HOCHSCHULE FURTWANGEN UNIVERSITY | HFU

Master Thesis

in the field of

Media Informatics

Neural Radiance Fields in the context of the Industrial Metaverse

supervised by: Prof. Dr. Uwe Hahne

cosupervised by: Dr. Jakob Lindinger

submitted on: 10.10.2023

submitted by: Sabine Schleise
matriculation number: 2701169
Wilhelmstr. 4, 78120 Furtwangen
sabine.schleise@hs-furtwangen.de

Accompanying Website

As part of this thesis, an accompanying website has been developed to supplement the content presented in this document. The website hosts various videos and animations that provide a dynamic understanding of certain concepts which might be hard to grasp through text and static figures alone. Wherever applicable, interactive demos or simulations are provided, enabling users to engage with the content and experiment in a hands-on manner.

For those reading this in print or in a non-interactive format, the website can be accessed at `https://sabinecelina.github.io/masterthesis-nerf_mv/`. I encourage all readers to explore the website to gain a richer and more comprehensive understanding of the topics discussed in this document.

Abstract

With the advancement of virtual and augmented reality technologies, the term "Metaverse" is becoming increasingly familiar. The Metaverse represents a convergent virtual world for interaction, work, and entertainment. Especially in the industrial sector, the Metaverse opens up new possibilities in terms of training, product development and collaboration. One of the key elements of the virtual world is the ability to display three-dimensional (3D) scenes. Traditionally, triangle meshes with textures, point clouds, volumetric grids, and implicit surface functions have been the preferred methods for $3D$ scene representation because of their clear structure and fast GPU/CUDA compatibility (Kerbl et al., 2023; Tewari et al., 2020). However, a new method called Neural Radiance Fields has recently emerged (Mildenhall et al., 2020). Neural Radiance Fields (NeRF) leverages continuous scene representations, typically employing a Multi-Layer Perceptron (MLP) optimized through volumetric ray-marching for the synthesis of novel views of captured scenes. Instead of directly reconstructing the complete $3D$ scene geometry, NeRF generates a volumetric representation called a "radiance field," which is capable of creating color and density at every point within the relevant $3D$ space.

However, using existing frameworks to generate a neural radiance field from real world data requires developer expertise and considerable effort, which significantly limits their adoption in industry. Therefore, we introduce a NeRF trainer to bridge this gap. This NeRF trainer implementation reduces the complexity of creating neural radiance fields by providing users with a user-friendly tool to construct their own 3D representations. We show the potential of the NeRF algorithm in generating high-fidelity $3D$ scenes from $2D$ images and how it can advance the Metaverse in the industrial context.

Zusammenfassung

Mit der Weiterentwicklung der Technologien für virtuelle und erweiterte Realität wird der Begriff "Metaverse" immer vertrauter. Das Metaverse stellt eine konvergente virtuelle Welt für Interaktion, Arbeit und Unterhaltung dar. Besonders im industriellen Bereich eröffnet das Metaverse neue Möglichkeiten für Training, Produktentwicklung und Zusammenarbeit. Eines der Schlüsselelemente der virtuellen Welt ist die Möglichkeit, $3D$-Szenen darzustellen. Traditionell waren Meshes mit Texturen, Punktwolken, volumetrische Gitter und implizite Oberflächenfunktionen aufgrund ihrer klaren Struktur und schnellen GPU/CUDA-Kompatibilität die bevorzugten Methoden zur Darstellung von $3D$-Szenen (Kerbl et al., 2023; Tewari et al., 2020). Vor kurzem wurde jedoch eine neue Methode namens Neural Radiance Fields vorgestellt (Mildenhall et al., 2020). NeRF nutzt kontinuierliche Szenendarstellungen, wobei typischerweise ein Multi-Layer Perceptron (MLP) eingesetzt wird, das durch volumetrisches Ray-Marching für die Synthese neuartiger Ansichten erfasster Szenen optimiert wird. Anstatt die komplette $3D$-Szenengeometrie direkt zu rekonstruieren, generiert NeRF eine volumetrische Darstellung, ein sogenanntes "Strahlungsfeld", das in der Lage ist, Farbe und Dichte an jedem Punkt im relevanten $3D$-Raum zu erzeugen.

Die Verwendung bestehender Frameworks zur Generierung eines neuronalen Strahlungsfeldes aus echten Daten erfordert jedoch das Fachwissen von Entwicklern und einen beträchtlichen Aufwand, was deren Einsatz in der Industrie erheblich einschränkt. Daher führen wir einen NeRF-Trainer ein, um diese Lücke zu schließen. Diese NeRF-Trainer-Implementierung reduziert die Komplexität der Erstellung neuronaler Strahlungsfelder, indem sie den Benutzern ein benutzerfreundliches Werkzeug zur Verfügung stellt, mit dem sie ihre eigenen $3D$-Darstellungen konstruieren können. Wir zeigen das Potenzial des NeRF-Algorithmus bei der Erzeugung von $3D$Szenen mit hoher Wiedergabetreue aus $2D$Bildern und wie er das Metaverse im industriellen Kontext voranbringen kann.

Acknowledgements

I would like to express my deepest gratitude to those who have accompanied, supported, and encouraged me throughout this journey. Special thanks to Dr. Jakob Lindinger, my co-supervisor, for his invaluable guidance and insights, which contributed significantly to the completion of this thesis. I am thankful for the consistent and eye-opening discussions that have deepened my understanding of my thesis. I am deeply indebted to my supervisor, Prof. Dr. Uwe Hahne, for giving me the creative freedom to design my thesis and for placing his trust in me. His broad interest in the subject has provided me with invaluable insights that have greatly benefited my work.

A big thank you goes to Michael Schelb. As a technical support, he was always present, especially when it came to complicated, company-related complications. Further thanks go to the members of the team I was a part of for their general support, feedback and help in answering various questions. I would also like to thank my team members for their unwavering support, constructive feedback, and assistance in answering a variety of questions. Their warm welcome and openness throughout my project, as well as our enlightening conversations over lunch and coffee breaks, deserve special mention. Special thanks are due to Per Verheyen for his significant contributions to the CI/CD pipeline; without his expertise, the project would not have reached its current level of excellence.

On a personal level, my deepest thanks go to my partner Sascha, whose emotional support has been crucial; without you, this work would not have the quality it has. You have helped me to stay focused and motivated even in times of lack of motivation. I would also like to thank my sisters who studied, wrote, and worked with me online. Your tireless support has not only enriched my work, but has also provided the necessary breaks for open conversations that have encouraged me to continue working with renewed energy.

Your support is the foundation on which this thesis is built.

Contents

List of Figures

List of Code Listings

List of Abbreviations

GPU Graphics Processing Unit

NeRF Neural Radiance Fields

instant-ngp Instant Neural Graphics Primitives

MLP Multilayer Perceptron

GUI graphical user interface

CUDA Compute Unified Device Architecture

mip-NeRF A Multiscale Representation for Anti-Aliasing Neural Radiance Fields

SSIM structural similarity

VR Virtual Reality

AR Augmented Reality

XR Extended Reality

MR Mixed Reality

IPE integrated positional encoding

PE Positional Encoding

USD Universal Scene Description

SfM Structure-from-Motion

SIFT Scale Invariant Feature Transform

BA Bundle Adjustment

SDF Signed Distance Function

PyPI Python Package Index

CI continuous integration

CD continuous delivery

CG computer graphics

IBRL Image-Based Relighting

OLAT one-light-at-a-time

SSAN Signed Surface Approximation Network

TSDF Truncated Signed Distance Field

IoT Internet of Things

AI Artificial Intelligence

1. Introduction

In this thesis we investigate the ability of the Neural Radiance Fields algorithm to convert two-dimensional (2D) images into three-dimensional (3D) scenes, particularly in the context of the Industrial Metaverse. We introduce a user-friendly framework that allows individuals without technical expertise to explore scene representations generated from neural radiance fields using their own captured data. Additionally, we explore the potential impact of this algorithm on the rapidly evolving landscape of the Industrial Metaverse.

1.1. Evolution of capturing the visual reality

The concept of recording visual reality deals with the process of representing the real world at a particular point in time. Since the earliest cave paintings, situations and experiences have been recorded for posterity. Throughout history, various techniques and technologies have been developed to provide more accurate representations (Reisig and Freytag, 2006).

The camera obscura laid the groundwork for advanced cameras with accurate representations of the real world. In 1850, the first stereophotographs were presented, making it possible for the first time to create a $3D$ impression in images. Stereography, which uses two slightly offset images, provides a sense of depth by presenting each eye with a slightly different perspective, mimicking the way our two eyes perceive depth in reality. Beyond stereography, another method called photogrammetry emerged to extract $3D$ data from $2D$ images. Derived from the Greek words "photos" (light), "gramma" (something written or drawn), and "metron" (measure), photogrammetry focuses on capturing and processing information from images. Its main objective is to determine the shape, size, and spatial positioning of objects, using multiple images from various viewpoints to create precise $3D$ reconstructions (Reisig and Freytag, 2006).

Photogrammetry is not without its challenges. One notable issue is the handling of transparent or reflective objects. These materials can lead to inaccurate reconstructions due to their unique light interaction properties. In addition, the process requires specialized software and considerable time, making it less accessible to those without technical expertise (Reisig and Freytag, 2006). Traditional photogrammetry focuses on reconstructing the geometric attributes of objects, neglecting the nuanced lighting

and color information that radiance fields encapsulate. It is in this gap that the NeRF algorithm has made a transformative difference (Mildenhall et al., 2020; Müller et al., 2022). Unlike traditional methods, NeRF generates a volumetric representation called a "radiance field". This field assigns color and density attributes to each point in $3D$ space, providing a more complete understanding of how light interacts with objects and scenes. This enhanced representation contributes to more realistic and visually appealing reconstructions. Currently, however, creating neural radiance fields is a complex process that requires developer expertise and specialized tools. This has resulted in their being accessible only to those with technical expertise. Given this barrier, this thesis presents an implementation of a NeRF framework. The aim of this framework is to make this advanced approach accessible to users regardless of their technical background. It allows individuals to engage with NeRF technology without being hindered by the complexities normally associated with its usage.

1.2. NeRF in the context of the Industrial Metaverse

The Industrial Metaverse represents the convergence of virtual and physical reality in industrial contexts, opening up new opportunities for the application of virtual technologies in manufacturing, product development, sales and training to create more efficient and collaborative workflows. In the Industrial Metaverse, companies can create virtual environments where employees, customers and partners can interact. These virtual environments can be digital representations of manufacturing facilities, virtual training environments, or virtual product development labs. Virtual Reality (VR) is a powerful virtual environment tool that offers remarkable levels of immersion by placing the user in a completely virtual environment. This immersive encounter actually gives the feeling of being in a different, tangible virtual setting. An important aspect of this concept is the creation of realistic $3D$ models, product environments and scenarios tailored to various industrial use cases.

However, as previously described, the current process for creating these complex $3D$ scenes is challenging. It requires modelling the scenes or photogrammetry with manual post-processing. To overcome this challenge, NeRF has emerged as a promising solution. Using neural network based algorithms, NeRF enables the generation of complex $3D$ scenes and objects from $2D$ images. This technique has the potential to improve the efficiency of converting $2D$ information into accurate and detailed $3D$ representations. While the theoretical potential of NeRF is considerable, its practical application in the industrial sector is still in its early stages. Currently, because of its novelty, its exploration is mostly limited to academics, developers and tech enthusiasts. This thesis aspires to address this oversight. The objective is not only to integrate NeRF into the industrial context, but also to thoroughly evaluate

its possibilities. Furthermore, this thesis aims to explore the various applications and implications of this innovative approach in the industrial domain. The aim of this thesis is to provide a comprehensive assessment of the potential role of NeRF in the advancement of industrial processes and methods, particularly in the context of the Industrial Metaverse.

1.3. Outline

Chapter 2 lays the theoretical foundation, touching on the elementary components of early vision to understand the concept of reality capture. We then go on to explain the workings of the NeRF algorithm, highlighting its subsequent advancements designed to address challenges and limitations. The focus of this chapter is on the instant-ngp algorithm, a recent improvement to the original NeRF algorithm, which is used in our thesis to generate NeRFs. Additionally, we discuss other research initiatives in the NeRF framework, highlighting the growing academic interest in this area.

Chapter 3 introduces existing frameworks that are capable of training, rendering and storing NeRF models. We evaluate the accessibility and intuitiveness of these frameworks and investigate how neural radiance fields can be created from user-generated data within these frameworks.

Then, in chapter 4, we explain the motivations for developing a custom framework. We dive deeper into the "NeRF-Trainer" implementation, which is accessible via a simple web application. This platform facilitates the training of neural radiance fields, requiring only the upload of user-generated data.

NeRF has a wide range of potential applications, especially in the context of the Industrial Metaverse. In Chapter 5, we explore the diverse applications of NeRF in the Industrial Metaverse, particularly where traditional approaches like photogrammetry fall short, such as in avatar creation, mesh generation or in rendering difficult material properties. However, we also identify limitations the algorithm has and present research that is attempting to overcome these challenges.

The final chapter 6 provides a concise summary and discussion of the thesis, as well as an outlook on potential future projects.

2. Basics

Computer Vision has made significant advances by using mathematical functions parameterized by the weights of a MLP to represent computer graphics primitives, which parameterizes physical properties of scenes or objects across space and time (Tewari et al., 2020; Xie et al., 2022). This chapter explores the NeRF algorithm, an approach that uses MLPs to regress a $5D$ input coordinate to a volume density and view-dependent color output. By querying the neural network for that output information, the algorithm uses classical rendering techniques to synthesize novel views. Before going into more detail on this method, we pick up on the idea of capturing reality by introducing the plenoptic function. Then, in Section 2.2, we explore the fundamentals of the NeRF approach and how it leverages the power of neural networks to model volumetric scenes, enabling the generation of high-quality $3D$ representations from $2D$ images. The NeRF method creates the volumetric scene representation based on input images and camera poses, which can lead to undersampling and aliasing artifacts. Furthermore, the training process is slow, and rendering can take a long time (Rabby and Zhang, 2023). We will therefore turn our attention to the instant-ngp (Müller et al., 2022) algorithm, which forms the basis of this thesis. This algorithm adeptly addresses the issues of training efficiency and rendering speed. Finally, we will present different definitions of the Metaverse in order to identify the main challenges and potential of using the NeRF algorithm in the context of the Industrial Metaverse.

2.1. Capturing Reality: The Elements of Early Vision

The question of the basic elements of early vision deals with the fundamental substances that underlie visual perception. In this context, the focus is less on discrete objects such as simple edges and corners, but rather on the essential building blocks of the visual process. The first stage of visual processing comprises a series of parallel pathways, each dedicated to a specific visual property, such as motion, color, orientation, and binocular disparity. These fundamental attributes are regarded as measurable units of early vision, with the focus on how the visual system quantifies the characteristics of objects, rather than how it labels them. The primary goal of this approach is to systematically derive visual elements and elucidate their connection to the structure of visual information in the surrounding world. All critical visual measurements can be

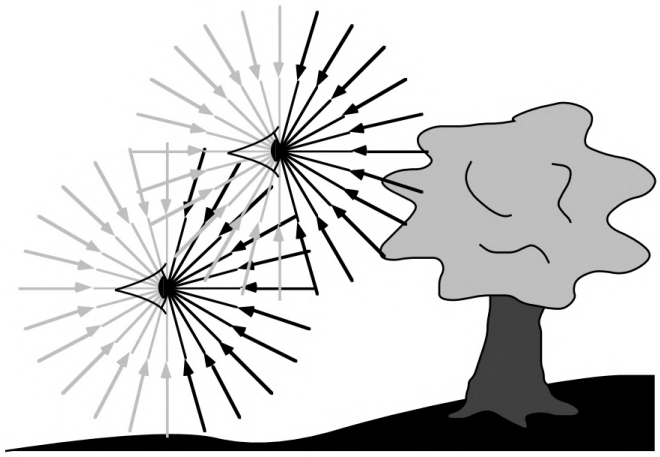


Figure 1.: The plenoptic function describes the information available to an observer at any point in space and time. This illustration depicts two schematic eyes, each with punctate pupils, capturing bundles of light rays. Notably, the plenoptic function encompasses rays originating from beyond the observer's viewpoint, which remain unseen (Adelson and Bergen, 1991).

characterized as local variations in one or more dimensions of a single function. This function describes the nature of incoming light to an observer and includes everything that has the potential for visual perception. This comprehensive function is called the "plenoptic function", derived from "plenus" meaning complete or full and "optical", which relates to optics. The importance of the plenoptic function lies in its role as an intermediary between $3D$ physical objects and the resulting retinal images in the eye of the observer. Rather than directly imparting attributes to objects, the plenoptic function imparts a pattern of light rays to the surrounding space. Ultimately, the observer samples this function to shape their visual perception. The plenoptic function acts as a link between the outside world and the inner eye. Introduced by Adelson and Bergen (1991), the plenoptic function attempts to capture the entirety of our visual reality and the information it contains. It represents a conceptual function with a $7D$ input denoted as $P = P(V_x, V_y, V_z, \theta, \phi, \lambda, t)$, where V represents an idealized eye at every possible location, recording the intensity of the light ray passing through the center of the pupil at every possible angle (θ, ϕ), for each wavelength λ, at each time t. The plenoptic function aims to model our visual reality comprehensively, including the $3D$ structure of the scene, surface features, motion, and other important visual aspects. It serves as the foundation for visual perception, providing raw data from which the visual system constructs our perception of the world. Figure 1 illustrates the idea of the plenoptic function (Adelson and Bergen, 1991).

Early attempts to address the theoretical concept of the plenoptic function include *Light Field Rendering* (Levoy and Hanrahan, 1996) and *The Lumigraph* (Gortler et al., 1996). *Light Field Rendering* captures and represents all the light rays passing

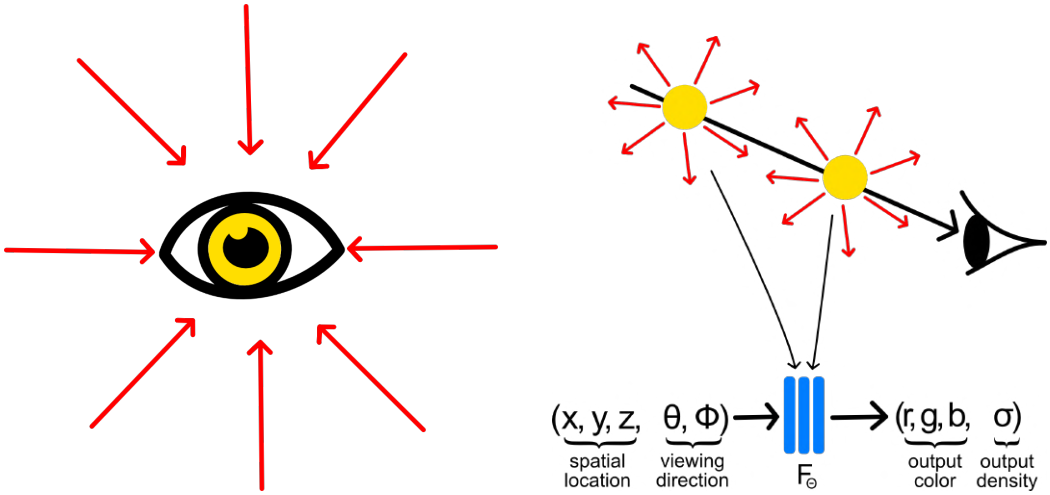
through a scene from various viewpoints and directions, providing a rich dataset that enables the reconstruction of different views and supports flexible rendering and post-processing. In contrast, *The Lumigraph* captures a set of images from a fixed set of viewpoints, allowing for the recreation of different views of the scene within the captured range. Both of these approaches represent significant advances in capturing and rendering the visual information contained within a scene, providing practical solutions for exploiting the vast amount of data encompassed by the plenoptic function. For further information about these two approaches, we refer to the paper by Levoy and Hanrahan, titled *Light Field Rendering* (Levoy and Hanrahan, 1996) and *The Lumigraph*, by Gortler et al. (1996) (Gortler et al., 1996), both presented at Siggraph in 1996.

Simplifying the plenoptic function by removing the dimensions of time and wavelength results in a static representation of the world. This simplification leads to a $5D$ function, $P = P(V_x, V_y, V_z, \phi, \theta)$, similar to the input for the network used in the NeRF algorithm. The plenoptic function models everything that comes into a single wavelength at any time and place. This is why, in theory, direct querying of this function would enable the rendering of diverse views from numerous vantage points. However, the NeRF algorithm takes a distinct approach. It diverges from the plenoptic function, which models the incoming visual data into a single eye. Instead, NeRF models the particles at every single point in space. These particles are characterized by their color and view-dependent radiance. This facilitates dynamic traversal through a scene, setting it apart from conventional light fields or prior methods. As a result, NeRF establishes a flexible representation, grounded in a robust underlying $3D$ structure. Each of these particles is further characterized by a function that takes into account $5D$ parameters and generates outputs for color and density, forming the core of NeRF's functionality (Kanawaza, 2023). Figure 2 visually contrasts the distinctions between the plenoptic function and the input employed by NeRF.

Through this comparison, it becomes evident that while the plenoptic function provides a comprehensive representation of visual information entering a single eye at a given moment, the NeRF approach diverges by considering each point in space along a ray (Kanawaza, 2023).

2.2. Neural Radiance Fields

NeRF is a method for synthesizing novel views of complex scenes by optimizing an underlying continuous volumetric scene function using a sparse set of $2D$ images. Given a set of images capturing a static scene from multiple angles, along with their corresponding poses, the neural network learns to represent that scene in a way that allows new views to be synthesized. The NeRF algorithm represents a continuous scene as a



(a) Plenoptic Function (b) NeRF Function

Figure 2.: Illustration highlighting distinction between the plenoptic function and the approach employed by NeRF. While the plenoptic function aims to capture all visual information that could enter a single eye at any given moment and location, NeRF instead feeds the network with information about every point along a ray. This ray is illustrated in Figure 2b, where it is marked in black, leading into the eye, and each point along it is indicated in yellow (Kanawaza, 2023)

$5D$ vector-valued function whose input is a $3D$ spatial location $X = (x, y, z)$ and a $2D$ viewing direction $d = (\phi, \theta)$, and whose output is the emitted color $c = (r, g, b)$ depending on each direction at each point and an view-independent volume density σ that ranges from $[0, \infty)$. To approximate this continuous $5D$ scene representation, an MLP network F_Θ is utilized, where $F_\Theta : (x, d) \longrightarrow (c, \sigma)$. The weights Θ of the network are optimized to map each $5D$ input coordinate to its corresponding volume density and directional emitted color. Part a and b of Figure 3 provide an overview of the NeRF scene representation (Mildenhall et al., 2020). Part c and d of this Figure will be explained later in this section.

2.2.1. Overview of the NeRF scene representation

The algorithm enables view synthesis through a three-step process, as illustrated in Figure 3. First, $5D$ coordinates are sampled along the camera rays and passed through an MLP to produce both color and volume density (steps a and b). Second, the output from the MLP is utilized to generate the individual pixels of an image using classical volume rendering techniques (step c). Finally, the differentiability of the rendering function is leveraged to optimize the weights of the MLP (step d). This is achieved by minimizing the loss function, which is simply defined as the total squared error

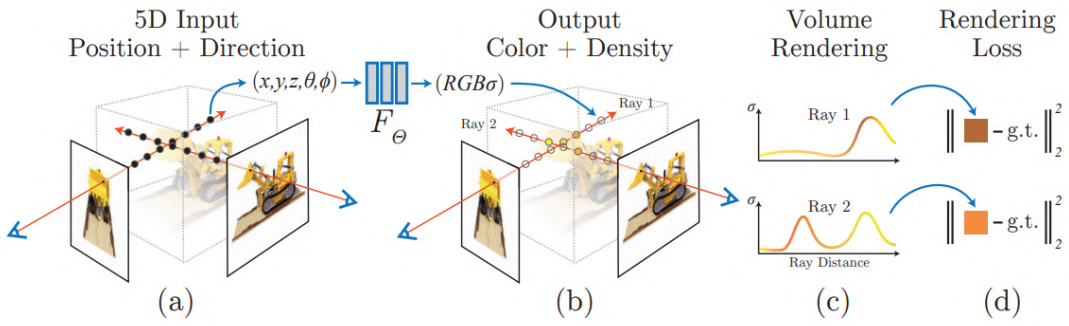


Figure 3.: An overview of the neural radiance field scene representation and differentiable rendering procedure. The synthesis of images is accomplished by sampling 5D coordinates, encompassing location and viewing direction, along camera rays (a). These sampled locations are then fed into a MLP to generate color and volume density values (b). By employing volume rendering techniques, these values are composited to create the final image (c). The rendering function is differentiable, enabling the optimization of the scene representation by minimizing the residual between the synthesized images and the observed ground truth images (d) (Mildenhall et al., 2020)

between the rendered and true pixel colors for both the coarse and fine rendering:

$$\mathcal{L} = \sum_{r \in \mathcal{R}} \left(\hat{C}_c(r) - C_{\text{gt}}(r) \right)^2 + \left(\hat{C}_f(r) - C_{\text{gt}}(r) \right)^2 \tag{2.1}$$

where $\mathcal{R}$ represents the set of rays in each batch and $C_{\text{gt}}(r)$, $\hat{C}_c(r)$ and $\hat{C}_f(r)$ denote the ground-truth, coarse volume predicted, and fine volume predicted pixel colors for ray r, respectively. The ray r passing through a pixel is determined based on the pixel's position and the camera's location. Due to the differentiability of the rendering function, the only input required to optimize this scene representation is a set of images with known camera poses (Mildenhall et al., 2020).

Detailed overview of the MLP

A scene is represented by using a fully-connected (non-convolutional) deep network, whose input is a single continuous $5D$ coordinate and whose output is the volume density and view-dependent emitted radiance at that spatial location.

The MLP, with an architecture of 8 layers and 256 neurons per layer, first takes the $3D$ spatial position coordinate as input and processes it through eight layers, returning two values: σ and a 256-dimensional feature vector. The feature vector is then concatenated with the viewing direction parameters $d = (\phi, \theta)$ and passed through another MLP layer, which produces the viewing direction dependent color $c = (r, g, b)$. Figure 4 illustrates the architecture of this MLP, highlighting the flow of information through the layers (Mildenhall et al., 2020). The variable $\gamma(x)$ in the Figure refers to the positional encoding explained in section 2.2.1, which helps

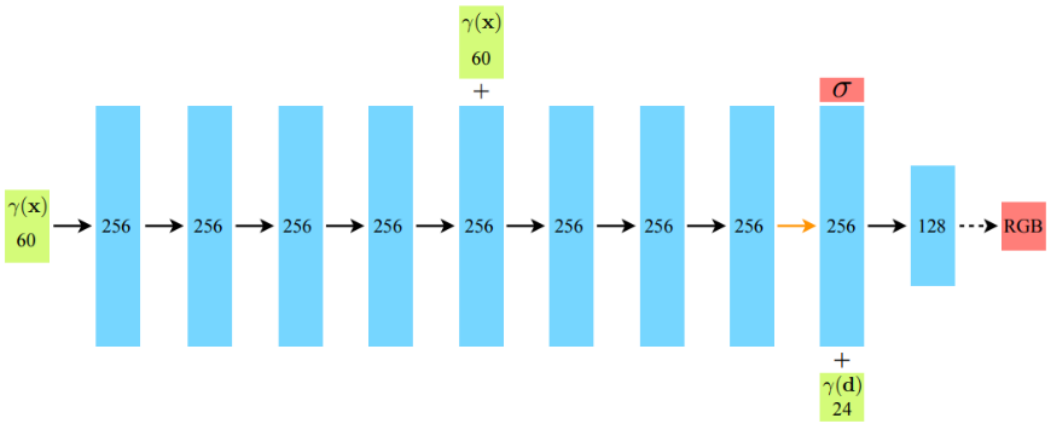


Figure 4.: The architecture of the MLP is depicted above. The green blocks represent the initial input vectors fed into the network, which undergo positional encoding denoted by $\gamma(x)$ as elaborated in section 2.2.1. The blue blocks illustrate the hidden layers and specify the number of neurons employed at each layer. Finally, the red blocks show the output vectors produced by the network. The MLP starts by taking the spatial location coordinate X as input. This input is processed through eight layers, resulting in the generation of σ and a 256-dimensional feature vector. The feature vector is then concatenated with the viewing direction parameters d, and this combined input is passed through an additional layer. The output of this layer represents the view-dependent color c (Mildenhall et al., 2020).

to process the input information effectively in the network. Once the neural network has optimized the weights to represent a scene, new views can be synthesized by rendering the neural radiance field from a particular viewpoint using classical rendering techniques.

Volume Rendering

With the output of the neural network, volume rendering is used to obtain the color $C(r)$ of any camera ray $r(t) = o + td$, with camera position o and viewing direction d using the classical rendering function represented as

$$C(r) = \int_{t_n}^{t_f} T(t) \cdot \sigma(r(t)) \cdot c(r(t), d) dt, \text{ where} \tag{2.2}$$
$$T(t) = \exp\left(-\int_{t_n}^{t} \sigma(r(s)), ds\right).$$

The rendering process, shown in the formula 2.2 involves integrating the emitted color $c(r(t), d)$, which depends on the $3D$ spatial location along the camera ray $r(t)$ and the viewing direction d, multiplied by the volume density $\sigma(r(t))$ of the scene at that location, over the range of distances along the ray from t_n to t_f.

This integral is attenuated by the factor $T(t)$, which represents the probability that the ray travels from t_n to t without hitting other particles, and accounts for the attenuation of light due to the density of the scene. The function $T(t)$ denotes the accumulated transmittance as the exponential of the integral of σ along the ray from t_n to t. Since σ is always non-negative, $T(t)$ can only decrease or stay the same as t increases along the ray. The final color $C(r)$ at a pixel $r(u, v)$ on the image plane is obtained by integrating the contributions of all points along the ray, each weighted by their emitted color, density, and attenuation factor, resulting in an accurate synthesis of novel views of complex scenes in NeRF. Rendering a view from the continuous neural radiance field requires estimating this integral $C(r)$ for a camera ray traced through each pixel of the desired virtual camera (Mildenhall et al., 2020).

The authors introduced several optimizations and adjustments to the original volume rendering function. Stratified sampling was incorporated to achieve higher resolution and flexibility. Furthermore, the computation of the color $\hat{C}(r)$ was modified to be trivially differentiable, which simplifies the optimization process. The final formula for $\hat{C}(r)$ is therefore:

$$\hat{C}(r) = \sum_{i=1}^{N} T_i \left(1 - \exp(-\sigma_i \delta_i)\right) c_i, \text{ where } T_i = \exp\left(-\sum_{j=1}^{i-1} \sigma_j \delta_j\right) \tag{2.3}$$

These adjustments make the algorithm better suited for the specific requirements of NeRF (Mildenhall et al., 2020; Tancik et al., 2020).

Positional Encoding

Allowing the F_θ network to operate directly on the $5D$ input coordinates often leads to poor rendering in terms of representing high-frequency variations in color and geometry (Mildenhall et al., 2020). Research by Rahaman et al. (2019) demonstrate that deep neural networks inherently exhibit a bias towards learning lower-frequency functions. Further, they reveal that transforming the inputs to a higher-dimensional space using high-frequency functions prior to network training significantly enhances the fit for data containing high-frequency variation (Rahaman et al., 2019). These findings have been leveraged in the context of neural scene representations. The NeRF algorithm encodes scalar positions as a multi-resolution sequence of $L \in \mathbb{N}$ sine and cosine functions as shown below (Mildenhall et al., 2020).

$$\gamma(x) = \Big(sin(2^0 x), sin(2^1 x), ..., sin(2^{L-1} x), cos(2^0 x), cos(2^1 x), .., cos(2^{L-1} x)\Big),$$

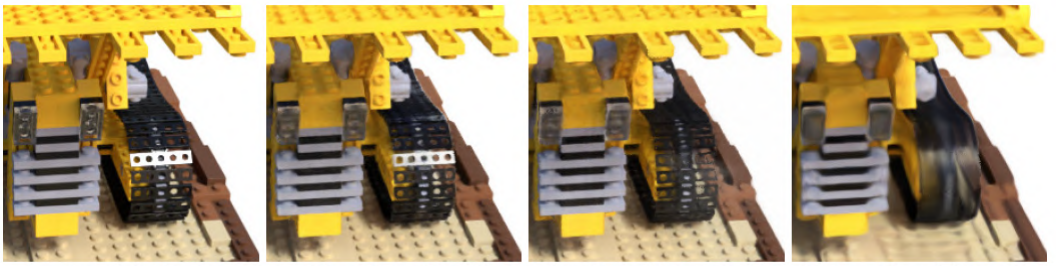


Figure 5.: This visualization demonstrates the advantages of incorporating view-dependent emitted radiance and utilizing high-frequency positional encoding in a full model. When view dependence is removed, the model fails to reproduce the specular reflection on the bulldozer tread. Likewise, the absence of positional encoding significantly diminishes the model's ability to represent intricate details of high-frequency geometry and texture, leading to an overly smoothed appearance (Mildenhall et al., 2020).

where L is a parameter representing the number of encoding functions used (Mildenhall et al., 2020; Müller et al., 2022).

This function maps continuous input coordinates to a higher dimensional space, enhancing the MLP's ability to approximate functions with higher frequencies (Mildenhall et al., 2020).

To conclude this section, Figure 5 visually represents the crucial need for the components that lead to high-quality representations. The first image (a) shows the ground truth, providing a reference for the scene's actual appearance and spatial relationships. Image 5(b) demonstrates the output of the complete NeRF model, which incorporates both view dependency and positional encoding. This image showcases the model's remarkable ability to synthesize accurate views of the scene. By considering these components, NeRF successfully captures intricate details, resulting in visually appealing reconstructions. Image 5(c) focuses on the impact of removing view dependency from the NeRF model. By neglecting variations in appearance caused by different viewpoints, the generated image may lack to reproduce the specular reflection. Lastly, Image 5(d) explores the implications of excluding positional encoding from the NeRF algorithm. Removing the positional encoding decreases the model's ability to represent high frequency geometry and texture, resulting in an oversmoothed appearance (Mildenhall et al., 2020).

2.2.2. Improvements of the NeRF algorithm

The NeRF approach has demonstrated its effectiveness in both synthetic data and real-world scenes. In synthetic settings, the consistency in viewing distance during training and testing makes it easier to achieve effective results. Moreover, NeRF outperforms other novel view synthesis algorithms such as Local Light Field Fusion

(LLFF) (Mildenhall et al., 2019) and Scene Representation Networks (SRN) (Sitzmann et al., 2020) when applied to real-world scenarios.

However, single-scene training alone requires a minimum of 12 hours, highlighting the computationally intensive nature of the process, which requires significant time and resources. In addition, the rendering process of a trained neural radiance field, which involves ray-marching through the scene, can be time consuming, posing a challenge for real-time or interactive rendering. Additionally, while NeRF excels in many areas, it still faces difficulties in accurately capturing the diversity and complexity inherent in real-world scenes, particularly concerning geometry, lighting, and unbounded conditions. To overcome these limitations, researchers have developed improvements to NeRF that aim to increase training efficiency, reduce rendering times, and enhance its ability to handle real-world scenes.

To provide a timeline of the improvements made to overcome these challenges, it's worth mentioning some key milestones. In 2020, NeRF was introduced, revolutionizing the field of novel view synthesis by using an MLP. Subsequently, the authors of NeRF released mip-NeRF in October 2021, which addresses the substantial limitations encountered when rendering images with varying resolutions or capturing scenes from different distances (Barron et al., 2021). NeRF also struggles to produce high-quality renderings for large unbounded scenes, where the camera can point in any direction and content can exist at any distance. In this context, conventional NeRF-like models often face challenges such as the generation of blurred or low-resolution renderings (resulting from the imbalance in detail and scale between near and far objects), slow training times, and potential artifacts arising from the inherent ambiguity of reconstructing a large scene from a limited set of images. To address these challenges, Barron et al. (2022) propose an extension of mip-NeRF that uses nonlinear scene parameterization, online distillation, and a novel distortion-based regularizer in June 2022. This model is called mip-NeRF 360 because it targets scenes in which the camera rotates 360 degrees around a point (Barron et al., 2022).

One notable drawback of NeRF lies in its performance challenges. Training the model on a single high-end GPU demands a substantial time frame of one to two days, making it a significant investment in terms of time. Furthermore, the process of generating a single image using the trained model through inference also consumes several minutes. Addressing this concern, Müller et al. proposed a promising innovation named instant-ngp in July 2022 (Müller et al., 2022). This advancement introduced a grid-based technique that not only accelerated training but also expedited the rendering process. With the implementation of this algorithm, the convergence of the NeRF model can now be observed within 20 seconds. The instant-ngp algorithm is used in this work to generate neural radiance fields and is therefore explained in more detail in section 2.2.3.

Further advancements were made in 2023 with the unveiling of Zip-NeRF, effectively addressing challenges related to aliasing, scaling, and training time, resulting in significant overall performance improvements. The authors show how ideas from rendering and signal processing can be used to construct a technique that combines mip-NeRF 360 and grid-based models such as instant-ngp to yield error rates that are 8%-77% lower than either prior technique, and that trains 24x faster than mip-NeRF360. The latest release on Siggraph in August 2023 came with MeRF and BakedSDF, which promise further developments to push the idea of NeRF forward.

For readers who wish to delve deeper, the appendix A includes detailed explanations of the algorithms Mip-NeRF, Mip-NeRF360, Zip-NeRF, MeRF and BakedSDF.

2.2.3. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding

An important element of the NeRF approach is, as described in section 2.2.1, the inclusion of positional encoding. While this technique is powerful for representing geometry, it introduces complexity into training and affects performance on Graphics Processing Unit (GPU)s where control flow and pointer chasing are costly (Müller et al., 2022). These challenges are addressed by the introduction of a new encoding method, proposed by Müller et al. in a paper titled "Instant Neural Graphics Primitives". The authors introduce a flexible approach for input encoding, enabling the utilization of a smaller network without compromising quality. This results in a reduction of floating-point operations and memory access, achieved by integrating a compact neural network with a multi-resolution hash table. The architecture benefits from the multi-resolution structure, allowing the network to effectively manage hash collisions and facilitating easy parallelization on modern GPUs. To maximize this parallelism, the entire system is implemented using fully-fused Compute Unified Device Architecture (CUDA) kernels, designed to minimize computational waste and memory bandwidth. The outcome is a remarkable speedup by several orders of magnitude, enabling rapid training of high-quality neural graphics components within seconds. An example of the rapid training dynamics is provided in Figure 6, where we showcase the training progression that converges within 20 seconds. The scene was captured from a 180-degree perspective, utilizing 145 frames for training, and trained using a Laptop GPU A3000.

Additionally, rendering tasks are expedited, taking mere tens of milliseconds even at a resolution of 1920×1080. This new encoding approach not only demonstrates efficiency improvements for the NeRF algorithm but also for other constructs known as "neural graphics primitives". These are mathematical functions that utilize neural networks to parameterize appearance. For instance, the Signed Distance Function (SDF) is used to represent shapes in $3D$ space by measuring the shortest distance to its surface, while Gigapixel Image Approximation aims to provide high-resolution

(a) Initial state at the beginning of training. (b) State after 20 seconds of training.

Figure 6.: Training process of a neural radiance field using instant-ngp. The object was captured from a 180-degree perspective, utilizing 145 images for training. In the training phase, rapid convergence is achieved in a mere 20 seconds, yielding a visually compelling scene. This performance was attained using a Laptop A 3000 GPU. This visualization underscores the remarkable speed of the training procedure. [For dynamic representation, see accompanying website videos.]

image reconstructions using neural network models, which learn the $2D$ to RGB mapping of image coordinates to colors (Müller et al., 2022).

The Multiresolution Grid Encoding

Given a fully connected neural network $m(y;\Phi)$, the focus lies in developing an encoding strategy for its inputs, denoted as $y = \gamma(x;\theta)$. This encoding scheme is designed to enhance the accuracy of approximations and expedite training across a diverse array of applications, all while ensuring that any resultant performance gains are achieved without introducing significant overhead. Within this neural network architecture, not only are the weight parameters Φ trainable, but also the encoding parameters θ. These parameters are organized into L levels, with each level accommodating up to T feature vectors, each with a dimensionality of F. To illustrate, Figure 7 outlines the sequential stages encompassed within the multiresolution hash encoding process.

