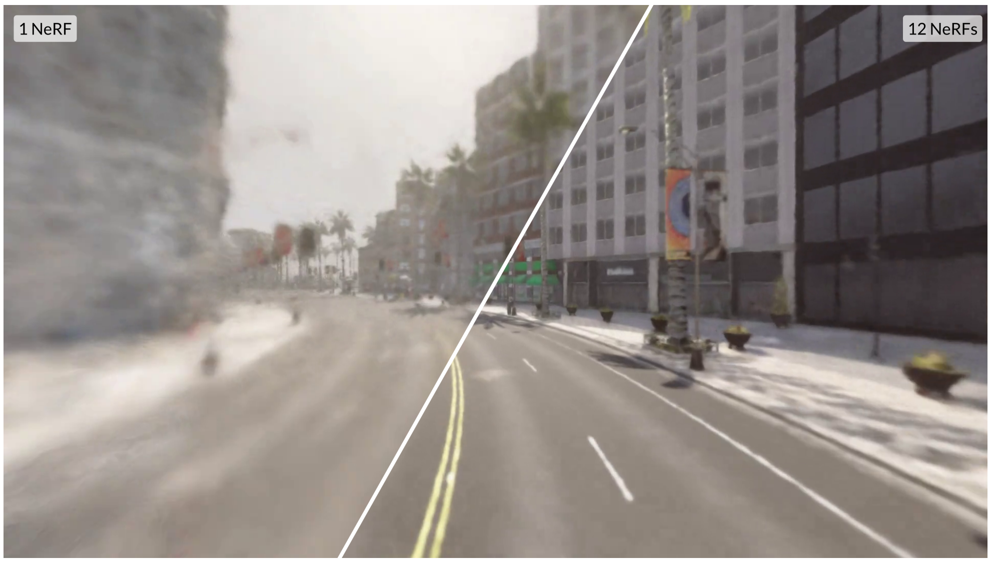

Figure 4.5: Comparison of two renders from models that have been trained on the same dataset. Left) A single NeRF trained on the full dataset, Right) 12 Block-NeRFs trained on equally spaced segments of the dataset.

enables high-quality image synthesis across large scenes, while a single NeRF trained on the same large scene experiences a decline in performance, resulting in poor image synthesis quality. These results suggest that Block-NeRF represents a promising approach for generating high-quality results for large-scale scenes.

4.4.1 Experiment 4.1: Removing Artifacts

Despite the significantly lower quality of the single NeRF implementation compared to the Block-NeRF implementation, the resulting render exhibits clear coherence without sudden changes in detail or the abrupt appearance of artifacts. In contrast, the naive Block-NeRF implementation suffers from this issue, as illustrated in Figure 4.6.

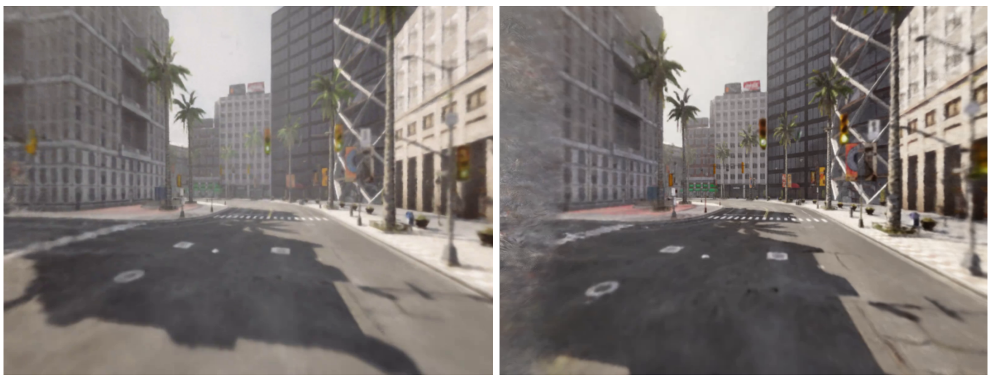

Figure 4.6: Comparison of two consecutive frames from a 12 Block-NeRF. The first frame (left) is the last frame rendered by Block-NeRF number 1. The second frame (right) is the first frame rendered by Block-NeRF number 2 and contains artifacts in the outer edges.

This issue could possibly be attributed to the hard cutoff between each Block-NeRF, in which Block-NeRF$_i$ only trains on a single data-bin data$_i$ where each data-bin is a disjunct collection of images and corresponding camera poses. A possible solution to the issue is to include an overlap of data from the previous and successive data-bin. Let data$_i$ be the i-th data-bin with the corresponding interval $[i, i+1]$. Then, the new interval for Block-NeRF$_i$ with overlap can be defined as $[i-\delta, i+1+\delta]$ where δ is the size of the overlap. In other words, the new interval includes not only the data-bin data$_i$, but also a portion of the previous data-bin data$_{i-1}$ and a portion of the successive data-bin data$_{i+2}$. A quantitative and qualitative comparison of the effects from this can be seen in Table 4.14 and Figure 4.7 respectively.

Table 4.14: Average across different Block-NeRF overlap configurations. The overlap becomes less visible with higher overlap values, but it comes at the cost of the previously explored capacity issue.

Description	**PSNR ↑**	**SSIM ↑**	**LPIPS ↓**
Average metrics with $\delta = 100$	23.80	0.746	0.248
Average metrics with $\delta = 50$	24.31	0.776	0.206
Average metrics with $\delta = 0$	24.80	0.802	0.158

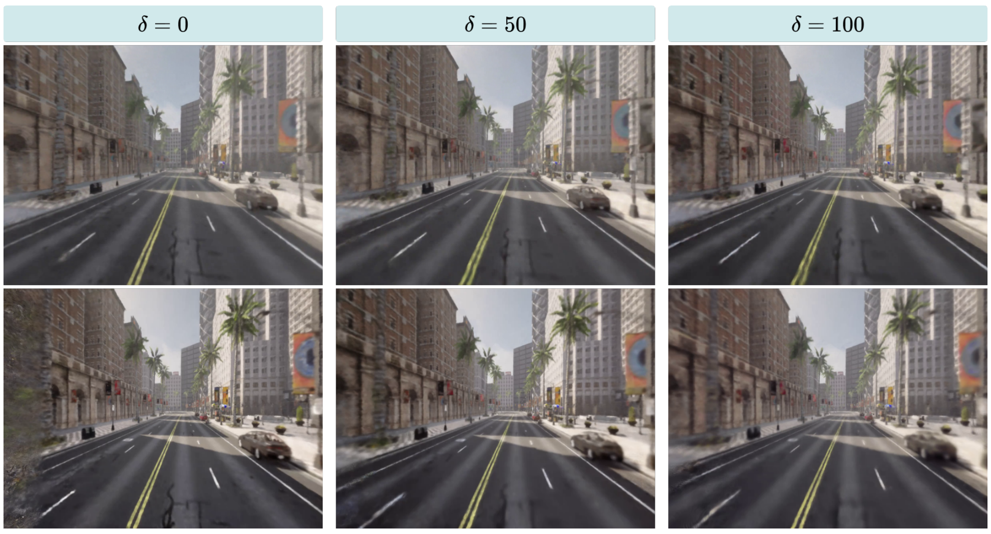

Figure 4.7: Comparison of Block-NeRF trained with 0, 50 and 100 images overlap, respectively. The top row corresponds to the last frame rendered by Block-NeRF number 0. The bottom row is the first frame rendered by Block-NeRF number 1.

