Circuits and Systems
Mekelweg 4,
2628 CD Delft
The Netherlands
https://cas.tudelft.nl/

CAS-2022-5283914

M.Sc. Thesis

Reconstruction and Rendering of Buildings as Radiance Fields for View Synthesis

Enpu Chen B.Sc.

TU Delft

Faculty of Electrical Engineering, Mathematics and Computer Science

Delft University of Technology

Reconstruction and Rendering of Buildings as Radiance Fields for View Synthesis

Thesis

submitted in partial fulfillment of the
requirements for the degree of

Master of Science

in

Electrical Engineering

by

Enpu Chen B.Sc.
born in Fushun, China

This work was performed in:

Circuits and Systems Group
Department of Microelectronics
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Delft University of Technology
Department of
Microelectronics

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science for acceptance a thesis entitled **"Reconstruction and Rendering of Buildings as Radiance Fields for View Synthesis"** by **Enpu Chen B.Sc.** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: 27 September 2022

Chairman: ______________________
dr.ir. Justin Dauwels

Advisor: ______________________
dr. Letao Liu

Committee Members: ______________________
dr. Xucong Zhang

____

Abstract

In inspection and display scenarios, reconstructing and rendering the entire surface of a building is a critical step in presenting the overall condition of the building. In building reconstruction, most works are based on point clouds because of their enhanced availability. In recent years, neural radiance fields (NeRF) have become a common function for implementing novel view synthesis. Compared to other traditional 3D graphic methods, NeRF-based models have a solid ability to produce photorealistic images with rich details that point clouds based methods cannot offer. As a result, we decided to investigate the performance of this technique in architectural scenes and look for ways to improve it for more significant scenes.

This thesis explores the ability to reconstruct large-field scenes with NeRF-based models. NeRF introduced a fully-connected network to predict the volume density and view-dependent emitted radiance at the special location, which will be projected into an image through classic volume rendering techniques. Due to the limitation of near-field ambiguity and parameterization of unbounded scenes, the original NeRF does not perform well on 360° input view, especially when the inputs are sparse. An inverted sphere parameterization that facilitates free view synthesis is introduced to address this limitation so that the foreground and background views can be trained separately. Besides that, we also compare the performance of tracing different light geometries, ray and cone, respectively. Meanwhile, to generate the reconstructed scene precisely, raw RGB images should be pre-processed to estimate the corresponding camera parameters. Finally, customized camera paths should be prepared to generate the final rendered video.

According to our experiments, training foreground and background separately is a promising method to solve practical large-scale scene reconstruction problems. A complete wrap-around view of the target building can be obtained using adjusted camera path parameters. Furthermore, introducing conical frustum casting into the original model also provides an alternative method to implement reconstruction. We named this method mip-NeRF++, which can contribute to the final results to some extent.

Acknowledgments

Two years of Master's study at TU Delft have been an unforgettable experience. The lockdown during the pandemic made me depressed, but I still gained wonderful campus life and friendship after reopening. First off, I would like to thank TU Delft for giving me such a great platform to grow as a graduate student. I also want to thank everyone who worked so hard to restore normalcy after the epidemic.

The first person to whom I would like to express my sincerest gratitude is my daily supervisor Dr. Justin Dauwels. Thank you for offering me the opportunity to access the most advanced technology. Your extensive experience plays a vital role in our work. Your detailed instructions and patient encouragement led us to finish the complicated subject successfully.

I would also like to express my gratitude to my advisor Dr. Letao Liu. Thank you for collecting images for us and giving us prompt and inspiring advice when we face difficulties. Meanwhile, my teammate Chuhan Wang also helped and inspired me a lot for my project. I would like to thank him for working together with me.

Afterward, I would like to say thank you to my family. Your support gives me the courage to overcome difficulties.

Finally, I also would like to thank all my friends who made my life in a foreign country not lonely.

Enpu Chen B.Sc.
Delft, The Netherlands
27 September 2022

Contents

List of Figures

List of Tables

Introduction 1

1.1 Background and motivation

Large-scale buildings play a crucial role in our modern world. They provide a variety of vital tasks in our everyday lives and the city's operations. In addition to all the practical usages, the artistic property of buildings also attracts people from all over the world to visit and study. People wish to look at some typical structures and investigate the specifics for a variety of reasons. Building safety inspectors, for example, require a vast amount of details to evaluate the building's state; rebuilding old structures may also provide historians with a thorough picture.