Each level, illustrated as red and blue in this Figure, operates independently, functioning as a repository for feature vectors positioned at the vertices of a grid. The resolution of this grid is meticulously selected to align with a geometric progression that bridges the most coarsest and finest resolutions $[N_{\text{min}}, N_{\text{max}}]$. The resolution of each grid increases by a fixed factor relative to the previous level, although it's important to note that the growth factor is not limited to two. In actuality, the authors deduce the scaling based on the selection of three distinct hyperparameters:

$$L := \text{Number of Levels}$$
$$N_{\text{min}} = \text{Coarsest Resolution}$$
$$N_{\text{max}} = \text{Finest Resolution}$$

and the grow factor computed with

$$b := \exp\left(\frac{\ln N_{\text{max}} - \ln N_{\text{min}}}{L-1}\right).$$

Therefore, the resolution of grid level l is $N_l := \lfloor N_{\text{min}} \cdot b^l \rfloor$, where $\lfloor \cdot \rfloor$ denotes the floor function that rounds down to the next lowest integer. Following the establishment

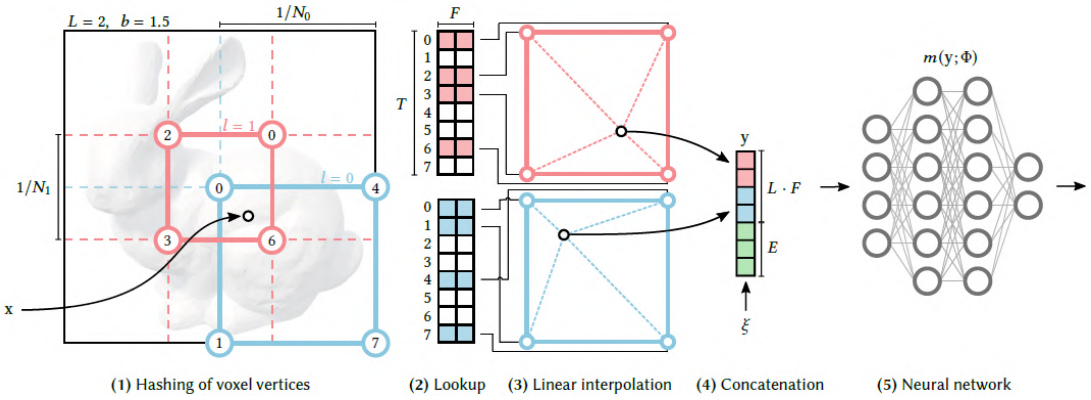


Figure 7.: Illustration of the multiresolution hash encoding in $2D$. (1) for a given input coordinate x, Müller et al. find the surrounding voxels at L resolution levels and assign indices to their corners by hashing their integer coordinates. (2) for all resulting corner indices, they look up the corresponding F-dimensional feature vectors from the hash tables θ_l and (3) linearly interpolate them according to the relative position of x within the respective l-th voxel. (4) the result of each level will be concatenated, as well as auxiliary inputs $\xi \in \mathbb{R}^E$, producing the encoded MLP input $y \in \mathbb{R}^{LF+E}$, which (5) is evaluated last. To train the encoding, loss gradients are backpropagated through the MLP (5), the concatenation (4), the linear interpolation (3), and then accumulated in the looked-up (2) feature vectors (Müller et al., 2022).

of the grid hierarchy, the next step involves defining the input transformation. For a given input coordinate x, the surrounding voxels are identified at L resolution levels. For each level l, the initial procedure entails the scaling of x by N_l and rounding to the nearest integer, thereby identifying the corresponding cell that envelops x. The indices to the corners of these voxels are then assigned using a hashing technique based on their integer coordinates. This process is illustrated in (1) in Figure 7 (Müller et al., 2022; Slater, 2022).

Next, for each resulting corner index, the corresponding F-dimensional feature vectors are retrieved from the hash tables θ_l depicted in (2) in Figure 7. Each corner coordinate is hashed with the employed function $h(x) = \bigoplus_{i=1}^{d} x_i \cdot \pi_i \bmod T$, where $\oplus$ denotes bit-wise exclusive-or and π_i are large prime numbers. To enhance cache coherence, the authors set $\pi_1 = 1$, along with $\pi_2 = 2654435761$ and $\pi_3 = 805459861$. To map the hash to a slot in each level's hash table, we simply modulo by T. The hash is then used as an index into a hash table. Each grid level has a corresponding hash table described by two more hyperparameters:

$T :=$ Hash Table Slot

$F :=$ Features per Slot

Each slot contains F learnable parameters. Throughout the training process, gradients are backpropagated through the entire network, extending even to the hash table entries. This dynamic optimization refines the entries to establish an effective input encoding strategy. The authors do not explicitly handle hash function collisions through traditional methods such as probing, bucketing, or chaining. Instead, they rely on the neural network to learn how to disambiguate hash collisions, thereby avoiding control flow divergence, simplifying implementation complexity, and improving performance. However, this adaptability presents a challenge when scaling down the encoding to very small hash table sizes—eventually, a large number of grid points are assigned to each slot (Müller et al., 2022; Slater, 2022).

The feature vectors F_i are then linearly interpolated based on the relative position of x within the respective l-th voxel to establish a resultant value at $\mathbf{x}$ itself (3). The authors note that interpolating the discrete values is necessary for optimization with gradient descent: it makes the encoding function continuous.

At this point, we have mapped x to L distinct vectors, each with a length of $\mathbf{F}$ - one vector for every grid level l. The results from each level are concatenated, along with auxiliary inputs (such as the encoded view direction and spatial location in neural radiance fields) $\xi \in \mathbb{R}^E$ (4) to produce $y \in \mathbb{R}^{\mathbb{L}\mathbb{F}+\mathbb{E}}$, which is the encoded input $\gamma(x;\theta)$ to the MLP $m(y;\Phi)$.

The encoded result y is then fed into a fully connected basic neural network with ReLU activations, similar to a multilayer perceptron. Remarkably, this network can be compact; for example, the authors use two hidden layers, each with a dimension of 64. During training, loss gradients are backpropagated through the MLP, the concatenation, and the linear interpolation. These gradients are then accumulated in the corresponding looked-up feature vectors, facilitating the training of the encoding process (Müller et al., 2022; Slater, 2022).

2.2.4. Importance of the NeRF algorithm

In recent times, NeRF models have attracted significant attention within the Computer Vision community. Numerous papers and preprints have surfaced on code aggregation platforms, and a substantial portion of these works have found their way into Computer Vision conferences. The original NeRF paper by Mildenhall et al. received more than 3000 citations and a growing interest with no signs of slowing down (Scholar Search, 2023). These research innovations have driven interests in a

wide variety of disciplines in both academia and industry.

Roboticists have ventured into leveraging NeRFs for a spectrum of applications including manipulation, motion planning, simulation, and mapping (Adamkiewicz et al., 2022; Byravan et al., 2022). Moreover, NeRF methods have extended their influence to tomography applications for reconstruction (Rückert et al., 2022), as well as the task of perceiving individuals within videos (Pavlakos et al., 2022). The potential of NeRFs has reverberated across domains such as visual effects and gaming studios, wherein the technology is being explored for production and the creation of digital assets. Even news outlets have embraced NeRFs to craft narratives using novel formats (Watson et al., 2022, September). The horizon of potential applications stretches extensively, and the surge of interest has even given rise to startups (Yu and Jain, 2023) dedicated to the deployment of this technology (Tancik et al., 2023).

We would like to point out some extension ideas that originated in the original NeRF algorithm, which have emerged, each addressing specific challenges and scenarios in Computer Vision. Some extensions focus on training NeRF models without known camera parameters (Wang et al., 2021; Meng et al., 2021; Yen-Chen et al., 2021; Heo et al., 2023), while others grapple with the challenges of managing unstructured sets of real-world photos (Martin-Brualla et al., 2021; Chen et al., 2022b; Jun-Seong et al., 2022). Further developments seek to tackle dynamic scenes characterized by intricate, non-static geometries (Pumarola et al., 2020; Tretschk et al., 2021; Weng et al., 2022; Li et al., 2023c). The realm of generative image synthesis has also benefited from NeRF-inspired extensions (Schwarz et al., 2021; Yu et al., 2020; Mildenhall et al., 2022). Lastly, there are extensions specifically tailored for novel view synthesis with LiDAR sensors (Tao et al., 2023; Zhang et al., 2023; Huang et al., 2023).

In conclusion, the NeRF algorithm has emerged as a cornerstone in the landscape of Computer Vision and beyond. Its ability to synthesize high-quality $3D$ reconstructions from $2D$ images has not only revolutionized our perception of scene representation but has also extended its influence across a multitude of disciplines. The algorithm's capacity to fuse data from various sources, including unconstrained photo collections and dynamic scenes with complex geometries, underscores its adaptability and potential. The importance of NeRF resonates deeply in academia, where it has inspired new paths of research and exploration. Its integration into Computer Vision conferences and sustained citation growth are testament to its impact and continued relevance. The industrial and practical applications of NeRF are equally remarkable. The technology's adoption in industries such as robotics, entertainment, simulation, and generative image synthesis speaks to its potential. As NeRF continues to inspire extensions and adaptations that address an ever-widening spectrum of challenges, its importance becomes even more apparent. This algorithm has opened up novel possibilities for how we interact with, interpret, and create visual content.

2.3. The Industrial Metaverse

As previously mentioned in the Introduction 1, NeRF offer extensive possibilities across a variety of domains. Within the context of the burgeoning Industrial Metaverse, NeRF holds particular promise. The algorithm's ability to generate high-fidelity, three-dimensional replicas of physical objects positions it as a critical asset for a variety of industrial scenarios. To fully appreciate the scope and utility of NeRF in this specific context, an understanding of the overarching concept of the Metaverse is essential.

The term 'Metaverse' traces its origins to the realms of science fiction, particularly a book by Stephenson. He didn't offer a rigid definition of the Metaverse, but he did describe a persistent virtual world that reached into, interacted with and influenced almost every aspect of human existence. It was a place of work and leisure, self-realization and physical exhaustion, art and commerce. At any one time, there were some 15 million human-controlled avatars on the Street, which Stephenson described as "the Broadway, the Champs Elysees of the Metaverse, but covering the entire surface of a virtual planet more than two and a half times the size of the Earth" (Ball, 2022).

The phenomenon of the Metaverse, given its recent emergence, faces a significant challenge in the absence of a universally accepted definition. This lack of consensus provides an opportunity to explore various attempts to define the Metaverse, highlighting its multifaceted nature and its potential for innovation. While different definitions continue to be proposed, there is a consensus in all ongoing debates that the Metaverse will be experienced in the form of a virtual world (Bitkom, 2022).

2.3.1. Defintions of the Metaverse

NVIDIA, a key player in graphics processing technology, describes the Metaverse as the internet's next evolutionary stage. Initially built on text-based interfaces, the internet has gradually integrated $2D$ images, multimedia, and social content. NVIDIA describes the Metaverse as a "$3D$ internet" - a network of persistent, interconnected virtual worlds that extends $2D$ web pages into $3D$ spaces (Caulfield, 2022). This allows for enhanced interactive online experiences that go beyond gaming and entertainment to include industrial and urban development projects.

NVIDIA has encapsulated this vision in "Omniverse," a platform introduced at the 2019 GTC conference, a global Artificial Intelligence (AI) conference for developer and IT experts, and later released as an open beta in 2020. Built on Pixar's open-source Universal Scene Description (USD) API and file framework, Omniverse enables real-time, seamless integration of various Metaverse elements. The use of USD allows for unparalleled real-time collaboration, ensuring all team members stay in sync within the Omniverse environment (NVIDIA, 2021; NVIDIA, 2023).

Another perspective comes from Bitkom, Germany's digital industry association. In this definition, the Metaverse should not be understood as a virtual parallel world with no connection to the real world, but rather as a multiplicity of connections in both directions. On the one hand, real world objects will find their virtual counterparts, effectively mirroring their existence. On the other hand, Extended Reality (XR) can be used to seamlessly integrate virtual objects into the tangible environment, enabling interactive experiences with these virtual entities (Bitkom, 2022).

> "The Metaverse is the next logical stage of the Internet: a real-time, three-dimensional Internet"(Bitkom, 2022).

The Metaverse demonstrates an increasing integration of virtual and real worlds. Extended Reality, which includes Augmented Reality, Mixed Reality and Virtual Reality technologies, serves as a gateway to the virtual world and plays a crucial role in the experience of the Metaverse. The immersive nature of XR technologies allows users to fully engage with the Metaverse. VR in particular offers total immersion, allowing users to experience the Metaverse in a fully virtual $3D$ environment. On the other hand, Augmented Reality (AR) and Mixed Reality (MR) broadcast virtual $3D$ objects into the real world, blurring the boundaries between the virtual and physical worlds.

An additional aspect of the Metaverse is the emergence of multidimensional user-generated content, particularly in form of $3D$ objects. These objects can be transferred, ported and traded within the context of the Metaverse, enabling dynamic interactions and experiences. Bitkom concludes that the Metaverse integrates aspects of XR with traditional Internet technologies, tailoring content according to user needs(Bitkom, 2022)

Matthew Ball, an authoritative writer on the metaverse, offers a nuanced framework for understanding it. He defines the Metaverse as a "massively scaled and interoperable network of real-time rendered 3D virtual worlds," emphasizing user synchrony and data persistence (Ball, 2022). This definition emphasizes the interoperable nature of the metaverse - composed of multiple interconnected virtual worlds - and its ability to provide real-time, persistent experiences. Synchrony ensures simultaneous user interaction, while persistence guarantees a stable and continuous digital realm. Data continuity, which includes elements such as identity and payment, enables seamless transitions between virtual worlds. Ball's definition thus informs scholarly discourse by highlighting the scale, interconnectedness, and complexity of the metaverse and encourages further exploration of this emergent digital domain (Ball, 2022).

Given the different perspectives and definitions of the Metaverse, it is clear that although the Metaverse is a multifaceted concept, there are certain key aspects that are

common to its various interpretations. For the purposes of this thesis, the Metaverse is defined as follows:

> The Metaverse is the next level of digitalization that connects (digital) ecosystems and extends the existing (mostly $2D$) internet with realistic $3D$ representations. It represents a persistent, shared, $3D$ virtual space that integrates real and virtual worlds, providing an interactive platform for a virtually unlimited number of users. It operates in real time, thereby enabling synchronous collaboration and communication. Drawing from Ball's definition, this virtual environment encompasses the continuity of data, such as individual identity, history, permissions, and objects, and supports various transactions including communication and payments.

This definition reflects the Metaverse's function as the next evolution of the Internet, a perspective acknowledged by Bitkom. Similar to NVIDIA's vision, it acknowledges the seamless integration of various elements of the real and virtual worlds and the creation of an immersive user experience. The Metaverse in this context enables high-fidelity representations of physical objects and processes, fostering enhanced collaboration and decision-making, particularly in industrial settings. Therefore, the Metaverse within the context of this thesis is a technologically advanced, interactive platform that is persistently evolving and blurring the boundaries between physical and virtual realities. This platform allows for both synchronous collaboration and individual exploration, with the potential to significantly alter the ways we interact, work, and innovate within the virtual industrial world.

Upon establishing a comprehensive understanding of the Metaverse, we can now delve into the concept of the Industrial Metaverse. Engaging the Chatbot "GPT-3" to define the Industrial Metaverse yields an insightful insight:

> The Industrial Metaverse is a digital simulation of the physical world, where physical and digital systems interact with each other. It utilizes digital twin technology to replicate physical systems, allowing for simulations and autonomous control of processes. The Industrial Metaverse allows businesses to create digital assets that generate expected income and value by connecting these isolated systems. It provides opportunities to unlock the potential of digital technologies and benefits all participants in the transformation (GPT-3, private conversation, 08-30-2023).

According to the definition provided by GPT-3, the Industrial Metaverse is characterized as a digital emulation of the physical world, where the interplay between physical and digital systems takes center stage. Facilitated by digital twin technology, this phenomenon replicates tangible systems digitally, resulting in the capacity for simulations and the autonomous control of processes. The Industrial Metaverse serves as a

platform for businesses to craft digital assets capable of generating anticipated value and income, achieved through the interconnection of previously disparate systems.

It's important to note, however, that the Industrial Metaverse is focused on specific industries, particularly the industrial sector. Siemens' delineation of the Industrial Metaverse underscores its purpose-driven nature, specifically designed to address real-world challenges and business needs. This specialized iteration recognizes the potential of the Metaverse to improve operational efficiency, optimize resource management and promote sustainable practices within the industrial landscape. Unlike the more general Metaverse, which spans a wide range of applications and domains, the Industrial Metaverse takes on a tailored role, addressing industry-specific challenges, streamlining industrial processes and facilitating data-driven decision making.Siemens further narrows down the concept, emphasizing its industry-specific applications (SIEMENS, 2023):

> While consumer metaverse applications are still evolving, the Industrial Metaverse focuses on purpose-driven use cases that address real-world challenges and business needs. The resource efficiencies enabled by Industrial Metaverse solutions can enhance business competitiveness and drive progress towards sustainability and resilience goals (SIEMENS, 2023).

By adopting Siemens' perspective, we highlight the Industrial Metaverse's specific role in utilizing digital technology to transform industries, emphasizing its focus on efficiency and sustainability.

2.3.2. The Current State of the Industrial Metaverse

As Industry 4.0 matures, integrating technologies such as the Internet of Things (IoT), AI, and cloud computing, manufacturing has undergone a profound transformation (IBM, 2023). These "smart factories" utilize data analytics for predictive maintenance and real-time decision-making, enhancing efficiency and responsiveness.

While Industry 4.0 mainly focuses on automation, the emerging concept of the Industrial Metaverse emphasizes human-centric experiences. It necessitates innovative solutions like $3D$ visualization to navigate its complex landscape (Soerensen, 2023). The Metaverse's growing prominence is underscored by new initiatives like the CUT (Connected Urban Twins) project and its new category in Time magazine's 'Best Innovation' awards (Reinecke et al., 2021; Time, 2022).

A significant stride towards materializing the Industrial Metaverse has been unveiled through a partnership between Siemens and NVIDIA. By synergizing Siemens' prowess in industrial automation, software, and infrastructure with NVIDIA's AI and accelerated graphics expertise, this collaboration aims to construct an Industrial Metaverse. The partnership's initial phases envision an amalgamation of Siemens Xceler-

ator and NVIDIA Omniverse, creating a domain where physics-based digital models from Siemens merge with real-time AI capabilities from NVIDIA. This unified platform empowers companies to make faster, more confident decisions, fostering an era of unparalleled industrial innovation (NVIDIA, 2022). Exemplifying the tangible impact of this trajectory, even revered car manufacturer Mercedes-Benz has embraced the digital-first ethos, leveraging NVIDIA Omniverse to revolutionize production strategies, marking a decisive step towards manufacturing within the Metaverse (Shapiro, 2023).

As Jensen Huang, CEO of NVIDIA, notes, "the metaverse will grow organically as the internet did — continuously and simultaneously across all industries, but exponentially, because of computing's compounding and network effects" (Caulfield, 2022). Thus, the Industrial Metaverse remains a fluid concept, subject to further evolution.

2.3.3. The Metaverse and it's opportunities for the industry

At the core of the Industrial Metaverse lies the innovative concept of "digital twins". These are virtual equivalents of real-world entities, ranging from individual objects and humans to intricate processes and complete production lines. These digital twins have a wide array of applications, including but not limited to, prototyping, testing, training, and simulation, thus allowing for a more sophisticated control over industrial operations (Stackpole, 2023, August).

The Industrial Metaverse is garnering significant attention for several compelling reasons. One major advantage is its facilitation of rapid problem-solving, coupled with the prospect of recombinant innovation. Digital twins of physical assets simplify the iterative process, negating the need for expensive prototypes and minimizing disruption to ongoing operations (Stackpole, 2023, August). As Marshall Van Alstyne, a professor at Boston University and visiting scholar at the MIT Initiative on the Digital Economy, points out, "the ability to reconfigure systems of components and their corresponding digital twins into novel combinations provides a fertile ground for innovation" (Stackpole, 2023, August).

Enhanced Design and Engineering The metaverse enables cross-functional teams from various locations to collaborate and innovate without the constraints of geographical distance or the overheads of physical prototyping. The synergy of photorealistic virtual environments with varied simulations augments the scope for testing and validating product designs or facility layouts, thereby fueling innovation (Stackpole, 2023, August).

Virtual Commissioning and Facility Planning Leveraging immersive digital twins, manufacturers can conceptualize, test, and commission their production floors

within the metaverse, thereby ensuring optimized, resilient operations. This approach enables the identification and rectification of errors without affecting current production cycles or incurring undue investment risks. Siemens, for example, employed digital twin technologies for the planning and simulation of a 73,000-square-meter factory in Nanjing, China, using a blend of factory, production line, and performance data to validate the facility prior to its construction (Stackpole, 2023, August).

Operational Enhancement Manufacturers can capitalize on simulations and real-time data collection within the metaverse to gain actionable insights. These insights can be employed to optimize machinery, reduce downtime, and anticipate and avert equipment failures (Stackpole, 2023, August).

Workforce Upskilling The metaverse offers remote access to specialized skills and training, making geographical location a non-issue. Employees can undergo comprehensive, practical training on intricate machinery without interrupting standard operations.

KLM, a national airline in the netherlands, serves as a case in point, pioneering the use of VR in pilot training. As Sebastian Gerkens, Senior Instructor Embraer at *KLM*, articulates, "Virtual Reality makes training more accessible. It allows for on-demand, location-independent training sessions, freeing pilots from the constraints of traditional classrooms or simulators. It encourages exploration within a safe, virtual environment". This innovative approach to training yields financial efficiencies as well. By reducing reliance on external suppliers and offering more flexible scheduling options for pilots, *KLM* has identified a cost-effective strategy that could serve as a model for other industries (KLM, 2020).

Sales and Customer Engagement What impact does the Industrial Metaverse have on sales in sectors primarily focused on hardware? The key change can be characterized as a pivot from a product-centric to an application-centric approach. While hardware companies often have deep product expertise, the true potential lies in understanding the actual applications that these products serve. This invaluable application-centric knowledge is typically held by customers and sales teams who understand the various functions and roles these products perform. The ultimate value of the product is realized when it seamlessly aligns with the customer's specific needs. An inherent challenge is that while customers excel at specifying their application needs, they may fall short in pinpointing the exact product solutions to meet those needs (Soerensen, 2023).

The Industrial Metaverse eloquently bridges this gap by providing a user-centric ecosystem optimized for both sales dialogues and consultative customer interactions.

Within this virtual realm, users can express their challenges while visualizing their industrial setups. This virtual representation helps identify and fine-tune potential solutions, allowing for rigorous testing within the metaverse before deployment in the real world. Rather than navigating a traditional, product-focused maze, the Metaverse unveils a dynamic arena where users can co-create customized solutions. This new environment facilitates immediate visual adjustments; for example, changing the specifications of a conveyor belt is no longer limited to static blueprints. The immersive nature of the metaverse enriches the demonstration of solutions. Using AR technologies, a proposed solution can be superimposed within the customer's actual workspace, enabling real-time compatibility assessments and adjustments for unanticipated spatial constraints (Soerensen, 2023).

In this metaverse-driven shift, hardware-based companies are on the threshold of redefining their approach to the industry. By adopting an user-centric strategy and leveraging immersive technologies, they can increase customer engagement, streamline project workflows, and provide instant insight into the effectiveness of their solutions, ushering in a new era of innovation in the industrial landscape (Soerensen, 2023).

The transformative potential of the Industrial Metaverse is undeniable and spans multiple operational areas, including prototyping, testing, training, simulation, collaboration, sales, and customer engagement. Utilizing digital twins and immersive technologies such as VR and AR, offers unparalleled efficiency, innovation, and customer focus. It enables real-time, collaborative problem-solving, filling gaps that have historically hampered performance. The scenarios discussed not only illustrate the expansive possibilities, but also point to a future in which the boundaries between the physical and digital worlds merge. The Industrial Metaverse stands as a pivotal innovation, setting the stage for technological integration and industry-wide advancements.

3. Related Work

In the context of NeRF research, a variety of frameworks have emerged, each with the aim of facilitating the development and the appicability of NeRF methods in different domains. In this chapter we delve into a selection of notable frameworks that contribute to the ongoing development and enhancement of NeRF. Our primary objective is to gain a comprehensive understanding of the functionalities and offerings provided by these frameworks. By thoroughly testing and interacting, we uncover both the progress and challenges within the domain of generating neural radiance fields, particularly when dealing with user-generated data.

Unlike frameworks that re-implemented original NeRF methods, such as NeRF-pytorch (Yen-Chen, 2020) - transitioning the original NeRF algorithm from the Tensorflow codebase to PyTorch - or HashNeRF-Pytorch (Bhalgat, 2022), a dedicated PyTorch-based implementation of the instant-ngp algorithm, we shift our focus to frameworks that aim to achive different goals.

We are exploring frameworks that seek to integrate different NeRF methods into a modular framework to make NeRF methods accessible to a wider audience, including scientists, artists, photographers, hobbyists and journalists. We will also look at a framework designed specifically for researchers to make it easy for them to develop their own NeRF methods. Additionally, our research includes frameworks that have successfully re-implemented the instant-ngp algorithm, aiming for a user-friendly design that not only removes the dependence on a graphics card, but also does not require deep technical knowledge. Lastly, we discuss LumaAI, a start-up app that offers the unique capability to generate neural radiance fields without any coding skills, using just an iPhone equipped with a LiDAR scanner. We emphasize that, to date, no frameworks are explicitly designed for industrial applications, leading us to our own implementation. We further underscore this aspect in the final discussion.

3.1. Overview over Frameworks offering several NeRF methods

NeRF is a rapidly growing field of research with diverse applications in computer vision, graphics, robotics and many other areas as elaborated in Chapter 2.2.2. Recognizing its importance, developers are building frameworks for NeRF methods to make them easier to use. These frameworks eliminate the need to set up each NeRF algorithm separately, making research more efficient and user-friendly. An outstand-

ing feature of these frameworks is their modular structure, which allows researchers and developers to easily implement different NeRF methods and adapt them to their specific needs. The modular design means that new methods can be easily integrated into existing frameworks without having to rewrite encodings, network architectures or ray marching functions. This is particularly useful as the field of NeRF is constantly evolving, with new methods and approaches emerging regularly.

Of particular interest for deeper exploration is *Nerfstudio*, which currently holds the distinction of being the largest and most well-known framework in this realm. Before taking a closer look at *Nerfstudio*, we would like to briefly discuss other frameworks that have dealt with the unification of different NeRF methods.

ArcNeRF is developed by Tencent's Applied Research Center Lab and focuses on providing a modular framework for integrating various NeRF methods. It uniquely offers a pronounced modular design, enabling seamless component changes and adjustments through configuration files. It also features a model subdivided into foreground and background, each tailored to specific geometric constraints, making it suitable for everyday video scenes. The training progress can be monitored in real-time through the *Nerfstudio* web viewer, although the interactive elements remain unimplemented. However, the framework's complex installation process and the steep learning curve for training on user-generated data can be challenging (Yue Luo, 2022)..

JNeRF presents a benchmarking framework based on Jittor, a high-performance deep learning framework with Python as front-end and CUDA/C++ as back-end. It supports a range of operations, including training models from scratch in predefined scenes like the Lego scenario, and testing pre-trained models stored in `pkl` format. Users can also render demonstration videos by specifying a particular camera path or extract meshes with color information. When using custom datasets, the preparation involves creating corresponding JSON files containing essential camera parameters. Although JNeRF is feature-rich, it lacks a graphical user interface and explicit data pre-processing instructions, making it less suitable for users seeking an all-in-one solution (Hu et al., 2020).

XRNeRF Part of the OpenXRLab project, XRNeRF is an open-source, PyTorch-based framework focused on NeRF research. It offers a comprehensive toolbox for a variety of NeRF methods, including scene-specific methods like NeRF (Mildenhall et al., 2020), Mip-NeRF (Barron et al., 2021), KiloNeRF (Reiser et al., 2021), and instant-ngp (Müller et al., 2022), among others. Additionally, *XRNeRF* caters to human-NeRF methods like NeuralBody (Peng et al., 2021b) and AniNeRF (Peng et al., 2021a), The framework's modular design organizes components into 'network,'

'embedder,' and 'renderer,' allowing for easy customization and integration. XRNeRF also supports both training and testing modes, adapting maximum iterations accordingly. The versatility of *XRNeRF* is evident in its ability to train models from scratch on specific scenes, such as the Lego scenario, or test pre-trained models, all while accommodating data selection through command-line arguments. While the framework provides ample functionalities, it does not offer explicit guidance on data pre-processing or incorporate a graphical user interface. In summary, *XRNeRF* is a versatile tool that provides researchers with a comprehensive toolkit for NeRF research and includes a wide range of functionalities, model components and scene implementations (Contributors, 2022).

NeRF-Factory is proving to be a robust framework in the field of NeRF, providing PyTorch re-implementations of several established variants, including NeRF (Mildenhall et al., 2020), NeRF++ (Zhang et al., 2020), DVGO (Sun et al., 2022), Plenoxels (Fridovich-Keil and Yu et al., 2022), Mip-NeRF 360 (Barron et al., 2022), and Ref-NeRF (Verbin et al., 2022). It also encapsulates seven of the popular NeRF datasets. While its portfolio is extensive, it is pertinent to highlight the absence of the specific instant-ngp implementation that is central to this thesis. The design ethos behind NeRF-Factory emphasises modularity, ensuring a seamless integration experience for its users. At its core, *NeRF-Factory* is optimized to facilitate training of neural radiance fields on pre-established datasets. This positions it as an invaluable asset for researchers and practitioners keen on exploring NeRF advancements, enabling a comparative analysis of diverse algorithms (Yoonwoo et al., 2022).

NerfAcc emerges as a PyTorch toolbox tailored for accelerating both training and inference processes within the context of NeRF. The toolbox predominantly centers on optimizing the sampling procedure within the volumetric rendering pipeline of radiance fields - a foundational aspect applicable and seamlessly integrable across a multitude of NeRF implementations. Notably, *NerfAcc* stands as a universal and plug-and-play solution compatible with most NeRF methodologies. This toolbox brings forth notable acceleration benefits with minimal requisite modifications to existing codebases, effectively expediting the training of diverse NeRF models presented in recent research papers. With a pure Python interface, *NerfAcc* further offers the advantage of flexible APIs, enhancing usability and adaptability (Li et al., 2023a).

Kaolin Wisp NVIDIA *Kaolin Wisp* was designed as a dynamic, research-focused library for neural fields to help researchers address the challenges of this discipline. The framework offers modular components for building complex neural fields, including datasets, image I/O, and mesh utilities. The library consists of modular building blocks

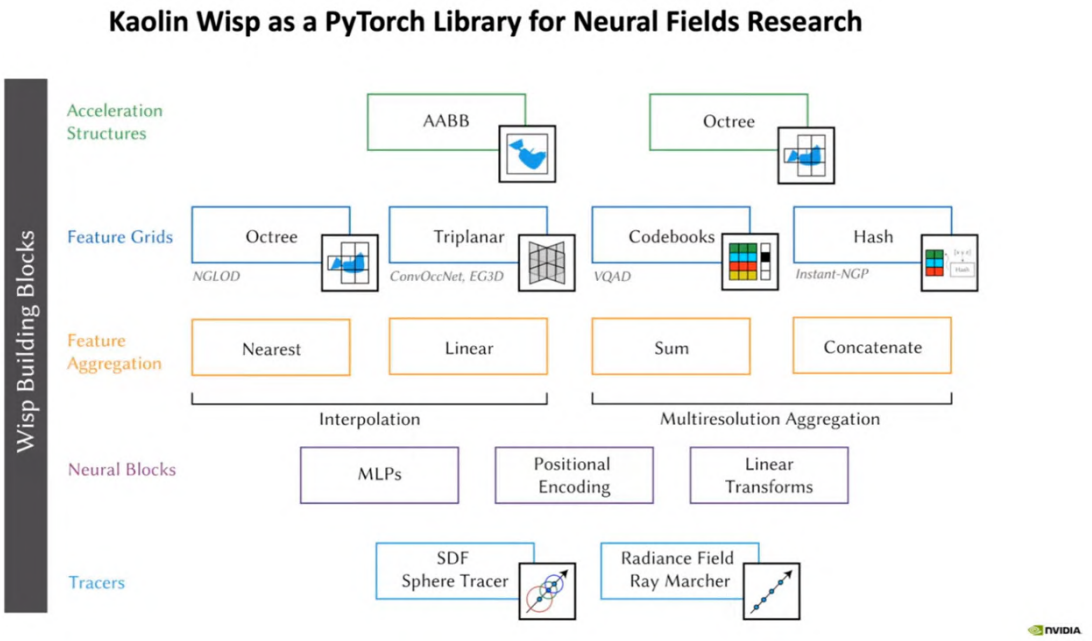


Figure 8.: NVIDIA Kaolin Wisp architecture and building blocks. Each pipeline component can be easily swapped out to provide a plug-and-play configuration for standard training (Takikawa et al., 2022)

that facilitate the construction of complex neural fields, and an interactive application for training and visualizing these neural fields. This framework is easily extensible for research purposes and features a modular pipeline where each component can be easily swapped out to provide a plug-and-play configuration for standard training. Its array of features encompasses datasets, image input/output, mesh manipulation, and ray-related utilities. Notably, Wisp also offers fundamental components like differentiable renderers and data structures (e.g., octrees, hash grids, triplanar features) that prove instrumental in constructing intricate neural fields. *Kaolin Wisp* is paired with an interactive renderer that supports flexible rendering of neural primitives pipelines like NeRF. Additionally, it incorporates debugging visualization tools, interactive rendering and training capabilities, logging functionalities, as well as trainer classes. Figure 8 provides an overview about the building blocks from Kaolin. The goal of *Kaolin Wisp* is to address limitations by offering modular building blocks that can be flexibly combined, akin to constructing with LEGO bricks. These Python-based modules enhance the accessibility of neural field research, allowing researchers to easily mix and match components according to their specific requirements (Takikawa et al., 2022). NVIDIA *Kaolin Wisp* offers an extensive range of utility functions for neural field research. However, it may not align perfectly with the specific objectives of this thesis project. While *Kaolin Wisp* does provide valuable tools for researchers exploring various aspects of neural fields, it may not directly address the unique challenges and requirements highlighted in this thesis (Takikawa et al., 2022).

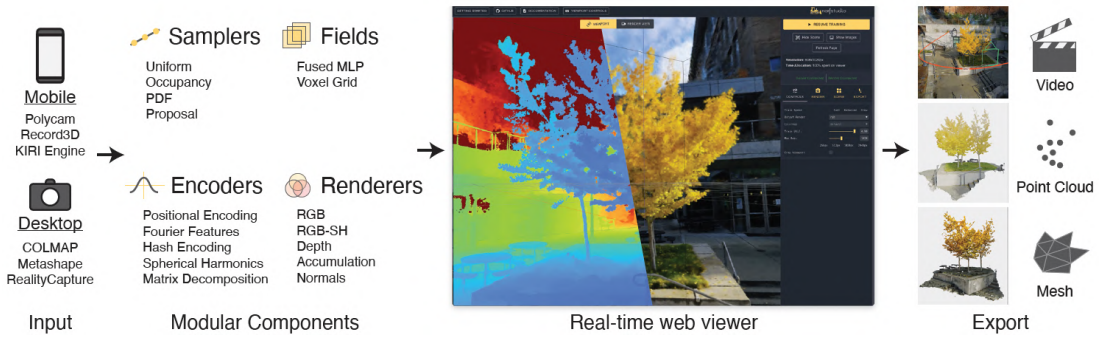


Figure 9.: *Nerfstudio* Framework Overview: *Nerfstudio* is a Python framework designed specifically for developing NeRF applications. It offers a range of features including support for various input data pipelines, a modular architecture based on core NeRF components, integration with a real-time web viewer, and versatile export capabilities. The primary objective of *Nerfstudio* is to streamline the development process of customized NeRF methods, facilitate real-world data processing, and enable seamless interaction with reconstructions (Tancik et al., 2023).

3.2. Nerfstudio - A Modular Framework for NeRF Development

Nerfstudio serves as a flexible and comprehensive framework for NeRF development. Its goal is to streamline various NeRF techniques into reusable, modular components, providing real-time visualization of neural radiance field scenes. With a set of user-friendly controls, *Nerfstudio* facilitates an end-to-end workflow for generating neural radiance fields from popular NeRF datasets and user-generated data. The design of the library simplifies the implementation of NeRF by dividing each element into modular units, enhancing interpretability and user experience. We will not delve into detailed explanations of these components in this chapter; interested readers are referred to the original paper for a more thorough understanding (Tancik et al., 2023). Figure 9 provides an overview of the pipeline process and the diverse possibilities within the *Nerfstudio* framework.

Nerfstudio is a highly interesting platform for the generation of neural radiance fields for researchers, developers, artists and more due to its four key components.

Firstly, *Nerfstudio's* user experience mirrors a plug-and-play model, allowing users to effortlessly incorporate NeRF into their projects. The development team is dedicated to providing extensive educational resources for all levels of NeRF expertise. Recognizing potential challenges for beginners and experienced users alike, *Nerfstudio* offers a wealth of tutorials, comprehensive documentation, and additional support materials. Secondly, *Nerfstudio* supports images and videos from different camera types and mobile capture applications such as *Polycam*, *Record3D*, and *KIRI Engine*. It also accepts outputs from popular photogrammetry software like *RealityCapture* and *Metashape*. This integration eliminates the need for time-consuming structure-from-motion tools such as COLMAP. Thirdly, *Nerfstudio's* real-time viewer, as described

by the authors, has been inspired by the real-time rendering during training as presented in instant-ngp (Müller et al., 2022). To address the difficulties associated with deploying local computing resources in remote settings, the authors developed a publicly accessible website that utilizes a ReactJS-based web viewer. The viewer is versatile, catering to users employing both local and remote GPUs. To simplify the use of remote compute, users only need to forward a port locally via SSH. Once training begins, the web interface provides real-time rendering of the neural radiance field, facilitating user interactions such as panning, zooming, and rotating around the scene during the optimization process and for the converged model. The viewer leverages ThreeJS for the implementation of camera controls and UI, allowing for the overlay of $3D$ assets like images, splines, and cropping boxes on the NeRF renderings. This enables users to intuitively compare the performance from different viewpoints. To maintain a steady frame rate and prevent lags in the user experience during fast camera movements, rendering resolution is adjusted, a strategy employed by Instant NGP (Müller et al., 2022). This approach also optimizes resource allocation, diverting more towards rendering in the viewer and reducing training time. The viewer is endowed with several features, including switching between different model outputs (such as RGB, depth, normals, semantics), creating custom camera paths with position and focal length interpolation, $3D$ visualization of captured training images, and crop and export options for point clouds and meshes. Navigation within the scene is facilitated through mouse and keyboard controls. The user interface of *Nerfstudio's* web viewer is depicted in Figure 10 (Tancik et al., 2023).

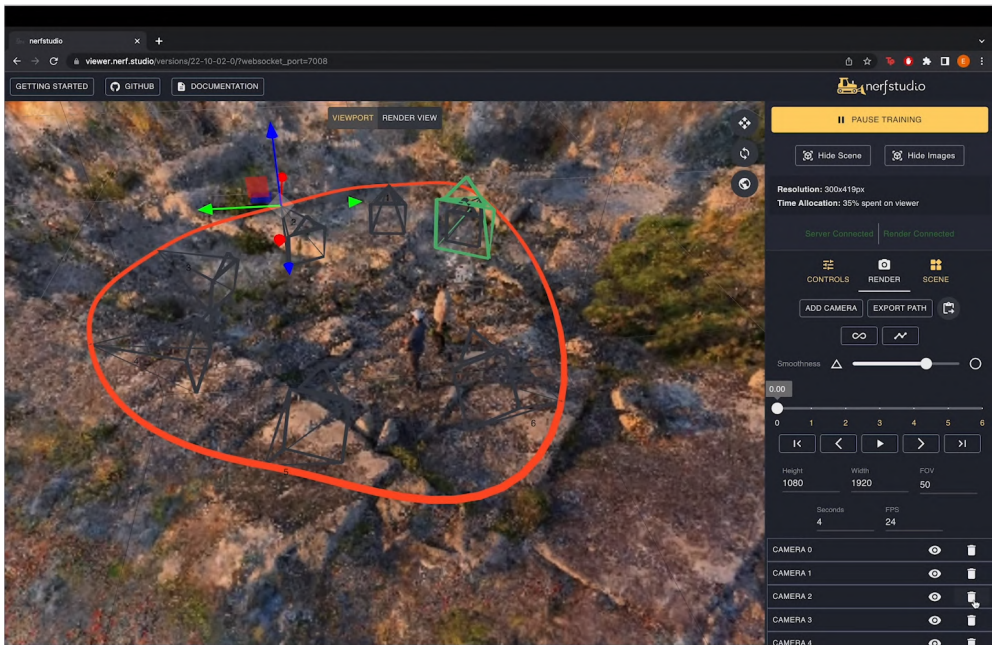


Figure 10.: The web viewer interface of *Nerfstudio*, providing real-time rendering and interactive exploration of the scene during training or model evaluation (Tancik et al., 2023).

Finally, *Nerfstudio* enhances its functionality by supporting various export methods, and the framework allows for easy incorporation of new export techniques as needed. The export interface, along with some of the supported formats, is demonstrated in Figure 11. These formats include point clouds, a truncated signed distance function (TSDF) to generate a mesh, and poisson surface reconstruction (Kazhdan et al., 2006). The texturing process of the mesh involves dense sampling of the texture image, using barycentric interpolation for identifying corresponding $3D$ point locations, and rendering short rays near the surface along the normals to obtain RGB values.

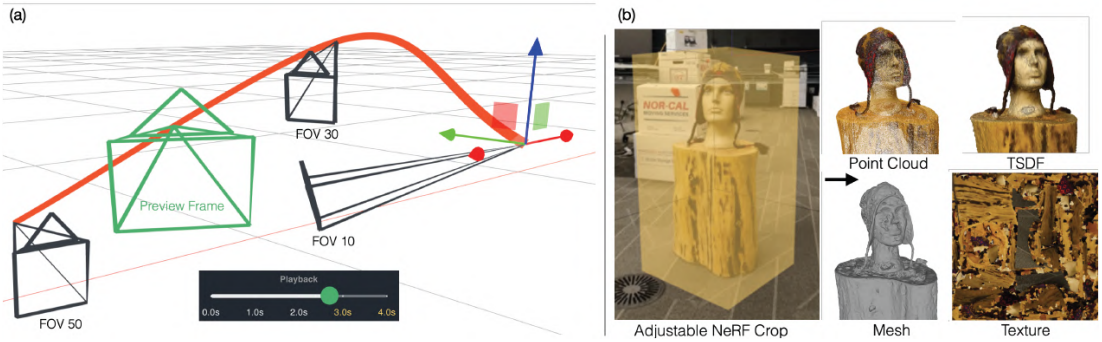


Figure 11.: Exporting videos and geometry with *Nerfstudio*. The interactive camera trajectory editor on the left facilitates the animation of poses, FOVs, and speed for rendering videos of NeRF's outputs. The cropping interface in the viewer and the resulting export formats, such as point clouds, TSDFs, and textured meshes, are presented on the right (Tancik et al., 2023).

Getting started with *Nerfstudio* can be a challenge. Although the framework is very powerful, there are barriers to installing the framework that prevent potential users, especially those with limited technical knowledge, from using this framework.