As shown in Figure 4.7, an increase in δ results in a significant reduction in the visible overlap between the blocks. Correspondingly, as indicated in Table 4.14, an increase in δ leads to a decline in NeRF quality across all three metrics. These findings suggest that it is necessary to identify a suitable value of δ that minimizes the visible overlap between blocks, while simultaneously maintaining high NeRF quality.

4.5 Experiment 5: Real Data

The implementation of the real data-capture pipeline and the subsequent `NAPLab-DataParser` allows the end-to-end pipeline to be run with data captured from the NAPLab car. The real data captured was split into three datasets and is presented in Figure 3.8. With the ability to conduct experiments on real data, we want to investigate the quality of the capture, if the captured data is suitable for training NeRFs, how the camera poses estimated from the GPS compare to the camera poses approximated with COLMAP, how splitting up the data and leveraging a Block-NeRF approach affects performance, and how camera optimization affects the results.

In order to test the aforementioned aspects, eight different pipeline runs are conducted on the three datasets previously presented in Figure 3.8. The experimental results for dataset 1 are presented in Table 4.15. As the results for datasets 2 and 3 show similar results, they have been moved to subsection A.2.3.

Table 4.15: Results from training NeRF on dataset 1. The transformation matrices are approximated with COLMAP or estimated from GPS-readings. "BN" is an abbreviation of Block-NeRF and the resulting metric score is averaged across the 4 NeRFs evaluations.

Description	PSNR ↑	SSIM ↑	LPIPS ↓	Time processing
COLMAP w/ optimizer	21.99	0.792	0.404	00:26:45
COLMAP w/o optimizer	24.59	0.862	0.272	00:26:45
COLMAP w/ optimizer, 4 BNs	22.58	0.794	0.252	00:26:45
COLMAP w/o optimizer, 4 BNs	27.26	0.899	0.198	00:26:45
GPS w/ optimizer	20.00	0.753	0.491	00:00:00
GPS w/o optimizer	17.04	0.738	0.550	00:00:00
GPS w/ optimizer, 4 BNs	20.12	0.741	0.397	00:00:00
GPS w/o optimizer, 4 BNs	19.37	0.737	0.511	00:00:00

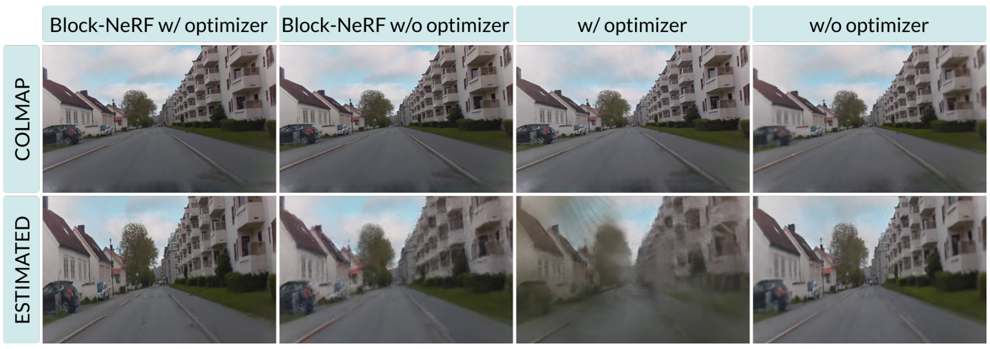

Figure 4.8: Comparison of the different approaches used to train the NeRF on real data. The data is from dataset 1, presented in Figure 3.8.

The results highlight interesting trade-offs between the applied approaches. Notably, the Block-NeRF runs with camera poses approximated by COLMAP without further optimization, yielded the best metrics across all the experiments. Disregarding both Block-NeRF runs, the NeRF trained on images with camera poses approximated by COLMAP without further optimization yielded the best quantitative results. In contrast, the experiments leveraging GPS readings for camera pose approximations with or without optimization were less successful across all metrics, with the non-optimized GPS approach scoring the lowest across all three metrics.

4.6 Experiment 6: Novel Views Along Altered Trajectory

An important application of NeRFs is their ability to synthesize high-quality images from novel viewpoints. Although there are many applications for this, one of those is training and evaluating AD systems. In this section, we will investigate how a NeRF trained on self-captured synthetic and real data can be used to create camera paths and render novel views not previously observed in the dataset. Figure 4.9 shows four different camera paths created in Nerfstudio for trained NeRFs.

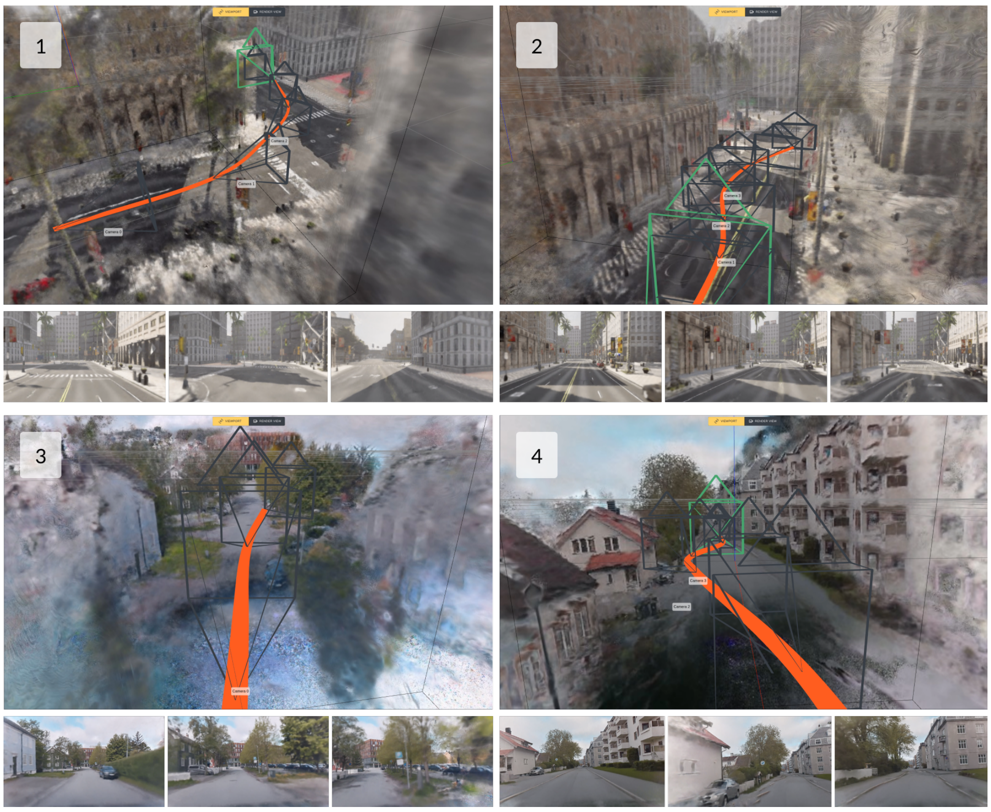

Figure 4.9: Previously unseen trajectories being rendered by defining a new camera path for the trained NeRF. Image 1 illustrates a trajectory deviating across a traffic light zone, while image 2 shows a trajectory veering into the opposing lane. Image 3, derived from dataset 3, portrays a trajectory directed towards a road sign, and Image 4, derived from dataset 1, presents a trajectory swerving over the curb. The NAPLab datasets are presented in Figure 3.8.