Many models have been constructed using modern graphics techniques to create photo-realistic visuals from hand-crafted scene representations. However, viewing a building remotely on another side of the planet and producing shapes, materials, lighting, and other characteristics of scenes remains a challenging task for people. In such a task, the first difficulty to be solved is the limitation of reconstructing a large-scale scene. Compared to small-scale processing scenes, many details of large-scale scenes may usually be lost because of the limitations of camera resolution and the interference between camera views. Besides that, reconstructing a building in an outdoor environment also increases the difficulty. This is most noticeable in the impact of changing lighting and a more intricate background. Furthermore, in order to broaden the scope of our project, we decided to explore a method that can reconstruct the object with purely using RGB images. As a result, some essential parameters must be estimated in advance to support the reconstruction.

As a result, this thesis will mainly focus on solving the aforementioned problems and developing a method to implement 3D reconstruction and rendering using neural radiance fields (NeRF) based techniques.

1.2 Problem statement

In this project, a batch of multi-view RGB images, which were captured by a drone, was given, and all of these images were shot at the target building, a modern office building in Singapore. The task is first to reconstruct a digital model containing surface information about the building. Then using the representation to render some continuous images to generate videos, which show the observation from a 360-degree angle of view towards the building.

To achieve this function, we aim to use the current state-of-the-art model, neural radiance fields (NeRF), as the core algorithm. NeRF provides the entire pipeline from generating graphical representation to rendering video results. The visualized presentation of this process is shown in figure 1.1. However, there are two critical restrictions:

large-scale scenes and lack of camera parameters. Researchers usually use standard and small-scale objects as experimental targets for academic purposes. These datasets usually contain only the object that needs to be rebuilt with high resolution to maintain detailed textures and clean backgrounds. However, in practical projects, the object we need to reconstruct is hundreds of times larger than the experimental subject. For this reason, more crucial details may be lost, and more inference noise will be included, making recreating the whole and clear object surface much more challenging. Meanwhile, because of the lack of camera parameters necessary for building's neural radiance field, pre-processing is also inevitable to estimate accurate camera information. This thesis will provide further information on how to solve these two challenges as well as a comparison of alternative solutions.

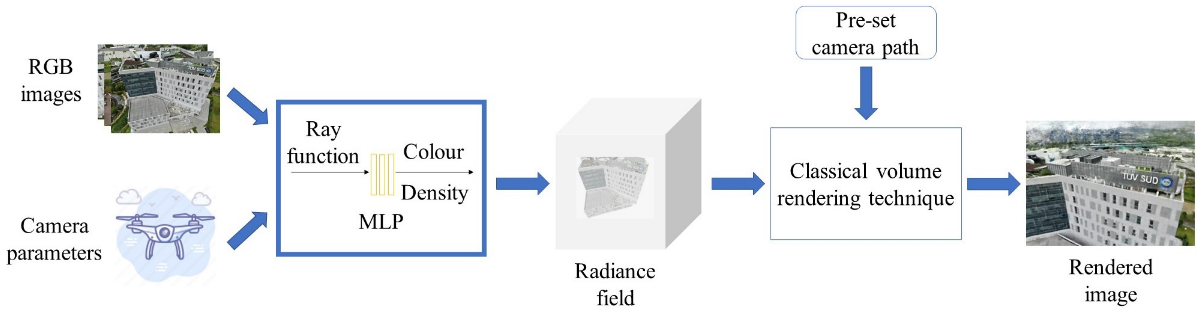

Figure 1.1: The whole pipeline.

1.3 Commonly-used neural rendering approaches

In this section, several of the most popular methods of neural rendering will be presented and compared. Finally, we will explain why we choose the NeRF-based method to complete this project and its application prospects.