Firstly, the installation process is far from being user-friendly. It is not as simple as downloading an executable file, but requires the use of Python, CUDA and PyTorch. These three installations are interdependent and their complex installation challenges are discussed in Chapter 4.2.3 and C. Not only does this affect the initial setup, but the entire installation process requires a significant amount of technical knowledge and coding skills that may be impossible for a layperson to master. Consequently, this complexity may limit the attractiveness of the framework and restrict its user base to those with the technical skills to manage the process.

Secondly, *Nerfstudio* is built on a command line interface, users cannot simply upload their data via a web viewer and specify the type of training, but have to execute the commands via a command line interface. The process starts with the execution of the command `ns-process-data {video,images,polycam,record3d} --data {DATA_PATH} --output-dir {PROCESSED_DATA_DIR}`, which processes the input data types such as videos, images, Record3D, etc. from the user, to produce the training input format. Subsequently, the command `ns-train {options}` is then used to start training a NeRF

Figure 12.: The figure illustrates the range of options available for training the instant-ngp algorithm in *Nerfstudio*, obtained via the `ns-train instant-ngp --help` command. The multitude of options highlights the complexity and flexibility inherent to the *Nerfstudio* platform.

algorithm, for example `ns-train instant-ngp {options}`. The option `ns-train --help` reveals additional options, some are shown in Figure 12. This shows the complexity and versatility of the training options.

Finally, the training process offers user-friendly features. One of them is the possibility to monitor the training progress via a web viewer. This visualization can be easily interpreted even by non-technical users. It provides an engaging way to observe how the neural radiance fields are created and refined. In addition, after completing the training, users have the option of creating a camera path in the web viewer that can be used for possible video rendering. However, the subsequent stages in the workflow — namely rendering the video or exporting a mesh — necessitate a return to the command line interface. Nonetheless, these commands can be pre-defined and entered through the web viewer, providing a degree of ease. The following code snippet illustrates the sequence of steps in which a point cloud (line 1), a mesh (line 2) and the rendering of a predefined path (line 3) are executed.

```
1 ns-export pointcloud --load-config instant-ngp/2023-07-17_120630/config.yml --num-
    points 1000000 --remove-outliers True --normal-method open3d --use-bounding-box
    True --bounding-box-min -1 -1 -1 --bounding-box-max 1 1 1
2 ns-export poisson --load-config instant-ngp/2023-07-17_120630/config.yml --output-dir
     exports/mesh/ --target-num-faces 50000 --num-pixels-per-side 2048 --normal-
    method open3d --num-points 1000000 --remove-outliers True --use-bounding-box True
     --bounding-box-min -1 -1 -1 --bounding-box-max 1 1 1
3 ns-render camera-path --load-config instant-ngp/2023-07-17_120630/config.yml --camera
    -path-filename ./camera_paths/2023-07-17_120630.json --output-path renders
```

```
/./2023-07-17_120630.mp4
```

Code Listing 1: Executing export commands in *Nerfstudio*

In conclusion, *Nerfstudio*, with its comprehensive capabilities, stands out as an essential resource for those committed to keeping pace with advances in NeRF technology. Its extensive features, while demonstrating its power and versatility, can be overwhelming, especially for users who are not well versed in the field. The complexity of its setup, combined with a command line interface that can be daunting for some, highlights a clear gap in user accessibility. Errors and problems occur during installation and execution that users have to fix themselves, which is a daunting task for non-programmers. This not only affects the user experience, but also adds a layer of complexity. While it offers immense potential for experienced researchers and enthusiasts, there's an obvious need for a more intuitive, straightforward alternative for a wider audience. This gap motivates the exploration and development of more a user-centric framework in the field of NeRF technology.

3.3. Torch-NGP - A Pytorch CUDA Extension

In our search for frameworks that could be used to generate neural radiance fields from user-generated data, we also took the approach of looking for alternative implementations that do not rely on specific graphics cards. While this approach may result in slower training and rendering processes, it is consistent with the goal of making the algorithm accessible to a wider audience, including users who do not have high-end graphics cards.

Torch-NGP offers various PyTorch re-implementations of NeRF technologies. Included in its portfolio are PyTorch realisations of the Signed Distance Function (SDF) and NeRF components found in instant-ngp (Müller et al., 2022), TensoRF (Tensorial Radiance Fields, Chen et al., 2022a), CCNeRF (Compressible-composable NeRF via Rank Residual Decomposition, Tang et al., 2022a), and D-NeRF (Neural Radiance Fields for Dynamic Scenes, Pumarola et al., 2020). The framework thus offers not only the algorithm instant-ngp, but also the possibility to prepare self-generated datasets for training input (Tang, 2022; Tang et al., 2022b) and a simple graphical user interface for training/visualizing neural radiance fields (Tang, 2022).

To generate training input from user-generated data, it is imperative to process the data into an appropriate format. This entails creating a JSON file that encapsulates essential camera parameters alongside their corresponding images. In order to effectively prepare and train with self-generated data, the user has to follow these steps:

- Data Acquisition: The procedure starts by capturing visual data, which can

take the form of a video or a series of images. The objective is to capture the target scene from a variety of perspectives.

- Data Organization: The captured data should be subsequently organized within a designated repository. For videos, storage occurs in the directory `./data/custom/video.mp4`, while images are stored in the directory `./data/custom/images/` as `*.jpg` files.

- Preprocessing Execution: The execution of a provided script is necessary to facilitate the transformation of the acquired data. Successful execution of the script necessitates the prior installation of both FFMPEG and COLMAP. Upon the execution of the preprocessing script, a `transform.json` file is generated, which encapsulates essential information extracted from the COLMAP process and serves as the fundamental input for subsequent stages of training.

The code snippet 2 shows the commands that need to be executed to get from the user input to the training input.

```
# For video data:
python scripts/colmap2nerf.py --video ./path/to/video.mp4 --run_colmap
# For image data:
python scripts/colmap2nerf.py --images ./path/to/images/ --run_colmap
# For dynamic scenes (suitable for D-NeRF settings):
python scripts/colmap2nerf.py --video ./path/to/video.mp4 --run_colmap --dynamic
```

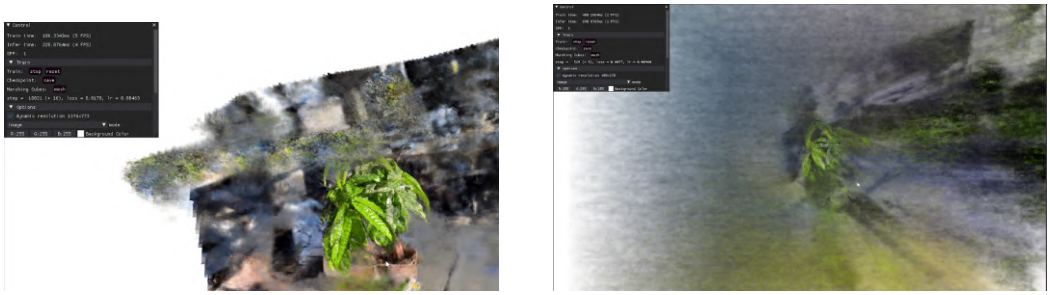
Code Listing 2: Prepare user-generated data in Torch-NGP

This comprehensive methodology ensures the seamless conversion of user-generated data into a suitable format and thus forms the basis for the subsequent training.

Torch-NGP provides flexibility in choosing the backbone, when training a neural radiance field with the instant-ngp algorithm. The following command-line instructions showcase the different backbone options available. These range from the traditional PyTorch-based Ray Marching method to options tailored for the default COLMAP dataset settings (like –bound 2 and –scale 0.33) and even those leveraging CUDA as the backend (Tang, 2022).

```
# Train with different backbones
python main_nerf.py data/fox --workspace trial_nerf # Standard mode with 32-bit
  floating point precision (fp32)
python main_nerf.py data/fox --workspace trial_nerf --fp16 # Enhanced speed with
  16-bit floating point precision (fp16) using PyTorch's automatic mixed precision
  (AMP)
python main_nerf.py data/fox --workspace trial_nerf --fp16 --ff # Incorporating
  FFMLP in 16-bit floating point precision (fp16) mode
python main_nerf.py data/fox --workspace trial_nerf --fp16 --tcnn # Utilizing the
  official tinycudann's encoder & MLP in 16-bit floating point precision (fp16)
  mode
```

Code Listing 3: Commands in Torch-NGP to train a neural radiance field with the instant-ngp algorithm



(a) Training progression after 5 minutes with CUDA support

(b) Training progression after 5 minutes without CUDA support

Figure 13.: A visual representation of neural radiance field training using the *torch-ngp* framework. The training data contains 288 images, captured over a 180-degree field of view. The left side 13a details results after 5 minutes of training with CUDA back-end, while the right side 13b does the same for a PyTorch-only session. [For dynamic representation, see accompanying website videos.]

Such a diverse set of command options enables users to fine-tune their training strategies to their specific needs, optimizing the performance of the NeRF models they work with. Figure 13 displays two training results after 5 minutes of training with different backbones. The figure's left side illustrates the progress with a CUDA-backed. In contrast, the right side showcases results without CUDA. A key takeaway from this visual comparison is the substantial speed reduction when opting for PyTorch over CUDA. Even with CUDA's capabilities, the results remain a step behind what the original instant-ngp framework can be seen in 6. Highlighting its user-centric design, *Torch-NGP* includes an intuitive graphical user interface that simplifies the process of training neural radiance fields. By simply incorporating the `--gui` flag, users gain effortless access to the GUI, allowing them to start, stop, and save training sessions with ease. Additionally, the process of mesh exporting is optimized through the integration of the Marching Cubes algorithm (Tang, 2022).

In conclusion, *Torch-NGP* is an excellent framework for users looking for an easy way to generate neural radiance field. However, it is important to recognize that the training process within *Torch-NGP* is significantly slower compared to the original instant-ngp algorithm. Consequently, opting for the original instant-ngp algorithm is the more logical choice in terms of training efficiency (Tang, 2022).

3.4. Luma AI

Launched in mid-2022, *Luma AI* marked its entry into the NeRF domain with the introduction of a private beta. Today, this pioneering application is available on the App Store for users with an iPhone 11 or newer. *Luma AI* boasts a unique feature set that empowers users to craft high-fidelity photo-realistic $3D$ assets and expansive environments swiftly using the NeRF algorithm as basis. Their underlying motivation

is eloquently captured in their observation:

> We live in a three dimensional world. Our perception and memories are of three dimensional beings. Yet there is no way to capture and revisit those moments like you are there. All we have to show for it are these 2D slices we call photos. It's time for that to change (Yu and Jain, 2023).

Luma AI's vision is not merely to record memories in $3D$ but to revolutionize how products are presented, memories are relayed, and spaces are explored digitally. With the goal of transcending the limitations of traditional $2D$ media, *Luma* aims to establish a vibrant $3D$ mixed reality. Their belief is that with the latest breakthroughs in neural rendering, deep learning, and computational capabilities, achieving photorealistic $3D$ capture is now within reach (Yu and Jain, 2023).

When starting the *Luma AI* app, users will find a user-friendly interface. They have the option of capturing either entire scenes or specific objects. When choosing to capture an object, the app seamlessly guides the user through a systematic process. Figure 14 describes this guiding. It starts with the user delimiting the target object, whereupon an adjustable bounding box appears around the object. This bounding box not only shows the boundaries, but also helps optimize the camera movement. The app then provides explicit guidance on camera movement with a real-time feedback system that highlights the captured camera angles, ensuring thorough data collection while minimizing redundancies. The user circles the selected object three times at different heights using the guide to ensure comprehensive data collection, which serves for optimal NeRF training. After data acquisition, the application moves to the crucial phase of training the neural radiation field. The final image in Figure 14 represents this phase and shows the training progress.

Figure 15 displays the resulting high-quality, $3D$ neural radiance field representation of the captured object.

A noteworthy aspect, already mentioned in the introduction, is the competence of NeRF in dealing with transparency. As can be seen in Figure 15 (e), Santa Claus remains well defined even without the surrounding glass, indicating an effective representation of both the glass and the figure within it. Furthermore, illustration (c) shows the clear influence of view dependence on glass rendering.

LUMA AI facilitates a variety of export options for the trained radiance field. Users can export scenes as point clouds in PLY files, as objects with varying polygonal details as OBJ files, or even utilize an integration plugin for Unreal Engine, underscoring its potential for gaming and simulations. An additional feature lets users chart camera paths to generate videos.

However, while these features seem promising, they come with their set of challenges in *LUMA AI*. Exporting as a point cloud, for instance, can yield a scene with

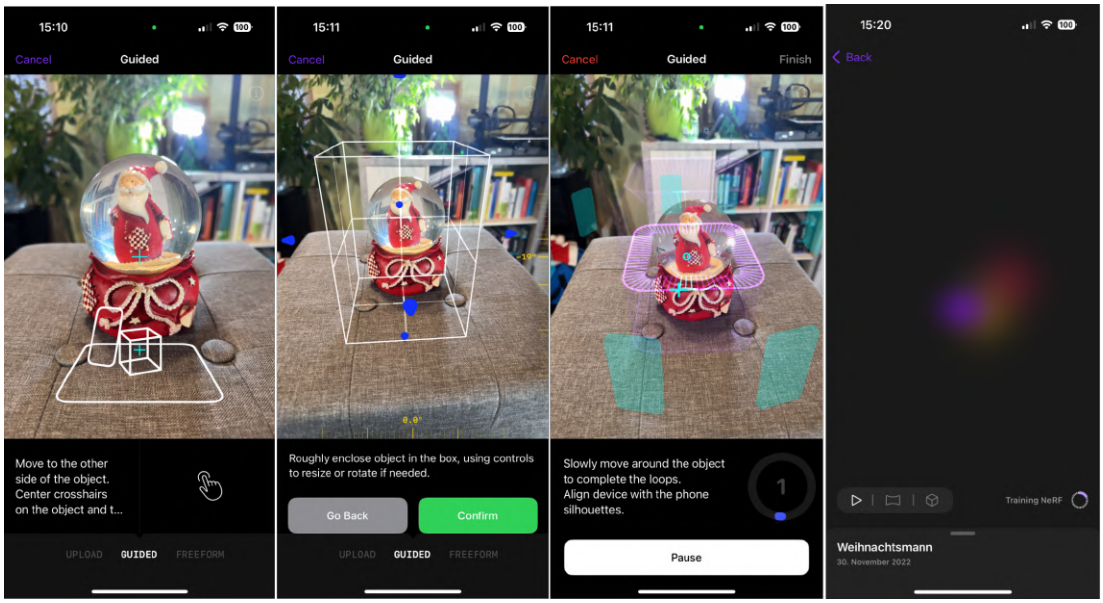


Figure 14.: The systematic capture process within Luma AI: (first) Users designate the object, initiating the bounding box (second) adjustable to match the object's dimensions. Subsequent steps guide the user on camera movement, showcasing captured angles (third). The neural radiance field's training status is represented in illustration four.

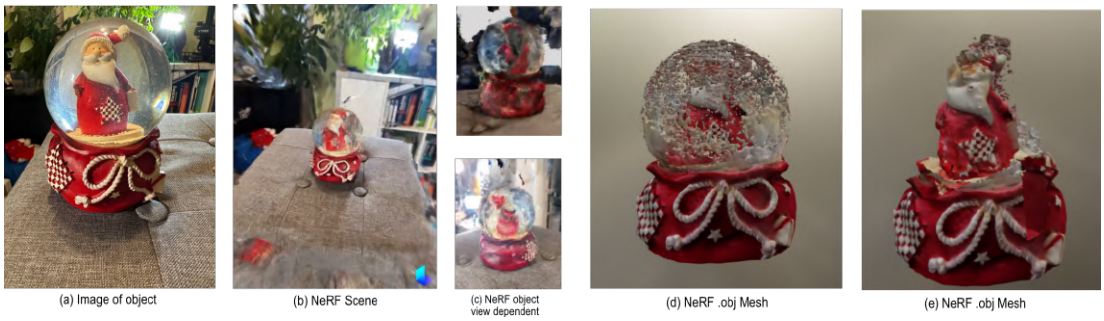


Figure 15.: The comprehensive visualization of the neural radiance field's training outcome for the Santa Claus object. Figure (a) is a reference photo of the object. (b) is the neural radiance field scene. (c) is an illustration of the view dependency, highlighting the significance of this aspect with the NeRF algorithm. (d)shows the extracted mesh showcasing the Santa Claus encapsulated within the transparent sphere. (e) highglights the Mesh without the surrounding sphere, emphasizing NeRF's capability to manage transparency; even with the glass removed, the central object remains intact, validating the efficient rendering of the glass alongside the contained object.

millions of points, demanding post-processing for object delineation, as depicted in Figure 16(a). The derived meshes, especially from dense point clouds, can contain millions of faces, which can be detrimental to real-time rendering performance, as shown in Figure 16(c).

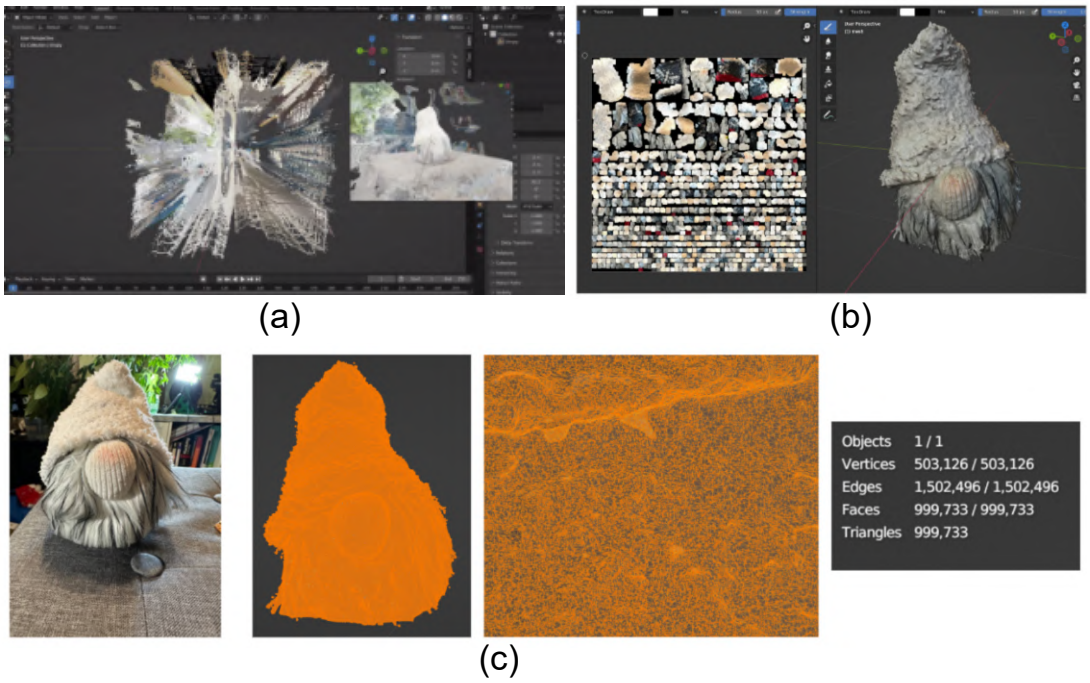


Figure 16.: A showcase of LUMA AI's export features and inherent challenges: (a) Point cloud depiction requiring post-process object extraction, (b) UV-Map of the exported texture, and (c) High-density mesh, highlighting rendering performance concerns.

Moreover, the neural radiance field doesn't always produce ideal results when depicting larger scenes, as shown in Figure 17. Aliasing artifacts become evident, and it often becomes challenging to distinctly perceive the scene from various viewpoints.

In the context of utilizing LiDAR sensors for creating neural radiance fields, challenges arises when encountering minor recording errors. These errors can lead to complexities in accurately comprehending the captured scene within the *Luma AI* app. As illustrated in Figure 18, these subtle recording inaccuracies can result in difficulties during scene interpretation and reconstruction.

LUMA AI has rapidly established itself as an innovative NeRF application since its launch in mid-2022, enabling users to create high-quality, photorealistic $3D$ assets and environments in a short time. Its user-friendly design steers individuals seamlessly through a systematic capture process, aimed at deriving optimal NeRF models. However, despite the impressive suite of functionalities it offers, *LUMA AI* is not without its limitations, particularly evident when tackling expansive scenes or exporting intricate models. Recent endeavors, such as the creation of a music video leveraging *LUMA AI's* capabilities, further highlight its potential in reshaping multimedia expe-

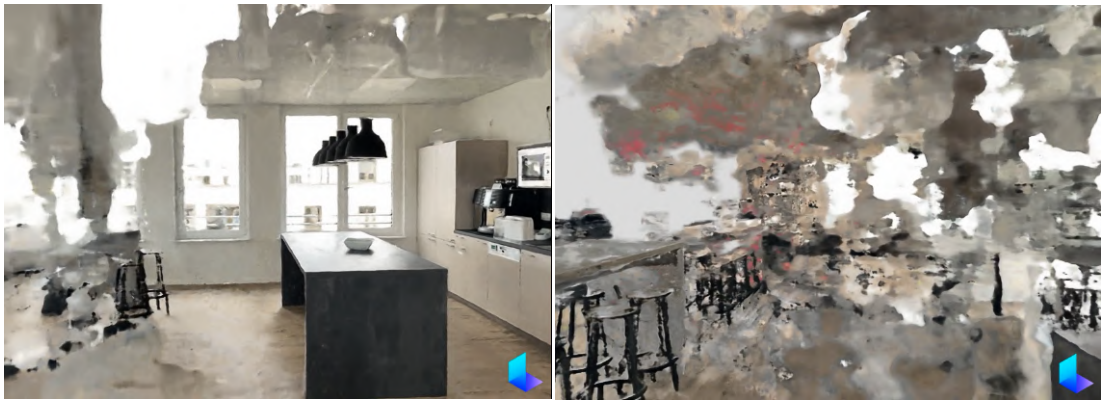


Figure 17.: *LUMA AI's* challenges with rendering large scenes: The left image presents the most accurately rendered view of a scene, while the right image reveals less-than-optimal renderings, marked by artifacts and overall challenges in discerning the scene. This juxtaposition highlights *LUMA AI's* constraints when dealing with large-scale environments.

Figure 18.: *LUMA AI's* challenges: The left image presents the most accurately rendered view of a scene, while the right image reveals less-than-optimal renderings. This image underscores that even minor recording inaccuracies can lead to significant distortions.

riences. Notably, the original developer of NeRF, Matthew Tancik, has become an integral part of this emerging start-up. With heavyweights like NVIDIA backing the vision, it sees a promising foundation forming for *LUMA AI* (Yu and Jain, 2023).

> "Luma is one of the first to bring to market new 3D NeRF capture technology, enabling non-technical users to create higher quality 3D renders than has previously been possible. The team has consistently demonstrated the ability to rapidly productize new research, and their long-term vision of democratizing access to 3D workflow tools for consumers has the potential to drastically lower the skills barrier to create & edit in 3D (Mohamed Siddeek, NVIDIA (NVentures)) (LUMA AI, 2023)."

3.5. The instant-ngp Framework

Following the release of NVIDIA's research on neural graphics primitives (Müller et al., 2022), NVIDIA has not only made the algorithm available as open-source code but has also introduced a comprehensive framework encompassing all necessary components, thereby offering an almost complete end-to-end solution. This framework offers pre-installations for popular NVIDIA graphics cards, a feature-rich GUI, and provides scripts accompanied by thorough explanations for preparing user-generated input data for the NeRF training. In the subsequent sections, we will delve deeper into the array of features and functionalities presented by instant-ngp.

NVIDIA provides pre-built installations for Windows users. People working on Linux platforms, who require special Python bindings, or who have GPUs such as Hopper, Volta or Maxwell iterations, will need to manually build instant-ngp. The instant-ngp framework is equipped with an interactive GUI, shown in Figure 19, which includes a wide range of features, each designed to enhance user interaction and functionality. These features include comprehensive controls that enable users to engage with neural graphics primitives in an interactive manner. Additionally, the framework offers a Virtual Reality and Augmented Reality mode, facilitating the visualization of neural graphics primitives through the use of virtual/augmented-reality headsets. Users also have the capability to save and load "snapshots," simplifying the sharing of graphics primitives online. The GUI further incorporates a camera path editor, empowering users to create customized videos, and supports seamless conversion between NeRF and Mesh representations. Notably, the framework provides tools for optimizing camera pose and lens parameters, ensuring optimal visual outcomes. In addition to these features, the instant-ngp GUI encompasses a host of other functionalities, all of which contribute to an enriched user experience while expanding the framework's overall capabilities. Figure 19 illustrates the instant-ngp framework (NVlabs, 2022). The GUI functions used in this thesis, such as the mesh export, are discussed in chapter 5.

Figure 19.: Illustration of the *instant-ngp* framework. This framework is armed with an interactive GUI that boasts a diverse array of features, all thoughtfully designed to amplify user interaction and enhance functionality. On the right side of this image is a GUI where users can make changes, save and load "snapshots", export a mesh and many more features e.g. debug options. The GUI on the bottom left contains a camera path editor that allows users to create custom videos and make adjustments to the camera for rendering (NVlabs, 2022).

In order to use instant-ngp on custom data, users should first establish a Python environment that includes the necessary libraries, as listed in the GitHub repository. The data, whether videos, images, or depth information, must then be prepared for training. This involves computing the camera parameters and storing them in a JSON format. To streamline this process, developers have crafted a Python script, `colmap2nerf.py`. This tool enables users to process video files or image sequences using the open-source COLMAP structure, which then outputs a JSON file. The script needs specific arguments as inputs, such as the path to the video file and others. Certain arguments, such as `aabb_scale`, explained in further detail in this thesis, are crucial for achieving high-quality neural radiance fields but may not be intuitively understood. The following code snippet provides an example on how to execute the script. It is crucial for users to comprehend these arguments and configure them appropriately to achieve the desired results.

```
1 python [path-to-instant-ngp]/scripts/colmap2nerf.py --video_in <video filename> --
    video_fps 2 --run_colmap --aabb_scale 32
```

Code Listing 4: Execute the colmap2nerf.py script from instant-ngp

Given the challenges involved, particularly for users without a technical background, the process of creating neural radiance fields using the instant-ngp framework can be difficult and time-consuming. Users must navigate through the intricacies of installation, data preparation, parameter selection for the JSON file generation, and

subsequent network training. Furthermore, the possession of an NVIDIA GPU is a prerequisite, which may not be readily available to all users. These factors significantly hinder the accessibility of this framework. It is also important to note that the installation process and requirements of the instant-ngp framework are also dependent on the user's operating system and the versions of the required dependencies. This adds an additional layer of complexity, as users need to ensure compatibility with their specific setup.

Following the successful initialization and preparation of the necessary parameters, one is poised to commence the NeRF model training. This can be initiated from the instant-ngp directory. Alternatively, for those who prefer a more direct graphical interaction, the designated .exe executable provides a user interface wherein one can easily integrate both the image datasets and the transforms.json configuration:

```
instant-ngp$ ./instant-ngp [path to training data folder containing transforms.json]
```

The scripts, `colmap2nerf.py` and `record3d2nerf.py`, operate under the premise that training images predominantly orient themselves towards a common focal point, which the scripts designate at the coordinate origin. This focal point is discerned by computing a weighted mean of the nearest points of intersection among rays emanating from the central pixel across all training image pairs. Operationally, this suggests an optimal performance when the training images are strategically captured with an inward orientation towards the primary object, even if they don't encapsulate a comprehensive 360-degree view. It should be noted that any backdrop behind the primary object remains subject to reconstruction, especially if the *aabb_scale* parameter surpasses a unitary value, as previously elaborated.

Figure 20 shows the training process using instant-ngp. For the purpose of debugging, cameras and the unit cube representing a scale of 1 have been mapped.

Figure 20.: This illustration shows the training in instant-ngp. Within the debugging visualization context, both the cameras and the unit cube—characterized by a scale of 1 - are incorporated (NVlabs, 2022).

3.6. Discussion

The instant-ngp framework stands out for its extensive features and the best implementation of the instant-ngp algorithm with CUDA. Alternative implementations such as *Torch-NGP* train more slowly and do not offer the same feature set. The primary goal of this thesis is to design a framework that allows users to efficiently train neural radiance fields with the instant-ngp algorithm, which is why *Nerfstudio* and similar frameworks are too extensive for the context of this thesis.

Throughout our research, we have taken an industrial perspective into account, acknowledging factors like the availability of administrator rights, which might not be universally accessible in an industrial environment, and the requirement of possessing an NVIDIA graphics card with RTX capabilities. We've also considered the scenario where users may lack coding expertise or familiarity with command-line interfaces. This exploration underscores the observation that existing frameworks, especially in an industrial context, do not offer a straightforward and efficient solution for generating neural radiance fields without the specific prerequisites mentioned above. Moreover, numerous current frameworks assume the availability of training input and do not extensively address this facet. Although methods to generate input from videos do exist, these steps often involve manual execution of commands or the use of external tools such as Colmap or FFMPEG. Given these intricacies, our decision was to develop a customized framework. Notably, our framework drew inspiration from the explored frameworks. For instance, *ArcNeRF* contributed insights into logger design and configuration files, while the approach taken in *instant-ngp* provided the foun-

dation for transitioning from video to input generation. In the next chapter (4), we take a closer look at the specific requirements that led us to embark on our unique implementation journey.

4. Implementation

The instant-ngp framework emerges as the most efficient and optimal solution for generating neural radiance fields through the use of multiresolution hash encoding. In addition to its core functionalities, the framework provides auxiliary scripts designed to transform self-captured data into a format suitable for training. To train a neural radiance field, users must execute these scripts in order to generate a compatible data format, which can then be fed into the instant-ngp framework to initiate the training process. It is important to note that by using the algorithm we are bound to an NVIDIA graphics card, as the scripts are written in CUDA, a low-level programming language designed to get the most out of its GPUs, to speed up the training process.

Industrial users face significant challenges in installing and using this framework. Many do not have the necessary administrative rights to install or run the framework, and additionally they can find themselves lost in the complexity, not just in comprehending the python commands but also in choosing the best parameter for their specific video type as input. Moreover, not everyone has access to a dedicated GPU, which is essential for training this NeRF model. Given these challenges and the complexity of data preparation, we saw the need for a more streamlined solution. To address this, we developed a custom framework that automates the entire pipeline, converting various data formats - such as videos, zipped image collections, and r3d captures from LiDAR-equipped iPhones - into neural radiance fields.

With this in mind, we developed a front-end application where users can easily upload data, adjust parameters, and download the resulting neural radiance field as a `.ingp` file, which can be used within the instant-ngp framework's GUI (Müller et al., 2022), enabling further exploration.

In Figure 21 we provide an overview of this pipeline. Initially, users are required to capture the scene and upload the data to our website. Upon clicking the "start" button, the pipeline process begins with a set of pre-configured settings, which can be customized using the "modify" button. The pipeline, represented by blue circles in the figure, initiates by extracting frames from the uploaded data, followed by the execution of COLMAP to estimate camera poses. Once COLMAP completes its computations, the data is transformed into a format suitable for training. This data, along with the corresponding images, serves as the input for training. Upon completion of the pipeline, users have the option to save the trained file, render a video, or download a mesh. The trained file allows users to visualize the neural radiance field and even

display it in VR within the instant-ngp framework.

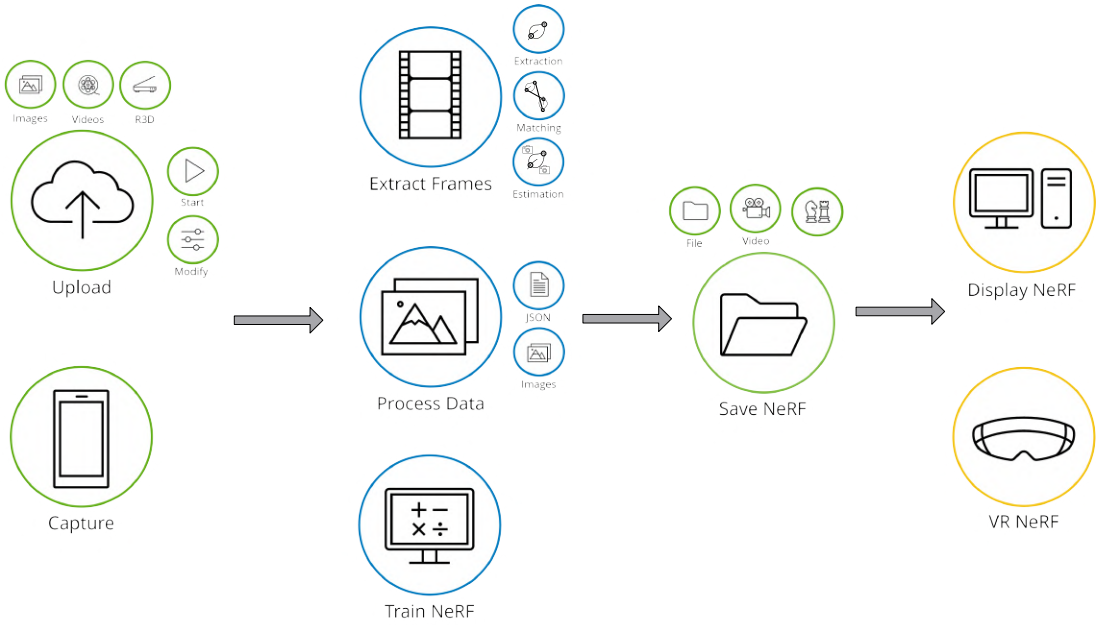


Figure 21.: A comprehensive overview of the custom pipeline designed to simplify the generation of Neural Radiance Fields for industrial users. The process starts when users upload their captured scene data to the web application. They can initiate the pipeline with pre-configured settings by clicking the 'start' button, or customize parameters using the 'modify' button. The pipeline consists of multiple steps, starting with frame extraction and followed by camera pose estimation via COLMAP. After this, the data is converted into a training-suitable format. The resulting training input is then processed to produce a neural radiance field, which can be saved, rendered into a video, or downloaded as a mesh. This output file is compatible with the instant-ngp framework and can be explored further in VR.

The back-end, built in Python, can run on any server equipped with an NVIDIA RTX GPU, eliminating the need for a local graphics card. This combination of a user-friendly interface and a separate back-end makes it simpler for users to create high-quality neural radiance fields. Figure 22 showcases the "SICK - NeRF" front-page, which has been designed with user experience in mind.

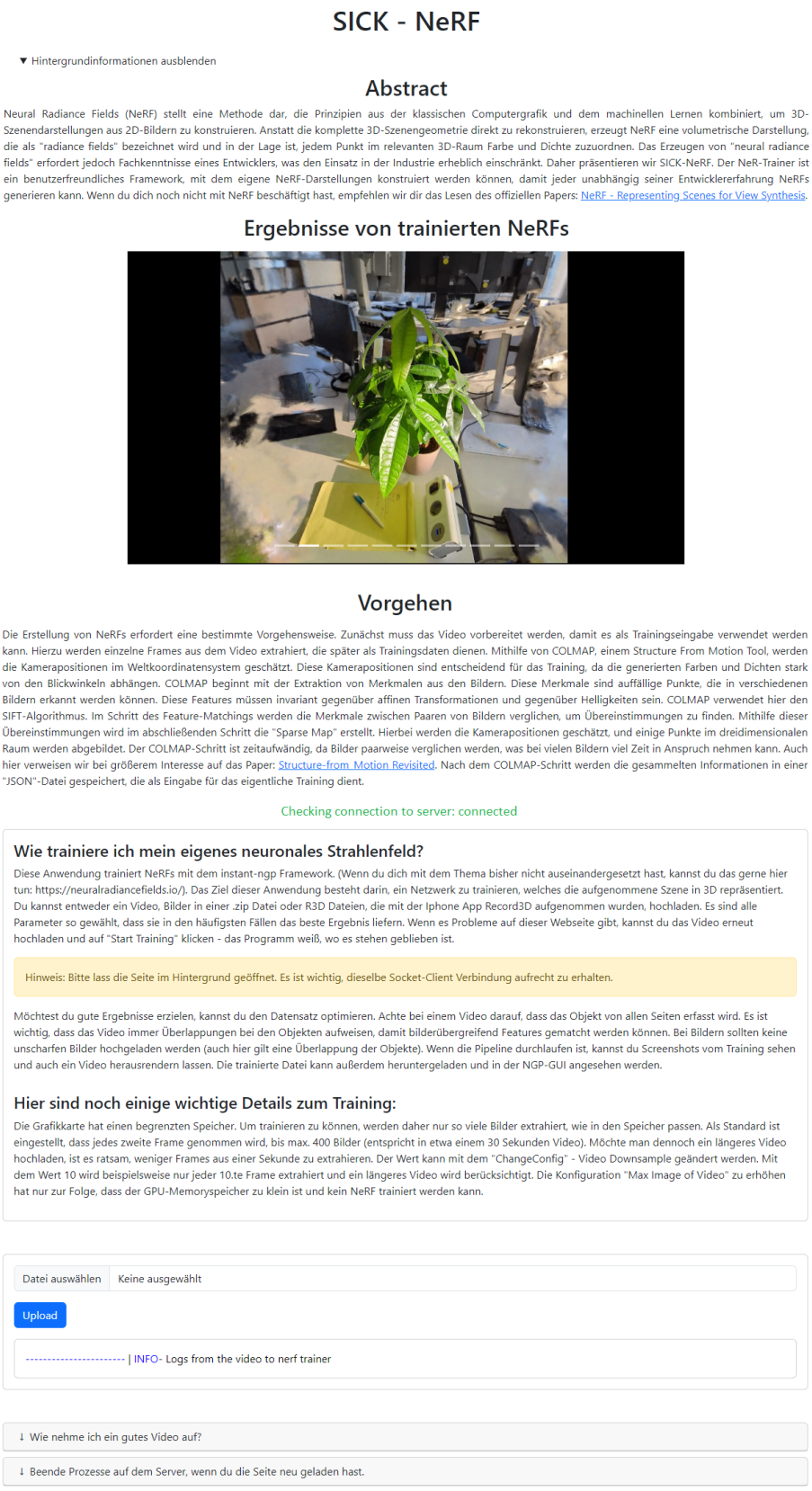


Figure 22.: The "SICK - NeRF" website first presents an abstract, followed by a video slider with results of trained neural radiance fields. Below is an information section describing how the pipeline works. The entire background section can be collapsed to focus on the main section. The main part of this page consists of the container box with an upload form that can be used to upload data and then train the neural radiance fields. Further down, expandable information accordions provide additional insights.

In this chapter, we provide a comprehensive overview of the methodologies and tools integral to our project. The discussion starts with Section 4.1, where we outline our technology stack and explain how we achieved system-independence and efficient package management using Docker and Poetry. Section ?? delves into the various development tools utilized, including build and dependency management with Poetry, system-independent installation methods via Docker, and our Flask and React-based web applications. We then move to Section 4.4 to dissect the pipeline of our project, detailing stages from image extraction to training with instant-ngp. The development of our web-based interface is laid out in Section 4.5, with specific focus on Flask and React technologies. The chapter concludes with a summary that encapsulates the key points discussed, offering a cohesive recapitulation of our project's technical architecture and implementation.

4.1. Technology Stack

In this section, we delve into the technical components that constitute the backbone of our application. By exploring the chosen programming languages and the key frameworks and tools, we provide a comprehensive understanding of the application's architecture.

4.1.1. Programming Languages

Python: The Back-end This application is entirely written in Python. Among programming languages, Python has seen a surge in popularity due to its versatility, extensive standard libraries, and robust community support. While the implementation for this project could have been achieved using various other programming languages, the choice of Python was influenced by the author's familiarity with the language and the wide range of available frameworks that expedite the development process (Python Software Foundation, 2023).

JavaScript: Powering the Front-end For the front-end, JavaScript is the language of choice. As one of the core technologies behind web development, JavaScript has matured into a versatile and powerful language. Its asynchronous capabilities and widespread browser support make it an ideal choice for creating dynamic and interactive user interfaces. The selection of JavaScript for the front-end was guided by its suitability for the project's requirements and the broad ecosystem of frameworks and libraries available to speed up the development process.

4.1.2. Frameworks, Tools, and Additional Features

The most important frameworks, libraries and other critical components and features we've developed and used for this application are briefly introduced below. Their exact use, including code examples, will be discussed in detail in the next sections.

Poetry Poetry is used for dependency management in the Python environment. It simplifies the process of managing project dependencies and streamlines package management, ensuring that all libraries are compatible and up-to-date (Poetry, 2023).

Docker Docker is employed to containerize the application, ensuring that it runs consistently across various computing environments. This aids in both development and deployment, allowing for easier scalability and manageability (Docker, 2023).

Flask Flask is the web framework chosen for the back-end of this application. Known for its simplicity and flexibility, Flask is particularly well-suited for small to medium-sized applications and services. It is used to handle API requests and serve data to the front-end (Pallets, 2023).

React React is the JavaScript library employed for developing the front-end. It is renowned for its component-based architecture and high performance, enabling dynamic and responsive user interfaces. React also boasts strong community support and a wide array of third-party libraries (Source, 2023).