Chapter 5

Discussion

This chapter discusses and analyzes the results of the experiments, highlighting their significance in the context of the research questions. Additionally, the limitations of the thesis are discussed.

5.1 Experiment 1: Defining a Baseline

In order to define a baseline for capturing data for training NeRFs, five experiments were selected and conducted: camera setup, capacity, number of frames, image resolution, and vehicle speed. These experiments were chosen based on heuristics and knowledge of what contributes to good NeRF results, such as well-lit scenes and non-blurry images. While the chosen experiments clearly consider important factors for capturing data for NeRFs, there may have been other experiments that could have been included in defining the baseline. It is also important to weigh the fact that the best-performing settings in one experiment may not generalize to other setups. Overall, the process of defining a baseline is an iterative one that requires careful consideration of various factors and a willingness to continuously improve and refine the baseline.

5.1.1 Experiment 1.1: Camera Setup

The results of the experiment showed relatively little difference in the quantitative metrics across the camera setups tested. However, the camera setup with two cameras at -10° and 10° yaw produced the highest SSIM and lowest LPIPS scores, indicating that it produced the most visually similar and perceptually pleasing images. A possible explanation for why this specific camera setup produces the best results is the level of overlap between the captured images the respective camera setup provides. Both cameras have a FOV of 90° and are mounted in the same location, resulting in a 70° overlap between the images in the training data. This overlap entails that when the model trains on an image from one of the cameras, it necessarily also trains on approximately three-quarters of the scene captured from the other camera. Because the evaluation set is a subset of the training images, it is fair to assume that the model should score high on the respective metrics.

The camera setups used in the experiment were arbitrary and may not reflect how cameras are typically rigged on cars. Future research could explore more realistic camera setups to improve the generalizability of the results. Additionally, the camera setups used in the study were sparse, with only a few cameras at specific angles. It is possible that other camera setups could yield even better results.

5.1.2 Experiment 1.2: Capacity

As stated in the experiment's section, the longest segment was selected for the baseline despite achieving the lowest scores across the metrics. This choice was made to ensure that the scene encompasses a diverse range of environments, including straight roads, curves, intersections, and varying lighting conditions.

In the qualitative analysis of the capacity results, it is evident that the quality of the renders degrades as the segment's length increase. The most prominent deterioration is the increase in blur; however, the PSNR remains relatively constant across the different experiments. A reason for this could be that PSNR has been shown to poorly capture the effects of blur [39], as exemplified by Figure 5.1. This example demonstrates the importance of evaluating the model across different metrics, as both SSIM and LPIPS are good metrics for capturing blur.

5.1.3 Experiment 1.3: Number of Frames

To comprehend the outcomes of this experiment, it would be beneficial to look into how the number of frames and the image resolution impact the training process. As described in the implementation details in section A.1, the model is trained for 15'000 iterations where each iteration uses 4096 pixels. This results in ~ 61 million pixels being sampled throughout a single training. With 225 training images and an image resolution of 600×450, ~ 61 million pixels would be sampled. That means that by the end of the training, approximately all of the input pixels were trained on. A dataset containing more than 225 training images and corresponding camera poses would leave abundant pixels, and any fewer would lead to pixels being trained on multiple times. This calculation entail that the results should be relatively similar for all the conducted experiments with 1231, 615, 411, 307, and 247 images respectively. But, there is a significant drop in PSNR from experiments 1 to 4. The drop in PSNR indicates that another important factor is the variety in the dataset.

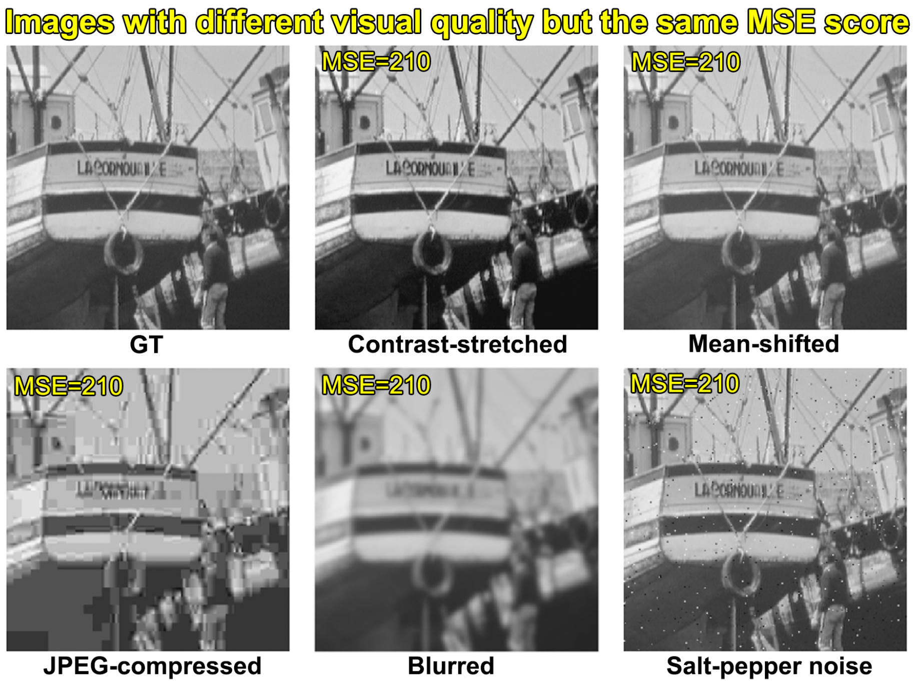

Figure 5.1: Comparison of the MSE score for an image that has been subjected to various types of distortions. The MSE score directly affect the PSNR, as PSNR builds upon MSE as described in subsection 2.7.1. Figure 3 from *Ways of cheating on popular objective metrics: blurring, noise, super-resolution and others* [28].

5.1.4 Experiment 1.4: Image Resolution

Based on the metric scores, it might seem counterintuitive that the lowest resolution of 200 × 150 performed best. However, a deeper look into the qualitative results in Figure 4.2 reveals a different narrative. Despite lower resolutions yielding higher scores on metrics, they seem to fail in capturing fine-grained details, resulting in less visually pleasing images. On the other hand, higher-resolution images allow the NeRF to capture and replicate more intricate details, leading to superior visual outcomes, albeit with lower metric scores. This difference can be attributed to the higher sensitivity of metrics to minor discrepancies and noise in high-resolution images, potentially causing significant metric score reductions despite only minimal perceptual differences.