By using traditional computer graphics, we can generate high-quality controllable imagery of a scene with all physical parameters provided. In recent years, with the rapid development of deep learning, using neural rendering methods to generate photo-realistic results in a controllable way has generally become the preferred method. The powerful learning ability of neural networks can overcome the difficulty of collecting physical parameters by estimating them. In general, the target of neural rendering is finding the mapping $\mathbf{I} = \mathcal{M}(\mathbf{c})$ between observations of real-world scenes and the corresponding output image $\mathbf{I} \in \mathbb{R}^{H \times W \times 3}$, where H and W are the height and width of images. However, the network cannot produce RGB results directly; several physics-based scene synthesis approaches disentangle the projection and other physical processes from the 3D scene representation. This procedure aims to utilize the interaction of the light and texture of the object as well as the camera to estimate the exterior of the target under observation from different views. Specifically, there are two main steps.

The first stage is scene representation which transfers visual sensory data into concise descriptions, including point clouds, implicit and parametric surfaces, meshes, and volumes methods. In general, all these approaches can be classified into two groups.

The first group only focuses on surface representation by calculating the properties of the surface of the object implicitly or explicitly. Another group does not estimate the appearance directly but stores volumetric properties such as densities, opacities, or occupancies and multidimensional features such as colours or radiance. These techniques will be elaborated in Section 1.2.1 and 1.2.2.

Afterwards, as the 3D geometry and the scene's appearance have been generated, these models need to be converted into images as the final result through rendering. In general, there are two main methods: ray casting and rasterization. More explanations of these two terms will be detailed in Section 1.2.3.

1.3.1 Surface representation

1.3.1.1 Point cloud

A point cloud consists of a collection of data points in Euclidean space. By arranging the specific position of a significant amount points in the XYZ coordinates, the object's surface can be viewed from some distance away. Meanwhile, more details such as RGB values or surface normals are also generated as additional attributes to obtain more photorealistic results. An example from the famous "Building Rome in a Day" [1] is shown in figure 1.2.

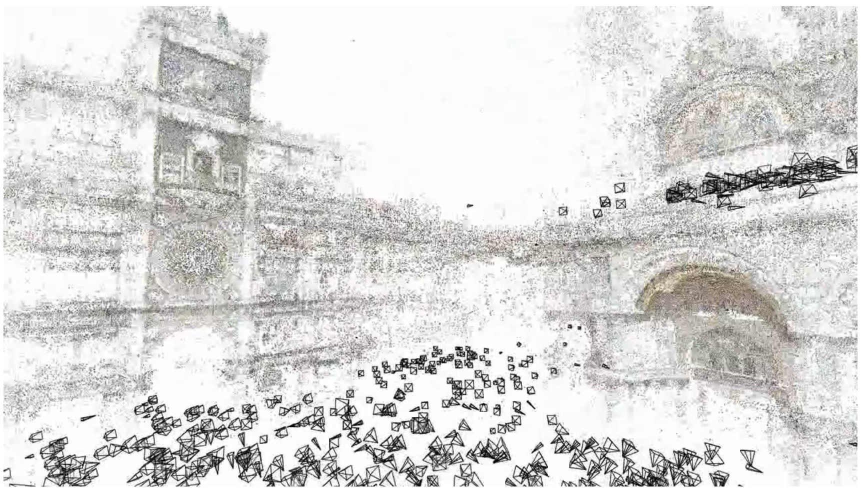

Figure 1.2: Effect of representing the target surface with point clouds [1].

Over the past decades, point-based rendering has been one of the most frequently used methods in computer graphics. In 1985 [36], M. Levoy and T. Whitted proposed a points-based display primitive model that first converts geometry into points and renders those points. After that, point-based representations introduced elliptical surface splats [82] to provide better approximation results. Since point-based models are usually acquired by range scanning or image-based reconstruction methods, which may cause problems of noise and artefacts, some techniques [52] [72] [10] are proposed from point-sampled geometries. Furthermore, more rendering strategies [21] [4] were also proposed to generate continues images and improve rendering efficiency.

In recent years, the development of deep learning has boosted the progress of point cloud-based rendering dramatically. For instance, [7] can generate high-quality point rendering images very efficiently, and [53] developed approaches that replace traditional hole-filling techniques to reduce blurriness. In ADOP [57], a stochastic point discarding technique was designed for efficient multi-layer rendering of large point clouds.