YAML `YAML`, an acronym for "YAML Ain't Markup Language," serves as a human-readable data serialization format that is easy to write and understand. It is designed to work seamlessly with version control systems like Git, making it highly suitable for managing configurations in a collaborative development environmente (Net et al., 2021). In our application, we utilize a `YAML` configuration file as a centralized location for storing all essential parameters and information, such as the name of the capture, the file path, training settings, and more. Below is a sample snippet from the project's configuration file, underscoring its organized layout that allows for straightforward modification of settings such as COLMAP's `match_type`:

```
data:
  colmap:
    match_type: sequential_matcher
  video_downsample: '3'
  video_path: ./uploads/VID_Buro.mp4
```

Code Listing 5: Example YAML Configuration used in this project

Real-Time Logging To augment the application's user-friendliness and transparency, a real-time logging mechanism is included alongside the key frameworks and libraries. This feature is particularly valuable during the execution of computational-intensive processes, such as the COLMAP phase. A specialized logger continually updates the user on the current state of the system, providing an informed overview of the ongoing activities. Below is an excerpt from the logging output, specifically detailing the steps and progress during the COLMAP phase:

```
2023-08-02 12:44:30.097 | INFO | - Start to run COLMAP and estimate cam_poses...
  Scene dir: data/Waki_Showroom
2023-08-02 12:44:30.097 | INFO | - Need to run construct from colmap...
2023-08-02 12:44:30.098 | INFO | - Starting Feature Extraction...
2023-08-02 12:44:30.233 | INFO | - Features extracted...
2023-08-02 12:44:30.233 | INFO | - Starting Feature Matching...
2023-08-02 12:44:30.871 | INFO | - Features matched...
2023-08-02 12:44:30.871 | INFO | - Starting create sparse map...
```

Code Listing 6: Logger output detailing the steps and progress

4.2. Development Tools

In this Section, we delve into the array of development tools and methodologies utilized to construct a robust, efficient, and easily deployable project. Understanding the intricacies of software development demands more than just writing code; it also necessitates an appreciation of build and dependency management, system portability, hardware optimization, and a streamlined process for incorporating updates into the live application.

4.2.1. Build and Dependency Management with Poetry

To streamline build and dependency management, we utilized Poetry - a Python-specific command-line tool designed to simplify the entire package management workflow. The primary focus of Poetry is to provide a comprehensive and user-friendly experience, prioritizing simplicity, ease of use, and consistency. With Poetry, developers can easily define project dependencies, manage virtual environments, build and package projects, and even publish them to the Python Package Index (Poetry, 2023).

Poetry centralizes its management functions through the `pyproject.toml` file, acting as the orchestrator for both the project and its associated dependencies. This file not only contains important configuration details but also serves as a central hub for managing various aspects of the project (Poetry, 2023). In particular, it's able to navigate complex dependency structures and automatically resolve any conflicts that may arise during the development. Listing 7 contains an excerpt from this file that illustrates the metadata associated with the project.

```
[tool.poetry]
name = "nerf_industrial_metaverse"
version = "0.1.0"
description = ""
authors = ["Sabine Schleise <user@email.de>"]
readme = "README.md"
packages = [{ include = "nerf_industrial_metaverse" }]
```

Code Listing 7: Metadata in Poetry

The subsequent code sample 8 illustrates how dependencies are declared within the `pyproject.toml` file.

```
[tool.poetry.dependencies]
python = "^3.9.13"
ffmpeg-python = "^0.2.0"
scipy = "1.9.3"
opencv-python-headless = "^4.7.0.72"
```

Code Listing 8: Poetry's dependency management

By documenting dependencies in the `pyproject.toml`, the flexibility to either specify fixed version numbers or define adjustable version ranges is afforded, thereby enabling seamless updates while maintaining compatibility. Within the `[tool.poetry.dependencies]` section, each dependency is articulated with either a specific version number or a version constraint that adheres to semantic versioning guidelines. The caret (^) symbol, for instance, permits updates that are confined to the specified major version, with ^3.9.13 allowing any subsequent version within the 3.x.x range but precluding updates to the major version. Despite the constraints outlined in the `pyproject.toml`, the versions that are ultimately installed may deviate based on these conditions and on the contemporaneous updates to external packages.

Invoking the `poetry install` command generates a `poetry.lock` file, a definitive record of the precise versions of all dependencies utilized during the development phase. This file serves as an authoritative reference for all ensuing installations, guaranteeing the uniformity of installed versions and negating unforeseen alterations stemming from dependency updates. The inclusion of the `poetry.lock` file in version control systems ensures that this uniformity is maintained across diverse development environments, thus contributing to a reliable and consistent build and deployment process (Poetry, 2023).

To further enhance the usefulness of our project, we took advantage of Poetry's ability to define and execute custom scripts. These scripts are defined within the `[tool.poetry.scripts]` segment of our `pyproject.toml` file, as shown in the code snippet 19. Each script, such as `hello-world`, is mapped to a specific function contained within our code base.

```
[tool.poetry.scripts]
hello-world="nerf_industrial_metaverse.tools.hello_world:entrypoint"
```

Code Listing 9: Custom Scripts for Facilitating Pipeline Executions

These custom scripts can be run using the `poetry run [script-name]` command. Alternatively, if the project has been installed with `pip install -e .`, these scripts can be invoked with `ngp-extract`.

In our exploration and installation of the academic projects presented in Chapter 4, it became evident that many of them are linked to certain prerequisites. Some projects require specific software and package versions, e.g. Python 3.7 or PyTorch 1.10+, others can only be installed on Ubuntu, while other installation procedures depend on the operating system used. In light of this, our incorporation of Poetry as a dependency and package management tool demonstrated its efficacy in simplifying the intricate landscape of project management. The use of the `pyproject.toml` file afforded us a centralized control center for managing project dependencies and their respective versions. Furthermore, Poetry's support for the creation of isolated virtual environments endowed our project with increased robustness and reproducibility. This feature allowed us to isolate the project from any conflicting system-level Python installations, thereby significantly mitigating potential disruptions.

4.2.2. System-Independent Installation with Docker

Docker is a platform that uses open source technology to facilitate the automated deployment and management of applications using lightweight, isolated containers. Each of these containers acts as a self-contained unit, housing both the application and its dependencies. One of the primary purposes of these containers is to ensure a consistent behavior across various environments. By design, Docker containers offer isolation, meaning each application operates in its unique sandboxed environment, free from interference by other applications or the underlying system. Containers also can be easily moved between different environments, such as development, testing, and production, without the concern of compatibility issues. When a container runs, it uses a dedicated and isolated file system, which is derived from a container image. This image is encapsulating everything required for the application to function - dependencies, configurations, scripts, binaries, and more (Docker, 2023).

To create a project-specific container image, we developed a `Dockerfile`, which acts as a scripted blueprint for constructing the container environment. A `Dockerfile` is a text document that contains all the commands a user could call on the command line to assemble an image.

This `Dockerfile` begins by specifying the base image, which in our project is `nvidia/cuda:11.8.0-devel-ubuntu22.04`. This particular image provides a pre-configured setting equipped with NVIDIA's CUDA 11.8 toolkit. The `devel` tag signifies that the image contains additional development packages and libraries, making it optimal for development and compilation activities that require CUDA capabilities. Furthermore, this image is built upon the Ubuntu 22.04 operating system, offering compatibility and a

familiar development environment.

Our `Dockerfile` incorporates multiple directives, delineated by keywords like `ARG` and `ENV`, for enhanced container customization. The `ARG` keyword allows developers to specify variables during the build process. Conversely, the `ENV` keyword is used to define environment variables accessible by the containerized application, providing a mechanism for application-level configuration. The provided Listing 10 demonstrates the initial setup and configuration in our `Dockerfile`.

```
FROM nvidia/cuda:11.8.0-devel-ubuntu22.04

ARG COLMAP_VERSION=dev
ENV CERES_SOLVER_VERSION=2.0.0
ARG CUDA_ARCHITECTURES=89
ENV PATH=/usr/local/cuda/bin:$PATH

RUN echo "Installing COLMAP ver. ${COLMAP_VERSION}..."
RUN git clone https://github.com/colmap/colmap.git
RUN cd colmap && git reset --hard ${COLMAP_VERSION}
```

Code Listing 10: Excerpt from Dockerfile for Container Construction

The `RUN` keyword is used to execute specific commands within the container during the build process. This enables developers to perform installations, set up dependencies, and carry out various configuration tasks to ensure the container has all the necessary components for running the desired application. Each `RUN` command is executed independently and in sequence, the && operator can be used to connect them (Docker, 2023), as demonstrated in Listing 10.

Additional Docker keywords like `ADD` and `COPY` support the transfer of files into the container, whereas `ENTRYPOINT` designates the default command to execute upon container instantiation. An example of how these commands are structured in our project is given in Listing 11:

```
ADD . nerf-industrial-metaverse
RUN cp instant-ngp/build/pyngp.cpython* /nerf-industrial-metaverse
RUN cd nerf-industrial-metaverse && pip install -e .
ENTRYPOINT [ "python3", "flask.py" ]
```

Code Listing 11: Example of Keywords used in this project

Our Docker configuration, detailed in Appendix B, outlines a sequential build process that starts with establishing the base image and installing essential packages. Subsequently, specialized software such as COLMAP and instant-ngp are installed to fulfill the project's specific requirements. Special attention is given to the compilation of instant-ngp, excluding the GUI. The decision to exclude the GUI is rooted in the inherent complexities and challenges associated with running graphical interfaces within Docker containers. Docker, by design, emphasizes the isolation of containerized applications from their host system. As a result, GUI applications, which require direct interaction with the host's X-Server, encounter barriers. Moreover, allowing

such interaction raises security concerns, as the X-Server lacks protective measures against applications accessing it. The final stage involves integrating and installing the application's source code into the container, thereby encapsulating all necessary components within a single, deployable unit.

Docker containers serve as self-contained environments that replicate the exact specifications defined in their corresponding `Dockerfile`. Upon initialization, a Docker container provides a tailored setting that includes the pre-configured tools and utilities. However, it's crucial to understand the inherent characteristics of these containers. Once a Docker container is instantiated, it provides users with an environment equipped with all the tools and utilities described in the `Dockerfile`. This ensures immediate and easy access to these tools without the need for further setup. On the other hand, any tools or utilities not specified during image creation will remain unavailable. Users can install additional tools within an active container. However, due to the isolated nature of Docker containers, all modifications within the container, whether they be configurations, installations, or file changes, are transient. This means that upon termination or restart of the container, these modifications are discarded, returning the container to its original state as defined by the image. For lasting changes to be integrated, they must be incorporated into the `Dockerfile` and the image must subsequently be rebuilt.

Upon the successful construction of the `Dockerfile`, a corresponding Docker image can be synthesized by executing the command `docker build -t nerf_industrial_metaverse` . To instantiate a container from this image, the `docker run` command is employed. Specifically, we use the `docker run --gpus all --rm --it name_of_container` command for container initialization. The `--gpus all` flag ensures that the container uses all available GPUs on the system. The `--rm` flag tells Docker to remove the container when it is finished, optimizing resource management. The `--it` flag stands for "interactive terminal", which allows users to interact directly with the container via the terminal.

This container serves as a specialized computational environment tailored for our project requirements. Within this encapsulated setting, we have the capacity to execute not only our pipeline through a command-line interface but also run specialized commands from software components like COLMAP and instant-ngp. The container thus offers a consistent, isolated, and resource-optimized ecosystem for the development of our application, guaranteeing that all dependencies and configurations are uniformly managed across different stages of the software lifecycle. In addition, it's worth noting that this container can also be used for remote development via Visual Studio Code's DevContainer feature. This alternative method is particularly beneficial for debugging and development tasks, as it combines the isolated environment of a Docker container with the rich feature set of VS Code, including its debugging

tools and extensions. This alternative method is particularly beneficial for debugging and development tasks, allowing for a unified development experience across different environments.

4.2.3. Compute Unified Device Architecture (CUDA)

Poetry and Docker provide robust and maintainable frameworks for our project. Nevertheless, the dependency of the instant-ngp algorithm on CUDA for GPU acceleration poses particular challenges that are not easily overcome due to the direct hardware connection. While Docker is designed to encapsulate and isolate its environment from the host, GPU acceleration requires more direct interaction with the hardware. To integrate CUDA within Docker's isolated environment, we employed the NVIDIA Container Toolkit, which facilitates the required hardware access while maintaining Docker's encapsulation principles. The toolkit guarantees backward compatibility, supporting even older containers as GPU drivers on the host are updated. Additionally, it provides mechanisms for GPU allocation in multi-GPU setups, enhancing resource management. A `setup-ubuntu.sh` script is supplied for toolkit installation.

A critical aspect of our setup is specifying the correct CUDA architecture, influenced by the varying compute capabilities of GPUs used in our project. Ranging from laptop-grade "Ampere A3000" (capability 86) to server-grade "4090 Ti" (85) and "A100 40 GB" (80), these capabilities not only determine the GPU's general architecture but also influence computational performance and feature compatibility. Incorrectly specifying the compute capability in our Dockerfile, for example by setting `ARG CUDA_ARCHITECTURES=89`, can lead to issues such as "illegal memory access" errors. Such issues can be daunting to diagnose, especially given their broad nature, often making them hard to trace back to the underlying hardware compatibility issue.

One limitation we encountered is the absence of an automated system to adapt the CUDA architecture setting. This manual requirement adds an additional layer of complexity, emphasizing the importance of correctly identifying and specifying the compute capabilities during both the development and deployment phases. Therefore, while the toolkit brings us closer to a more system-independent setup, this CUDA-related setting remains a manual step, underlining its significance and the need for caution during the setup process.

Some of the frameworks discussed in Chapter 3 used CUDA, PyTorch and/or tiny-CUDA-NN in their projects. Installing these components presented its own sets of challenges. For readers interested in a more comprehensive discussion of these challenges, including solutions and workarounds, refer to Appendix C.

4.2.4. Flask Application

Flask (Pallets, 2023) is a lightweight yet powerful web framework for Python that is particularly suitable for developing small to medium sized web applications. It provides the flexibility needed to create a variety of back-end services, including REST APIs, web pages and more (Pallets, 2023). In this section we will look at the implementation in Flask in more detail.

The core structure and functionalities in Flask are encapsulated within a main file named `app.py`. This file not only hosts the essential configurations but also forms the backbone of our Flask setup. The configurations employed in our application are depicted in below 12.

```
from flask import Flask, request, jsonify, session, g, send_file
from werkzeug.utils import secure_filename
from flask_cors import CORS
from flask_socketio import SocketIO

app = Flask(name)

CORS(app, resources={
  r"/*": {"origins": ""}})

socketio = SocketIO(app, cors_allowed_origins="*", async_mode='threading')

SECRET_KEY = secrets.token_hex(16)
app.secret_key = SECRET_KEY

if name == 'main':
socketio.run(app, host='0.0.0.0', port=5000, debug=True)
```

Code Listing 12: Configuration of the Flask application in `app.py`

We begin by importing necessary modules and functionalities. The Flask class is indispensable for setting up any Flask application. Additional imports like `request`, `jsonify`, `session`, `g`, and `send_file` offer functionalities for handling client requests, JSON formatting, managing user sessions, a special object for temporary storage during a request, and sending files as responses, respectively. The `secure_filename` function from the `werkzeug` library aids in ensuring that the uploaded files have secure names, a security measure preventing malicious uploads. The `CORS` function from `flask_cors` package is pivotal in handling Cross-Origin Resource Sharing (CORS), allowing the front-end and back-end to communicate without any restrictions, which is especially significant if they are hosted on different domains or ports. Furthermore, to enable real-time communications, we employ `SocketIO` from the `flask_socketio` library. This ensures bidirectional communication between the web clients and servers, a requirement for many modern web applications. The line `app = Flask(__name__)` initiates our Flask app. For security, we have generated a secret key for our application, which ensures the confidentiality of user sessions and prevents potential tampering. This is established using the `secrets.token_hex(16)` function, which produces a random token

that we set as the `app.secret_key`. Lastly, the `if __name__ == '__main__'` block signifies that if this script is run directly, the Flask app will start in debug mode on the specified host and port. The use of `socketio.run` instead of the conventional `app.run` allows for the integration of Flask with `Socket.IO's` functionalities.

Flask's route system stands out due to its intuitive nature. Using the `@app.route()` decorator, functions are easily mapped to specific URLs. The following code snippet is an example of this route system in Flask:

```
@app.route('/upload', methods=['POST'])
def upload():
[shortened for clarity]
return jsonify({'status': 'error', 'message': 'Invalid file type'}), 400
```

Code Listing 13: Route System in Flask

The route `@app.route('/upload', methods=['POST'])` defines the endpoint `/upload` and specifies that it accepts only `POST` requests. This route essentially acts as a gateway for users to upload files.

In this implementation, we established specific routes to interact with the front-end. For instance, the route `@app.route('/start-training', methods=['POST'])` initiates the pipeline when the "Start Training" button is clicked. Similarly, the route `@app.route('/update_yaml_config', methods=['POST'])` updates the YAML configuration based on changes made in the front-end. We also defined routes for transmitting screenshots as well as the rendered video, as exemplified in the schemas `@app.route('/api/images/path:image_path')` and `@app.route('/api/images')`.

Concluding this implementation, we've seamlessly integrated essential functionalities of the pipeline into a back-end web application, enabling a streamlined workflow for training neural radiance fields with uploaded data. Moreover, the platform provides visual feedback, equipping users with optional utilities such as downloading data, rendering videos, and exporting meshes.

4.2.5. React Application

Our React application, based on the open-source JavaScript library by Meta (Source, 2023), is engineered for user-friendly interface and efficient server-client communication. The architecture revolves around modular "components" that render specific UI elements, thereby making the code easily maintainable.

```
src
├── components
│   ├── UploadForm.jsx
│   └── HandleTraining.jsx
├── App.jsx
├── App.css
├── index.js
└── [...]
```

The primary directory, 'src', holds all React code. Components like `UploadForm.jsx` manage data uploading, while `HandleTraining.jsx` handles training processes. `App.jsx` serves as the main component integrating various other components.

In React, components seamlessly integrate with HTML, as demonstrated in the code snippet below. The visual rendering of the `UploadForm` component, as depicted in Figure ??, also incorporates the `LogViewer` component, which is nested within `UploadForm`. Additionally, this snippet showcases the use of Bootstrap to enhance the visual appeal of the application.

```
<div className="card my-5">
  <div className="card-body">
    <UploadForm />
  </div>
</div>
```

Code Listing 14: Integrating React components

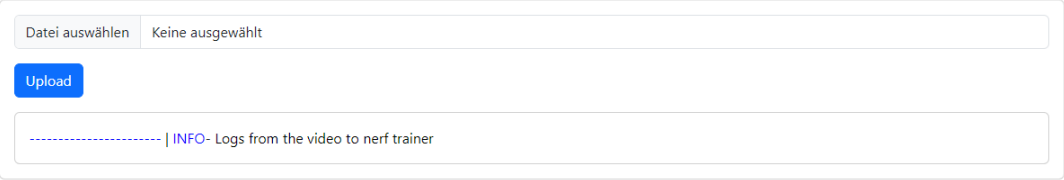


Figure 23.: Rendered view of the `UploadForm.jsx` component

We've incorporated real-time data flow between server and client using WebSockets, with robust reconnection strategies for handling potential server unavailability 15.

```
import { io } from 'socket.io-client';

export const socket = io('URL-to-backend:5000', {
  cors: {
    origin: "URL-to-frontend:PORT",
    methods: ["GET", "POST"],
    credentials: true,
    transports: ['websocket', 'polling'],
  },
  allowEIO3: true,
  reconnection: true,
  reconnectionAttempts: 10,
  reconnectionDelay: 10000,
  reconnectionDelayMax: 50000
});
```

Code Listing 15: Socket Connection in React

Our development leverages GitLab's CI/CD pipeline for automatic rebuilds and deployments. The front-end operates independently of the back-end, allowing for a flexible hosting strategy and uninterrupted user experience.

4.2.6. Continuous Integration and Continuous Delivery

The principle behind continuous integration (CI) is the automation of code integration processes, which markedly accelerates the product development cycle. With CI, developers regularly submit their code alterations to a communal repository. Each submission initiates an automatic build and testing sequence for the entire project, enabling rapid identification of problematic commits. This practice minimizes the time required for code integration and elevates overall productivity (Red Hat, 2022).

For the scope of this project, we leveraged GitLab's CI/CD capabilities for the front-end and the back-end. The architecture of our construction and deployment processes is defined in a `.gitlab-ci.yml` file. A representative excerpt from this configuration file, which outlines our CI pipeline for the back-end, is displayed below for elucidation (16):

```
variables:
... [shortened for clarity]
stages:
publish

publish:
tags:
[...]
stage: publish
image: ${IMAGE_PUBLISH}
script: |
echo ${ARTIFACTORY_TOKEN} | docker login -u=${ARTIFACTORY_USER} --password-stdin ${
  REGISTRY}
docker build --file ${DOCKERFILE_PATH} -t ${REGISTRY}/${CI_PROJECT_NAME}:latest .
docker push ${REGISTRY}/${CI_PROJECT_NAME}:latest
```

Code Listing 16: Deployment configuration of our back-end

The script section delineates the orchestrated sequence of actions: initially, authentication is performed against our private Docker registry by utilizing stored credentials. Subsequently, a Docker image is constructed based on the predefined `Dockerfile`. Finally, this freshly created image is uploaded to the designated registry. This entire automated workflow is triggered with every code commit to the shared repository, ensuring that the back-end's Docker image is perpetually up-to-date. To complete the automation cycle, our `Dockerfile` concludes with an `ENTRYPOINT [ "python3", "app.py" ]` directive, guaranteeing that the Flask application embodied in the `app.py` script is executed through Python3 every time the container is instantiated.

Following the successful implementation of CI through GitLab's CI capabilities, the next imperative step was to establish a continuous delivery (CD) pipeline. CD acts as an evolutionary advancement over CI, concentrating on the automation of the software deployment process. By embracing CD methodologies, software can be systematically released to end-users in a secure, expedient, and sustainable fashion. This approach not only expedites the product's market introduction but also considerably mitigates

the risks tied to manual deployment methods. As a result, the software development lifecycle benefits from increased resilience and reliability (Red Hat, 2022).

The integration of both CI and CD ensures a comprehensive automation framework that continually builds, tests, and deploys both our front-end and back-end components. This synchronous mechanism guarantees that our software solutions remain consistently updated and aligned with the latest development paradigms and user requirements.

4.3. Directory Structure and Functionality

The architecture of this project adheres to modular programming principles, facilitating efficient code management, extensibility, and maintainability. As the project undergoes further development, this modular approach enables the seamless addition of new features. The implementation strategy is anchored by several pivotal directories, each of which serves a distinct functional purpose.

1. **Common Directory:** Constituting the project's foundational layer, the *common* directory incorporates utility functions. Noteworthy among these are the `config_util` for YAML configuration management, `logger_utils` for logging mechanisms, and `image_utils` for image-related operations.

2. **Scripts_ngp Directory:** This directory features modified versions of instant-ngp scripts, tailored to execute the project-specific training procedures.

3. **Tools Directory:** This compartment contains Python scripts, each dedicated to executing individual tasks in the pipeline, ranging from frame extraction from videos to running COLMAP commands and data pre-processing for training.

4. **Viewer Directory:** Serving as the user interface gateway, the *viewer* directory is bifurcated into two primary subdirectories:

    a) *Client:* Tasked with rendering the web-based user interface.

    b) *Server:* This segment is charged with managing back-end processes, including request handling and data manipulation.

An illustrative depiction of this directory and script hierarchy is presented in Figure 24.

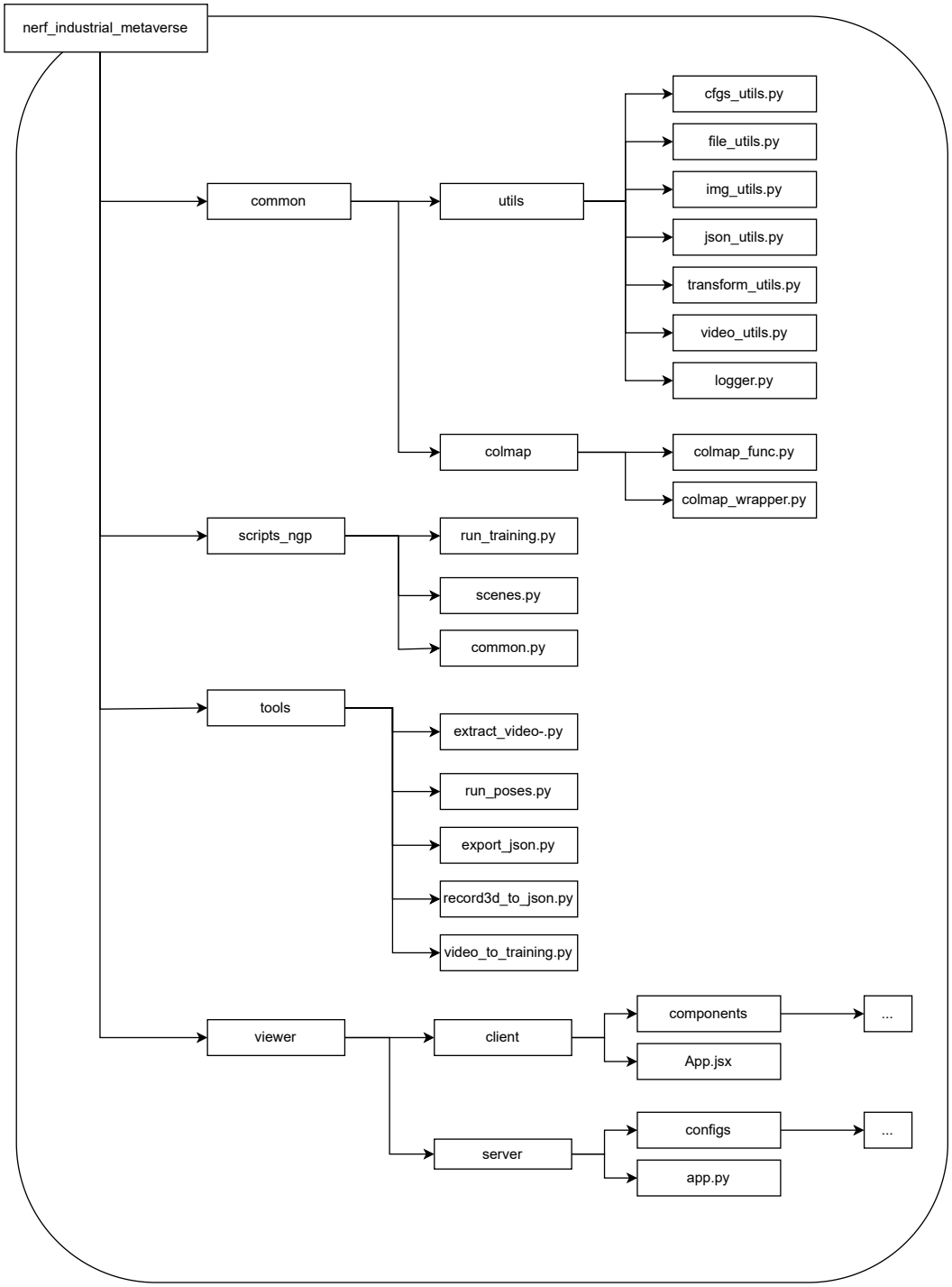


Figure 24.: The project consists of four main components: (1) *common* - containing auxiliary colmap scripts and utilities, (2) *scripts_ngp* - customized scripts from the instant-ngp framework, (3) *tools* - essential pipeline steps like video extraction and JSON file export, and (4) *viewer* - includes server and client scripts.

The Docker containerization process encompasses the entire directory of the project, ensuring that all necessary files are included within the encapsulated environment.

This is accomplished using the following `Dockerfile` commands, which not only transfer the directory but also proceed to install the project into the environment.

```
ADD . nerf-industrial-metaverse
RUN cp instant-ngp/build/pyngp.cpython* /nerf-industrial-metaverse
RUN cd nerf-industrial-metaverse && pip install -e .
```

Code Listing 17: Dockerfile commands for directory transfer and project installation

Developers are afforded the capability to execute the containerized environment using the command `docker run --gpus all --rm -it nerf_industrial_metaverse -v path/to/files/folder:/nerf_industrial_metaverse/videos`. This command initializes the Docker container, allocates all available GPUs (–gpus all) to the container, removes the container upon exit (–rm), and provides an interactive terminal (-it). Additionally, it maps a local files directory to a corresponding folder within the container. Users can thereby interact directly with the containerized application via the command-line interface. The flexibility to run the container either locally or on a server enhances the adaptability and scalability of the project, a feature that proved particularly advantageous during the pipeline implementation phase.

In addition to the command line interface, users can also launch the web application within the Docker container by running the `npm start` command in the `client` folder and `flask run` in the `server` folder. These commands are part of the Node.js and Flask ecosystems, respectively. Once activated, the entire application is accessible via `localhost:3000`, thus offering a cohesive user experience. This configuration provides users with the capability to instantiate the Docker container on a remote server equipped with high-performance GPUs. However, the setup is equally suited for local execution. When the container is run locally, the web application is readily accessible via `localhost:3000` without the need for any additional configurations. If the container is hosted on a remote server, users can still access and interact with the project's pipeline via the web interface. This is made possible through Visual Studio Code configurations that facilitate the redirection of specific ports. As a result, the entire application can be run either remotely on a resource-intensive server or locally on a personal machine, while still offering a seamless web-based interaction.

For local pipeline execution, the application consequently offers two execution modes:

1. Running the Docker container using the command-line interface, which can be done either locally or on a remote server.

2. Running the web application within the Docker container. This can be accomplished in two ways:

    a) By running the container locally, accessible via `localhost:3000`.

b) By hosting the Docker container on a remote server, while still making it accessible locally through port redirection, facilitated by configurations in Visual Studio Code.

For enterprise-wide or global deployments, we have restructured the application into two distinct components: the back-end and the front-end. Specifically, the "client" folder, which constitutes the front-end component, has been decoupled from the original directory and migrated to a separate repository. This separation allows us to host the back-end on a server equipped with an NVIDIA GPU, while the front-end can smoothly operate on a standard server, readily accessible via a simple URL. This approach offers significant advantages. Users can seamlessly explore the world of neural radiance fields without the need for complicated installations or specialized graphics hardware. In addition, the web application's user-friendly design encourages users to experiment with the algorithm, providing a more accessible alternative to complex installations, command line interfaces or script execution. With robust server-client interactions and a detailed logging system in place, we guarantee transparency and active user involvement at every step. This project implementation ensures that users, regardless of their technical prowess, can access and benefit from our solution.

4.4. Implementation of the Pipeline

In this section, we explain the implementation of our automated pipeline shown in Figure 25. Our pipeline is designed to process video data, images and R3D files. When processing video data, we start by extracting individual frames. Following this, camera parameters are ascertained utilizing the computational methodologies provided by COLMAP. Post-completion of these preliminary steps, the derived data undergoes a transformation into JSON format. This JSON-formatted data, in conjunction with the correlated images, constitutes the training dataset for the Neural Radiance Field.

For iPhone users, especially the newer models such as iPhone 12 and its successors, we offer an alternative method of data collection. With the Record3D application, based on ARKit technology, we enable the creation of recordings where the camera parameters are already stored in a metafile. This data is then read from the file and can be converted directly into the required JSON format, eliminating the need for camera calibration via COLMAP. This eases the transition to the training phase and makes the whole process more efficient.

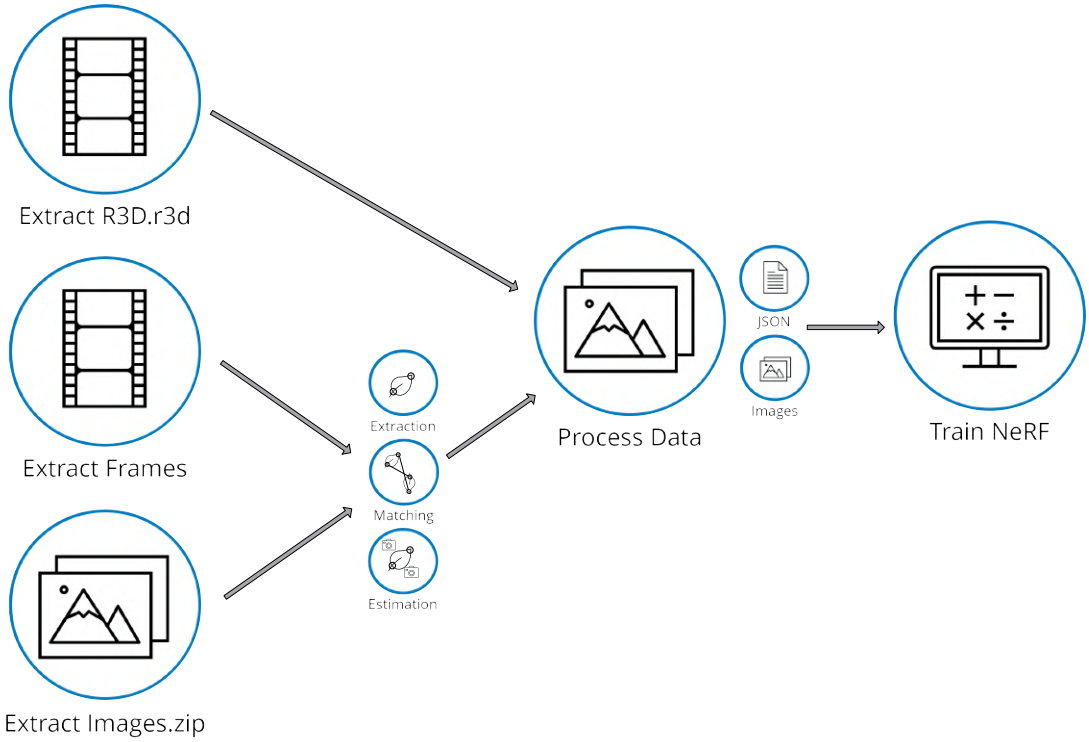


Figure 25.: Comprehensive overview of our automated pipeline: The diagram delineates the distinct phases involved when utilizing either video or R3D files. For video-based workflows, the pipeline commences with frame extraction, progresses through data processing with COLMAP, and culminates in the creation of JSON files. These JSON files, along with the corresponding images, serve as the input for Neural Radiance Field training. Alternatively, R3D files bypass the need for COLMAP and are directly converted into the requisite JSON format, expediting the transition to the training phase.

The provided Listing 18 illustrates the whole pipeline process implemented in Python.

```
cfgs = load_configs_yaml(cfs_path)
if not valid_key_in_cfgs(cfgs, 'iphone'):
  if not osp.isdir(osp.join(scene_dir, "images")):
    extract_video_cmd = ["ngp-extract", "--configs", args.configs]
    extract_video_output = subprocess.check_output(
    extract_video_cmd, universal_newlines=True)
  if not osp.exists(osp.join(scene_dir, "colmap_output.txt")):
    run_colmap_cmd = ["ngp-colmap", "--configs", args.configs]
    run_colmap_output = subprocess.check_output(
    run_colmap_cmd, universal_newlines=True)
  if not osp.exists(osp.join(scene_dir, cfgs.json.name)):
    logger.add_log('--- EXPORT TRANSFORM.JSON ---')
    export_json_cmd = ["ngp-json", "--configs", args.configs]
    export_json_output = subprocess.check_output(
    export_json_cmd, universal_newlines=True)
else:
  if not osp.exists(osp.join(scene_dir, "metadata")):
    exit()
  if not osp.exists(osp.join(scene_dir, cfgs.json.name)):
    export_json_cmd = ["ngp-iphone", "--configs", args.configs]
    export_json_output = subprocess.check_output(
    export_json_cmd, universal_newlines=True)
if osp.exists(osp.join(scene_dir, cfgs.json.name)):
  train_ngp_cmd = ["ngp-train", "--configs", args.configs]
  train_ngp_output = subprocess.check_output(train_ngp_cmd, stdin=None, stderr=None,
    universal_newlines=True)E ---')
```

Code Listing 18: Converting input data to input for the training

The code consists of conditional blocks that execute specific procedures based on the source of input data - either standard video or R3D files from an iPhone. Upon reading and storing configurations from the YAML file, the code first checks for the presence of the "iPhone" key within the configurations (Line 2). If this key is absent, the code proceeds with the COLMAP-based camera calibration routine. Conversely, if the "iPhone" key is detected, the required metadata is read directly from the metafile, thereby bypassing the COLMAP stage. Following the generation of the JSON file, a validation step is incorporated to verify the file's existence. Once this validation succeeds, the training phase is automatically initiated.

Leveraging Poetry's capability for custom script definitions, we've configured various scripts like `ngp-colmap` to correspond to specific functions within our project - `nerf_industrial_metaverse.tools.run_poses:entrypoint`, specifically. These custom scripts are illustrated in Listing 19.

```
[tool.poetry.scripts]
ngp-extract="nerf_industrial_metaverse.tools.extract_video:entrypoint"
ngp-colmap="nerf_industrial_metaverse.tools.run_poses:entrypoint"
ngp-json="nerf_industrial_metaverse.tools.export_json:entrypoint"
ngp-iphone="nerf_industrial_metaverse.tools.record3d_to_json:entrypoint"
ngp-train="nerf_industrial_metaverse.scripts_ngp.run:entrypoint"
ngp-trainvideo="nerf_industrial_metaverse.tools.video_to_training:entrypoint"
```

Code Listing 19: Customized commands for executing the pipeline steps

In order to execute these custom commands within the Python environment, Python's native `subprocess.check_output()` method is utilized, as illustrated in lines 4, 8, 13, 20, and 24 of Listing 18. This approach allows for the seamless incorporation of external commands into the pipeline, resulting in a highly automated workflow.

With these script definitions in place, users have the flexibility to execute each step of the pipeline individually within the Docker container, as specified by the various `ngp-*` commands. Alternatively, the `ngp-trainvideo` command can be used to execute the entire block of code outlined in Listing 18. To further aid understanding, the following sections provide a detailed explanation of each pipeline step. This deep dive is intended to provide the reader with a thorough understanding of how each phase contributes to the overall generation of neural radiance fields.

4.4.1. Extraction of the input data

The initial stage of our pipeline is centered around data extraction, encompassing three primary tasks: extracting frames from a video, unpacking images saved in a `.zip` file, and managing `.r3d` extraction. Each of these tasks prepares the input data for the subsequent phases of the pipeline.

The video extraction is handled by the script `extract_video.py`. This script offers configurable parameters, specifically the `video downsampling factor` and `maximum frames`, both of which are configurable via the `YAML` file. These parameters are designed to adapt to the limitations and capabilities of the available GPU. Specifically, these parameters ensure that the GPU's memory is used efficiently, thereby avoiding memory allocation errors. For instance, in the case of larger videos, users can increase the downsampling factor to ensure that the entire scene is captured without overloading the GPU. On the other hand, for shorter videos, a lower downsampling factor can be selected to maximize image extraction. The process continues until it either reaches the set maximum frame count or completes the video.

The video extraction function begins by verifying the file's existence and then employs OpenCV's (OpenCV, 2023) `VideoCapture` class for frame iteration. Frames are saved in the "xxxxx.jpg" format, where each 'x' is a zero-padded digit. The process concludes by releasing the video capture object, freeing up valuable system resources.

For image and `.r3d` files, the extraction process is straightforward and sets the stage for subsequent processing steps.

4.4.2. Estimate camera poses from images with COLMAP

To determine the viewing direction for each image - a essential input for NeRF - we leverage Structure-from-Motion (SfM) (MathWorks, 2023) algorithms. These algorithms facilitate camera calibration, which is pivotal for obtaining both intrinsic

and extrinsic camera parameters.

The intrinsic parameters - focal length, principal point coordinates, and lens distortion coefficients - elucidate the internal configuration of the camera. Meanwhile, extrinsic parameters, encompassing rotation and translation matrices for each image, specify the camera's relative orientation and position within a scene (Bradski and Kaehler, 2011).

For this purpose, our research utilizes COLMAP, an open-source tool that robustly implements SfM algorithms (Schönberger and Frahm, 2016). Triggering the `ngp-colmap` command initiates a sequence of COLMAP sub-commands, each corresponding to a distinct stage in the SfM pipeline, ranging from feature extraction to bundle matching. To deepen the reader's understanding of SfM's role in camera calibration, we have detailed these algorithms in Appendix D.

For an overview of the SfM process we visualized the pipeline in Figure 26. The SfM pipeline in COLMAP starts with a collection of images covering different perspectives of a scene. In the next phase, called "correspondence search", characteristic features of an image are extracted. These features are then matched and their geometric consistency is verified. Once the corresponding features are identified, the system enters the "Incremental Reconstruction" phase. Here, the process begins with initialization, followed by image registration, triangulation of feature points, and finally bundle matching to optimize the $3D$ structure and filter out outliers. The pipeline ensures systematic and efficient calibration of the camera and reconstruction of the scene (Schönberger and Frahm, 2016).

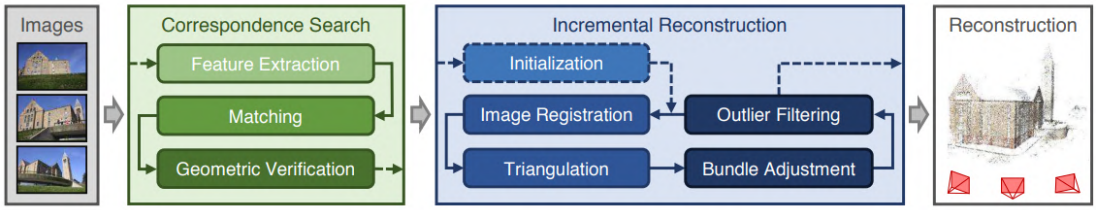


Figure 26.: The COLMAP pipeline starts with a set of images and proceeds through various stages including correspondence search, feature extraction, matching, and geometric verification. It then moves on to incremental reconstruction, encompassing initialization, image registration, triangulation, bundle adjustment, and outlier filtering.

To offer visual insights into this complex process, we employ COLMAP's GUI, which illustrates sparse scene reconstruction and indicates the spatial positions of successfully reconstructed cameras, as shown in Figure 27.