Given these considerations, the chosen resolution of 400 × 300 for the following experiments is a balanced choice, considering both the quantitative and qualitative results. Furthermore, it aligns well with the chosen number of frames from the Experiment 1.3, as the combined configuration will allow training to sample about 84% of the input pixels.

5.1.5 Experiment 1.5: Vehicle Speed

The reason why the runs with higher vehicle speeds achieve worse results might be attributed to the motion blur and temporal artifacts that can occur when the vehicle is moving fast. In CARLA these effects are added as part of the post-processing of the captured camera image. At higher speeds, the motion of the vehicle can cause blurring and distortion in the captured images, which can reduce the quality of the data and make it more difficult for the NeRF to learn the underlying 3D scene structure and appearance. In contrast, slower vehicle speeds can reduce the amount of motion blur and temporal artifacts, resulting in clearer and more detailed images.

Another side effect of driving slower is that it leads to an increased amount of images captured. At 50% speed, the dataset consists of 1095 images, in contrast to the 429 images captured at 200% speed. An increased dataset size proved to be beneficial in Experiment 1.3, and could positively affect the results in this experiment.

5.1.6 Experiment 1.6: Assessing the Combined Baseline

RQ 1: What are the critical factors that need to be considered when capturing synthetic data for training NeRF models, and how do they impact the performance of the resulting models?

From the experiments discussed above, it is evident that all the configurations contribute to the quality of the data capture, which in turn contributed to the quality of the image synthesis from the resulting NeRF. Nevertheless, it is challenging to quantify the extent to which each of the different configurations affects the quality of the final baseline results.

Looking at the experiments separately, image resolution had the largest span between the quantitatively best and worst metrics. Nevertheless, the qualitative results indicated that the quantitative comparison did not convey a fair comparison of the render-quality. Due to this, the capacity-experiment could be the experiment with the most impact on the final result.

In conclusion, when capturing synthetic data for training NeRF models, it is important to consider the camera setup, segment length or scene size, dataset size, image resolution, and vehicle speed. Each of these parameters presents unique influences on the quality of the data captured, and in turn, the performance of the resulting NeRF models.

5.2 Experiment 2: Simulated Noise Conditions

RQ 2: How does the initial camera pose accuracy and segment length impact the final reconstruction? Can rough initial camera poses be optimized to achieve comparable results to those obtained from tools such as COLMAP?

This research question addresses the effects of imperfect camera poses, frequently encountered in real-world scenarios due to GPS/GNSS inaccuracies, on the performance of camera pose optimization. We examine this by adding Gaussian noise to camera poses obtained from the CARLA pipeline, simulating real-world inaccuracies in these camera poses.

Despite Gaussian noise not perfectly replicating real-world noise distributions, its application allows an indicative examination of camera pose optimization within the Nerfacto-pipeline. It is noteworthy that while the quantitative distinction between the non-optimized and optimized camera pose results might appear marginal, the qualitative contrasts depicted in Figure 4.3 deliver strong evidence of the camera pose optimization's effectiveness.

Particularly, the qualitative comparison highlights that the optimized camera poses generate consistently higher quality renders, even under significant noise levels. This is more noticeable within shorter segments, where the optimization seems more efficient than in the larger baseline scene. This finding may be attributed to the previously discussed limitations in capacity, given that the camera optimization treats camera pose parameters as jointly optimized learnable parameters along with the RGB values.

Interestingly, the only instance where non-optimized poses yield superior results is when no noise is introduced. In such cases, the resulting render from non-optimized camera poses appears considerably sharper than its optimized counterpart, which appears comparatively more blurred. This outcome is likely unique to datasets with near-perfect camera poses, such as the synthetic dataset used in this experiment.

When examining the effectiveness of COLMAP pre-processing against the joint optimization of initial camera poses along with the other learnable parameters, it becomes evident that COLMAP tends to deliver superior performance. This observation holds particularly when the initial rough camera poses are influenced by a rather minor noise adhering to a Gaussian distribution with a standard deviation of 0.1^2. Under these conditions, all three evaluation metrics (PSNR, SSIM, LPIPS) associated with the NeRF lag behind those achieved by COLMAP, as can be seen from Table 4.9 and Table 4.10. Furthermore, when the noise's standard deviation is escalated to 0.5^2, which is fair to assume could occur during real-world data capture, the metrics degrade even more significant compared to what COLMAP can deliver. While the processing time required by COLMAP might seem prohibitively lengthy for large datasets, this concern can be addressed by splitting the data into smaller sections. Thus, despite the initial time investment, COLMAP's superior performance merits consideration, particularly in scenarios with significant noise in initial camera poses.

5.3 Experiment 3: Comparing NeRF-models

RQ 3: How do different NeRF methods (Instant-ngp [3], Mip-NeRF [4], Nerfacto [5]) perform on unbounded scenes in terms of reconstruction quality and computational efficiency?

Upon examination of the different NeRF methods, namely Nerfacto, Instant-ngp, Nerfacto-big, and Mip-NeRF, it is evident that their performance in terms of reconstruction quality and computational efficiency on unbounded scenes is rather comparable, with the exception of Mip-NeRF.

Both Nerfacto and Instant-ngp demonstrate high performance on the evaluated metrics, showcasing their proficient ability to effectively learn and represent the unbounded scene. The largest deviation between the two models is their LPIPS score, where Nerfacto outperforms Instant-ngp. Nerfacto-big is a differently configured Nerfacto-model featuring, among other configurations presented in Table A.2, an expanded hidden layer width which increases the model's capacity. Despite being trained for a significantly larger number of iterations it fails to exceed the performance of its less complex counterparts.

Mip-NeRF, which is designed for bounded scenes, predictably falls short in learning the unbounded scene. It yields the lowest scores across all metrics. The qualitative evaluation further support this, showing no perceivable scene structure in its renderings, despite producing some correct color representations.

Given these findings, Nerfacto emerges as a strong candidate for subsequent experiments. Furthermore, it should serve well as a backbone-model for applications to build upon, thanks to its balance between reconstruction quality and computational efficiency.

5.4 Experiment 4: Large-Scale NeRF

RQ 4: What are the technical challenges and considerations for implementing a functional approach for large-scale NeRF within the Nerfstudio API, and how does it compare to approaches not optimized for large-scale in terms of scalability, efficiency, and rendering quality?

Implementing a functional approach for large-scale NeRF within the Nerfstudio API has been technically challenging, as elaborated upon in section 3.6. However, the initial naive Block-NeRF approach has displayed encouraging performance in both quantitative and qualitative aspects. The experimental results confirm that compared to training single NeRFs on large scenes, the Block-NeRF approach yields superior outcomes. The image synthesis produced by the Block-NeRF approach exhibits sharper details and an overall enhancement in the visual quality.