1.3.1.2 Polygonal meshes

Mesh-based representation approaches aim to collect vertices, edges, and faces that make up a 3D object. Triangles, quadrilaterals, or other simple convex polygons are combined to represent a piece-wise linear approximation of a surface. Figure 1.3 shows an example of using polygonal meshes to represent the surface of a dolphin. Because representing a surface with polygonal meshes needs to calculate linear equations, which is simple and fast, it is the most commonly-used boundary representation for a 3D graphics object.

In practice, the vertex's location needs to be predicted using trained neural networks. Since renderers are usually differentiable, the vertex positions and the vertex attributes can be optimized to reproduce an image. Besides using vertex attributes, texture map is another commonly-used strategy to store surface attributes with a neural network, e.g., to predict dynamically changing surfaces [8]. Then the next step is to determine the specific coordinate of each 2D texture attached to the mesh's vertices. Using barycentric interpolation, discrete texture coordinates can be computed for any point in a triangle, and the attribute can be retrieved from the texture using bilinear interpolation. Meanwhile, continuous textures can be generated by parameterizing the function through an MLP to predict colour values for each surface point [50]. Compared to discrete representation, continuous methods overcome the limitation of low resolution and the heavy dependence on task-specific shape parameterization.

1.3.1.3 Implicit surfaces

A continuous zero-level-set equation defines implicit surfaces in Euclidean space, which allows intuitive handling of complex topology. For instance, a unit sphere can be expressed using the implicit function $f(\mathbf{x}) = 1 - |\mathbf{x}|$ for points $\mathbf{x} \in \mathbf{R}^3$, points on the sphere are coordinates when $f(\mathbf{x}) = 0$. If the function equals a positive value, the point is inside the surface and vice versa. Figure 1.4 illustrates an example of this technique.

For classical surface representation methods, there are two main branches: blobby surface and algebraic surface [67]. The implicit functions of a blobby surface are the sum of radially symmetric functions with a Gaussian with a particular centre and standard deviation. The two blobby spheres will melt together when they are close enough to each other. The blobby function is usually expressed as an exponential function. Sometimes, authors [49] may also use a piecewise polynomial function to simplify the arithmetic process. Meanwhile, various shapes can be created with the blobby approach by using ellipsoidal rather than spherical functions. On the other hand, algebraic surfaces are described by one or more polynomial expressions in x, y and z depending on their complexity. Much of the work on this method has been devoted to fitting an algebraic surface to a given collection of points. At the same time, an entire surface can also be generated easily by piecing together many separate algebraic surface patches. Nevertheless, the drawback of this approach requires much machinery to smooth joins across patch boundaries.

Most recent studies have focused on model implicit functions with multi-layer perceptron (MLP) because of its powerful expressive representation and low memory consumption. There are two main application scenarios. The first is implementing geometry and appearance scene representation [18] [50] and another one is rendering free-view scenes [63] [58].

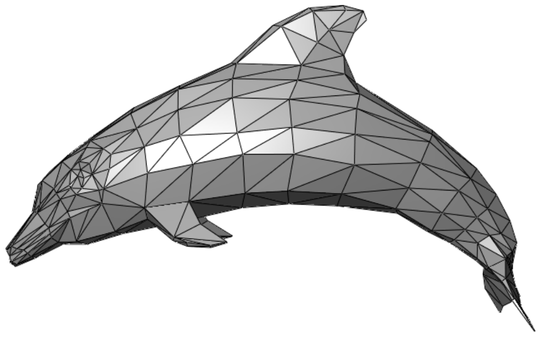
Figure 1.3: Polygonal meshes example.

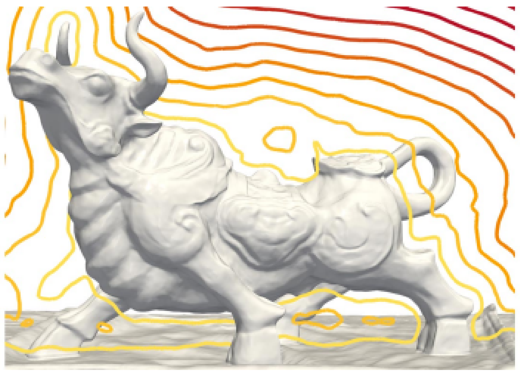
Figure 1.4: Implicit surfaces example [75].