Upon completion, COLMAP outputs three text files that are crucial for subsequent analyses: *cameras.txt*, *images.txt*, and *points3D.txt*. Specifically, for our next steps, we utilize the *cameras.txt* and *images.txt* files, which contain the intrinsic parameters

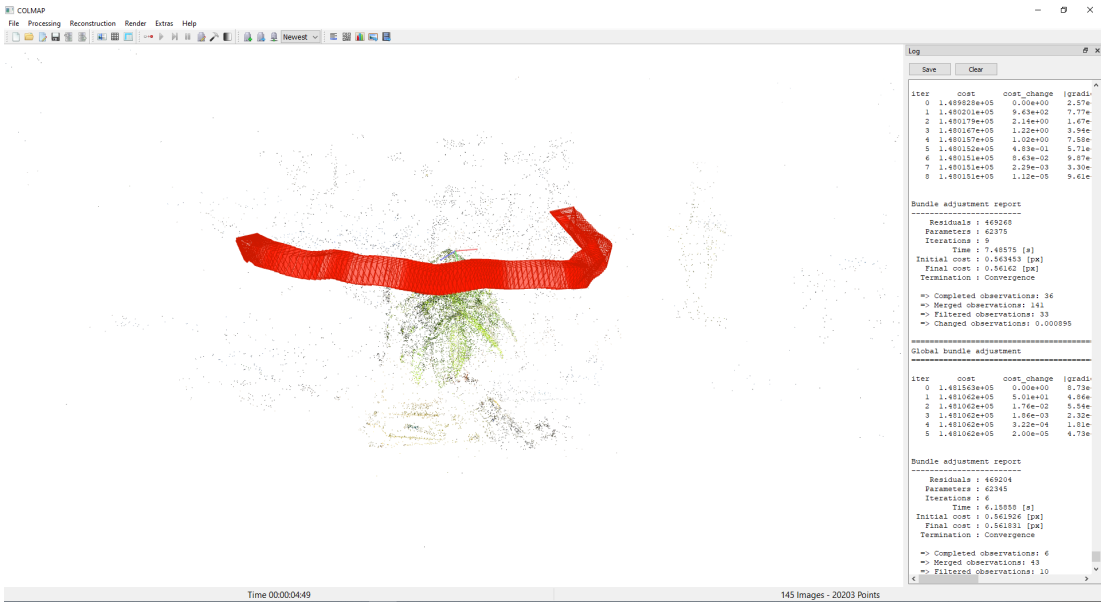


Figure 27.: Screenshot of the COLMAP GUI after Pose Estimation: Red symbols denote reconstructed cameras, while the sparse point cloud illustrates estimated 3D spatial points.

and the pose and keypoints of all reconstructed images, respectively.

4.4.3. Generating the JSON Input

Having already saved a set of images for our training input, our next goal is to create a JSON file in which we store the intrinsic camera parameters and, for each existing image, also the corresponding extrinsic camera parameters. To create this file, we run our customized script `export_json.py`. The resulting JSON file is structured as follows 20.

```
{
  "camera_angle_x": 0.7599288285742521, // The camera angles of view in radians
  "camera_angle_y": 1.2336458746664418, //
  "fl_x": 1072.0, //  The focal lengths in pixels for the x and y axes
  "fl_y": 1068.0,
  "cx": 553.0, // The principal points (x,y) in pixels, representing the image center
     coordinates
  "cy": 969.0,
  "w": 1080, //  The image width w and height h in pixels
  "h": 1920, //
  "k1": 0.0312, // Distortion parameters (k) used for radial distortion correction
  "k2": 0,
  "k3": 0,
  "k4": 0,
  "p1": 0, // Distortion parameters (p) used for tangential distortion correction
  "p2": 0,
  "aabb_scale": 4,
  "frames": [] // A section where per-frame extrinsic parameters are specified
}
```

Code Listing 20: Input file for training neural radiance fields with instant-ngp

The parameters `camera_angle_x` and `camera_angle_y` denote the horizontal and vertical fields of view of the camera, respectively. In essence, these angles define the angular span that the camera captures, representing how much of the scene is visible to the lens. The fields `fl_x` and `fl_y` denote the focal length for the x and y axes. The focal length `fl_x` and `fl_y` specify the focal length for the x and y axes. The focal length gives information about the angle of view (how much of the scene is captured). Every camera sensor possesses a central point, often referred to as the 'principal point.' In our configuration, this is denoted by `cx` and `cy` parameters, representing the image center in pixel coordinates. The resolution of the image is captured by `w` and `h`, indicating its width and height in pixels. A common issue in photography is radial distortion, where images get distorted, especially at the edges. The parameters `k1` and `k2 - k4`, are coefficients of a polynomial that helps in rectifying this distortion. Similarly, tangential distortions can be described using `p1` and `p2`. Such a detailed calibration allows the transformation from $3D$ world points to $2D$ image points (Bradski and Kaehler, 2011).

Extrinsic camera parameters provide information about the position and orientation of a camera in a $3D$ space. In our case, these parameters are represented in the form of a transformation matrix, denoted as $M = [R|t]$. Here, R, represents a 3x3 rotation matrix, and t represents a 3x1 translation vector. To explain further, the rotation matrix R adjusts the camera's coordinate system to the world's standard coordinates. In parallel, the translation vector t moves the camera's origin to coincide with that of the world. Given the context of homogeneous coordinates, the transformation matrix M contains an additional component that expands it into a 4x4 matrix. The advantage of using a 4x4 matrix in homogeneous coordinates lies in its efficiency: it consolidates rotation and translation into a single matrix multiplication, thus streamlining the transformation process. Below is a representative example of how the transformation matrix is structured within the JSON file.

```
{
  // ...
  "frames": [
  {
    "file_path": "images/frame_00001.jpeg",
    "transform_matrix": [
    [-1.0, 0.0, 0.5, 0.0],
    [0.0, 1.0, 0.0, 0.0],
    [0.0, 0.0, 1.0, 1.0],
    [0.0, 0.0, 0.0, 1.0]
    ]
  }
```

Code Listing 21: Example of extrinsic parameters stored in the JSON file

In this structure, the field `frames` defines an array, where each entry corresponds to a particular frame. Within each frame, the `file_path` provides the path to the image

associated with the frame. Following the file path, we have the `transformation_matrix`, which captures the extrinsic parameters.

Converting the extrinsic parameters calculated by COLMAP into the transformation matrix suitable for instant-ngp necessitates several mathematical manipulations. This complexity arises because COLMAP stores extrinsic information in Quaternion format (QW, QX, QY, QZ), while instant-ngp employs its own set of transformation conventions, as demonstrated in Equation 4.1. For a more in-depth mathematical dissection of this transformation matrix, the reader is directed to Appendix E.

$$C2W_{transformed} = \begin{bmatrix} +X_0 & +Y_0 & +Z_0 & Z \\ +X_1 & +Y_1 & +Z_1 & X \\ +X_2 & +Y_2 & +Z_2 & Y \\ 0.0 & 0.0 & 0.0 & 1 \end{bmatrix} \tag{4.1}$$

The `transforms.json` file serves not only as a repository for essential camera parameters but also as a mechanism to introduce additional parameters that fine-tune both the training efficiency and the visual output quality of the NeRF model. The `aabb_scale` parameter is a key ngp-specific variable. Set to 1 by default, it defines the spatial extent of the scene. This value is particularly relevant when the scene incorporates elements that extend beyond the initial bounding box area, as it expands the ray projection onto a larger bounding box. A higher value, up to a maximum of 128, is recommended for natural scenes. The `scale` field allows for a global scaling of the entire scene. While the default value is 0.33, it can be adjusted depending on the specific requirements of the dataset. The `offset` array provides displacements along the X, Y, and Z axes, defaulting to [0.5, 0.5, 0.5]. This option is particularly useful when the camera positions are not optimally situated within the unit cube and require adjustment. All these parameters can be directly modified within the `transforms.json` file, obviating the need to rerun any pre-existing scripts.

4.4.4. Training Neural Radiance Fields with instant-ngp

The training of neural radiance fields is initiated using a pre-generated `transform.json` file, in conjunction with the images that have been previously extracted. The pipeline triggers an adapted script for this purpose. The script is based on the GitHub repository of NVLAB (NVlabs, 2022) and is invoked using the `ngp-train` command.

Configuration parameters for the training can be customized using a YAML file, which we have adapted from the original instant-ngp parameters. While the default neural network architecture and encoding settings are immutable in this script, specific parameters such as `training_steps` can be adjusted to better suit the project's needs.

Upon the successful completion of the training, the neural radiance field model

is saved as `.ingp` file extension. Additionally, the configurations can be modified to include options for rendering screenshots and videos, or extracting mesh geometries.

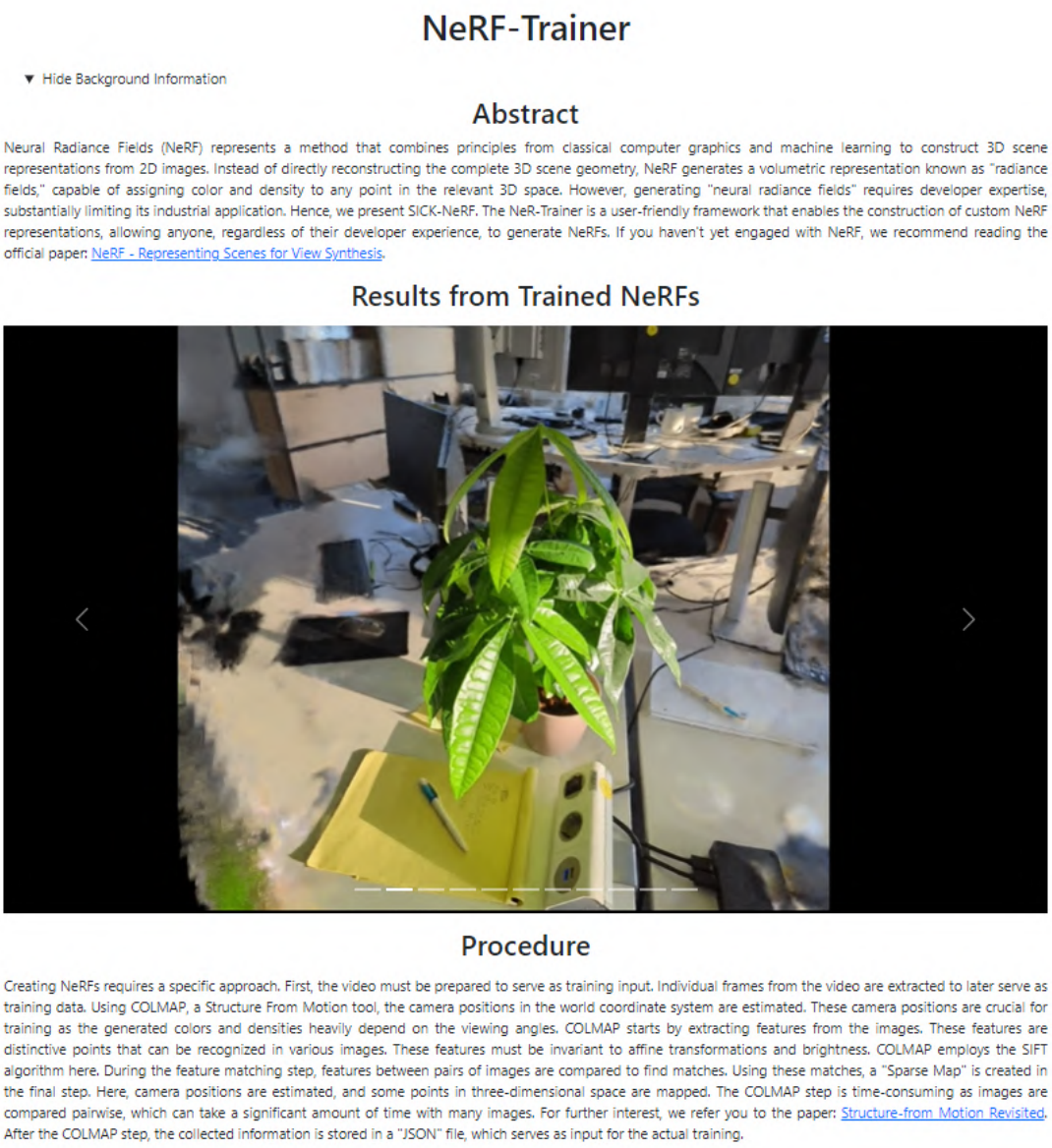


Figure 28.: Rendered view of the background information section on the website. This section includes an introduction to what functionalities the website offers, followed by examples of previously trained NeRF models for qualitative evaluation, and concludes with an outline of the NeRF training pipeline process.

4.5. Implementation of the Website

To facilitate greater accessibility and user-friendliness, we extended the pipeline described in previous sections to include a web-based interface. This online platform serves two main purposes: first, to offer an accessible medium for non-developers, and second, to eliminate the need for local installations, manual configurations, and specific hardware requirements such as a local NVIDIA graphics card. This section

details the architecture, technologies, and functionalities of the web-based interface.

4.5.1. Website Structure

The website is designed to initially provide background information on the training process, clarifying the functionalities available on the platform. Following this introduction, we present results from neural radiance fields to offer users an opportunity for qualitative evaluation. These results also serve to inform users who may not intend to train a NeRF but are interested in understanding its capabilities. Subsequently, the pipeline process for NeRF training is elaborated. Figure 23 displays the section that provides the background information on the website. Although the website was originally written in German, the content is reproduced in English in this context.

These background details can be toggled to focus on the core functionalities of the website, which pertain to NeRF training. Initially, the system verifies the connection status with the back-end server. Subsequently, additional information and specifics about the training process are presented within a distinct card element, as shown in Figure 29. This card also contains an upload field for submitting the data with which to train a neural radiance field.

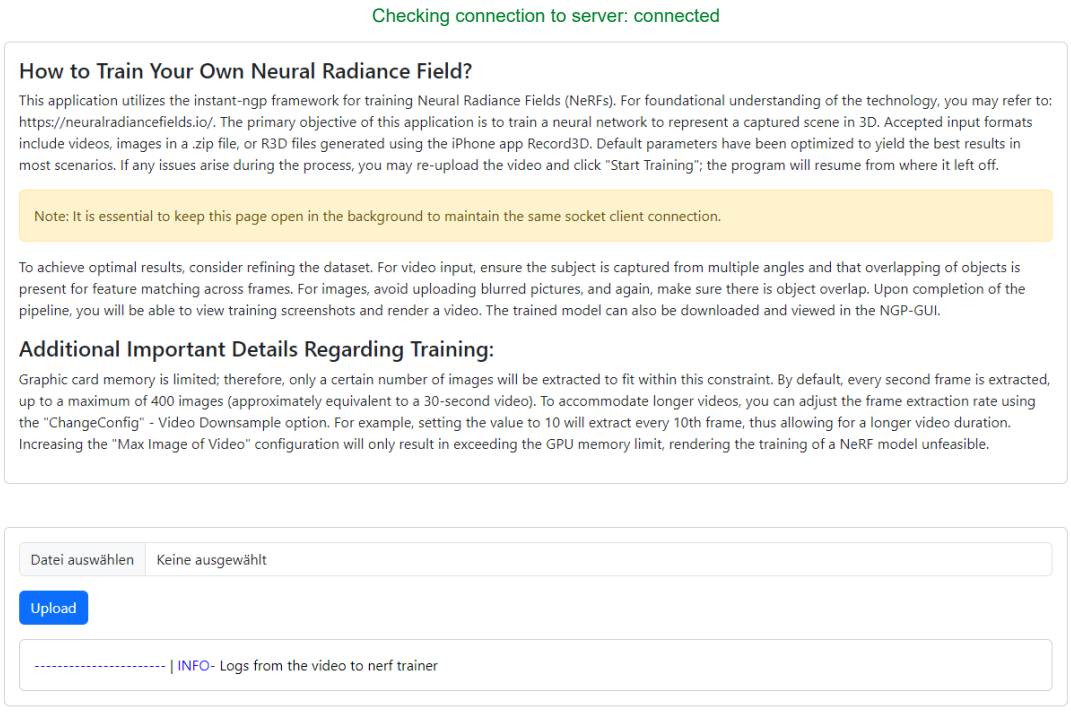


Figure 29.: Rendered view of the NeRF training interface on the website. This section allows the user to toggle background information for a focused experience. It incorporates initial server connection checks, elaborates on the nuances of the training process within a dedicated card layout, and includes an upload field for submitting training data.

At the bottom of the web interface, additional accordions offer extended functionalities for optimal user interaction, as illustrated in Figure ??. One accordion contains a sketch illustrating both good and bad camera motions to guide users in capturing scenes effectively. Another accordion features a 'Terminate' button designed to interrupt any ongoing processes running in the background. This feature can be particularly useful if the connection between the front-end and the back-end is lost. This button is intended primarily for easy testing during deployment and is not a permanent feature of the interface.

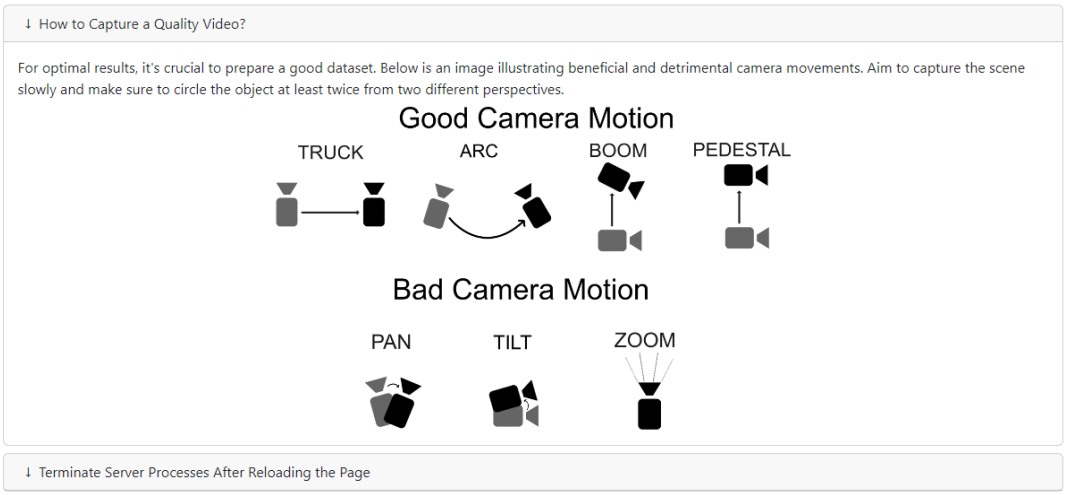


Figure 30.: The section presents optional information for capturing a video and offers an interface for user terminate the COLMAP process.

In summary, the website serves as an integrated platform that offers a robust set of functionalities for training neural radiance fields. It not only provides background information to assist both novice and experienced users but also showcases previously trained NeRF models for qualitative evaluation. The site is designed with user experience in mind, featuring an intuitive interface and offering detailed instructions on various aspects of the training pipeline. One of the website's main strengths is its user-friendly design, allowing individuals with varying levels of expertise to train their own NeRF models. Moreover, the website is engineered to handle training processes efficiently, providing real-time updates to the users and allowing them to terminate server-side processes if necessary.

4.5.2. Training Management via the Web Interface

After successfully uploading the necessary data, two buttons appear next to the upload button: "Start Training" and "Modify." By clicking on the "Modify" button, users are given the opportunity to adjust a variety of parameters related to the training process. This interactive feature ensures that users have the flexibility to fine-tune

settings before initiating the training. An overview of these functions is provided in Figure 31.

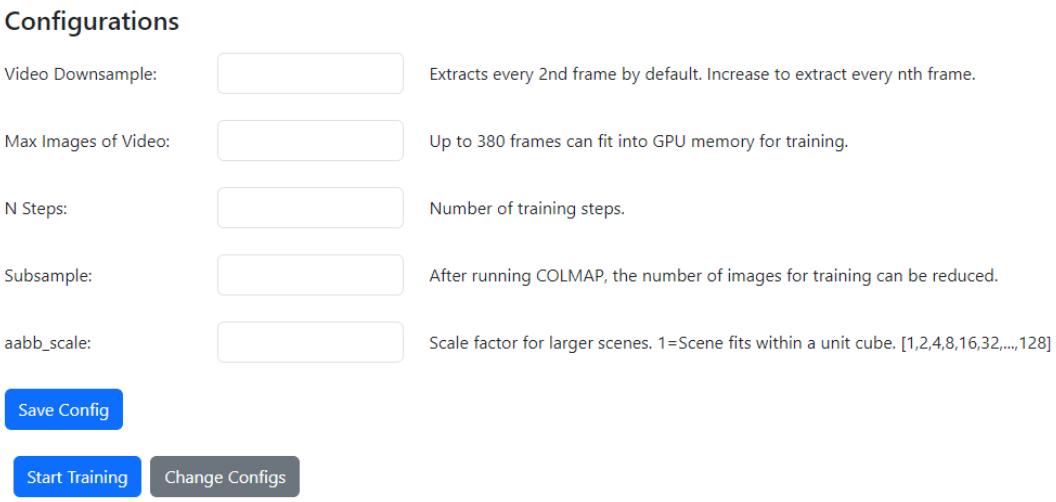


Figure 31.: A detailed view of the configuration options available on the web interface. These settings include 'Video Downsample' for frame extraction frequency, 'Max Images of Video' for GPU memory limitations, 'Subsample' for refining the image set after running COLMAP, and 'aabb_scale' for scene scaling.

After the training concludes, the pipeline automatically displays screenshots on the front-end, offering a visual validation of the successful training. On this interface, users can either opt to render videos following a pre-defined camera path or download the trained `.ingp` file. Additionally, users can export a mesh, enabling them to utilize the generated $3D$ geometry. This possibilities require the addition of particular keys, followed by the re-initiation of the `ngp-train` step with the updated YAML parameters.

```
keys_to_remove_list = [
['train.save_snapshot'],
['train.save_mesh'],
['train.screenshot_transforms', 'train.screenshot_dir']
]
new_config = {
  'train.video_output': configurations['video_output'],
  'train.load_snapshot': configurations['snapshot'],
  'train.video_camera_path': "configs/base_cam.json"
}
```

Code Listing 22: Keys for rendering a video from a trained snaphot

Essentially, this process entails the elimination of specific tags and the integration of new ones, which ensures the `run-ngp.py` script aligns with the updated configurations.

Figure 32 shows the web page after a successful training.

The website is thus an easily accessible and user-friendly platform specifically designed for neural radiance field training. It successfully simplifies the typically complex process of neural network training and makes it accessible to users with varying levels of technical expertise. The website provides an intuitive interface where users

can upload data and either begin training immediately or make simple but effective changes to various parameters. In this way, a harmonious balance between usability and customization is achieved, allowing users to tailor the training process to their specific needs.

4.6. Results of neural radiance fields in capturing reality

Before delving into the implications of NeRF for the Industrial Metaverse, we present empirical results from our website to demonstrate the efficacy and speed of the instant-ngp algorithm in generating novel views with our Website. For these experiments, we utilized videos of approximately 30 seconds in duration, yielding 200-300 frames after extraction, as the input for our models. In terms of computational time, the COLMAP stage consumed about 2 hours, while the subsequent training phase was notably completed in less than 2 minutes. In addition to the impressive speed of the training phase, the quality of the rendered images is also noteworthy. The generated views were selected to be close approximations of, but not identical to, the original input frames. Despite this, the algorithm succeeded in producing photorealistic renderings that captured both the geometry and the texture of the scenes with remarkable accuracy. This fidelity in representation is evident in Figure 33, which showcases several instances of synthesized views alongside their corresponding input frames for comparison. The high quality of these renderings reinforces the utility of the instant-ngp algorithm as an effective tool for generating $3D$ representations of static scenes, suitable for applications in the Industrial Metaverse.

4.7. Conclusion

In conclusion, the architecture detailed in this chapter signifies a comprehensive, end-to-end implementation that encapsulates the entire data flow – from initial data input to the training of neural radiance fields. This implementation, comprising system-independent installations, streamlined package management, efficient data upload processes, training mechanisms, and real-time monitoring, is brought together seamlessly through an intuitive web interface. Notably, this pipeline represents the first of its kind, pioneering an approach that integrates the initial data input through to the training of Neural Radiance Fields. By amalgamating diverse tools such as Poetry, Docker, CUDA, and front-end technologies like Flask and React, we have crafted a robust system that not only ensures efficiency but also greatly enhances user accessibility and experience.

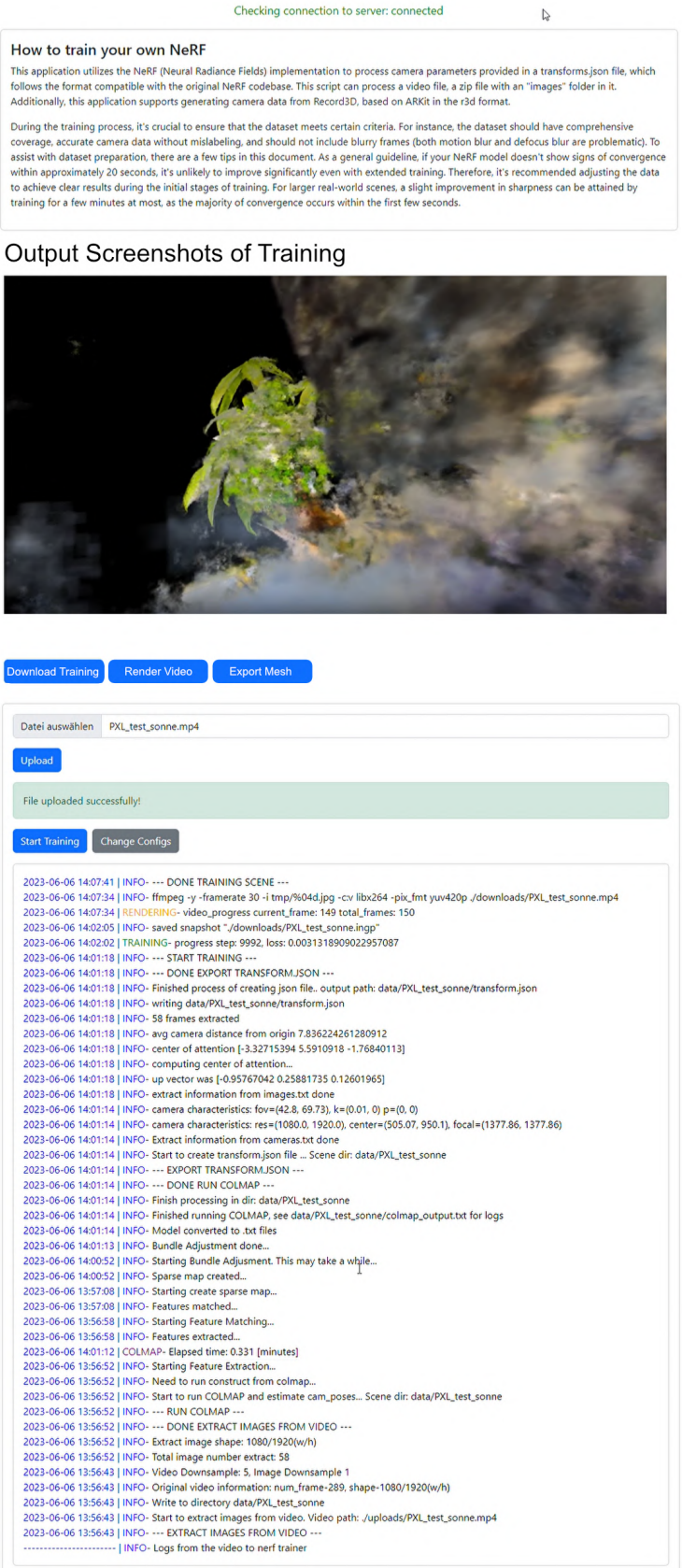


Figure 32.: This figure presents the user interface on the webpage, encompassing backend logs. Through a socket connection, users receive ongoing updates, ensuring they remain updated about the training's progression and outcome. Such transparent feedback and live interactions elevate the user experience, making the entire training journey more intuitive. [For dynamic representation, see accompanying website videos.]

Figure 33.: The figure illustrates NeRF's prowess in generating $3D$ representations of static scenes. Derived from shorter videos, the COLMAP computations for these scenes lasted approximately 2 hours, with an impressive training duration of less than 2 minutes. The displayed renderings are intended to be close approximations, but not exact replicas, of the original input views. Despite minor discrepancies, the outcomes are highly commendable. [For dynamic representation, see accompanying website videos.]

5. Neural Radiance Fields in the Context of the Metaverse

The introduction of the NeRF algorithm marked the birth of the domain of Neural Scene Rendering, a subfield within computer graphics focused on generating $3D$ scenes using neural networks (Tewari et al., 2020; Xie et al., 2022). In general, neural scene rendering is a rapidly growing field with the potential to revolutionize the way we generate $3D$ content, offering data-driven approaches that make the process more accessible and efficient than traditional methods.

Despite the comprehensive research highlighted in Chapter 2.2.4, defining practical applications for this innovation has proven challenging. This chapter addresses these challenges by focusing on the potential applications of NeRF within the Industrial Metaverse. Throughout this examination, it is important to keep in mind that it is not about how perfect the results are, but about the potential of those results.

To delve deeper into this subject, we begin by exploring the concept of capturing reality in an industrial setting. Building on that foundation, we investigate complicated scenarios where traditional methods like photogrammetry fall short, revealing NeRF as a compelling alternative. We will then simulate two scenarios in the context of the Industrial Metaverse to determine the efficiency of NeRF in these scenarios. We round out this chapter by highlighting the specific limitations of NeRF that make it difficult to apply in practice and provide an overview of ongoing research efforts aimed at addressing these very challenges.

5.1. Capturing Reality at a single moment

The NeRF algorithm represents a static scene as a volumetric representation that captures intricate details, depth information, and light interactions. With the idea of an accurate representation of the real environment in a virtual environment, we can explore the possible use cases, especially in capturing the reality at specific times. Equally important to mention, but outside the scope of our current context, is the ability to continuously capture and update dynamic real world scenarios. This ensures near real-time synchronization with dynamic changes in the physical environment.

With the ability to capture reality so accurately, we are now turning our attention to the many applications that arise from this accurate mapping of the physical world. We would like to note that these options refer specifically to capturing reality in real-world conditions, while the options we mentioned in Chapter 2.3.3 refer to building

digital twins in the Industrial Metaverse.

Precision and Fidelity The ability to capture intricate details, including lighting, spatial relationships, and environmental factors, increases the accuracy of the virtual rendering. This is a universal feature that spans multiple applications and ensures that all tasks performed in virtual space have been reliably transferred from the real world.

Continuity of Operations The virtual environment's replication of real-world scenarios allows for a seamless continuity of operations across diverse settings, be it production lines, machinery, or other complex processes. This capacity for uninterrupted operation translates not just to heightened efficiency, but also to substantial cost savings, as it negates the need to halt or modify activities in the real world.

Remote Functionality By precisely capturing the nuances of real-world environments, the virtual space offers an unparalleled level of remote functionality. This replication allows for accurate operation, analysis, and innovation from afar, effectively eliminating the need for physical presence and consequently reducing associated expenses.

Iterative Refinement The digital twin's exact replication of real-world conditions allows for a cycle of continuous optimization and adaptation. This enables stakeholders to make well-informed adjustments in the virtual setting before executing changes in the physical realm, thus ensuring both effectiveness and risk minimization.

Interactivity and Engagement Leveraging the high-fidelity capture of real-world scenarios, virtual environments elevate user engagement, whether in sales presentations or training modules. This heightened sense of immersion not only improves the effectiveness of training but also deepens comprehension and enhances communication in customer-facing interactions.

Comprehensive Documentation The capture of real-world conditions in the virtual environment allows for an enriched form of documentation that goes beyond traditional text-based manuals. Because it faithfully replicates the actual setting, users can interact with a virtual guide that is both contextually accurate and readily accessible. This enhances understanding and retention of information, making it especially beneficial in complex scenarios where a deep understanding of the environment is crucial.

Even though the above applications show promising possibilities, it is important to note that these ideas are primarily based on brainstorming the idea of capturing reality. Although NeRF has showcased remarkable proficiency in producing high-quality $3D$ scene representations, the algorithm, in its current form, presents certain constraints that may challenge the realization of these concepts. This chapter will delve deeper into these limitations and the ongoing research addressing them. The practical viability of these ideas awaits further exploration and validation.

5.2. NeRF and it's ability to handle complex scene conditions

A distinctive strength of NeRF lies in its capability to handle complex material properties, which has historically been a challenge in computer graphics and $3D$ reconstruction. We conducted experiments with a selection of challenging objects to evaluate the capability of NeRF in faithfully and realistically representing them.

5.2.1. Generating Avatars with NeRF

Human skin, with its inherent subsurface scattering where light penetrates and scatters within, brings its own set of complexities. This phenomenon, important for creating lifelike human representations, has always been a challenge to capture authentically. Furthermore, the dynamic nature of clothing, with its folds, wrinkles, and drapes, poses another layer of complexity for traditional computer graphics (CG), as does the task of realistically depicting the intricacies of the human face, from the lips to the eyes and the subtleties of expression. We evaluated the NeRF algorithm's ability to manage these complexities with the objective of generating avatars suitable for remote collaboration, training, and interactive experiences within the Industrial Metaverse. Utilizing NeRF for avatar creation enables the generation of virtual experiences that closely mimic human presence. By using NeRF to generate avatars, it is possible to create virtual experiences that authentically replicate human presence. Figure 34 displays a human neural radiance field captured with Record3D. This method allowed us to prepare the data input and train this model within 2 minutes, which is quite impressive.

In this context, it is important to note that while the neural radiance field is really good at capturing the detail and illumination of this human, it is essentially a static representation that does not provide the ability to simulate rigid body dynamics or animation. This is a limitation when trying to recreate the movements and dynamics of human figures in real time.

The instant-ngp algorithm was designed for static scenes. If the scene changes slightly during a shot, this leads to visible distortions in the resulting $3D$ representation, as shown in Figure 35.

Figure 34.: This Figure presents a neural radiance field of a human avatar captured with Record3D. Remarkably, the entire process of data preparation and model training took a mere 2 minutes. While there are some areas that present challenges, the overall representation of the avatar as a neural radiance field is commendably accurate and realistic. It is also worth noting that this avatar is a static representation; it lacks a rigid body or other physical components, limiting its range of motion and interaction. [For dynamic representation, see accompanying website videos.]

Figure 35.: This illustration underscores the challenge posed by the minimal motion and dynamics of humans. While NeRF is adept at capturing static scenes, even subtle movements can introduce significant distortions to the representation. Despite these intricacies, the resultant avatar in the neural radiance field remains notably accurate and lifelike. [For dynamic representation, see accompanying website videos.]

While instant-ngp falls short in addressing the dynamic aspects of avatars specific to the Industrial Metaverse, alternative algorithms exist that specialize in dynamic rendering. For instance, *NeuralBody* (Peng et al., 2021b) is adept at reconstructing a moving human figure from a single-camera video, while *AniNeRF* (Peng et al., 2021a) is tailored to produce animatable human models from multi-view videos. Recent research by Gafni et al. (2021) even extends the scope to capturing both the appearance and dynamics of the human face (Gafni et al., 2021). As it stands, selecting an algorithm specialized for the particular requirements of each use case remains imperative for optimal results.

5.2.2. Generating hands with NeRF

Generating $3D$ representations of hands from images is inherently challenging due to the intricate anatomy and vast range of possible hand poses. Each hand possesses unique features, from varying finger lengths to distinct knuckle shapes. Furthermore, hands can adopt numerous poses, and during many of these poses, fingers or parts of the hand can overlap or hide other parts, leading to self-occlusion. This occlusion makes it difficult to reconstruct the $3D$ representation from $2D$ images as not all parts of the hand are always visible. Adding to this, hands often display fine details, such as wrinkles and fingernails, that are important for a realistic representation but can be challenging to capture from images. Interaction with other objects, skin deformation during movement, and changing lighting conditions further complicate the modeling process. In the Industrial Metaverse, $3D$ models of human hands can be used to perform safety assessments. For example, the intrusion of a human hand into a production line can be simulated to evaluate the effectiveness of existing safety protocols and mechanisms, and also to show the customer what happens when an employee's hand gets into the equipment.

Figure 36 illustrates our result of training a neural radiance field with the instant-ngp algorithm based on the video of a hand.

Figure 36.: This figure illustrates an attempt to create a neural radiance field of a hand using Record3D. Notably, influences from varying light conditions posed challenges to the modeling process. The result, while showcasing the potential of NeRF, also underlines the intricacies and difficulties inherent in accurately capturing and representing hands. [For dynamic representation, see accompanying website videos.]

As previously noted, the generated neural radiance field merely constitutes a vol-

umetric scene function; it lacks physical properties and a rigid body structure. Consequently, this generated radiance field currently has no capability for interaction or dynamic behavior.

5.2.3. Generating large scenes with NeRF

The instant-ngp scripts operate under the assumption that all training images are oriented towards a common focal point, which is positioned at the origin. The determination of this focal point is achieved by taking a weighted average of the nearest points of convergence between the rays passing through the central pixel of all pairs of training images. This inherently implies that optimal results are obtained when the training images are captured with their focus inwards towards the object, although a complete 360-degree view isn't mandatory (NVlabs, 2022). However, driven by the urge to push the boundaries of this tool, we embarked on an exploratory mission to determine its efficacy in capturing larger scenes.

In our experiment, we trained two larger scenes, with the data preparation stage involving COLMAP taking a significant amount of time—approximately 12 hours. However, the actual training was impressively fast, concluding in merely 5 minutes. This highlights the efficiency of the instant-ngp algorithm once provided with the requisite input data. Figure 37 depicts a room featuring information boards. The first image in this Figure displays the entire neural radiance field within its specified *aabb_scale* parameter, a critical instant-ngp-specific setting that determines the extent of the scene. The default *aabb_scale* value is set to 1, which scales the scene so that the camera positions maintain an average distance of 1 unit from the origin. This default setting is optimal for small synthetic scenes like the original NeRF dataset, ensuring rapid training. However, in natural scenes with backgrounds that extend beyond this bounding box, NeRF models can struggle and may generate artifacts known as "floaters" at the box boundaries. As the rendering progresses frame by frame, it zooms in on the information panels, revealing the distinct lighting conditions in the room. While the neural radiance field effectively captures the spaciousness of the room, it does suffer from noticeable artifacts and a lack of sharp focus on the information panels. With this experiment, we wanted to simulate a room to put the user in a virtual customer center where they can explore the products on display in a VR environment.

In a further exploration of the instant-ngp algorithm's capabilities, we captured another expansive environment: an office space featuring two desk blocks. The resulting renderings, which demonstrate the detailed aspects of the scene, are presented in Figure 38. While the capture of such a complex environment is not flawless, the final rendering is notably impressive, offering a realistic recreation of the office. This radiance field allows users to experience the space through VR, gaining an under-

Figure 37.: This figure shows renderings of a large scene. The first image shows the entire scene within its bounding box. Subsequent renderings dive deeper into this radiance field and show each captured information platform. The overall result turned out very well except for the artifacts. [For dynamic representation, see accompanying website videos.]

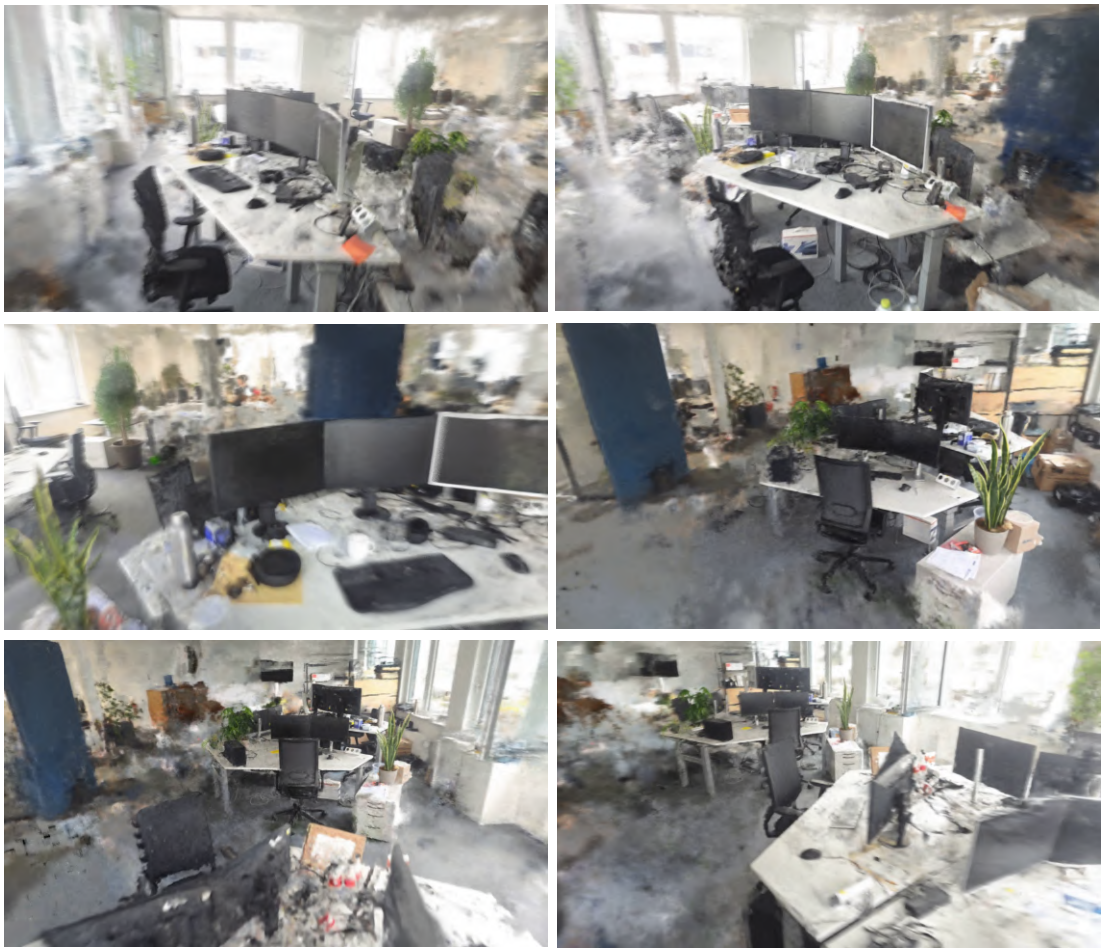


Figure 38.: This figure showcases renderings from an office setting with two distinct desk blocks. The initial four images highlight the first desk block, while the remaining two focus on the second. Despite the presence of artifacts, the results are commendable. [For dynamic representation, see accompanying website videos.]

standing of NeRF capabilities within the context of an environment they are already familiar with.