While the naive implementation of Block-NeRF does not inherently exhibit optimal scalability, it holds significant potential for improvement. In its current state, each Block-NeRF is trained sequentially, creating a bottleneck that limits scalability. However, the architecture provides an opportunity for horizontal scaling, which could be achieved by parallelizing the unique training processes across multiple Graphics Processing Units (GPUs). This adjustment would drastically improve efficiency and reduce training time, thereby enhancing scalability.

Nevertheless, it is important to acknowledge that this process is computationally intensive, and care should be taken to find a balance between the quality of the results and the efficiency of the process. Future work could focus on optimizing this balance to make large-scale NeRF more viable and efficient, while maintaining the high-quality image synthesis that the Block-NeRF approach has demonstrated thus far.

5.4.1 Experiment 4.1: Removing Artifacts

The introduction of an overlap δ between the naive Block-NeRF's data resolves the problem of visible artifacts during the transition between Blocks. However, as δ increases, the dataset size for the respective Block-NeRFs increase, and the collective render quality decreases. This finding is consistent with the results from Experiment 1.2, which indicate that the scene size is inversely correlated with render quality.

While the approach of incorporating overlap between the Block-NeRF's data resolves the problem of visible artifacts during the transition between Blocks, it is evident from Figure 4.6 that the final frame rendered by a Block-NeRF produces renders of lower quality compared to the initial frame rendered by the same Block-NeRF. If the second Block-NeRF's dataset contains images captured closer to the respective motive, indicating more detailed images, this difference is to be expected. However, the difference could potentially be mitigated by experimenting with image-merging techniques, for example, by rendering the view from both blocks and merging them with techniques like *inverse distance weighing*.

5.5 Experiment 5: Real Data

The naive Block-NeRF approach emerges as the clear frontrunner in the experiments on real data, producing both the best quantitative and qualitative results. This finding aligns with expectations considering the scale of the scene being learned, which potentially surpasses the capacity of a single NeRF. Among all conducted experiments, the Block-NeRF run, with camera poses approximated by COLMAP without further optimization, demonstrated a distinct supremacy across all metrics, affirming the advantage of COLMAP-based pose estimation over GPS-derived alternatives.

In the subset of experiments excluding the naive Block-NeRF approach, the approach utilizing COLMAP without subsequent optimization yielded the best results. Among the runs with camera poses estimated from GPS-readings, those involving subsequent camera pose optimization produced superior metrics. These findings further support the observation that camera poses that are near-perfect tend to be disadvantaged by subsequent optimization, while imperfect camera poses benefit from this process. This observation is further reinforced by the qualitative evaluation of the different runs, presented in Figure 4.8, where the output from runs utilizing near-perfect camera poses tended to exhibit blurriness upon subsequent optimization. In contrast, the use of subsequent optimization of imperfect camera poses enabled the NeRF to generate more precise and clear renderings.

5.6 Experiment 6: Novel Views Along Altered Trajectory

One of the benefits of using NeRFs for training AD systems, is the ability to generate data along altered trajectories. While this can be achieved with a simulator, as discussed in subsection 2.8.6, the application of NeRF becomes particularly useful when a high-quality NeRF has been trained on real data. In addition to evaluating the autonomous vehicle in a novel environment, the NeRF can be used to expand the training dataset for the autonomous vehicle by creating an arbitrary number of altered camera paths and synthesizing photo-realistic novel views of altered trajectories. This can help improve the robustness and generalizability of the AD system by exposing it to a wider range of scenarios and viewpoints.

Looking into the experimental results from Experiment 6, it is important to note that the absence of ground truth images from the altered path inherently constrains the ability to conduct quantitative analysis beyond the initial evaluations. The qualitative assessment is thus the primary mode of evaluation. From the renderings depicted in Figure 4.9, we can clearly see the potential of this application. Although the novel-rendered perspectives do not always match the quality and coherence found in the original evaluation images, they are largely successful in producing clear and structurally accurate visualizations. This affirms their potential and the effectiveness of the approach.

5.7 Shortcomings

Several areas of potential improvement can be recognized within this study. Firstly, while the research has illuminated key aspects of data capture from vehicles, the time constraints of the project did not allow us to validate these findings through real-world testing. Synthetic data sets were leveraged extensively and provided evidence that the combined configurations for the baseline were promising. However, the validity of these results has yet to be confirmed in real-world capture.

Secondly, there exist limitations in the capture of real data, specifically in the estimation of transformation matrices. The camera poses' translational components are derived from raw GNSS-readings, and the rotational components are estimated through trigonometric comparisons of adjacent GNSS-readings. Such a method is inherently susceptible to errors and inaccuracies. To improve the accuracy and reliability of data capture, a better approach could be to utilize both of the vehicle's GNSS sensors and integrate the vehicle's accelerometer data to achieve precise roll and pitch estimations.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

The primary research goal of this thesis was to design and develop an end-to-end pipeline for generating NeRFs, leveraging vehicle-captured video sequences and corresponding camera poses with varying degrees of accuracy.

Initially, a data capture pipeline was created for CARLA, which provided synthetic data from a controlled environment. Connecting this with the Nerfstudio pipeline established the end-to-end pipeline and enabled the creation of a performance baseline for further experiments. During the creation of the baseline, multiple configurations and settings were evaluated, underscoring their significance on the resulting image synthesis. Some of the configurations that proved important were camera setup, segment length, dataset size, image resolution, and vehicle speed.

Having obtained a baseline, further experiments were conducted, revealing important findings: NeRFs trained with jointly optimized camera poses consistently achieved higher scores than non-optimized camera poses on the metrics PSNR, SSIM and LPIPS, and proved particularly effective on shorter segments. However, pre-processing an approximation of the camera poses with COLMAP outperformed camera pose optimization, especially in high-noise situations, arguing for its consideration despite an upfront processing-time investment. These two findings support the finding that camera poses that are near-perfect tend to be disadvantaged by subsequent optimization, while imperfect camera poses benefit from this process.

Progressing into the exploration of large-scale NeRF approaches, a naive prototype was implemented. Despite being a prototype with rudimentary features, the high-quality results were consistent across multiple scenes. The approach provided the ability to represent large scenes while still retaining sharp details and high visual quality.

In order to expand the end-to-end pipeline to enable the input of real data, a data capture pipeline with a custom data parser was created for the NAPLab car. The extension to real data resulted in photo-realistic renderings, and many of the findings from the experiments conducted in the controllable, virtual environment held true. Optimizing the rough camera poses captured from the NAPLab vehicle performed better than not optimizing them. However, COLMAP outperformed joint camera optimization overall. Additionally, the naive large-scale NeRF implementation performed better than a single NeRF.

Both the synthetic and real data were used to conduct experiments on how well the vehicle-captured data could provide a NeRF capable of rendering unseen trajectories. Although the novel-rendered perspectives did not match the quality and coherence found in the original evaluation images, they were largely successful in producing clear and structurally accurate renderings. This affirms their potential and the effectiveness of the approach.

6.2 Future Work

This section provides ideas related to this thesis that were not covered, but would be interesting to explore in future work. Given that the large-scale approach provided the best results across all experiments when compared to single-NeRF approaches, it is a promising approach to pursue. Following are some techniques and approaches that would provide interesting tracks for future work.