5.3. Leveraging NeRF in User-Centric Applications

In chapter 2.3.3, we shifted the focus from a product-centric approach to an application-centric approach. Rather than sifting through static images and product descriptions on a company's website, users can generate neural radiance fields to represent their specific environments and challenges. This virtual representation enables a deeper exploration and testing of sensor solutions in real-world conditions, assuring the identification of the most fitting technologies. Beyond mere integration, users can employ these neural radiance fields to develop virtual training modules, experiment with new methodologies, and optimize their specific applications.

Users not only have the option to generate a neural radiance field based on their unique application needs, but they can also do so for the sensor technologies they

are considering. This allows application specialists to virtually insert the sensor into a digital twin of the environment, evaluating its performance and compatibility in various conditions to create the most effective solution. Additionally, Augmented Reality (AR) technology can help visualize how these sensor solutions seamlessly integrate into the user's specific setting.

In the following sections, we will conduct a practical examination, generating neural radiance fields across a range of user environments and sensor technologies.

5.3.1. Neural Radiance Fields for industrial environments

Users can convert their industrial environments into a neural radiance field, allowing them to embed and evaluate potential product integrations within their unique environment. This provides a tangible way to visualize how these products would function and integrate within their specific contexts.

In a pilot test, we transformed a production facility into a neural radiance field using footage captured with a smartphone. The rendered images, illustrated in Figure 39, demonstrate NeRF's aptitude in capturing spatial intricacies and attributes of the facility with high fidelity. For immersive visualization, the instant-ngp framework supports integrating the product's application into VR/AR environments. Upon connecting our headset to Steam VR, we initiate the VR mode using the "Connect to VR/AR headset" button. This immersive VR setting equips users with multiple navigation and interaction tools, as delineated by NVlabs (2022). They can seamlessly traverse the environment, modify the camera's vantage point, rotate, zoom and even erase points from the neural radiance field. Figure 40 shows the GUI of the framework, in which the rendered VR views are displayed.

The current version of instant-ngp primarily provides a viewing platform for neural radiance fields. Although it allows users to visualize results, adjust camera positions for video rendering, and navigate through scenes, it falls short in offering tools to modify these radiance fields. This lack of editability prevents us from our idea of embedding products in a neural radiance field.

Despite this challenge, we were determined to test this approach, and to use the resources currently available in the framework. Therefore, we turn our attention to the rendering capabilities provided by instant-ngp. instant-ngp offers considerable flexibility in the choice of camera settings for image rendering. This means that users can adjust various parameters related to camera placement, orientation, field of view and focal length to create images from different perspectives.

In an exploratory scenario, consider an user managing a complex logistics environment replete with packages requiring continuous tracking via barcode reading. The individual is interested in evaluating the efficacy of various sensor technologies, such as cameras, in resolving this specific application challenge. An innovative application

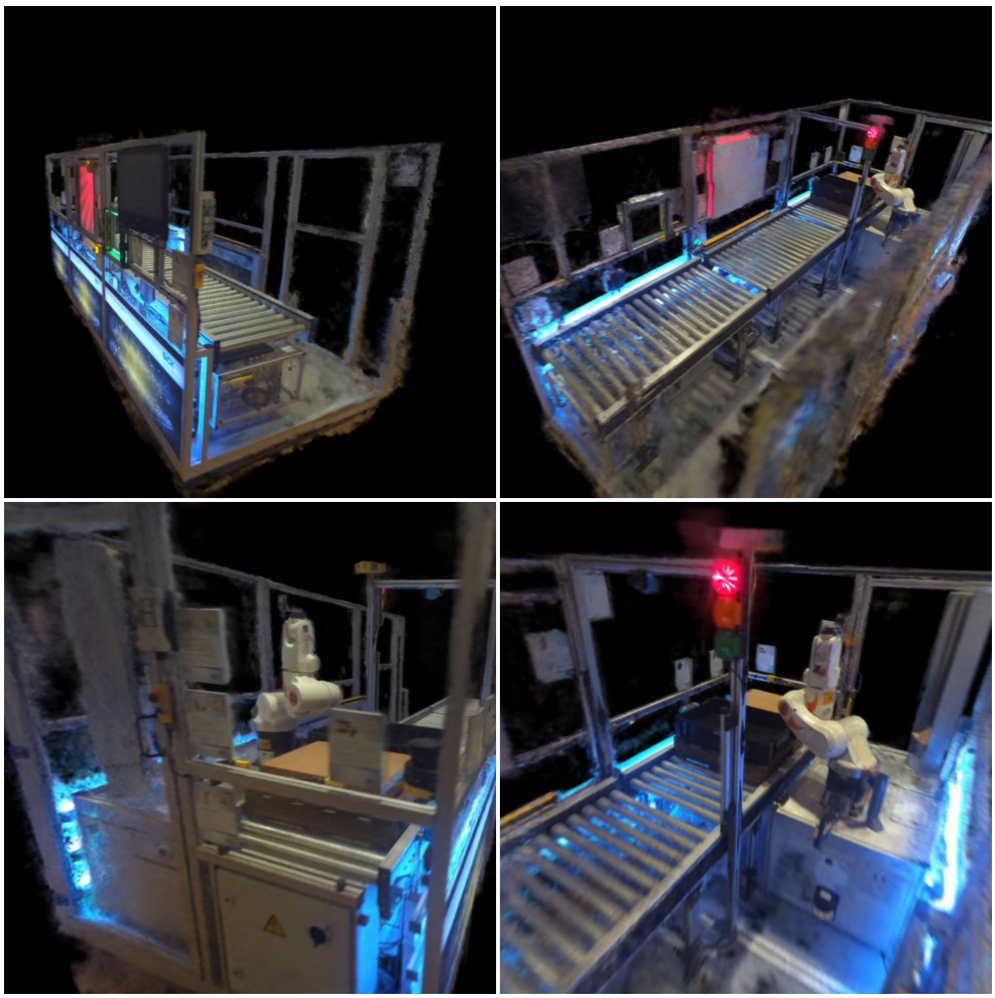


Figure 39.: Rendered views from the neural radiance field of a production application. The renderings accurately capture the spatial details of the scene. [For dynamic representation, see accompanying website videos.]

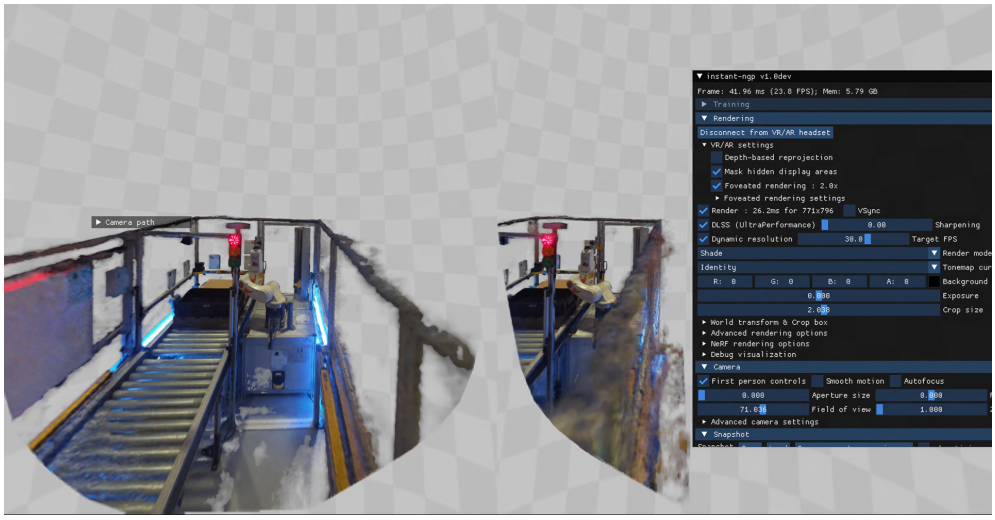


Figure 40.: Screenshot of the instant-ngp application rendering the views currently displayed in the VR headset. [For dynamic representation, see accompanying website videos.]

Figure 41.: This illustration delineates the rendering results from divergent sensor placements and their corresponding parameter settings.

has been developed to facilitate the generation of a neural radiance field to represent this logistical scenario.

Upon the successful generation of this neural radiance field, the user has the ability to virtually place sensors at specific locations within the model to view the resulting renderings. Given the pre-defined settings for these sensors, rapid assessment of their effectiveness in different configurations is possible. This computational simulation and evaluation methodology empowers the user with the necessary analytical tools to select an optimal sensor solution that aligns precisely with the unique requirements of their logistics environment. Figure 41 shows such an example of a user testing various sensors in a simulated production environment.

The above scenario, executed within the context of the instant-ngp framework, not only highlights the available capabilities but also signals the potential for broader applicability. This allows various stakeholders to customize the technology to their specific contextual needs. While the current iteration utilizes a graphical user interface (GUI) for visualization, future developments could incorporate virtual reality interfaces, thereby amplifying the capacity for spatial rendering and interactive editing. Such a transition could offer a more immersive user experience, providing an enriched interactive milieu within which users can virtually navigate their application landscape to adjust and position sensors as needed.

5.3.2. Generate neural radiance fields as digital twins

Using NeRF, one can construct an accurate digital representation of physical objects for diverse applications. For instance, once an object is digitally rendered, it can be integrated into various virtual environments like a simulated production line or a virtual testing ground. This integration allows for proactive problem-solving, such as identifying scanning inefficiencies, or optimization tasks like repositioning elements for optimal interaction. Figure 42 illustrates the degree of accuracy in capturing intricate radiance properties, including transparency and reflections from different perspectives.

Figure 42.: This figure displays a captured neural radiance field of an object with transparent elements. The neural radiance field effectively captures and represents the radiance properties with varying reflections from different angles. [For dynamic representation, see accompanying website videos.]

Although the neural radiance fields can be saved, incorporating them into existing digital twins remains a challenge. A key requirement for this integration is the capacity to render the neural radiance field independently of the digital twin's existing environment. This feature is not yet implemented in existing platforms like Omniverse. Consequently, we investigated the feasibility of extracting a mesh from the neural radiance field for integration into a digital twin environment.

Although the neural radiance fields can be saved, incorporating them into existing digital twins remains a challenge. A key requirement for this integration is the capacity to render the neural radiance field independently of the digital twin's existing environment. This feature is not yet implemented in existing platforms like Omniverse. Consequently, we investigated the feasibility of extracting a mesh from the neural radiance field for integration into a digital twin environment.

5.3.3. Generate Meshes in instant-ngp

The result obtained from the neural radiance field, as depicted in Figure 42, is undeniably impressive. It stands as a compelling demonstration of the $3D$ scene representation derived from $2D$ images through the framework of instant-ngp. instant-ngp provides the functionality to extract a mesh from the radiance field, utilizing the Marching Cubes algorithm for the extraction process.

Technical Background of Marching Cubes

The Marching Cubes algorithm, developed by Lorensen and Cline (1987), is a method used to create triangle models of constant density surfaces from $3D$ volume elements or polygons. It takes volumetric data, represented as a $3D$ grid of voxels, each containing information about the density of the $3D$ scene. By selecting a desired surface density value, the algorithm identifies the voxels that cross this threshold, distinguishing between the inside and outside of the object. The $3D$ grid is divided into small cubes, usually with eight voxels each. These cubes represent local regions of the volume and hold density values at their eight corners. Since each cube has eight vertices and two states (inside and outside), there are only 256 ways a surface can intersect the cube. The algorithm enumerates these cases and creates a table to look up surface-edge intersections, based on the labeling of a cube's vertices. This table contains the edges intersected for each case. For every cube, the algorithm determines which edges are intersected by the isosurface (where the density crosses the threshold). Then, based on the intersected edges, a unique 4-bit index is generated to represent the isosurface configuration within the cube. Using a pre-computed lookup table, the algorithm maps this 4-bit index to a set of triangles that approximate the isosurface within the cube. These triangles are defined by interpolating the positions of the intersected edges. The triangles generated for each cube are combined to form the final $3D$ mesh representing the isosurface of the object (Lorensen and Cline, 1987).

instant-ngp also provides parameters to optimize the resulting mesh. These parameters are *Laplacian Smoothing*, *Density Push*, and *Inflate*. *Laplacian Smoothing* is a technique used to improve the quality of a $3D$ mesh by adjusting the positions of its vertices. It works by moving each vertex toward the average position of its neighboring vertices. This process helps to reduce sharp angles and irregularities in the mesh, resulting in a smoother and more aesthetically pleasing surface. The amount of smoothing applied can be controlled by adjusting the *Laplacian Smoothing* parameter. Increasing this value will apply more smoothing, while decreasing it will retain more of the original mesh details (Sorkine, 2005). *Density push* is a parameter that allows users to control the expansion or contraction of the $3D$ mesh based on the underlying density information. It works by pushing the vertices of the mesh away from or towards the regions with higher or lower density values, respectively. This can be useful in cases where the original mesh may not align perfectly with the desired density surface. By adjusting the *Density Push* parameter, users can fine-tune the level of mesh deformation to better match the density distribution. The *Inflate*

parameter is used to adjust the overall size or thickness of the $3D$ mesh. Increasing the inflation value will expand the mesh, while decreasing it will shrink it. This parameter can be helpful in adjusting the thickness of the generated surface to achieve the desired visual appearance or physical properties. By providing users with control over these mesh optimization parameters, the instant-ngp framework enables the creation of more refined and customized $3D$ meshes, tailored to specific requirements or visual preferences.

In Figure 43 we provide an illustrative example of a mesh output, specifically one derived using the default settings provided by the framework. This particular figure is a composite of three distinct images, each emphasizing a different aspect of the process. The first image portrays the original neural radiance field, giving viewers an understanding of the raw data from which the mesh is derived. The second image offers a visualization of the mesh, with an emphasis on vertex normals. This provides insight into the mesh's geometric features, highlighting how surfaces are oriented in three-dimensional space. The third image within the figure displays the mesh, but with a focus on vertex colors. This showcases the color information associated with each vertex, allowing for a comprehensive view of both the mesh's structure and its appearance.

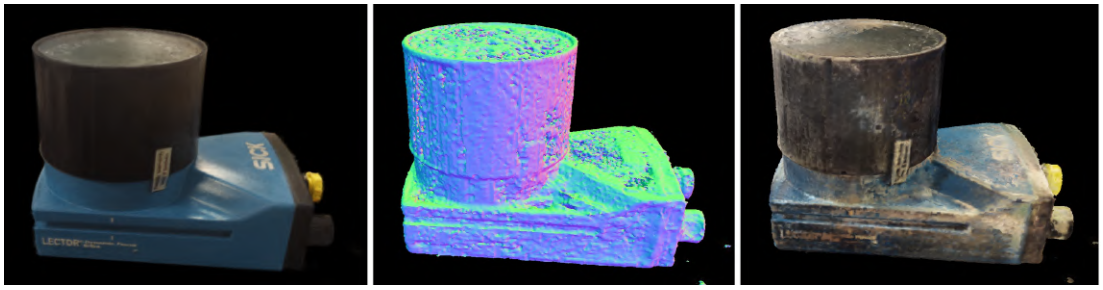


Figure 43.: Exported mesh with default values. The first image represents the neural radiance field, the second image shows the mesh with vertex normals, and the third image displays the mesh with vertex colors.

To further enhance the mesh's quality and usability, manual optimizations have been applied, as depicted in Figure 44. This demonstrates that a better overall appearance can be achieved with small modifications using e.g. Blender Modifier.

The extracted mesh loses the radiance field properties that it initially had. While materials within a radiance field are distinctly identifiable, exporting to mesh only generates a framework without associated material data. This means that transparent objects are not extracted with their transparency intact. However, if the primary goal is to integrate just the object's shape from the radiance field into a digital twin, this method proves suitable. Furthermore, it is faster than traditional photogrammetry techniques. The manual enhancements also significantly elevate the visual fidelity and adaptability of the mesh for its application within the digital twin environment.

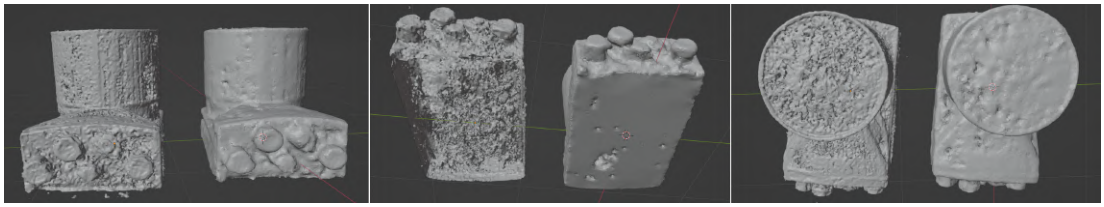


Figure 44.: Optimized exported mesh with manual editing. The left side of each of the images shows the original Mesh Export without any manual editing, while on the right side, Blender edits have been applied to achieve a better result. Although the mesh may not be perfect, it provides valuable insights into the structure of the subject.

instant-ngp is primarily centered around the concept of neural radiance fields. In our quest for improved results, we've integrated the user-friendly and artist-oriented framework known as *Nerfstudio*. *Nerfstudio* offers diverse export methods tailored to the needs of creators and artists, facilitating the generation of point clouds or meshes for further processing and seamless integration into downstream tools such as game engines. Figure 45 provides an illustration of an exported point cloud.

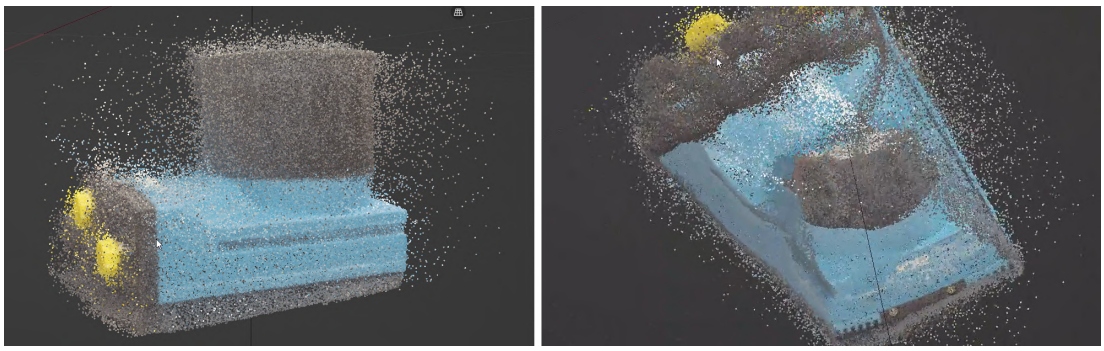


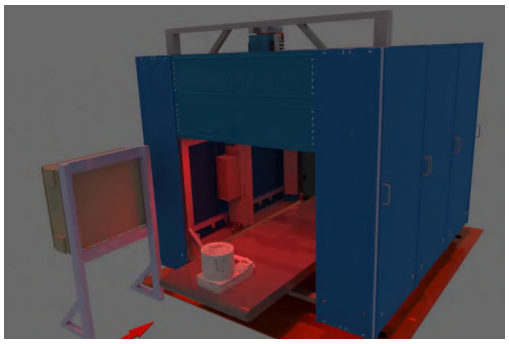
Figure 45.: Point cloud export from *Nerfstudio* highlighting the challenges in mesh extraction. The presence of outliers and the absence of points under the object - unreachable due to its placement on a table - make the extraction process non-trivial. Despite these obstacles, the point cloud offers valuable insights for future analysis and potential improvements in mesh extraction.

Examining the point cloud reveals the reasons why a mesh export is not straightforward. The presence of many outliers can complicate the process of mesh extraction, as these irregular data points can disrupt the smooth surface representation necessary for creating a coherent mesh. Additionally, the absence of points on the bottom of the subject (which was occluded during video capture) rules out the possibility of accurately connecting the mesh in this region. The mesh generated with *Nerfstudio* does not surpass the quality achieved using instant-ngp.

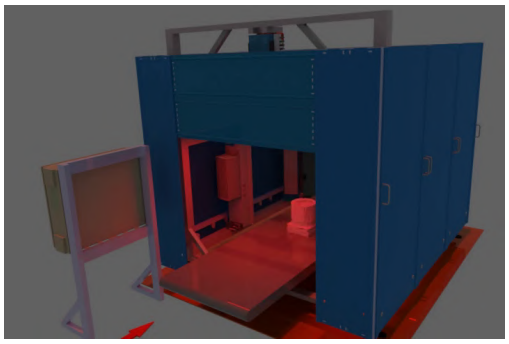
Mesh generation is computationally intensive, resulting in meshes with a high number of triangles, making their use in other programs challenging. Manual post-processing becomes essential if one aims to create a useful mesh despite these limita-

tions. The lack of a standardized NeRF training file format adds further complexity, as it hinders seamless transitions between different applications without re-running the entire process. This difficulty also affects the ability to easily load neural radiance field for further editing within a web viewer.

Nevertheless, we extracted a mesh and integrated it into a digital twin for practical application. As shown in Figure 46, we demonstrate the feasibility of placing a NeRF-generated object within a digital twin environment. Here, the digital twin enables practical testing; for instance, determining the optimal positioning of a label for sensor detection.



(a) Object before scanning



(b) Object after scanning

Figure 46.: Integration of a NeRF-generated object into a digital twin for practical evaluation. [For dynamic representation, see accompanying website videos.]

Consequently, we can conclude that the NeRF algorithm may not be the optimal choice for extracting meshes. Currently, without generating a mesh, the created neural radiance field with the instant-ngp framework cannot be integrated into other platforms like *Unity*, *Omniverse*, or *Unreal Engine* due to the absence of an API for these applications.

5.4. Limitations of the NeRF Algorithm

While the NeRF (Mildenhall et al., 2020; Barron et al., 2021; Barron et al., 2022; Müller et al., 2022; Barron et al., 2023) approach has gained significant attention for its innovative volumetric scene representation by using encodings, neural networks, and volume rendering, it is not without its limitations. At its core, the algorithm is a novel view synthesis technique that represents a scene as a continuous volumetric function, parameterized by MLPs that provide the volume density and view-dependent emitted radiance at each location. Moreover, the neural network is specifically trained on a dataset of images characterized by unique lighting conditions, enabling it to generate novel views solely for that particular scene.

As elaborated in Chapter 2.2.4, the success of NeRF has led to a range of follow-up methods that focus on enhancing both image quality and computational speed. The

methods have largely centered on optimizing training and rendering speeds by incorporating three primary design elements: spatial data structures for feature storage, different encoding mechanisms, and MLP architectures optimized for computational efficiency (Kerbl et al., 2023). Most notable of these methods is instant-ngp, which we explained in Chapter 2.2.3. This approach uses a hash grid and an occupancy grid to accelerate computation and a smaller MLP to represent density and appearance. The current state-of-the-art is represented by MeRF (Reiser et al., 2023) presented in August 2023, which has made advancements in both of these aspects (speed and quality).

The algorithm excels in its designed purpose, which is the synthesis of novel views for static scenes. However, when applied to divergent tasks like mesh generation or scene editing, it encounters limitations and often requires modifications to perform effectively.

This section intends to offer a comprehensive analysis of the intrinsic difficulties associated with the algorithm's specific nature, as well as the strategies being explored to surmount these challenges.

5.4.1. Editing NeRFs

A limitation of NeRF is that it relies on a dense field for scene representation. Therefore, any modifications to the scene require adjustments to this radiance field, a process that is both computationally demanding and time-intensive (Rabby and Zhang, 2023).

However, several algorithmic solutions such as *NeRFShop* and *NeuralEditor* are emerging to mitigate this limitation (Jambon et al., 2023; Chen et al., 2023).

NeRFShop aims to provide an interactive editing solution for NeRF that operates on entire scenes and offers free-form direct manipulation capabilities, encompassing deformation, translation, scaling, rotation, and object duplication, all within the volumetric representation. It builds on the instant-ngp algorithm to offer these functionalities in a more computationally efficient manner (Jambon et al., 2023).

At the core of the *NeRFShop* framework is the idea of interactive selection and deformation of objects through a technique known as cage-based transformations. Users initiate the editing process by "scribbling" on the area they wish to manipulate. This scribbling feature provides a fine level of control for selecting specific regions or objects within the scene. Once the user completes the initial scribbling, the framework estimates the selected area in a three-dimensional context. This reprojection step translates the scribbled regions into a volumetric subset of the scene that can be further manipulated. For users who wish to expand the area of influence, the framework provides a `GROW REGION` function. This enlarges the selected area, offering greater flexibility in the editing process. Upon finalizing the desired area for ma-

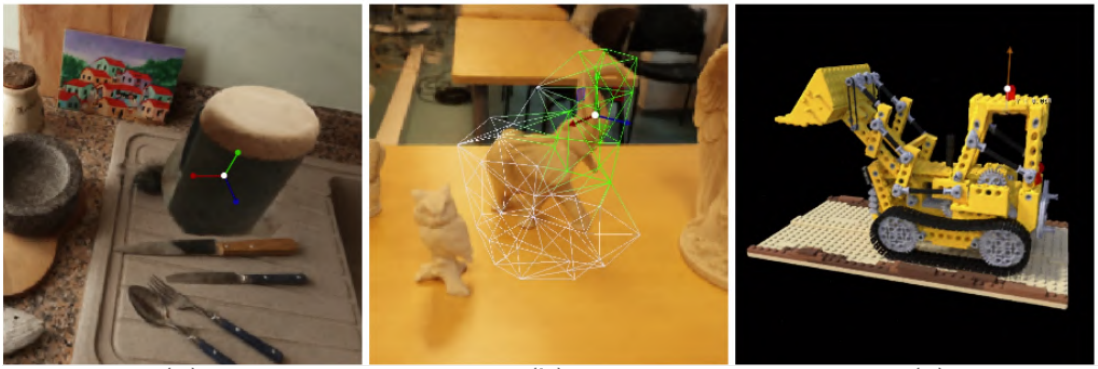


Figure 47.: This figure demonstrates the manipulation capabilities within a scene. (a) displays a cup accompanied by a coordinate system, allowing for movement of the cup in the x, y, z directions. It was already moved from the left side along the x-axis (red arrow). (b) highlights the cage-based system. Within this cage, specific points can be selected for editing. In the Lego representation (c) the roof is translated in the y direction, illustrating the possibility of translation manipulation (Jambon et al., 2023).

nipulation, users activate the COMPUTE PROXY function to construct a three-dimensional "cage" around the selected objects or regions. The cage serves as a geometric proxy that allows for various transformations. Users can rotate the NeRF view to ensure the cage accurately encloses the intended areas. Specific sections of the cage can be highlighted and edited by clicking and dragging the mouse over the region. This enables targeted adjustments to the object within the constructed cage. *NeRFShop* introduces a post-editing step to reduce artifacts that may arise from the transformation process. It utilizes a volumetric membrane interpolation technique, inspired by Poisson image editing, to achieve smoother transitions. Additionally, the framework provides a "distillation" process that integrates the edits into a standalone NeRF representation, making the changes more permanent and cohesive.

By incorporating these functionalities, *NeRFShop* substantially advances the state-of-the-art in interactive scene editing within NeRF (Jambon et al., 2023). Figure 47 provides a visual overview of the robust scene editing capabilities enabled by these techniques.

Another approach, *NeuralEditor*, integrates point clouds with NeRF to enhance shape editability. This is rooted in the understanding that NeRF rendering essentially involves projecting a $3D$ point cloud onto a $2D$ image plane. For rendering, the framework employs K-D tree-guided density-adaptive voxels, enabling deterministic integration. This not only elevates rendering quality but also yields precise point clouds suitable for editing tasks. This approach balances computational efficiency with rendering fidelity.

The core functionality of *NeuralEditor* lies in its ability to map relevant points

between point clouds for shape modifications. This mapping process enables a wide array of editing tasks, from basic shape deformations to more complex scene morphing. The result is a more natural, intuitive editing experience compared to other *NeRF* editing methods. The framework can perform high-fidelity renderings in a zero-shot inference manner, with options for subsequent refinement, thus serving a wide array of editing requirements.

By incorporating these core principles and functionalities, *NeuralEditor* serves as a pioneering tool that bridges the existing gap between high-quality NeRF rendering and flexible $3D$ scene editing. Its codebase, benchmarks, and demo videos are publicly available, inviting further advances in $3D$ shape and scene editing research (Chen et al., 2023).

5.4.2. Relighting NeRFs

While NeRF effectively models emitted radiance, it does not explicitly consider material properties like reflectance, roughness, or transparency. Although these properties are important for realistic rendering in classical methods, NeRF achieves similar levels of realism without needing to know them. Since NeRF entangles material and lighting information in its view-dependent radiance, it is not straightforward to alter the lighting conditions without retraining the model. Unlike traditional $3D$ modeling, where material and lighting are distinct parameters, NeRF integrates both into its model. This integration makes it challenging to adjust lighting without also affecting material properties (Rabby and Zhang, 2023). As such, a model captured within a particular lighting environment cannot be rendered photorealistically within another environment that has its own, distinct lighting conditions. Additionally, the nature of neural networks makes it difficult to intuitively modify the model's weights for specific changes.

In addressing these limitations, a recent paper titled "ReNeRF: Relightable Neural Radiance Fields with Nearfield Lighting" (Xu et al., 2023) was published in October 2023. The paper introduces a novel technique for constructing high-quality, relightable Neural Radiance Fields (ReNeRFs). These ReNeRFs are generated from images captured under controlled studio conditions and offer full control over the camera viewpoint. Importantly, they allow for adaptive relighting under any illumination conditions, including near-field lighting. The methodology outlined in the paper leverages the principles of Image-Based Relighting (IBRL), a data-driven approach used for modeling complex global illumination in scenes. IBRL uses the concept of light superposition to relight scenes by linearly combining a set of basis images. These basis images are captured under one-light-at-a-time (OLAT) conditions. However, the IBRL method is primarily $2D$ and assumes distant lighting and fixed viewpoints. In contrast, the *ReNeRF* approach extends this methodology to a $3D$ context and al-

lows for novel viewpoints. It also accounts for the proximity between $3D$ points in the scene and the light sources, thereby capturing the nuances of near-field lighting. Instead of linearly combining basis images, *ReNeRF* learns to perform a nonlinear interpolation of a continuous OLAT basis at varying distances. This effectively results in OLAT super-resolution and enables the model to adapt to new lighting conditions. Another notable feature of the *ReNeRF* methodology is its efficiency in learning. The model can be trained using only a limited number of area light sources commonly used in studio photogrammetry, eliminating the need for complex and dense light stage setups. Results of this method are shown in Figure 48 (Xu et al., 2023).

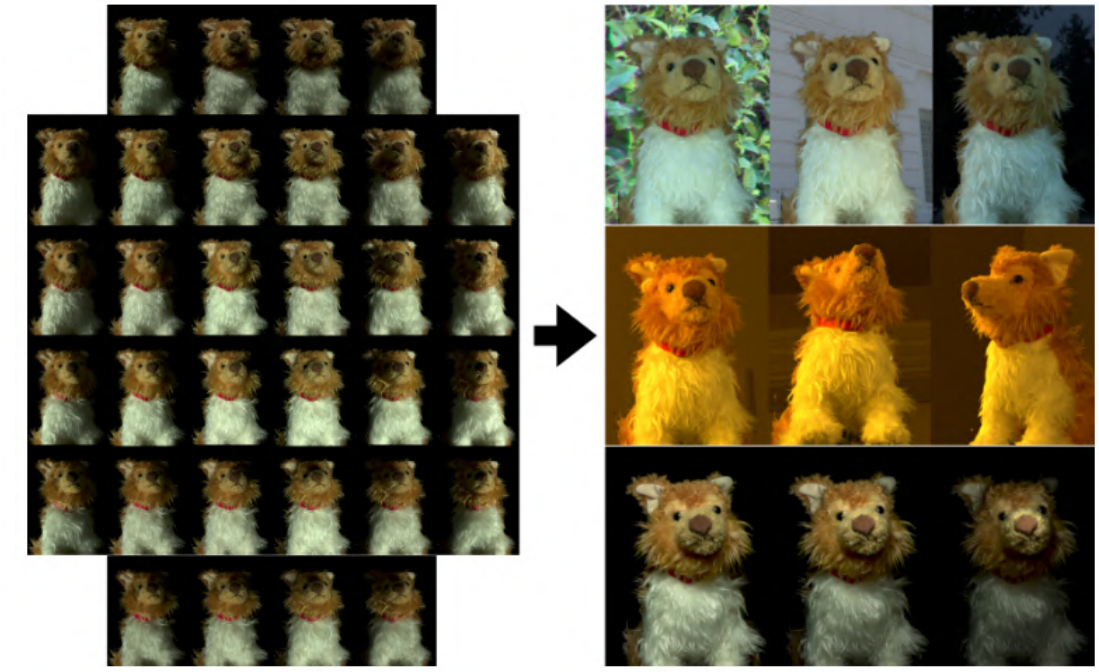


Figure 48.: Renderings from *ReNeRF* models are displayed within two distinct lighting environments. The left side displays the training images, while the right side shows renderings from *ReNeRF* models under various lighting conditions (Xu et al., 2023).

5.4.3. Generating NeRFs of Large Scenes

Applying NeRF to large-scale scenes environments typically leads to significant artifacts and low visual fidelity due to limited model capacity along with memory and compute constraints, as our results showed in Section 5.2.3. Additionally, obtaining training data for expansive environments under uniform conditions in a single capture session is improbable. Instead, data for various segments of the environment may have to be gathered from multiple collection efforts. This introduces variability in scene geometry - such as changes due to construction work or parked cars - as well as in appearance, which can be affected by factors like weather conditions and time of day (Tancik et al., 2022).

Figure 49.: The illustration showcases the capability of *Block-NeRF* to execute large-scale scene reconstructions by leveraging multiple compact NeRFs, each optimized for memory. During the inference process, *Block-NeRF* adeptly integrates renderings from pertinent NeRFs tailored for the specific zone. This image highlights a reconstruction of the Alamo Square neighborhood in San Francisco, derived from data aggregated over a span of 3 months. A noteworthy aspect of *Block-NeRF* is its capacity to refresh distinct environment blocks without necessitating retraining on the entire landscape, evident from the developmental progress on the right (Tancik et al., 2022).

The algorithm we utilized in our experiments, instant-ngp, is specifically designed for object-level representation rather than for capturing expansive, scene-level details. The idea to train and then to merge multiple NeRFs to represent a large-scale scene presents its own complex challenges. Such an approach would require the development of a rendering algorithm that can discern which neural radiance field saturates first, based on integrated density and viewing angle, before outputting the corresponding color.

To address the scalability challenges associated with representing large-scale environments through NeRF, Tancik et al. (2022) introduce a variant known as *Block-NeRF*. Unlike other approaches that employ a single NeRF model, the authors propose segmenting the environment into multiple *Block-NeRFs*. Each of these can be trained independently and in parallel. This modular design allows for the addition of new blocks or the updating of existing ones without requiring the retraining of the entire environment, as illustrated in Figure 49 (Tancik et al., 2022)

During the rendering process, relevant *Block-NeRFs* are dynamically selected and composited. This compositing employs optimized appearance codes and interpolation weights, calculated based on each *Block-NeRF's* proximity to the new viewpoint. The appearance codes are specifically designed to match lighting conditions across blocks. For efficiency, the authors employ two filtering mechanisms to select only those *Block-NeRFs* that are relevant to the target viewpoint. The first is a radial distance filter, discarding blocks that fall outside a set radius from the target viewpoint. The second is a visibility threshold, calculated independently of the color network, which omits blocks with low average visibility. After these filtering steps, typically one to three *Block-NeRFs* remain for the final composition process.

The authors also address the issue of visual consistency across different Block-NeRFs during compositing. Appearance latent codes are utilized to match the visual attributes, such as lighting and color balance, across different blocks. A 3D matching location is identified between adjacent Block-NeRFs to optimize the appearance code of the target block, thus ensuring a more uniform look and feel across the entire scene. This optimized appearance code is then iteratively propagated through the environment, achieving visual consistency.

Through these innovative methods for training, rendering, and compositing multiple *Block-NeRFs*, the authors offer a scalable and flexible approach to representing large-scale $3D$ environments. This is particularly advantageous for applications that require both high-quality rendering and the capability to incrementally update or expand the modeled environment. For instance, in the field of robotics, especially in closed-loop robotic simulations, the ability to precisely model and update large-scale environments is invaluable. Another application lies in autonomous driving systems. These systems often rely on re-simulating previously encountered driving scenarios for evaluation. Any deviation from the recorded trajectory would require high-fidelity novel view renderings, a capability provided by the *Block-NeRF* approach (Tancik et al., 2022).

5.4.4. Standardization of storing and rendering NeRFs

As the field of NeRF expands, the need for standardization in data storage becomes increasingly apparent. Frameworks such as instant-ngp and *Nerfstudio* rely on their proprietary file formats, posing significant challenges for data interoperability and compatibility across different platforms and algorithms. Each framework also has specific requirements for data preparation, such as how the input data is stored, which adds another layer of complexity.

The problem here is that NeRF algorithms introduce different network architectures, encoding and rendering techniques. For example, standard NeRF uses positional encoding, while mip-NeRF employs integrated positional encoding, and instant-ngp opts for hash encoding. These different approaches bind NeRF models to their respective network queries, encoding and rendering environments, making it difficult to share or transfer models between different frameworks. The different rendering and query methods make the development of a universal viewer or renderer complex, as it would have to take into account the specifics of each NeRF variant.

Currently, the *Nerfstudio* framework, discussed in Chapter 3.2, provides a localized solution by allowing different types of NeRFs to be trained and stored within its environment through the modular design approach. However, this is more of a workaround than a sustainable solution, as it only facilitates interoperability within the confines of the *Nerfstudio* platform.

In order to work with NeRFs in the Industrial Metaverse, there is an urgent need

for an industry standardized file format, whereby for example a global renderer etc. can be written to allow seamless data exchange.

5.4.5. Integrating NeRF in other Environments

The integration of NeRF into existing game and simulation frameworks like Unity, Unreal, or Omniverse presents multiple challenges that extend beyond technical compatibility. NeRF operates on differentiable rendering methods that are fundamentally different from traditional $3D$ scene representation techniques employed in existing engines. Integration of NeRF would necessitate engines to adapt these specific rendering methods, which could lead to extensive modifications in their optimized rendering pipelines. Conventional $3D$ models in these engines offer separate control over geometry, material properties, and lighting conditions. However, NeRF ties material and lighting within its volumetric representation, making it less adaptable to varying lighting conditions. The challenge, therefore, is in reconciling the differing lighting models when inserting a NeRF into an existing scene.

Luma AI (Yu and Jain, 2023) launched a plugin in April 2023 that successfully integrated NeRF into Unreal Engine 5 and has since advanced it to version 3. The plugin works through fully local volumetric rendering, eliminating the need for adjustments to mesh formats, geometry, materials, or streaming protocols. This feature, called Luma Fields, introduces a new paradigm in photorealistic rendering. Features include easy-to-use crop rotation and centering controls, customizable quality settings from "minimal" to "extreme," and alert mechanisms for outdated Luma Fields that need to be updated. All of these controls are intuitively placed within the Blueprint detail view of the Unreal Engine interface (Rubloff, 2023a; AI, 2023).

After the groundbreaking announcement by *Luma AI* (Yu and Jain, 2023) regarding the integration of NeRF into Unreal Engine 5, *Volinga AI* has also stepped up to further simplify this process. Following its launch, the company announced the first-ever native NeRF file format, `.nvol`. Meanwhile, the `.ingp` format is exclusive to the instant-ngp framework and aims to establish itself as the first global standard for NeRF files. The `.nvol` format not only streamlines the file handling but also supports drag-and-drop functionality directly into Unreal Engine 5 (Rubloff, 2023c, July). To complement the `.nvol` format, *Volinga AI* has released the Volinga Exporter tool within *NeRFstudio*. This tool facilitates a hassle-free conversion of NeRF models to `.nvol` files, further smoothening the integration workflow in Unreal Engine 5 (Rubloff, 2023c, July). In addition, the exporter is loaded with an array of user-friendly features, such as Plug-and-Play NeRF integration, advanced translation and rotation controls, simplified crop features, and enhanced user experience.

Fresh off the heels of Volinga AI's Volinga Exporter announcement, another company has announced a integration of NeRF into Unreal Engine. This new contender

named "VELOX XR" has released a compelling video demonstrating an export from instant-ngp to Unreal Engine. The video serves as both a proof-of-concept and an indicator of the rapidly growing interest and competition in this field (Rubloff, 2023b).

As NeRF technology becomes more widespread, more companies are exploring its integration into game and simulation engines, particularly Unreal Engine 5. During the GTC Conference in March 2023, discussions were held regarding the incorporation of NeRF into NVIDIA's Omniverse platform. The proposed approach involves utilizing two distinct rendering engines within Omniverse—one for rendering NeRF models and another for the overall scene. A key aspect of this strategy was the synchronization of lighting conditions between the two rendering methods, addressing a fundamental challenge in NeRF integration (State, 2023).

5.4.6. Dynamic NeRF

The performance of NeRFs in representing static scenes is impressive, but standard NeRFs reach their limits when it comes to dynamic environments. In such scenarios, objects may move or change, requiring time-dependent reconstruction beyond the capabilities of the original NeRFs and their improvements.

There is much ongoing research focused on filling this gap, particularly through the use of MLP-based methods. A detailed overview of these methods can be found in Section 4.3 of the article "Advances in Neural Rendering" by Tewari et al. (2022) (Tewari et al., 2022).

By incorporating dynamic aspects, NeRFs could be used in a wider range of applications, from real-time entertainment to complex simulations in industry.

5.4.7. Mesh Generation with Neural Rendering

The primary aim of NeRF is to create volumetric scene representations that excel in capturing complex lighting conditions and view-dependent effects. However, translating these representations into meshes presents challenges, as displayed in Section 5.3.3. The volumetric nature of NeRF means that it captures data in a continuous $3D$ space rather than discrete surfaces, which are the basis of mesh representations. While NeRF models the radiance and density of each point in $3D$ space, it doesn't explicitly capture the geometric surface of the objects. This makes it difficult to identify and extract well-defined boundaries that can be used to construct a mesh. In addition, the transition from NeRF to mesh results in the loss of view-dependent properties that are essential to the algorithm's ability to capture realistic lighting and material conditions. Conventional mesh formats cannot retain these angle-dependent properties, resulting in reduced visual fidelity. Converting NeRF's volumetric data to a mesh also demands additional computational resources. While methods like iso-

surface extraction (e.g., marching cubes in instant-ngp) can be employed, they often require further refinement to produce high-quality meshes, adding complexity and computational load to the process.

In addressing the challenge of mesh generation from neural radiance fields, a solution named *NeRFMeshing* has been proposed (Rakotosaona et al., 2023). *NeRFMeshing* introduces an advanced pipeline that adeptly extracts geometrically consistent meshes from trained NeRF models. This approach, while being time-efficient, yields meshes that are geometrically precise and equipped with neural colors, facilitating real-time rendering on standard hardware. Central to this is the Signed Surface Approximation Network (SSAN), a post-processing component of the NeRF pipeline, calibrated to accurately represent both surface and appearance. SSAN works by estimating a Truncated Signed Distance Field (TSDF) in conjunction with a feature appearance field. The outcome is the extraction of a $3D$ triangle mesh that represents the scene, which is subsequently rendered using an appearance network to generate view-dependent color gradients (Rakotosaona et al., 2023). When benchmarked against other techniques, *NeRFMeshing* stands out in multiple facets. It is compatible with diverse NeRF architectures, making the assimilation of future advancements seamless. The technique is adept at managing unbounded scenarios and intricate, non-lambertian surfaces. Furthermore, *NeRFMeshing* preserves the high-resolution details characteristic of neural radiance fields, encompassing view-dependent nuances and reflections, solidifying its viability for real-time novel view synthesis (Rakotosaona et al., 2023).

As mentioned, the goal of NeRF is indeed to capture the radiance field of a specific scene. However, the extraction of meshes from these radiance fields currently leads to the loss of this valuable property. While NeRF results may look visually appealing, the generated meshes lack structure and detail.

In response, ongoing research focuses on novel approaches of mesh generation with other neural rendering techniques. *Neuralangelo* (Li et al., 2023b) introduces a framework for high-fidelity $3D$ surface reconstruction from RGB video captures using the signed distance function. This algorithm utilizes the multi-resolution hash encoding from instant-ngp to represent positions. The encoded features are then fed into a neural signed distance function representing the underlying $3D$ scene. Additionally, a color MLP is employed to synthesize images using neural surface rendering. *Neuralangelo* effectively recovers dense scene structures of both object-centric captures and large-scale indoor/outdoor scenes with extremely high fidelity, enabling detailed large-scale reconstruction from RGB videos (Li et al., 2023b).

Figure 50 illustrates the results of the *Neuralangelo* algorithm. Three main aspects can be seen here: (a) Result of RGB view synthesis, where realistic color views are generated from the captured data, (b) Visualization of the reconstructed surface

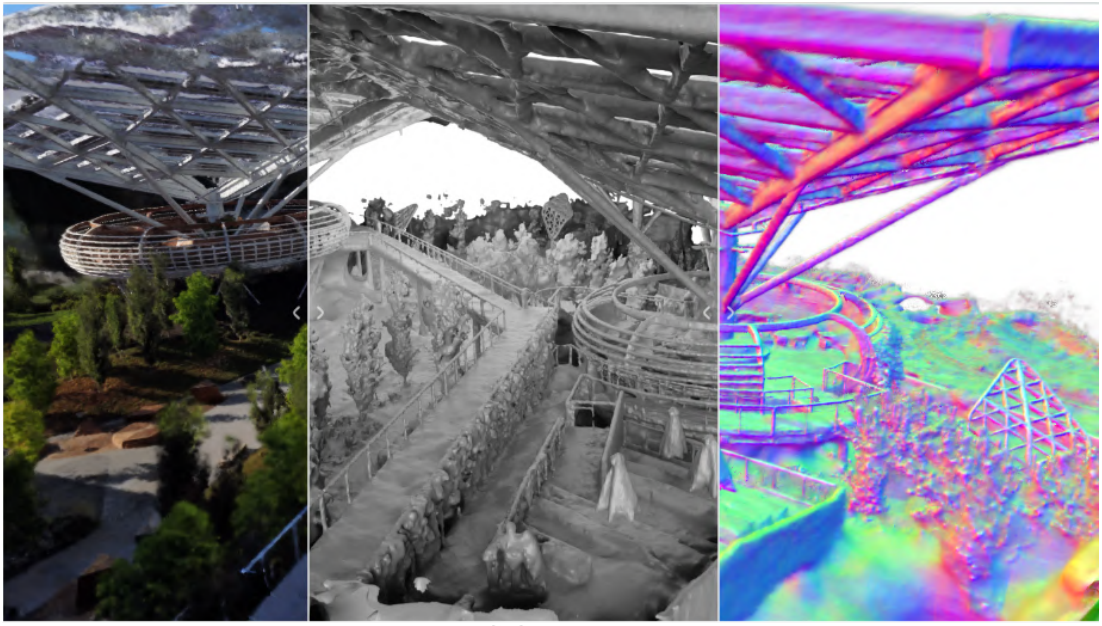


Figure 50.: Results of *Neuralangelo*: (a) RGB View Synthesis, (b) Surface Reconstruction, (c) Surface Normals. These visualizations illustrate the capabilities of *Neuralangelo* in generating realistic views, reconstructing surfaces, and depicting surface normals. These abilities hold promising prospects for highly accurate $3D$ model creation from image and video data (Li et al., 2023b).

derived from the given data, (c) Surface normals, which provide important information about the orientation and structure of the reconstructed surface. These visualizations illustrate the ability to recover dense scene structures with accuracy (Li et al., 2023b).

Neuralangelo presents a novel approach for photogrammetric neural surface reconstruction. The technique relies on numerical gradients for higher-order derivatives and employs a coarse-to-fine optimization strategy. This approach unleashes the potential of multi-resolution hash encoding, which is modeled as Signed Distance Functions (SDF) for neural surface reconstruction. *Neuralangelo* is shown to effectively recover dense structures in various settings, from object-centric captures to large-scale indoor and outdoor scenes, all with remarkable fidelity. The technique allows for high-quality, large-scale scene reconstructions using RGB videos. The current version of *Neuralangelo* relies on lengthy training iterations due to its random pixel sampling, without accounting for statistics and errors. Future work aims to improve the efficiency of the sampling strategy to speed up the training process.

5.5. Conclusion

The utility of NeRF in capturing reality with intricate detail opens up promising avenues for applications in the Industrial Metaverse. NeRF excels particularly in rendering objects that may pose challenges in photogrammetry, and gains efficiency

through the use of hash-based algorithms, making it faster than conventional methods.

However, there are limitations that hinder its adoption in industrial settings. First, the static nature of NeRFs makes them less suitable for dynamic industrial environments where conditions and configurations are constantly changing. This limits their application in real-time scenarios. It is worth noting that extensive research is currently underway to develop dynamic NeRFs 5.4.6. Second, NeRFs are primarily geared toward rendering rather than interaction, restricting their usability in industrial contexts where adaptability is key. The lack of interactive or modifiable capabilities in the generated models poses a significant drawback. Third, the algorithm lacks integrated physics for simulating the behavior of models in industrial settings. Without this, NeRF remains a somewhat superficial tool for visual representation but not for behavioral simulation.

Another barrier to widespread adoption is the lack of standardization. Current developments in NeRF are tailored for specific tasks, making it difficult to create a unified platform that can handle a variety of tasks without specialized modifications.

The "black box" nature of neural networks, on which NeRF is based, also presents difficulties. This opacity hampers fine-tuning for specific requirements and makes it complicated to integrate different NeRF models due to specialized weightings and parameters. Furthermore, the computational requirements of NeRF models can be a critical factor, particularly for real-time applications.

Despite these limitations, the potential for reality capture through neural radiance fields remains significant. The ongoing research in this domain points to the development of increasingly efficient and versatile methods. Adaptations that integrate dynamic scenarios, real-time interactivity, and physics-based simulation are not beyond reach, and several works in the current literature are already advancing in these directions. The fast-paced growth of research in this area indicates that many of the present challenges could be overcome in the near future.

6. Conclusions and Outlook

We conclude this thesis with a summary, followed by a brief discussion on the potential implications of our approach, as well as the limitations of our solution. Finally, we outline avenues for future work and aim to encourage the reader to contribute further to enhancements in the area of Neural Rendering.

6.0.1. Summary

In this thesis, we introduced a new framework for generating Neural Radiance Fields, after a careful examination of the trade-offs in existing state-of-the-art methodologies. The framework we present, based on the instant-ngp algorithm, addresses the limitations of current state-of-the-art frameworks that are primarily geared towards research users and less accessible to individuals without a technical background. Our framework provides a user-friendly, end-to-end solution for generating Neural Radiance Fields. Featuring a modular architecture split between a front-end and a back-end, it allows the back-end to be hosted on a server equipped with an NVIDIA graphics card. This negates the need for users to possess specialized hardware, offering instead an accessible user interface that eliminates the requirement for programming skills.

More specifically, we presented the following content in this thesis: The overarching theme addressed is "Capturing Reality." Starting from this foundational concept, we introduce the background and delve into the Neural Radiance Field Algorithm, which is capable of capturing reality, including its light properties depending on the viewing angle, with remarkable precision. This algorithm has garnered significant attention in the Computer Vision Community and shows a growing interest with no signs of slowing down. Consequently, we introduced improvements to NeRF, including a detailed examination of the instant-ngp algorithm that served as the basis for this thesis. To conclude this chapter, we have introduced the concept of industrial metaverse, presented its current state, and explored its potential applications.

Then,in Chapter 3 we scrutinized existing frameworks that are closely related to the aims of our research. We presented an array of frameworks that offer different solution in terms of generating neural radiance fields. Notably, we focused on Nerfstudio, a leading framework in NeRF development, and Torch-NGP, a user-friendly PyTorch extension that alleviates the need for specialized graphics hardware. We also introduced LUMA AI, an innovative startup that allows the creation of NeRFs

using an iPhone. Furthermore, we provided an in-depth analysis of the instant-ngp framework, outlining its features and capabilities.

Throughout our survey, we recognized the absence of a one-size-fits-all solution in existing research, particularly when considering industrial use-cases. Issues like the necessity for administrator rights, which are not universally available in an industrial environment, and the dependency on NVIDIA RTX graphics cards were acknowledged. Moreover, the existing frameworks often presuppose users to be comfortable with coding and command-line interfaces. These observations led us to conclude that there is a gap in the market for a more user-friendly and comprehensive solution.

Our framework, therefore, draws inspiration from the existing work and sets the stage for Chapter 4, where we delved into the implementation of our framework. The architecture of our framework serves as an exhaustive, end-to-end solution that manages the complete pipeline in generating neural radiance fields. Our implementation features system-independent installations, streamlined package management, effective data upload procedures, as well as robust training and real-time monitoring mechanisms - all accessible through an easy-to-navigate web interface. By merging a variety of tools and technologies - including Poetry, Docker, CUDA, and libraries such as Flask and React - we have developed a robust system that not only optimizes efficiency, but also improves the user experience and the experience related to NeRF generation.

In the final chapter 5, we focused on the question of the impact of neural radiance fields in the Industrial Metaverse domain. This chapter was structured to provide a comprehensive analysis of the capabilities, limitations, and applications of NeRF. First, we explored the fundamental concept of "capturing reality in a single moment in terms of the Industrial Metaverse". We then conducted experiments that addressed this research question, specifically highlighting NeRF's ability to manage complex scene conditions and generate avatars, hands, and large scenes. In addition, we created two use cases for NeRF in different sales and application contexts to evaluate the current efficiency of the algorithm. During the experimentation phase, we encountered limitations inherent to the algorithm's design. In response, we engaged with existing research that addresses these constraints, with a special focus on editing, re-lighting, and generating NeRFs for extensive scenes and dynamic purposes. We also discussed the challenges and opportunities concerning standardization and integration with existing engines. Our findings demonstrate that although NeRFs appear promising, practical limitations currently preclude their effective deployment in the Industrial Metaverse. Nevertheless, as a conclusion it remains to say that the potential of this algorithm is remarkable.

In conclusion, we have presented a comprehensive analysis and contribution to the application of Neural Radiance Fields in the context of the Industrial Metaverse. We

found that existing research does not offer a one-size-fits-all solution, particularly not for industrial applications that have specific requirements regarding access rights and hardware. In response, we introduced a new framework that offers a user-friendly, system-independent implementation, complemented by an intuitive web interface. Through multiple experiments, we analyzed the efficiency and practical limitations of the NeRF algorithm, particularly in handling complex scenarios and in various sales and application contexts. While NeRFs yield promising results, we identified that practical constraints currently prevent their effective utilization in the Industrial Metaverse. However, we also highlighted the considerable potential of this algorithm, which could drive future research and development. Lastly, we proposed solutions to the existing limitations of the algorithm and emphasized the need for standardization and better integration into existing systems. Overall, our work demonstrates that a solid understanding of the capabilities and limitations of NeRFs is crucial to fully exploit the algorithm's potential in the industrial metaverse. Our findings can serve as a foundation for future research projects in this exciting and rapidly evolving field.

6.1. Discussion

This thesis makes significant contributions in the areas of computer graphics and computer vision, specifically in the context of Neural Radiance Fields (NeRFs) in the Industrial Metaverse. Our user-centric approach significantly lowers the technical threshold for implementing NeRFs in industrial scenarios, allowing for extensive real-world testing.

During the course of this thesis, numerous NeRF-related studies have been presented at major conferences such as CVPR, ICCV, and Siggraph. While the sheer volume of research has prevented full coverage of all aspects, it does highlight the versatile and expanding field of applications for this technology. Despite existing limitations, our results and ongoing research indicate that neural networks for $3D$ representation have enormous potential for future applications.

Finally, our thesis highlights the understudied role of NeRFs in the Industrial Metaverse. Our work is the first to provide a comprehensive analysis and application of NeRFs in this particular context. We have shown that NeRFs can play a key role in the realization of complex 3D scenarios in industrial applications. However, we have also pointed out the current limitations and challenges that currently prevent a broader application of NeRFs in the Industrial Metaverse. These findings provide a solid foundation for future research and potential industrial applications.

6.2. Outlook

The true potential of NeRF lies in its role as a proof-of-concept for neural fields in general. Subsequent approaches have extended this idea by including not only position and direction as inputs, but also additional variables such as labels or depth. The outputs of these networks have been enriched to include not only traditional features but also additional elements like labels, as evidenced by SemanticNeRF (Zhi et al., 2021). Developments have even diversified into new domains such as neural sound fields (Luo et al., 2022). This suggests that while NeRF itself has its limitations, the underlying concept of using neural networks to represent fields offers a broader and growing range of applications. The influence of NeRF has sparked a broader interest in innovative methods for scene representation. Our examination of the most recent award-winning paper, "3D Gaussian Splatting for Real-Time Radiance Field Rendering" (Kerbl et al., 2023), underscores this dynamic landscape. It tackles several limitations commonly linked with NeRF by removing the neural network component.

The authors of this paper introduces three key advancements that elevate both visual quality and computational efficiency. It achieves real-time novel-view synthesis at a 1080p resolution with a frame rate of at least 30 fps. The first breakthrough employs $3D$ Gaussians, generated from sparse calibration points, to represent the scene. These Gaussians retain the beneficial traits of continuous volumetric radiance fields, thereby streamlining scene optimization by bypassing unnecessary calculations in void spaces. Secondly, the research applies an interleaved optimization and density control of these $3D$ Gaussians, with a particular focus on optimizing anisotropic covariance for a more faithful scene depiction. This results in a compact, yet precise scene representation, involving 1 to 5 million Gaussians across all tested cases. Thirdly, Kerbl et al. unveil a fast, visibility-aware rendering algorithm that incorporates anisotropic splatting, thus speeding up both the training and real-time rendering processes.

With the publication of this paper (Kerbl et al., 2023), a new foundational layer has been established that warrants further investigation. The authors not only provide an innovative approach in scene representation but also pave the way for further research in this rapidly evolving field. Complementing this static approach, a dynamic variant has recently been introduced (Luiten et al., 2023), marking yet another significant advancement that underscores the continual growth and relevance of this research field.

As a conclusion of this thesis, we can say that the investigated area has opened a new and promising field of research. The groundwork laid by NeRF provides a solid starting point from which future research can proceed in many different directions. This work underscores not only the importance of NeRF as pioneering work, but also the far-reaching possibilities it opens up for further studies and applications.

Bibliography

Adamkiewicz, M., Chen, T., Caccavale, A., Gardner, R., Culbertson, P., Bohg, J., & Schwager, M. (2022). Vision-only robot navigation in a neural radiance world.

Adelson, E., & Bergen, J. R. (1991). The Plenoptic Function and the Elements of Early Vision.

AI, L. (2023). Luma Unreal Engine Plugin [Accessed: 2023-10-05].

Ball, M. (2022). *The metaverse: And how it will revolutionize everything*. Liveright Publishing Corporation, a division of W.W. Norton; amp; Company.

Barron, J. T., Mildenhall, B., Tancik, M., Hedman, P., Martin-Brualla, R., & Srinivasan, P. P. (2021). Mip-nerf: A Multiscale Representation for Anti-Aliasing Neural Radiance Fields. *ICCV*.

Barron, J. T., Mildenhall, B., Verbin, D., Srinivasan, P. P., & Hedman, P. (2022). Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields. *CVPR*.

Barron, J. T., Mildenhall, B., Verbin, D., Srinivasan, P. P., & Hedman, P. (2023). Zip-NeRF: Anti-Aliased Grid-Based Neural Radiance Fields. *ICCV*.

Barron, J. (2021, March). MIP-NeRF: A multiscale representation for anti-aliasing neural radiance fields.

Bhalgat, Y. (2022). HashNeRF-pytorch [Accessed: 2023-09-05].

Bitkom, e. (2022). A guidebook to the metaverse: Leitfaden 2022 [Accessed: 2023-09-28].

Bradski, G. R., & Kaehler, A. (2011). *Learning OpenCV: Computer vision with the OpenCV library* (1. ed., [Nachdr.]). O'Reilly.

Byravan, A., Humplik, J., Hasenclever, L., Brussee, A., Nori, F., Haarnoja, T., Moran, B., Bohez, S., Sadeghi, F., Vujatovic, B., & Heess, N. (2022). Nerf2real: Sim2real transfer of vision-guided bipedal motion skills using neural radiance fields.

Caulfield, B. (2022). What Is the Metaverse? [Accessed: 2023-09-27].

Chen, A., Xu, Z., Geiger, A., Yu, J., & Su, H. (2022a). TensoRF: Tensorial Radiance Fields. *European Conference on Computer Vision (ECCV)*.

Chen, X., Zhang, Q., Li, X., Chen, Y., Feng, Y., Wang, X., & Wang, J. (2022b). Hallucinated neural radiance fields in the wild. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 12943–12952.

Chen, J.-K., Lyu, J., & Wang, Y.-X. (2023). NeuralEditor: Editing Neural Radiance Fields via Manipulating Point Clouds. *CVPR*.

Contributors, X. (2022). OpenXRLab Neural Radiance Field Toolbox and Benchmark [Accessed: 2023-09-04].

Docker. (2023).

Fridovich-Keil and Yu, Tancik, M., Chen, Q., Recht, B., & Kanazawa, A. (2022). Plenoxels: Radiance Fields without Neural Networks. *CVPR*.

Gafni, G., Thies, J., Zollhöfer, M., & Nießner, M. (2021). Dynamic Neural Radiance Fields for monocular 4d Facial Avatar Reconstruction. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 8649–8658.

Gortler, S. J., Grzeszczuk, R., Szeliski, R., & Cohen, M. F. (1996). The Lumigraph. *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, 43–54.

Heo, H., Kim, T., Lee, J., Lee, J., Kim, S., Kim, H. J., & Kim, J.-H. (2023). Robust Camera Pose Refinement for Multi-Resolution Hash Encoding.

Hu, S.-M., Liang, D., Yang, G.-Y., Yang, G.-W., & Zhou, W.-Y. (2020). Jittor: a novel deep learning framework with meta-operators and unified graph execution. *Science China Information Sciences*, *63*(222103), 1–21.

Huang, S., Gojcic, Z., Wang, Z., Williams, F., Kasten, Y., Fidler, S., Schindler, K., & Litany, O. (2023). Neural LiDAR Fields for Novel View Synthesis. *Proceedings of the IEEE/CVF International Conference on Computer Vision*.

IBM. (2023). How Industry 4.0 technologies are changing manufacturing [Accessed: 2023-09-30].

Jambon, C., Kerbl, B., Kopanas, G., Diolatzis, S., Leimkühler, T., & Drettakis, G. (2023). NeRFshop: Interactive Editing of Neural Radiance Fields". *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, *6*(1).

Jun-Seong, K., Yu-Ji, K., Ye-Bin, M., & Oh, T.-H. (2022). HDR-Plenoxels: Self-Calibrating High Dynamic Range Radiance Fields. *ECCV*.

Kanawaza, A. (2023). Nerfstudio: A Modular Framework for Neural Radiance Field Development.

Kazhdan, M., Bolitho, M., & Hoppe, H. (2006). Poisson Surface Reconstruction. In A. Sheffer & K. Polthier (Eds.), *Symposium on geometry processing*. The Eurographics Association.

Kerbl, B., Kopanas, G., Leimkühler, T., & Drettakis, G. (2023). 3d Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Transactions on Graphics*, *42*(4).

KLM. (2020). KLM Cityhopper introduces Virtual Reality training for pilots [Accessed: 2023-09-18].

Levoy, M., & Hanrahan, P. (1996). Light Field Rendering, 31–42.

Li, R., Gao, H., Tancik, M., & Kanazawa, A. (2023a). NerfAcc: Efficient Sampling Accelerates NeRFs. *arXiv preprint arXiv:2305.04966*.

Li, Z., Müller, T., Evans, A., Taylor, R. H., Unberath, M., Liu, M.-Y., & Lin, C.-H. (2023b). Neuralangelo: High-Fidelity Neural Surface Reconstruction. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Li, Z., Wang, Q., Cole, F., Tucker, R., & Snavely, N. (2023c). DynIBaR: Neural Dynamic Image-Based Rendering. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Lorensen, W. E., & Cline, H. E. (1987). Marching cubes: A high resolution 3D surface construction algorithm. *ACM SIGGRAPH Computer Graphics*, *21*(4), 163–169.

Luiten, J., Kopanas, G., Leibe, B., & Ramanan, D. (2023). Dynamic 3d gaussians: Tracking by persistent dynamic view synthesis. *preprint*.

LUMA AI. (2023). LUMA AI - Series A [Accessed: 2023-09-12].

Luo, A., Du, Y., Tarr, M., Tenenbaum, J., Torralba, A., & Gan, C. (2022). Learning neural acoustic fields. *Advances in Neural Information Processing Systems*, *35*, 3165–3177.

Martin-Brualla, R., Radwan, N., Sajjadi, M. S. M., Barron, J. T., Dosovitskiy, A., & Duckworth, D. (2021). NeRF in the Wild: Neural Radiance Fields for

Unconstrained Photo Collections. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 7206–7215.

MathWorks. (2023). Stereo camera calibrator [Accessed: 2023-10-08].

Meng, Q., Chen, A., Luo, H., Wu, M., Su, H., Xu, L., He, X., & Yu, J. (2021). GNeRF: GAN-based Neural Radiance Field without Posed Camera. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 6331–6341.

Mildenhall, B., Srinivasan, P. P., Ortiz-Cayon, R., Kalantari, N. K., Ramamoorthi, R., Ng, R., & Kar, A. (2019). Local Light Field Fusion: Practical View Synthesis with Prescriptive Sampling Guidelines. *ACM Transactions on Graphics (TOG)*.

Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., & Ng, R. (2020). NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis [arXiv:2003.08934 [cs]].

Mildenhall, B., Hedman, P., Martin-Brualla, R., Srinivasan, P. P., & Barron, J. T. (2022). NeRF in the Dark: High Dynamic Range View Synthesis from Noisy Raw Images. *CVPR*.

Müller, T., Evans, A., Schied, C., & Keller, A. (2022). Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. *ACM Trans. Graph.*, *41*(4), 102:1–102:15.

Net, I., Müller, T., Antoniou, P., Aro, E., & Smith, T. (2021).

NVIDIA. (2021). The Metaverse Begins: NVIDIA Omniverse and a Future of Shared Worlds [Accessed: 2023-09-28].

NVIDIA. (2022). Siemens und nvidia: Aufbau des industriellen metaverse [Accessed: 2023-09-28].

NVIDIA. (2023). NVIDIA Brings Millions More Into the Metaverse With Expanded Omniverse Platform [Accessed: 2023-09-28].

NVlabs. (2022). NVlabs/instant-NGP: Instant Neural Graphics Primitives: Lightning Fast Nerf and more [Accessed: 2023-10-06].

OpenCV, T. (2023). OpenCV [Accessed: 2023-10-08].

Pallets. (2023). Flask [Accessed: 2023-10-08].

Pavlakos, G., Weber, E., Tancik, M., & Kanazawa, A. (2022). The one where they reconstructed 3d humans and environments in tv shows.

Peng, S., Dong, J., Wang, Q., Zhang, S., Shuai, Q., Zhou, X., & Bao, H. (2021a). Animatable Neural Radiance Fields for Modeling Dynamic Human Bodies. *ICCV*.

Peng, S., Zhang, Y., Xu, Y., Wang, Q., Shuai, Q., Bao, H., & Zhou, X. (2021b). Neural body: Implicit Neural Representations with Structured Latent Codes for Novel View Synthesis of Dynamic Humans. *CVPR*.

Poetry. (2023). Python packaging and dependency management made easy [Accessed: 2023-08-28].

Pumarola, A., Corona, E., Pons-Moll, G., & Moreno-Noguer, F. (2020). D-NeRF: Neural Radiance Fields for Dynamic Scenes. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Python Software Foundation. (2023). Python [Accessed: 2023-10-08].

Rabby, A. S. A., & Zhang, C. (2023). Beyondpixels: A Comprehensive Review of the Evolution of Neural Radiance Fields.

Rahaman, N., Baratin, A., Arpit, D., Draxler, F., Lin, M., Hamprecht, F. A., Bengio, Y., & Courville, A. (2019). On the spectral bias of neural networks.

Rakotosaona, M.-J., Manhardt, F., Arroyo, D. M., Niemeyer, M., Kundu, A., & Tombari, F. (2023). NeRFMeshing: Distilling Neural Radiance Fields into Geometrically-Accurate 3D Meshes.

Red Hat. (2022). What is CI/CD? [Accessed: 2023-10-08].

Reisig, W., & Freytag, J.-C. (2006). *Informatik - Aktuelle Themen im historischen Kontext*. Springer Berlin Heidelberg.

Reinecke, D. N., Hinzen, L., Mühlpfort, M., & Wafa, D. H. A. E. (2021). Urban Digital Twins for urban development of the future [Accessed: 2023-09-28].

Reiser, C., Peng, S., Liao, Y., & Geiger, A. (2021). Kilonerf: Speeding up Neural Radiance Fields with Thousands of Tiny MLPs. *International Conference on Computer Vision (ICCV)*.

Reiser, C., Szeliski, R., Verbin, D., Srinivasan, P. P., Mildenhall, B., Geiger, A., Barron, J. T., & Hedman, P. (2023). MERF: Memory-Efficient Radiance Fields for Real-time View Synthesis in Unbounded Scenes. *SIGGRAPH*.

Rubloff, M. (2023a). Luma AI releases V 0.3 of Unreal engine 5 nerf plugin: Neural radiance fields.

Rubloff, M. (2023b). Velox XR announces instant NGP to Unreal engine 5: Neural radiance fields [Accessed: 2023-10-05].

Rubloff, M. (2023c, July). Volinga announces release of version 0.2.0 Unreal Engine plugin: Neural radiance fields [Accessed: 2023-10-05].

Rückert, D., Wang, Y., Li, R., Idoughi, R., & Heidrich, W. (2022). NeAT: Neural adaptive tomography. *ACM Transactions on Graphics*, *41*(4), 1–13.

Schönberger, J. L., & Frahm, J.-M. (2016). Structure-from-Motion Revisited. *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Schwarz, K., Liao, Y., Niemeyer, M., & Geiger, A. (2021). GRAF: Generative Radiance Fields for 3D-Aware Image Synthesis.

Scholar Search. (2023). Neural Radiance Fields Mildenhall et. al. [Accessed: 2023-08-28].

Shapiro, D. (2023). Manufactured in the metaverse: Mercedes-benz assembles next-gen factories with nvidia omniverse [Accessed: 2023-09-28].

SIEMENS. (2023). The emergent industrial metaverse [Accessed: 2023-09-28].

Sitzmann, V., Zollhöfer, M., & Wetzstein, G. (2020). Scene Representation Networks: Continuous 3D-Structure-Aware Neural Scene Representations.

Slater, M. (2022). Exploring Neural Graphics Primitives [Accessed: 2023-09-28].

Soerensen, A. (2023). Das Industrielle Metaverse und die Chancen fuer SICK.

Sorkine, O. (2005). Laplacian mesh processing. In Y. Chrysanthou & M. Magnor (Eds.), *Eurographics 2005 - State of the Art Reports*. The Eurographics Association.

Source, M. O. (2023). React [Accessed: 2023-09-10].

Stackpole, B. (2023, August). Solve real-world problems with the industrial metaverse.

State, G. (2023). 3D by AI: Using Generative AI and NeRFs for Building Virtual Worlds [Accessed: 2023-10-08].

Sun, C., Sun, M., & Chen, H. (2022). Direct Voxel Grid Optimization: Super-fast Convergence for Radiance Fields Reconstruction. *CVPR*.

Takikawa, T., Perel, O., Tsang, C. F., Loop, C., Litalien, J., Tremblay, J., Fidler, S., & Shugrina, M. (2022). Kaolin Wisp: A PyTorch Library and Engine for Neural Fields Research.

Tancik, M., Srinivasan, P. P., Mildenhall, B., Fridovich-Keil, S., Raghavan, N., Singhal, U., Ramamoorthi, R., Barron, J. T., & Ng, R. (2020). Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains.

Tancik, M., Casser, V., Yan, X., Pradhan, S., Mildenhall, B., Srinivasan, P., Barron, J. T., & Kretzschmar, H. (2022). Block-NeRF: Scalable Large Scene Neural View Synthesis. *arXiv*.

Tang, J., Chen, X., Wang, J., & Zeng, G. (2022a). Compressible-composable NeRF via Rank-residual Decomposition.

Tang, J., Chen, X., Wang, J., & Zeng, G. (2022b). Compressible-composable NeRF via Rank-residual Decomposition. *arXiv preprint arXiv:2205.14870*.

Tancik, M., Weber, E., Ng, E., Li, R., Yi, B., Wang, T., Kristoffersen, A., Austin, J., Salahi, K., Ahuja, A., Mcallister, D., Kerr, J., & Kanazawa, A. (2023). Nerfstudio: A Modular Framework for Neural Radiance Field Development. *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Proceedings*.

Tang, J. (2022). Torch-ngp: a PyTorch implementation of instant-ngp [Accessed: 2023-09-08].

Tao, T., Gao, L., Wang, G., Lao, Y., Chen, P., Zhao, H., Hao, D., Liang, X., Salzmann, M., & Yu, K. (2023). LiDAR-NeRF: Novel LiDAR View Synthesis via Neural Radiance Fields.

Tewari, A., Fried, O., Thies, J., Sitzmann, V., Lombardi, S., Sunkavalli, K., Martin-Brualla, R., Simon, T., Saragih, J., Nießner, M., Pandey, R., Fanello, S., Wetzstein, G., Zhu, J.-Y., Theobalt, C., Agrawala, M., Shechtman, E., Goldman, D. B., & Zollhöfer, M. (2020). State of the Art on Neural Rendering.

Tewari, A., Thies, J., Mildenhall, B., Srinivasan, P., Tretschk, E., Wang, Y., Lassner, C., Sitzmann, V., Martin-Brualla, R., Lombardi, S., Simon, T., Theobalt, C., Niessner, M., Barron, J. T., Wetzstein, G., Zollhoefer, M., & Golyanik, V. (2022). Advances in neural rendering.

Time. (2022). The best inventions [Accessed: 2023-07-12].

Tretschk, E., Tewari, A., Golyanik, V., Zollhöfer, M., Lassner, C., & Theobalt, C. (2021). Non-Rigid Neural Radiance Fields: Reconstruction and Novel View Synthesis of a Dynamic Scene From Monocular Video. *IEEE International Conference on Computer Vision (ICCV)*.

Verbin, D., Hedman, P., Mildenhall, B., Zickler, T., Barron, J. T., & Srinivasan, P. P. (2022). Ref-NeRF: Structured View-Dependent Appearance for Neural Radiance Fields. *CVPR*.

Wang, Z., Bovik, A., Sheikh, H., & Simoncelli, E. (2004). Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, *13*(4), 600–612.

Wang, Z., Wu, S., Xie, W., Chen, M., & Prisacariu, V. A. (2021). NeRF−−: Neural Radiance Fields Without Known Camera Parameters. *arXiv preprint arXiv:2102.07064*.

Watson, K., Devaux, A., Koppel, N., Chavar, A., & Whidden, P. (2022, September). Creating workflows for nerf portraiture.

Weng, C.-Y., Curless, B., Srinivasan, P. P., Barron, J. T., & Kemelmacher-Shlizerman, I. (2022). HumanNeRF: Free-Viewpoint Rendering of Moving People From Monocular Video. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 16210–16220.

Williams, L. (1983). Pyramidal parametrics. *Proceedings of the 10th annual conference on Computer graphics and interactive techniques*, 1–11.

Xie, Y., Takikawa, T., Saito, S., Litany, O., Yan, S., Khan, N., Tombari, F., Tompkin, J., Sitzmann, V., & Sridhar, S. (2022). Neural Fields in Visual Computing and Beyond. *Computer Graphics Forum*.

Xu, Y., Zoss, G., Chandran, P., Gross, M., Bradley, D., & Gotardo, P. (2023). Renerf: Relightable Neural Radiance Fields with Nearfield Lighting. *ICCV*.

Yariv, L., Hedman, P., Reiser, C., Verbin, D., Srinivasan, P. P., Szeliski, R., Barron, J. T., & Mildenhall, B. (2023). BakedSDF: Meshing Neural SDFs for Real-Time View Synthesis. *arXiv*.

Yen-Chen, L., Florence, P., Barron, J. T., Rodriguez, A., Isola, P., & Lin, T.-Y. (2021). iNeRF: Inverting Neural Radiance Fields for Pose Estimation. *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1323–1330.

Yen-Chen, L. (2020). NeRF-pytorch [Accessed: 2023-09-04].

Yoonwoo, J., Seungjoo, S., & Kibaek, P. (2022). Kakaobrain/nerf-factory: An awesome pytorch nerf library.

Yu, A., Ye, V., Tancik, M., & Kanazawa, A. (2020). pixelNeRF: Neural Radiance Fields from One or Few Images. *CoRR*, *abs/2012.02190*.

Yu, A., & Jain, A. (2023). Luma AI - Join Us.

Yue Luo, Y.-P. C. (2022). Arcnerf: nerf-based object/scene rendering and extraction framework.

Zhang, K., Riegler, G., Snavely, N., & Koltun, V. (2020). NeRF++: Analyzing and Improving Neural Radiance Fields. *arXiv:2010.07492*.

Zhang, J., Zhang, F., Kuang, S., & Zhang, L. (2023). NeRF-LiDAR: Generating Realistic LiDAR Point Clouds with Neural Radiance Fields.

Zhi, S., Laidlow, T., Leutenegger, S., & Davison, A. J. (2021). In-Place Scene Labelling and Understanding with Implicit Scene Representation. *ICCV*.

Utilization of Artificial Intelligence Technologies

In the era of digital transformation, AI has emerged as a key element in various fields. This chapter explains the AI technologies used in this thesis.

6.3. DeepL Write (`https://www.deepl.com/write`)

DeepL Write is an advanced text-editing service that leverages machine learning technologies. The tool is specifically designed to augment the quality of text through spelling, grammar, and sentence structure corrections.

DeepL Write was utilized in this research for multiple objectives:

- Text Correction: The tool played a significant role in enhancing spelling, grammar, and sentence structure.
- Quality Enhancement: DeepL Write contributed to elevating the overall textual quality of the thesis to an academically rigorous standard.

6.4. Chat-GPT (`https://chat.openai.com/`)

The Generative Pre-trained Transformer (GPT), commonly known as Chat-GPT, is an advanced conversational agent developed by OpenAI. Built on the Transformer architecture, it has been trained on a vast dataset that includes diverse portions of the internet. The model excels in generating human-like text and has a wide array of applications, ranging from automating customer service to aiding in academic research .

Chat-GPT was deployed for multiple purposes in this thesis:

- Caption Annotation: Chat-GPT assisted in the labeling and descriptions of captions.
- Content Review: Various sections of this research were reviewed and enhanced through Chat-GPT's generative capabilities.
- Formulation Assistance: In instances where phrasing or formulation posed challenges, Chat-GPT was employed to provide alternative expressions or clarify complex ideas.

Statutory declaration

I herewith declare that I have completed the present report independently, without making use of other than the specified literature and aids. All parts that were taken from published and non-published texts either verbally or in substance are clearly marked as such. The use of AI applications is detailed in the relevant part of the thesis, in terms of type and extent.

This report has not been presented to any examination office in the same form.

I am aware that a false declaration may have legal consequences.

Furtwangen, 10.10.2023 Sabine Schleise

A. Improvements of NeRF

A.1. A Multiscale Representation for Anti-Aliasing Neural Radiance Fields

This section addresses the substantial limitations encountered when rendering images with varying resolutions or capturing scenes from different distances, as depicted in Figure 51. While the renderings produced with NeRF are visually impressive at full resolution and achieve a high structural similarity score (structural similarity (SSIM)) Wang et al. (2004), which serves as a quantitative measure of the perceptual quality and similarity between the rendered image and the ground truth, problems arise when the renderings are viewed from a distance or at different resolutions. These problems manifest themselves in the form of jagged artifacts or aliasing, resulting in a significant drop in the SSIM score. The following video 52 provides a visual demonstration of these problems, where the ground truth is displayed on the right side and the jagged artifacts are observed in the NeRF renderings on the left side, which were generated using training inputs at the same distance, labeled as *Single Scale Training* in the video. Trying to address this problem by training neural radiance fields on multi-resolution data have not provided a satisfactory solution. While there is a slight improvement in performance at low resolutions, the quality worsens at higher resolutions, which is undesirable. In particular, the NeRF renderings exhibit excessive blurring in close-up views and exhibit aliasing artifacts in distant views. To overcome these challenge, the authors Barron et al. of the original NeRF proposed the mip-NeRF algorithm, a multiscale representation for anti-aliasing Neural Radiance Fields. This approach aims to reduce the aliasing and scale-related issues encountered with rendering the original neural radiance field.

mip-NeRF (multum in parvo NeRF as in "mipmap" Barron et al. (2021)) is inspired by the mipmapping Williams (1983) technique commonly used in computer graphics rendering pipelines to mitigate aliasing artifacts. Mipmapping involves generating a series of scaled-down versions, known as mipmaps, of a signal such as an image or texture map. The selection of an appropriate mipmap scale for a ray is based on the projection of the pixel footprint onto the intersected geometry. However, implementing this technique directly for neural volumetric representations like NeRF can be computationally intensive, as it requires numerous MLP evaluations to render a single ray and considerable time to reconstruct a scene Barron et al. (2021).

Figure 51.: In the original NeRF paper Mildenhall et al. (2020), the main results were on scenes of objects floating in space, with a hemisphere of cameras around each object pointing inwards, visualized as camera frustums. Notably, all cameras are placed equidistant from the object, allowing NeRF to perform view synthesis without scale or aliasing considerations. However, the introduction of additional cameras placed at varying distances from the object reveals the limitations of NeRF as a single-scale model attempting to address a multi-scale problem Barron (2021, March).

The proposed solution, mip-NeRF, extends the capabilities of NeRF by simultaneously representing the prefiltered radiance field across a continuous range of scales as shown in figure 53. In mip-NeRF, a $3D$ Gaussian is used as input to define the region over which the radiance field should be integrated. Renderings of prefiltered pixels are obtained by querying mip-NeRF at intervals along a cone, utilizing Gaussians to approximate the conical frustums corresponding to each pixel. To encode a $3D$ position and its surrounding Gaussian region, a new encoding called integrated positional encoding (IPE) is introduced. IPE is an extension of NeRF's Positional Encoding (PE) and enables the compact representation of a region in space, rather than just a single point. While PE maps a single point in space to a feature vector, IPE considers Gaussian regions of space, allowing the network to reason about sampling and aliasing. This enables a natural way to input a region of space as a query to a coordinate-based neural network. As the region widens, higher frequency features automatically diminish, providing lower-frequency inputs to the network. Conversely, as the region narrows, these features converge to the original positional encoding. By utilizing IPE, NeRF is trained to generate anti-aliased renderings. Instead of casting infinitesimal rays through each pixel, mip-NeRF casts a cone. For each queried point along a ray, the associated 3D conical frustum is considered. Different cameras viewing the same point can result in distinct conical frustums. To incorporate this information into the NeRF network, a multivariate Gaussian is fitted to the conical frustum, and the IPE is employed to create the input feature vector for the network.