Wayve's pipeline for large-scale NeRF, discussed in subsection 2.8.6, includes the use of COLMAP for individual segments, reducing the complexity and time used to approximate camera poses. Given this thesis' findings of COLMAP's performance in contrast to Nerfstudio's camera optimization, it seems worthwhile to explore this approach and integrate it into the large-scale NeRF pipeline.

When capturing real data that span large areas, it is inevitable to capture a lot of transient objects in the image data, such as moving cars and pedestrians. This data contributes to blurry renderings and artifacts because the object's position changes from frame to frame. A solution to this problem is the employment of object segmentation models and masking. By employing a segmentation model and subsequent masking of transient objects, the data becomes more consistent across frames and should produce higher-quality renderings.

Although the naive Block lookup-table works well when rendering the same path the vehicle had when capturing the data, it does not necessarily adapt to the free roaming of the 3D scene. In order to remedy this, a visibility prediction network could be employed. This network's responsibility would be to predict if a certain point would be "visible" from a specific Block-NeRF, in other words, predict if a Block-NeRF's render would provide additional information to the final render. Combining this technique with image-merging techniques, such as merging multiple rendered images with inverse distance weighting, could provide a higher-quality and more coherent final render. This is because multiple Block-NeRFs could contribute to rendering a point they've only partially "seen", resulting in a more complete representation of the scene. Both of these techniques are explored in Block-NeRF [2], as discussed in subsection 2.8.3.

The Nerfacto-model and most other state-of-the-art NeRF models leverage a space-warping algorithm designed for front-facing or object-centric trajectories. The capture of data from a vehicle is not inherently object-centric, and changing the space-warping algorithm could potentially have a great impact on the performance. F2-NeRF [40] has proposed a new space-warping method designed to handle arbitrary trajectories, called *perspective warping*. Implementing F2-NeRF as the backbone model for large-scale NeRF would be an interesting track to pursue.

These related ideas present potential avenues for further research and improvement, offering opportunities to refine and enhance the existing methodology and outcomes of the study.

Bibliography

[1] M. Debbagh, *Neural radiance fields (NeRFs): A review and some recent developments*, arXiv:2305.00375 [cs, eess], Apr. 2023. [Online]. Available: http://arxiv.org/abs/2305.00375 (visited on 05/31/2023).

[2] M. Tancik, V. Casser, X. Yan, S. Pradhan, B. Mildenhall, P. P. Srinivasan, J. T. Barron, and H. Kretzschmar, *Block-NeRF: Scalable Large Scene Neural View Synthesis*, arXiv:2202.05263 [cs], Feb. 2022. [Online]. Available: http://arxiv.org/abs/2202.05263 (visited on 09/12/2022).

[3] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant Neural Graphics Primitives with a Multiresolution Hash Encoding," *ACM Transactions on Graphics*, vol. 41, no. 4, pp. 1–15, Jul. 2022, arXiv:2201.05989 [cs], ISSN: 0730-0301, 1557-7368. DOI: 10.1145/3528223.3530127. [Online]. Available: http://arxiv.org/abs/2201.05989 (visited on 09/02/2022).

[4] 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*, arXiv:2103.13415 [cs], Aug. 2021. [Online]. Available: http://arxiv.org/abs/2103.13415 (visited on 09/02/2022).

[5] M. Tancik, E. Weber, E. Ng, R. Li, B. Yi, J. Kerr, T. Wang, A. Kristoffersen, J. Austin, K. Salahi, A. Ahuja, D. McAllister, and A. Kanazawa, *Nerfstudio: A Modular Framework for Neural Radiance Field Development*, arXiv:2302.04264 [cs], Feb. 2023. DOI: 10.48550/arXiv.2302.04264. [Online]. Available: http://arxiv.org/abs/2302.04264 (visited on 03/06/2023).

[6] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[7] M. Tancik*, E. Weber*, E. Ng*, R. Li, B. Yi, T. Wang, A. Kristoffersen, J. Austin, K. Salahi, A. Ahuja, D. McAllister, and A. Kanazawa, *Nerfstudio: A framework for neural radiance field development*, 2022. [Online]. Available: https://github.com/nerfstudio-project/nerfstudio.

[8] *NAPLab*, https://www.ntnu.edu/idi/naplab, Accessed on May 25, 2023, n.d.

[9] S. Tomar, “Converting video formats with ffmpeg,” *Linux Journal*, vol. 2006, no. 146, p. 10, 2006.

[10] 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*, arXiv:2003.08934 [cs], Aug. 2020. [Online]. Available: http://arxiv.org/abs/2003.08934 (visited on 08/30/2022).

[11] J. T. Barron, B. Mildenhall, D. Verbin, P. P. Srinivasan, and P. Hedman, *Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields*, arXiv:2111.12077 [cs], Mar. 2022. [Online]. Available: http://arxiv.org/abs/2111.12077 (visited on 09/02/2022).

[12] J. L. Schonberger and J.-M. Frahm, “Structure-from-Motion Revisited,” en, in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA: IEEE, Jun. 2016, pp. 4104–4113, ISBN: 978-1-4673-8851-1. DOI: 10.1109/CVPR.2016.445. [Online]. Available: http://ieeexplore.ieee.org/document/7780814/ (visited on 11/26/2022).

[13] I. J. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[14] R. A. Drebin, L. Carpenter, and P. Hanrahan, “Volume rendering,” *SIGGRAPH Comput. Graph.*, vol. 22, no. 4, Jun. 1988, ISSN: 0097-8930. DOI: 10.1145/378456.378484. [Online]. Available: https://doi.org/10.1145/378456.378484.

[15] N. Max, “Optical models for direct volume rendering,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 1, no. 2, pp. 99–108, Jun. 1995, ISSN: 10772626. DOI: 10.1109/2945.468400. [Online]. Available: http://ieeexplore.ieee.org/document/468400/ (visited on 11/11/2022).

[16] J. Pawasauskas, *Volume visualization with ray casting*, https://web.cs.wpi.edu/~matt/courses/cs563/talks/powwie/p1/ray-cast.htm, Accessed on June 12, 2023, 1997.

[17] P. Lacroute and M. Levoy, “Fast volume rendering using a shear-warp factorization of the viewing transformation,” in *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH ’94, New York, NY, USA: Association for Computing Machinery, 1994, pp. 451–458, ISBN: 0897916670. DOI: 10.1145/192161.192283. [Online]. Available: https://doi.org/10.1145/192161.192283.

[18] K. Engel, M. Bauer, G. Greiner, and T. Ertl, “Interactive volume rendering on standard pc graphics hardware using multi-textures and multi-stage rasterization,” *Proceedings of the SIGGRAPH/Eurographics Workshop on Graphics Hardware*, vol. 147, Jun. 2000. DOI: 10.1145/346876.348238.

[19] L. Westover, “Footprint evaluation for volume rendering,” *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, 1990.