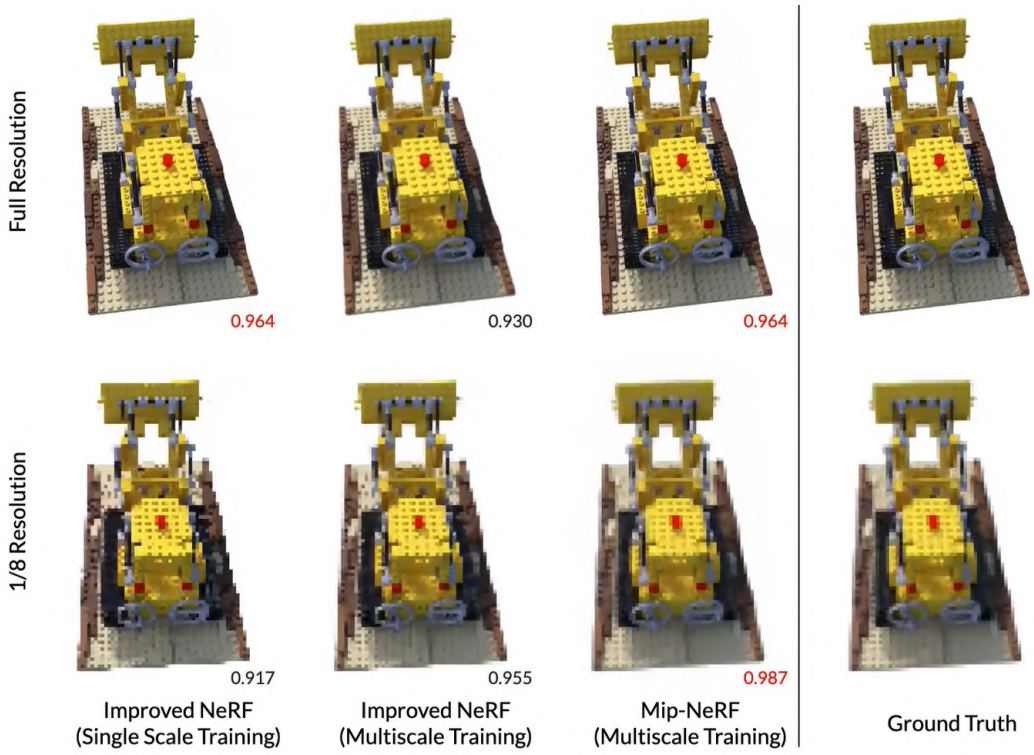


Figure 52.: The image showcases the challenges faced by NeRF in rendering scenes. On the right side, the ground truth images are displayed. The first left side shows the results of training a neural radiance field with single-scale training, while the next image demonstrates the results after applying multiscale training. These renderings exhibit jagged artifacts and aliasing when viewed at different resolutions. Finally, next to the ground truth, the results from the new approach mipnerf are displayed, which aims to address these challenges and improve the rendering quality.

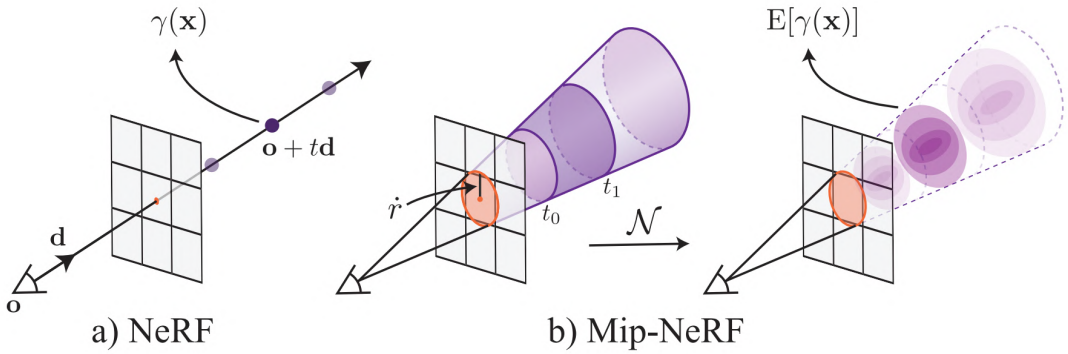


Figure 53.: NeRF (a) employs point sampling along rays traced from the camera's center of projection through each pixel. These points are then encoded using a PE)(γ), generating the corresponding feature $\gamma(x)$. In contrast, Mip-NeRF (b) operates by reasoning about the $3D$ conical frustum associated with a camera pixel. These conical frustums are featurized using an IPE method. The IPE approximates the frustum with a multivariate Gaussian and computes the closed-form integral $E[\gamma(x)]$ over the positional encodings of the coordinates within the Gaussian Barron et al. (2021).

In summary,mip-NeRF renders anti-aliased conical frustums instead of rays, reducing aliasing artifacts and improving the ability to represent fine details. It demonstrates a rendering speed that is 7% faster and a model size that is half that of NeRF. Moreover, mip-NeRF achieves a 17% reduction in average error rates on the dataset used with NeRF and a significant 60% reduction on a challenging multiscale variant of the dataset Barron et al. (2021).

A.2. Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields

NeRF also struggles to produce high-quality renderings for large unbounded scenes, where the camera can point in any direction and content can exist at any distance. In this context, conventional NeRF-like models often face challenges such as the generation of blurred or low-resolution renderings (resulting from the imbalance in detail and scale between near and far objects), slow training times, and potential artefacts arising from the inherent ambiguity of reconstructing a large scene from a limited set of images. To address these challenges, the authors Barron et al. (2022) propose an extension of mip-NeRF that uses nonlinear scene parameterisation, online distillation, and a novel distortion-based regulariser. This model "is called "mip-NeRF 360" because it targets scenes in which the camera rotates 360 degrees around a point Barron et al. (2022).

The authors Barron et al. address these issues by introducing several key enhancements to mip-NeRF. Firstly, they employ a nonlinear scene parameterization to handle the representation of unbounded scenes, as mip-NeRF inherently requires a bounded domain. This is achieved by wrapping the Gaussian distributions used in mip-NeRF into a non-Euclidean space using a technique reminiscent of an extended Kalman filter. This allows for the representation of scenes that extend indefinitely and will be explained in detail. Furthermore, the authors tackle the challenge of handling highly detailed large scenes. Simply increasing the size of the neural network underlying NeRF leads to slow training times. To address this, they employ a two-stage optimization process. They train a small proposal MLP to approximate the geometry predicted by a larger NeRF MLP. This approach enables faster training while still capturing the intricate details present in the scene. Lastly, the inherent ambiguity of 3D reconstruction in larger scenes often results in artifacts. To mitigate this, the authors introduce a novel distortion-based regularizer specifically designed for mip-NeRF ray intervals. This regularizer helps improve the overall quality of the rendered images by addressing the ambiguity and reducing artifacts. By incorporating these enhancements, mip-NeRF 360 offers improved capabilities for rendering large unbounded scenes, addressing issues related to representation, training efficiency, and artifact reduction Barron et al. (2022).

In mip-NeRF, the parameterization of unbounded 360-degree scenes in an infinite Euclidean space presents a challenge of imposing a bounded domain on the 3D scene coordinates. To address this, a Kalman-like approach is employed, utilizing Gaussians for parameterization. The scene parameterization is visualized in Figure 54. The contract operator, represented by the function contract$(\cdot)$, maps the coordinates onto a ball of radius 2. Points within a radius of 1 remain unaffected, while points outside this range are contracted towards the origin. Mathematically, the contract$(\cdot)$ operation can be defined as

$$\text{contract}(x) = \begin{cases} x, & ||x|| \leq 1 \\ \left(2 - \frac{1}{||x||}\right)\left(\frac{x}{||x||}\right), & ||x|| > 1. \end{cases} \tag{A.1}$$

This contract$(\cdot)$ operation is applied to the Gaussians of mip-NeRF in the Euclidean 3D space, resembling the behavior of a Kalman filter. The resulting contracted Gaussians, represented by red ellipses in the visualization, ensure that their centers lie within a ball of radius 2. By combining the contract$(\cdot)$ design with linearly spaced ray intervals based on disparity, rays cast from a camera positioned at the origin of the scene exhibit equidistant intervals within the orange region, as demonstrated in 54. This parameterization technique enables the effective representation of unbounded scenes while maintaining a bounded domain for the 3D coordinates in mip-NeRF Barron et al. (2022).

For further details and in-depth explanations of the efficiency considerations and the mitigation of ambiguity in mip-NeRF 360, please refer to the original paper Barron et al. (2022).

A.3. Zip-NeRF: Combining Grid-Based Techniques and mip-NeRF 360

NeRF training can benefit from the use of grid-based representations in mapping spatial coordinates to colors and volumetric density. However, these grid-based approaches often lack an explicit understanding of scale, leading to aliasing issues such as jaggies or missing scene content. To address this challenge, the concept of anti-aliasing was previously introduced in mip-NeRF 360, which operates by reasoning about sub-volumes along a cone instead of individual points along a ray. However, this approach is not natively compatible with current grid-based techniques. In this paper, the authors Barron et al. present Zip-NeRF, a novel technique that combines the benefits of mip-NeRF 360 and grid-based models like Instant NGP. Zip-NeRF leverages ideas from rendering and signal processing to achieve superior performance compared to previous methods. Specifically, Zip-NeRF achieves error rates that are 8% to 77% lower than either prior technique and offers a training speed that is 24

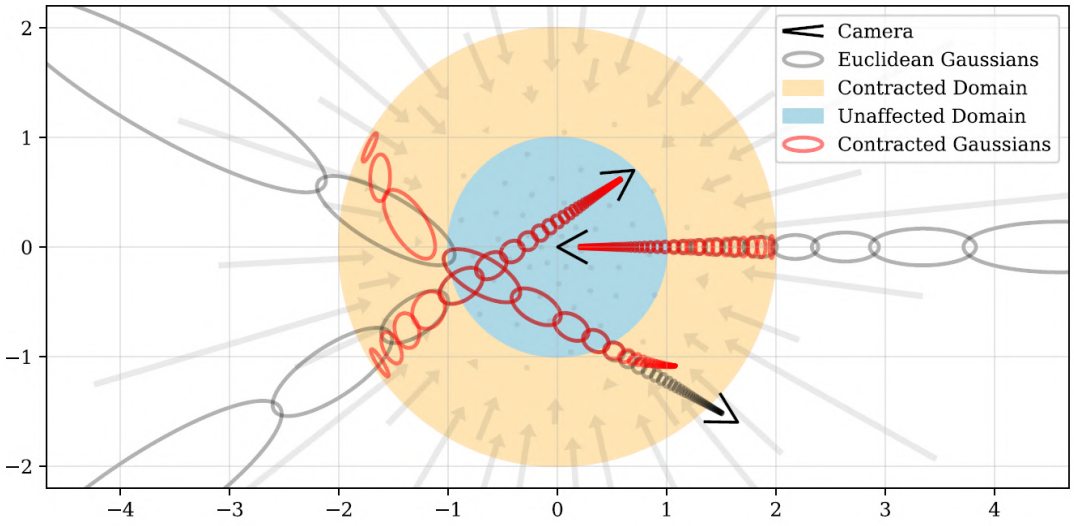


Figure 54.: $2D$ visualization of the scene parameterization. The contract$(\cdot)$ operator maps coordinates onto a ball of radius 2 (orange), where points within a radius of 1 (blue) remain unaffected. The contract$(\cdot)$ operation is applied to the Gaussians of mip-NeRF in Euclidean 3D space (gray ellipses), similar to a Kalman filter. The contracted Gaussians (red ellipses) guarantee that their centers lie within a ball of radius 2. The combination of the contract$(\cdot)$ design and the linear spacing of ray intervals based on disparity ensures equidistant intervals for rays cast from a camera at the origin of the scene, as demonstrated in this visualization Barron et al. (2022).

times faster than mip-NeRF 360. The combination between these methods yields promising results, as showcased in Video 57.

One of the key components of Zip-NeRF is the use of multisampling to approximate the average NGP feature over a conical frustum. The authors construct a 6-sample pattern that precisely matches the first and second moments of the frustum. During training, they randomly rotate and flip each pattern, while during rendering, they deterministically flip and rotate each adjacent pattern by 30 degrees as shown in figure 55. This multisampling technique enhances the accuracy of Zip-NeRF's predictions and reduces aliasing artifacts. Another important aspect addressed by Zip-NeRF is the artifact known as z-aliasing, which arises from the proposal network used for resampling points along rays in mip-NeRF 360. Z-aliasing manifests as foreground content intermittently appearing and disappearing as the camera moves towards or away from the scene. This artifact occurs when the initial set of samples from the proposal network is not dense enough to capture thin structures, resulting in missed content that cannot be recovered by subsequent rounds of sampling. To mitigate z-aliasing, Zip-NeRF introduces improvements to the proposal network supervision, resulting in a prefiltered proposal output that preserves foreground objects consistently across frames.

For more detailed information and comprehensive results regarding Zip-NeRF, we

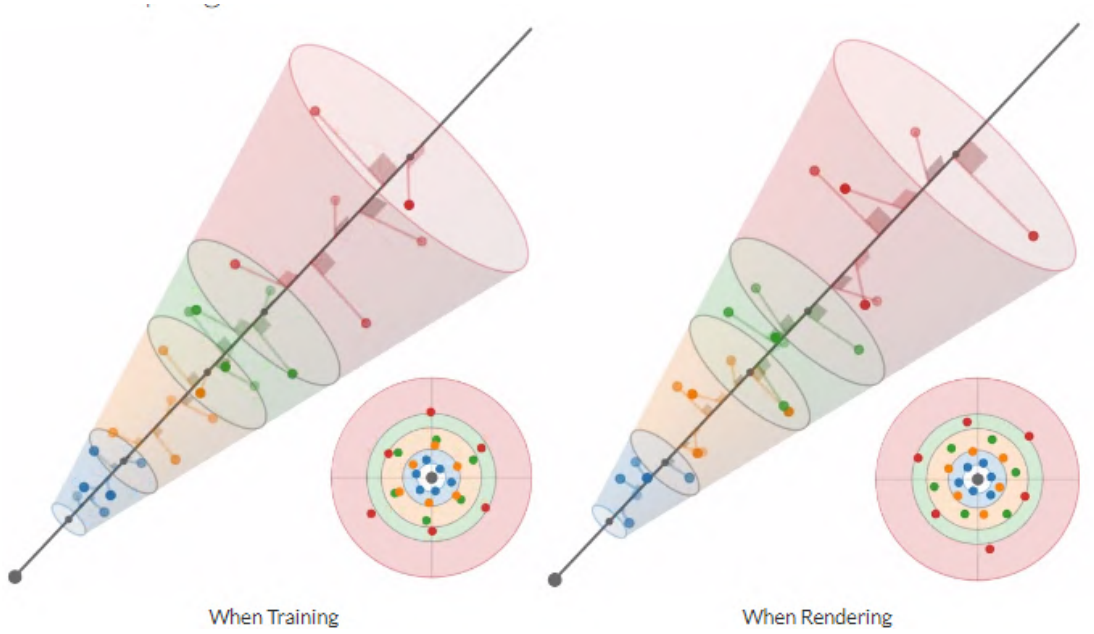


Figure 55.:]
Approximating the average NGP feature over a conical frustum using multisampling. A 6-sample pattern is constructed to precisely match the frustum's first and second moments. During training, patterns are randomly rotated and flipped (along the ray axis), while during rendering, adjacent patterns are deterministically flipped and rotated by 30 degrees Barron et al. (2023).

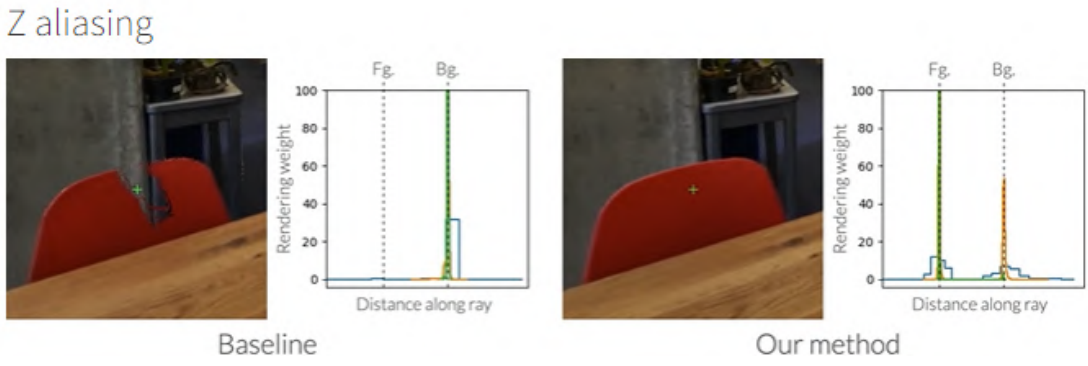


Figure 56.: The improvements to proposal network supervision result in a prefiltered proposal output that preserves the foreground object for all frames in this sequence. The plots above depict samples along a ray for three rounds of resampling (blue, orange, and green lines), with the y axis showing rendering weight (how much each interval contributes to the final rendered color), as a normalized probability density. Barron et al. (2023).

Figure 57.: This figure displays a screenshot of a Zip-NeRF rendered video. The high-quality result is impressive.

refer interested readers to the original paper Barron et al. (2023).

A.4. MeRF and BakedSDF

Existing representations of radiance fields suffer from either high computational demands, making real-time rendering impractical, or excessive memory requirements, limiting scalability for large scenes. Recognizing these challenges, the authors of NeRF have recently focused on overcoming these limitations. In their latest publications, they have made significant efforts to strike a balance between computational efficiency and memory usage, resulting in more practical and scalable implementations of radiance fields.

One notable contribution is the introduction of MERF, a memory-efficient Radiance Field representation that enables real-time rendering of large-scale scenes in a browser. MERF achieves this by reducing the memory consumption of previous sparse volumetric radiance fields through a combination of a sparse feature grid and high-resolution 2D feature planes. Additionally, to support large-scale unbounded scenes, the authors propose a novel contraction function that maps scene coordinates into a bounded volume while still facilitating efficient ray-box intersection. Moreover, the authors have designed a lossless procedure to bake the parameterization used during training into a model. This approach ensures that the resulting model achieves real-time rendering capabilities while preserving the photorealistic view synthesis quality characteristic of a volumetric radiance field Reiser et al. (2023).

In addition to the advancements in MERF, the authors also present another method

called "BakedSDF" that addresses the reconstruction of high-quality meshes for large unbounded real-world scenes, facilitating photorealistic novel view synthesis. The authors optimize a hybrid neural volume-surface representation designed to have well-behaved level sets that accurately correspond to the surfaces within the scene. This representation is then baked into a high-quality triangle mesh, which is augmented with a fast and straightforward view-dependent appearance model based on spherical Gaussians. The authors further optimize this baked representation to faithfully reproduce the captured viewpoints, enhancing the fidelity of the model. By leveraging accelerated polygon rasterization pipelines on commodity hardware, the approach enables real-time view synthesis. Notably, this method outperforms previous scene representations in terms of accuracy, speed, and power consumption, making it a superior choice for real-time rendering. Additionally, the high-quality meshes generated by the approach offer numerous possibilities for applications such as appearance editing and physical simulation Yariv et al. (2023).

By integrating the capabilities of BakedSDF and MERF, the authors have made substantial progress in achieving realistic and scalable rendering of radiance fields for various applications.

B. Dockerfile

```
FROM nvidia/cuda:11.8.0-devel-ubuntu22.04

ARG COLMAP_VERSION=dev
ENV CERES_SOLVER_VERSION=2.0.0
ARG CUDA_ARCHITECTURES=89
ENV PATH=/usr/local/cuda/bin:$PATH

RUN apt-get update && apt-get install -y python-is-python3 libsm6 libxrender1
    libfontconfig1
# Prevent stop building ubuntu at time zone selection.
ENV DEBIAN_FRONTEND=noninteractive

RUN apt-get update &&  \
apt-get install -y python3-pip
RUN pip install --upgrade --no-cache-dir pip

# Prepare and empty machine for building.
RUN apt-get update && apt-get install -y \
ffmpeg \
git \
cmake \
ninja-build \
build-essential \
libboost-program-options-dev \
libboost-filesystem-dev \
libboost-graph-dev \
libboost-system-dev \
libboost-test-dev \
libeigen3-dev \
libflann-dev \
libfreeimage-dev \
libmetis-dev \
libgoogle-glog-dev \
libgflags-dev \
libsqlite3-dev \
libglew-dev \
qtbase5-dev \
libqt5opengl5-dev \
libcgal-dev \
libceres-dev \
nano \
python-is-python3 \
ffmpeg \
curl \
wget

# Install Node.js
RUN mkdir temp && cd temp && wget https://nodejs.org/dist/v16.16.0/node-v16.16.0-
    linux-x64.tar.xz && \
tar -xf node-v16.16.0-linux-x64.tar.xz --strip-components=1 -C /usr/local
```

```
49
50 RUN npm config set registry=https://deagxartifactory.sickcn.net/artifactory/api/npm/
       npm/
51 RUN npm config set proxy=http://cloudproxy-sickag.sickcn.net:10415
52 RUN npm config set https-proxy=http://cloudproxy-sickag.sickcn.net:10415
53 RUN npm config set noproxy=localhost,127.0.0.1,localaddress,.localdomain.com,.sickcn.
       net
54 RUN npm config set cafile=/etc/ssl/certs/ca-certificates.crt
55
56 ENV PATH="/usr/local/bin:${PATH}"
57
58 RUN rm -r temp
59 # Install Ceres-solver (required by colmap).
60 RUN echo "Installing Ceres Solver ver. ${CERES_SOLVER_VERSION}..." \
61 && cd opt \
62 && git clone https://github.com/ceres-solver/ceres-solver \
63 && cd ceres-solver \
64 && git checkout ${CERES_SOLVER_VERSION} \
65 && mkdir build \
66 && cd build \
67 && cmake ../ -DBUILD_TESTING=OFF -DBUILD_EXAMPLES=OFF \
68 && make -j \
69 && make install && \
70 cd ../.. && \
71 rm -rf ceres-solver
72
73 RUN echo "Installing COLMAP ver. ${COLMAP_VERSION}..."
74 RUN git clone https://github.com/colmap/colmap.git
75 RUN cd colmap && \
76 git reset --hard ${COLMAP_VERSION} && \
77 mkdir build && \
78 cd build && \
79 cmake .. -GNinja -DCMAKE_CUDA_ARCHITECTURES=89 && \
80 ninja && \
81 ninja install && \
82 cd .. && rm -rf colmap
83
84 # Build instant-nerf wihout gui
85 RUN echo "Installing Instant-NeRF"
86 RUN  git clone --recursive https://github.com/nvlabs/instant-ngp
87 RUN cd instant-ngp && \
88 cmake -DNGP_BUILD_WITH_GUI=off ./ -B ./build && \
89 cmake --build build --config RelWithDebInfo -j 16
90
91 ADD . nerf-industrial-metaverse
92 RUN cp instant-ngp/build/pyngp.cpython* /nerf-industrial-metaverse
93 RUN cd nerf-industrial-metaverse && pip install -e .
```

Code Listing 23: Dockerfile created for this project

C. Installation of Pytorch with CUDA and Tiny-Cuda-NN

Some of the frameworks discussed in Chapter 3 included CUDA, PyTorch and/or tiny-CUDA-NN in their projects. Installing these components presented a number of challenges, which are discussed in more detail in this section.

To use CUDA, it is essential to first install the compatible CUDA version for the used graphics card. The list of CUDA versions supported by various graphics cards can be referenced on the Wikipedia page about CUDA. In parallel, it is crucial to ensure that the graphics driver aligns with the intended CUDA version. This compatibility information is available on on NVIDIA's official driver download page. In our specific case, the graphics card were compatible with CUDA version 12, but it's worth noting that it's possible to downgrade to an earlier version if necessary.

Following the establishment of the CUDA environment, the installation of PyTorch is the subsequent step. PyTorch is a comprehensive framework for developing deep learning models and is written in Python, making it accessible and user-friendly for machine learning developers. A direct PyTorch installation via `pip install torch` defaults to the latest version without CUDA support. It is therefore recommended to install a PyTorch binary that supports the CUDA version of the graphics card. Configuration guidelines and recommendations are provided on the official PyTorch website. For example, to install PyTorch 2.0.1 with CUDA 11.8, use the following command:

```
pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/
  whl/cu118
## or
python3 -m pip install torch==2.0.1+cu118 torchvision==0.15.2+cu118 --extra-index-
  url https://download.pytorch.org/whl/cu118
```

Code Listing 24: PyTorch Installation with CUDA

We can check for successful installation with the following Python commands, which will return `true` if CUDA is available:

```
import torch
torch.cuda.is_available()
```

Code Listing 25: Verify Pytorch Installation

At this point I would like to refer to the Stack Overflow question page, where the user Jodac, who qualifies as a research scientist, has written a detailed and clear instruction.

The installation of the framework Tiny-CUDA-NN is also non-trivial. This framework is a concise, independent system proficient in training and querying neural net-

works. It features a rapid "fully fused" multi-layer perceptron, a flexible multiresolution hash encoding, and supports a broad spectrum of input encodings, losses, and optimizers. Tiny-CUDA-NN offers a PyTorch extension, enabling users to utilize its swift MLPs and input encodings within a Python environment. However, if PyTorch is already installed, Tiny-CUDA-NN can only be successfully installed if the PyTorch installation has the correct CUDA version. If the basic PyTorch installation was done via pip, the installation of Tiny-CUDA-NN will fail as it relies on the CUDA compute capability, which needs to be specified as an environment variable.

```
ENV TCNN_CUDA_ARCHITECTURES=${CUDA_ARCHITECTURES}
RUN python3.10 -m pip install git+https://github.com/NVlabs/tiny-cuda-nn.git@v1.6#
  subdirectory=bindings/torch
```

With accurate graphics card specifications and adherence to the correct installation processes, including the selection of the appropriate CUDA version, both PyTorch and tiny-CUDA-NN can be installed without complications.

D. Structure from Motion with COLMAP

In our research, we used COLMAP (Schönberger and Frahm, 2016), an open-source tool that embodies the algorithms of the SfM method. By executing the command `ngp-colmap`, inspired by the code in (Mildenhall et al., 2020; Müller et al., 2022), a set of COLMAP commands is activated. These commands represent different stages of the process, ranging from feature extraction to bundle matching. In the following section we elaborate on these algorithms to clarify how SfM calibrates the camera.

The initial phase involves performing a correspondence search to detect scene overlaps within the input images, denoted as $I = \{I_i \mid i = 1..N_I\}$. This search aims to identify the projections of identical points that appear in multiple overlapping images. The outcome of this process consists of a collection of image pairs, $\overline{C}$, which have been geometrically verified, along with a graph that represents the image projections for each identified point.

Using the `colmap feature extractor` command, COLAMP processes each image I_i and obtains a set of local features $F_i = \{(x_j, f_j) \mid j = 1..N_{F_i}\}$ specific to that image. Here, N_{F_i} represents the total number of local features detected in image I_i. These features are carefully designed to maintain invariance under both radiometric and geometric variations, enabling SfM to accurately and uniquely recognize them across multiple images. Among the various feature extraction options available, Scale Invariant Feature Transform (SIFT) stands out as the gold standard in terms of robustness and will be utilized in this context (Schönberger and Frahm, 2016). This command can be fine-tuned with various parameters to enhance the output. For instance, when working with a singular camera, the Camera Model can be preset to *OPENCV*. The following code 26 succinctly demonstrates this feature extraction procedure, including the tailored parameters:

```
feature_extractor_args = [
"colmap",
'feature_extractor',
'--database_path',
os.path.join(scene_dir, 'database.db'),
'--image_path',
os.path.join(scene_dir, 'images'),
'--ImageReader.camera_model=OPENCV',
'--SiftExtraction.use_gpu=true',
'--SiftExtraction.estimate_affine_shape=true',
'--SiftExtraction.domain_size_pooling=true',
'--ImageReader.single_camera=1',
'--SiftExtraction.gpu_index=0'
```

```
]
logger.add_log('Starting Feature Extraction...')
run_subprocess(feature_extractor_args, logger, logfile)
logger.add_log('    Features extracted...')
```

Code Listing 26: The command `colmap feature_extractor` with its input options

Once features have been extracted from a set of images using techniques like SIFT, the next step in SfM is feature matching, executed using the command `colmap feature_matching`. At its core, this process aims to identify shared features across different images to deduce which images depict overlapping parts of a scene. A straightforward approach to this involves testing every possible image pair for scene overlap. This is achieved by searching for feature correspondences, where each feature in image I_a is compared to every feature in image I_b using a similarity metric that compares their appearances f_j. However, this naive approach has a computational complexity of $O(N_I \cdot N_{F_i}^2)$, making it impractical for large image collections. To address this challenge, various approaches have been developed to make the matching process scalable and efficient. The ultimate output of this stage is a set of potentially overlapping image pairs $C = \{\{I_a, I_b\} \mid I_a, I_b \in I, a < b\}$, along with their associated feature correspondences $M_{ab} \in F_a \times F_b$. These correspondences are essential for establishing the relationships between different images and are crucial for subsequent $3D$ reconstruction and scene understanding (Schönberger and Frahm, 2016).

Notably, in our setup, the default matching type is set to the `sequential matcher`. This mode proves useful when the images are captured sequentially, as is the case with video camera footage. As consecutive frames typically have visual overlap, there's no need to exhaustively match all image pairs. Instead, the consecutively captured images are matched against each other. By modifying the YAML file, one can switch the matcher type, for instance, to `exhaustive`, which might be more suited for standalone images.

```
feature_matcher_args = [
"colmap",
"sequential_matcher",
'--database_path',
os.path.join(scene_dir, 'database.db'),
'--SiftMatching.use_gpu=true',
'--SiftMatching.guided_matching=true',
'--SiftMatching.gpu_index=0',
]
logger.add_log('Starting Feature Matching...')
run_subprocess(feature_matcher_args, logger, logfile)
logger.add_log('    Features matched...')
```

Code Listing 27: The command `colmap feature_matching` with its additional input

The matching stage also involves the verification of potentially overlapping image pairs $\overline{C}$. As matching is based solely on appearance, it cannot guarantee that the corresponding features actually correspond to the same scene points. To address

this, SfM attempts to estimate transformations that map feature points between images using projective geometry. Depending on the movements of the camera and the nature of the scene, different mathematical models can be used to describe the relationship between features in different images. For example, a homography H is used to describe the transformation of a purely rotating or moving camera capturing a planar scene. Epipolar geometry, on the other hand, is used to describe the relation for a moving camera through the essential matrix E (calibrated) or the fundamental matrix F (uncalibrated). To verify the matches, SfM attempts to find a valid transformation that maps a sufficient number of features between the images. This process is essential to establish the geometric accuracy of the matched pairs. However, since the correspondences from matching may be contaminated with outliers, robust estimation techniques like RANSAC are employed to handle the outliers effectively. The output of this stage is a set of geometrically verified image pairs $\overline{C}$, their associated inlier correspondences $\overline{M}_{ab}$ and optionally a description of their geometric relation G_{ab}. COLMAP creates with the `feature matching` command a so-called scene graph with images as nodes and verified pairs of images as edges (Schönberger and Frahm, 2016).

This scene graph works as input to the reconstruction stage. The main objective of this stage is to deduce pose estimates $P = \{P_c \in SE(3) \mid c = 1...N_P\}$ for previously registered images and to specify the reconstructed spatial scene structure as a set of points $X = \{X_k \in \mathbb{R}^{\not\vdash} \mid k = 1..N_X\}$. To achieve this, we utilize the `colmap mapper` command. The process begins with the creation of a basic model derived from a carefully selected two-view reconstruction. After obtaining a metric reconstruction, new images can be registered to the refine the existing model using the Perspective-n-Point (PnP) problem, which involves feature correspondences with already triangulated points in registered images ($2D$-$3D$ correspondences). In case of calibrated cameras, the PnP problem's pose estimation often employs RANSAC and a minimal pose solver, whereas uncalibrated cameras may use various minimal solvers or sampling-based approaches. For a newly registered image, COLAMP observes existing scene points and expands the scene's coverage by adding new points to the set of points X through triangulation. To triangulate and incorporate a new scene point X_k into X, COLMAP requires registration of at least one additional image that offers a different viewpoint of the new scene. Triangulation is a critical step in SfM as it enhances the stability of the existing model through redundancy and enables the registration of new images by providing additional $2D$-$3D$ correspondences. Image registration and triangulation are separate procedures, but their outcomes are highly correlated, as uncertainties in camera pose affect triangulated points and vice versa. Further, additional triangulations may enhance the initial camera pose through increased redundancy.

By default, COLMAP keeps the principal point constant during the reconstruction, as principal point estimation is an ill-posed problem in general. Once all images are reconstructed, the problem is most often constrained enough that you can try to refine the principal point in global bundle adjustment, especially when sharing intrinsic parameters between multiple images. After constructing a sparse model of the scene, we apply Bundle Adjustment to fine-tune the principal point by minimizing the "reprojection error" E. This error essentially measures the difference between the observed positions of points in an image and the projected positions of these points based on our current understanding of camera parameters and point locations. Mathematically, this error can be represented as:

$$E = \sum_j p_j \left(||\pi(P_C, X_k) - x_j||_2^2 \right)$$

The task of minimizing projection error is tackled by several algorithms within the field, two of which are noteworthy. The Levenberg-Marquardt algorithm, a commonly employed approach, is widely recognized for its effectiveness. Additionally, the Schur complement trick is another strategy motivated by the unique parameter structure encountered in Bundle Adjustment (BA) problems. This technique involves a two-step process: firstly, solving the reduced camera system, followed by the subsequent update of points via back-substitution, as detailed in the work by Schönberger and Frahm (2016).

E. Generating the transformation matrix

To address the absence of documentation on the transformation matrix in instant-ngp, we designed a simple `transform.json` file, which allows for specific and explicit changes to the parameters, providing a clear view of how the transformation matrix is defined in instant-ngp. In this JSON structure, we integrated frames accompanied by black images, simultaneously varying the parameters of the transformation matrix. Figure 58 depicts the coordinate system alongside the cameras we instituted via our JSON in instant-ngp. The distinct axes are color-coded for clarity: blue signifies the forward (z-axis) direction, red symbolizes the side (x-axis) perspective, and green indicates the upward (y-axis) orientation. We strategically positioned cameras at every vertex of this coordinate system, marking their respective translation matrices. For instance, the primary camera, centrally placed at the coordinate $(0, 0, 0, 1)$, remains with no translation and has the transformation matrix:

$$\text{transform\_matrix} = \begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

Adjustments to the first translation value result in camera translation along the z-axis. Modifying the second translation value causes the camera to traverse the x-axis, and setting the translation to $(0, 0, 1, 1)$ moves the camera vertically along the y-axis. The transformation matrix can therefore be represented as:

$$\text{transformation\_matrix} = \begin{bmatrix} 0 & 0 & 0 & Z \\ 0 & 0 & 0 & X \\ 0 & 0 & 0 & Y \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We can gain further insight into this transformation matrix by examining the instant-ngp script, which implemented how the COLMAP input is converted into the matrix. Additionally, we intend to investigate the rotations as part of our comprehensive analysis.

The *images.txt* file offers details on the pose and keypoints of every reconstructed image within the dataset, with each image being described over two lines.

```
# Image list with two lines of data per image:
# IMAGE_ID, QW, QX, QY, QZ, TX, TY, TZ, CAMERA_ID, NAME
```

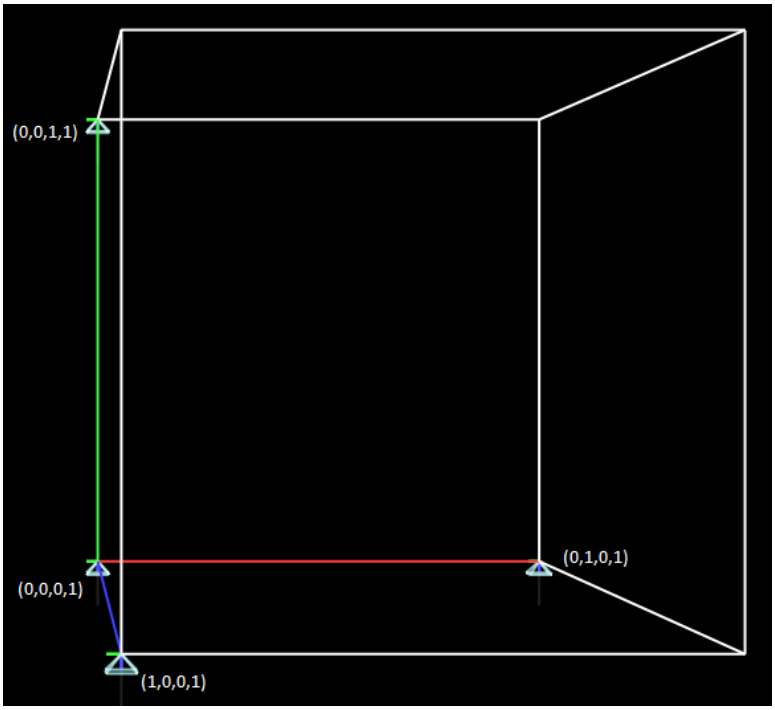


Figure 58.: The coordinate system as applied in instant-ngp. The blue, red, and green axes represent the z, x, and y dimensions, respectively. Cameras are strategically positioned at each system corner, with their corresponding translation matrices annotated. The central camera, situated at $(0, 0, 0, 1)$, remains stationary. Adjusting the matrix's first value moves the camera along the z-axis, the second value adjustment translates it across the x-axis, and setting it to $(0, 0, 1, 1)$ elevates the camera along the y-axis.

```
# POINTS2D[] as (X, Y, POINT3D_ID)
# Number of images: 2, mean observations per image: 2
1 0.851773 0.0165051 0.503764 -0.142941 -0.737434 1.02973 3.74354 1 P1180141.JPG
2362.39 248.498 58396 1784.7 268.254 59027 1784.7 268.254 -1
2 0.851773 0.0165051 0.503764 -0.142941 -0.737434 1.02973 3.74354 1 P1180142.JPG
1190.83 663.957 23056 1258.77 640.354 59070
```

The pose of the reconstructed image is represented through a quaternion (Q_W, Q_X, Q_Y, Q_Z) and a translation vector (T_X, T_Y, T_Z). This quaternion adheres to the Hamilton convention, a standard used by the Eigen library. The camera's local coordinate system is oriented such that the X-axis points rightward, the Y-axis downward, and the Z-axis forward, all from the perspective of the image (Schönberger and Frahm, 2016).

The adaptive code, sourced from *instant-ngp*, initiates its operation by loading the `images.txt` file and subsequently transforms this input into a matrix. Below is a snippet illustrating this transformation:

```
qvec = np.array(tuple(map(float, elems[1:5])))
tvec = np.array(tuple(map(float, elems[5:8])))
R = qvec2rotmat(-qvec)
t = tvec.reshape([3, 1])
m = np.concatenate([np.concatenate([R, t], 1), bottom], 0)
c2w = np.linalg.inv(m)
c2w[0:3, [1, 2]] *= -1  # flip the y and z axis
```

```
8  c2w = c2w[[1, 0, 2, 3], :]
9  c2w[2, :] *= -1  # flip whole world upside down
```

Code Listing 28: Converting COLMAP's extrinsics to the transformation matrix

To understand the given Python code more deeply, let's break down its operation mathematically:

- A quaternion $q = [q_w, q_x, q_y, q_z]$ is extracted from the input and is converted to a 3x3 rotation matrix R using the given `qvec2rotmat` function. The matrix R is represented as (NVlabs, 2022):

$$R = \begin{bmatrix} 1 - 2q_y^2 - 2q_z^2 & 2q_xq_y - 2q_wq_z & 2q_xq_z + 2q_wq_y \\ 2q_xq_y + 2q_wq_z & 1 - 2q_x^2 - 2q_z^2 & 2q_yq_z - 2q_wq_x \\ 2q_xq_z - 2q_wq_y & 2q_yq_z + 2q_wq_x & 1 - 2q_x^2 - 2q_y^2 \end{bmatrix}$$

- This rotation matrix R is then concatenated with the translation vector t to form a 4x4 transformation matrix M:

$$M = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}$$

- Next, in line 5 in 28, the matrix is inverted to convert the camera-to-world transformation into a world-to-camera transformation.

- Certain coordinate system transformations are applied:
  - The y and z axes are flipped (line 7).
  - The rows of the x and y axes are swapped (line 8).
  - The entire orientation of the world is reversed (line 9).

- After these adjustments, the final camera-to-world matrix $C2W$ is represented as:

$$C2W_{transformed} = \begin{bmatrix} +X_0 & +Y_0 & +Z_0 & Z \\ +X_1 & +Y_1 & +Z_1 & X \\ +X_2 & +Y_2 & +Z_2 & Y \\ 0.0 & 0.0 & 0.0 & 1 \end{bmatrix}$$

The operations and their corresponding mathematical representations elucidate the transformation process of COLMAP's extrinsics into a matrix format, which defines the camera's position in $3D$ space.

In our in-depth analysis, we determined the coordinate system of the transformation matrix. Given its non-conventional nature, we found an issue discussed on the GitHub

page of instant-ngp concerning this matrix. Alex Evans, one of the paper's authors, clarified the situation:

> We apologize for the shift in the coordinate system. Internally, NGP employs an entirely 0-1 bounding box, as exhibited in the GUI, with cameras oriented towards the positive z-direction. This convention was an early choice. Nonetheless, we aimed for compatibility with the original NeRF datasets, which define the origin at 0, scale the cameras to a distance approximately three units from the origin, and follow a distinct convention for orientation. Therefore, the snippet you encountered is essentially a translation from the original NeRF paper conventions to NGP's conventions. Over time, the original NeRF format of transforms.json became the primary method for data integration into NGP, which might seem confusing due to the lack of explicit discussion on the mapping (Alex Evans).

In summary, the design remains as intended. If required, users can modify the scale and offset components of the transformation by including additional parameters such as 'offset', 'scale' and 'aabb' in the JSON. For now, however, the coordinate inversion is hard-coded and cannot be changed (Alex Evans).