[20] Wikipedia, *Volume ray casting — Wikipedia, the free encyclopedia*, http://en.wikipedia.org/w/index.php?title=Volume%20ray%20casting&oldid=1119695339, [Online; accessed 28-November-2022], 2022.

[21] 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*, arXiv:2111.11426 [cs], Apr. 2022. [Online]. Available: http://arxiv.org/abs/2111.11426 (visited on 11/05/2022).

[22] M. Tancik, P. P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. T. Barron, and R. Ng, *Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains*, arXiv:2006.10739 [cs], Jun. 2020. [Online]. Available: http://arxiv.org/abs/2006.10739 (visited on 11/28/2022).

[23] C.-H. Lin, W.-C. Ma, A. Torralba, and S. Lucey, *BARF: Bundle-Adjusting Neural Radiance Fields*, arXiv:2104.06405 [cs], Aug. 2021. [Online]. Available: http://arxiv.org/abs/2104.06405 (visited on 11/27/2022).

[24] Z. Wang, S. Wu, W. Xie, M. Chen, and V. A. Prisacariu, *NeRF–: Neural Radiance Fields Without Known Camera Parameters*, arXiv:2102.07064 [cs], Apr. 2022. [Online]. Available: http://arxiv.org/abs/2102.07064 (visited on 11/30/2022).

[25] J. L. Schönberger, E. Zheng, M. Pollefeys, and J.-M. Frahm, "Pixelwise view selection for unstructured multi-view stereo," in *European Conference on Computer Vision (ECCV)*, 2016.

[26] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004, ISSN: 1573-1405. DOI: 10.1023/B:VISI.0000029664.99615.94. [Online]. Available: https://doi.org/10.1023/B:VISI.0000029664.99615.94.

[27] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[28] Video Processing, Compression and Quality Research Group, *Ways of cheating on popular objective metrics*, https://videoprocessing.ai/metrics/ways-of-cheating-on-popular-objective-metrics.html, Sep. 2021.

[29] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, *The Unreasonable Effectiveness of Deep Features as a Perceptual Metric*, arXiv:1801.03924 [cs], Apr. 2018. [Online]. Available: http://arxiv.org/abs/1801.03924 (visited on 05/10/2023).

[30] L. Williams, "Pyramidal parametrics," ACM, Detroit, Michigan, United States, 1983, pp. 1–11. DOI: 10.1145/800059.801126. [Online]. Available: http://doi.acm.org/10.1145/800059.801126.

[31] B. Mildenhall, P. P. Srinivasan, R. Ortiz-Cayon, N. K. Kalantari, R. Ramamoorthi, R. Ng, and A. Kar, “Local light field fusion: Practical view synthesis with prescriptive sampling guidelines,” *ACM Transactions on Graphics (TOG)*, 2019.

[32] A. F. Agarap, *Deep Learning using Rectified Linear Units (ReLU)*, arXiv:1803.08375 [cs, stat], Feb. 2019. [Online]. Available: `http://arxiv.org/abs/1803.08375` (visited on 05/31/2023).

[33] R. Martin-Brualla, N. Radwan, M. S. M. Sajjadi, J. T. Barron, A. Dosovitskiy, and D. Duckworth, *NeRF in the Wild: Neural Radiance Fields for Unconstrained Photo Collections*, arXiv:2008.02268 [cs], Jan. 2021. [Online]. Available: `http://arxiv.org/abs/2008.02268` (visited on 09/17/2022).

[34] P. Sokolski, “Building city-scale neural radiance fields for autonomous driving,” Mar. 2023. [Online]. Available: `https://www.nvidia.com/en-us/on-demand/session/gtcspring23-s51770/`.

[35] CARLA, *BRMC_9*, [Photograph], n.d. [Online]. Available: `https://d26ilriwvtzlb.cloudfront.net/8/83/BRMC_9.jpg`.

[36] D. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2017. arXiv: `1412.6980v9 [cs.LG]`.

[37] Swift Navigation, *Duro specification and user manual*, `https://www.swiftnav.com/sites/default/files/duro_product_summary.pdf`, Accessed May 19, 2023, n.d.

[38] M. Hirsch, *PyGemini*. DOI: `10.5281/zenodo.3262738`.

[39] Video Processing AI, *Ways of cheating on popular objective metrics*, `https://videoprocessing.ai/metrics/ways-of-cheating-on-popular-objective-metrics.html`, Accessed: May 20, 2023, n.d.

[40] P. Wang, Y. Liu, Z. Chen, L. Liu, Z. Liu, T. Komura, C. Theobalt, and W. Wang, *F^{2}-NeRF: Fast Neural Radiance Field Training with Free Camera Trajectories*, arXiv:2303.15951 [cs], Mar. 2023. [Online]. Available: `http://arxiv.org/abs/2303.15951` (visited on 04/04/2023).

Appendix A

Additional Material

A.1 Parameters for the training in Nerfstudio

This section provide the full configuration for the models trained in Nerfstudio.

Table A.1: An overview of the parameters in the default Nerfacto model. Rows prefixed with *DF* describe the model's corresponding *Density Fields*.

Description	Default Value
Max number of iterations	15000
Number of rays per batch	4096
Optimizer	Adam
How far along the ray to start sampling.	0.05
How far along the ray to stop sampling.	1000.0
Number of samples per ray for the nerf network.	48
Sample every n steps after the warmup	5
Scales n from 1 to `proposal_update_every` num steps	5000
Number of proposal network iterations.	2
Use the same proposal network.	False
Number of samples per ray for each proposal network	256 & 96
Dimension of hidden layers	64
Maximum resolution of the hashmap for the base MLP	2048
Proposal loss multiplier.	1.0
Distortion loss multiplier.	0.002
Orientation loss multipier on computed noramls.	0.0001
Predicted normal loss multiplier.	0.001
Use proposal weight annealing.	True
Use average appearance embedding or zeros for inference.	True
Slope of the annealing function for the proposal weights	10.0
Max num iterations for the annealing function.	1000

Continued on next page

Table A.1 – continued from previous page

Description	Default Value
Use single jitter or not for the proposal networks.	True
Predict normals or not.	False
DF: Dimension of hidden layer	16
DF: Hashmap size	2^{17}
DF: Number of levels of the hashmap	5
DF: Maximum resolution of the hashmap (density field 1)	64
DF: Maximum resolution of the hashmap (density field 2)	256

Table A.2: An overview of the parameters in the Nerfacto-big model. Rows prefixed with *DF* describe the model's corresponding *Density Fields*.

Description	Default Value
Max number of iterations	100000
Number of rays per batch	4096
Optimizer	Rectified Adam
How far along the ray to start sampling.	0.05
How far along the ray to stop sampling.	1000.0
Number of samples per ray for the nerf network.	128
Sample every n steps after the warmup	5
Scales n from 1 to `proposal_update_every` num steps	5000
Number of proposal network iterations.	2
Use the same proposal network.	False
Number of samples per ray for each proposal network	512 & 256
Dimension of hidden layers	128
Maximum resolution of the hashmap for the base MLP	3000
Proposal loss multiplier.	1.0
Distortion loss multiplier.	0.002
Orientation loss multipier on computed noramls.	0.0001
Predicted normal loss multiplier.	0.001
Use proposal weight annealing.	True
Use average appearance embedding or zeros for inference.	True
Slope of the annealing function for the proposal weights	10.0
Max num iterations for the annealing function.	1000
Use single jitter or not for the proposal networks.	True
Predict normals or not.	False
DF: Dimension of hidden layer	16
DF: Hashmap size	2^{21}
DF: Number of levels of the hashmap	5
DF: Maximum resolution of the hashmap (density field 1)	64
DF: Maximum resolution of the hashmap (density field 2)	256

Table A.3: An overview of the parameters in the default Instant-ngp model

Description	**Value**
Max number of iterations	15000
Number of rays per batch	4096
Optimizer	Adam
Whether to create a scene collider to filter rays.	False
Number of samples in field evaluation.	24
Resolution of the grid used for the field.	128
Contraction type.	Unbounded Sphere
Cone angle	0.004
Minimum step size for rendering.	0.01
How far along ray to start sampling.	0.05
How far along ray to stop sampling.	1e3
Whether to use an appearance embedding.	False
Whether to randomize the background color.	True

Table A.4: An overview of the parameters in the default Mip-NeRF model

Description	**Value**
Max number of iterations	300000
Number of coarse samples	128
Number of fine samples	128
Optimizer	Rectified Adam
Whether to create a scene collider to filter rays.	True
Near plane collider-plane.	2.0
Far plane collider-plane.	6.0
The loss coeficcient for the coarse MLP.	0.1
The loss coeficcient for the fine MLP.	1.0
Number of rays per chunk during eval	1024

A.2 Results

This section presents the complete results of the experiments where some results were condensed to convey the primary findings.

A.2.1 Simulated Noise Conditions

Table A.5: Results for Simulated Noise Condition experiment on both the baseline and shorter segments. The shorter segments are 10% the size of the baseline segment, approximately 50m in length.

Description	**PSNR ↑**	**SSIM ↑**	**LPIPS ↓**	**PSNR ↑**	**SSIM ↑**	**LPIPS ↓**
	Baseline segment					
	With camera optimizer			**Without camera optimizer**		
$\mathcal{N}(0,0.0)$	23.41	0.714	0.321	24.70	0.793	0.179
$\mathcal{N}(0,0.1^2)$	22.51	0.674	0.321	22.46	0.677	0.281
$\mathcal{N}(0,0.2^2)$	21.34	0.617	0.338	21.21	0.602	0.367
$\mathcal{N}(0,0.3^2)$	20.52	0.577	0.346	20.47	0.561	0.399
$\mathcal{N}(0,0.5^2)$	19.08	0.501	0.372	19.37	0.499	0.474
$\mathcal{N}(0,1.0^2)$	17.67	0.434	0.433	18.21	0.444	0.560
$\mathcal{N}(0,3.0^2)$	16.66	0.408	0.637	16.32	0.386	0.648
	Shorter segment					
	With camera optimizer			**Without camera optimizer**		
$\mathcal{N}(0,0.0)$	24.83	0.825	0.102	25.68	0.862	0.077
$\mathcal{N}(0,0.1^2)$	23.03	0.753	0.115	23.32	0.755	0.147
$\mathcal{N}(0,0.2^2)$	20.67	0.614	0.149	21.38	0.629	0.209
$\mathcal{N}(0,0.3^2)$	20.50	0.596	0.155	21.10	0.612	0.233
$\mathcal{N}(0,0.5^2)$	19.06	0.483	0.197	20.07	0.513	0.299
$\mathcal{N}(0,1.0^2)$	18.09	0.407	0.338	18.29	0.418	0.405
$\mathcal{N}(0,3.0^2)$	15.71	0.345	0.613	15.08	0.363	0.699

A.2.2 Block-NeRF

Table A.6: Results for each segment when the baseline-segment spanning the entire block has been split into 4 Block-NeRFs.

Description	**PSNR ↑**	**SSIM ↑**	**LPIPS ↓**
Segment 1	24.96	0.815	0.162
Segment 2	25.66	0.824	0.164
Segment 3	23.91	0.755	0.194
Segment 4	23.36	0.765	0.216

Table A.7: Results for each segment when the larger segment spanning approximately 1.2km has been split into 12 Block-NeRFs.

Description	**PSNR ↑**	**SSIM ↑**	**LPIPS ↓**
Segment 1	25.05	0.827	0.139
Segment 2	25.70	0.831	0.156
Segment 3	24.92	0.830	0.150
Segment 4	22.73	0.713	0.207
Segment 5	25.47	0.838	0.132
Segment 6	25.67	0.842	0.147
Segment 7	25.66	0.821	0.164
Segment 8	25.07	0.777	0.154
Segment 9	24.80	0.815	0.154
Segment 10	23.69	0.754	0.186
Segment 11	23.94	0.757	0.179
Segment 12	24.58	0.808	0.144
Average metrics of 12 Block-NeRF	24.77	0.801	0.159
Average metrics of 1 Block-NeRF	22.52	0.654	0.424

A.2.3 Real data

Table A.8: Data from dataset 2 with transformation matrix approximated with COLMAP or GPS-readings. BNs an abbreviation of Block-NeRF and the resulting metric score is averaged across the 4 NeRFs evaluations.

Description	PSNR ↑	SSIM ↑	LPIPS ↓	Time processing
COLMAP w/ optimizer	19.94	0.724	0.535	00:10:25
COLMAP w/o optimizer	20.99	0.757	0.468	00:10:25
COLMAP w/ optimizer, 4 BNs	20.37	0.760	0.428	00:10:25
COLMAP w/o optimizer, 4 BNs	20.45	0.761	0.425	00:10:25
GPS w/ optimizer	18.88	0.711	0.573	00:00:00
GPS w/o optimizer	15.29	0.693	0.649	00:00:00
GPS w/ optimizer, 4 BNs	18.41	0.690	0.560	00:00:00
GPS w/o optimizer, 4 BNs	14.58	0.667	0.656	00:00:00

Table A.9: Data from dataset 3 with transformation matrix approximated with COLMAP or GPS-readings. BNs an abbreviation of Block-NeRF and the resulting metric score is averaged across the 4 NeRFs evaluations.

Description	PSNR ↑	SSIM ↑	LPIPS ↓	Time processing
COLMAP w/ optimizer	19.84	0.682	0.542	00:08:45
COLMAP w/o optimizer	22.31	0.752	0.431	00:08:45
COLMAP w/ optimizer, 4 BNs	20.97	0.699	0.397	00:08:45
COLMAP w/o optimizer, 4 BNs	23.40	0.794	0.329	00:08:45
GPS w/ optimizer	18.17	0.653	0.598	00:00:00
GPS w/o optimizer	13.80	0.627	0.719	00:00:00
GPS w/ optimizer, 4 BNs	18.40	0.638	0.545	00:00:00
GPS w/o optimizer, 4 BNs	14.37	0.619	0.683	00:00:00