1.3.2 Volume representation

1.3.2.1 Voxel grids

A voxel block grid is a globally sparse and locally dense data structure to represent 3D scenes by storing values on a regular grid in a 3D space. In traditional methods that represent a voxel-based structure, the entire 3D space must first be divided into block grids. Then blocks containing surfaces are organized in a hash map by 3D coordinates and are further divided into dense voxels that array indices can access. Additionally, sometimes people need to generate voxels from 3D discrete points to obtain a higher-level representation [74]. In this case, voxels may be compared to pixels in a picture. They are abstracted three-dimensional units with predefined volumes, locations, and properties that may be used to structurally represent discrete points in a topologically explicit and information-rich way.

In 2019, Sitzmann et al. proposed DeepVoxels [64], which is a learned representation that encodes the view-dependent surface of a 3D scene by storing features in the voxel grid. These DeepVoxels can be seen as volumetric neural textures, which can be directly optimized using backpropagation. Because of the feature of the learned soft visibility, higher-quality results and better generalization can be obtained by explicit occlusion reasoning. However, the memory usage efficiency of 3D CNN is relatively low due to the heavy computational complexity. To resolve this problem, octree data structure [34] is widely used to represent the volume sparsely. As a hierarchical data structure, octrees do not need to be explored to their full depth. This means that a system can extract a small subset of voxels as needed. In addition, octrees permit smoothing of the underlying data to help with anti-aliasing. Furthermore, several studies represent the voxel grid with a sparse approximation. For example, [29] used parameterized shape templates to represent a deformable model, [71] used an enhancement of multiplane image (MPI) to reproduce the view-dependent model, [5] introduced a collection of spherical images which represents panoramic light field content.

1.3.2.2 Neural volumetric representations

Besides using a voxel grid to store features of the object, these quantities can also be produced by a neural network similar to implicit surfaces. In this field, the most commonly-used framework used to parameterize volumes is a multilayer perceptron (MLP) network. And the majority of the approaches can be classified into using global [76] or local [19] networks. More specifically, local methods represent an implicit function as a mixture of local deep implicit functions. In contrast, the global methods use the global implicit function, which is decomposed into the sum of N local implicit functions. If we use grids and neural networks together as a hybrid representation, a trade-off can be made between computational and memory efficiency [26].

1.3.3 Differentiable image formation

Because the 3D scene model was created using prior methods, transforming the 3D model to a 2D picture is critical. There are two basic ways in graphics techniques: rasterization and ray casting. By differentiating the rendering, the gap between 2D and 3D processing methods is merged because the neural network is allowed to optimize 3D entities while operating on 2D projections. Therefore, 3D scene representations can be obtained by backpropagating the gradients concerning the rendering features. In practice, the generated image can be seen as a photo taken from a specific angle of view toward the object. Most systems employ the pinhole camera paradigm, with all camera rays traveling through a single point in space. By casting each camera ray, the rendered image can be achieved.

1.3.3.1 Rasterization

In computer graphics, rasterization has long been considered the dominant technique that displays a 3D object on a 2D plane because of its fast speed and outstanding performance. Briefly, rasterization reproduces the scene from a mesh of virtual triangles or polygons. The corner of each different triangle, so-called vertices, are intersected together and stores information, including its position in space, colour, and texture. For the next step, the information at the corner will be projected to corresponding pixels on the 2D screen with perspective projection. Then loop over all pixels in the image to check if all of them lie within the object region. Finally, if they are correct, the object's colour will be filled in.

Even though scaling with the number of visible vertices in a scene makes rasterization processing fast, it also has limitation in capturing specific effects, such as lighting, shadows, and reflections. In recent years, soft rasterization [39] was proposed to solve this problem using differentiable rendering. Specifically, it learns to render mesh silhouettes with an accurate soft approximation of the rasterized triangles. Then a differentiable pipeline is established so that the connection between the derivatives of pixel intensities and the attributes of the anticipated object will be established, and a gradient-based optimization will be available to formulate the 3D inference problem without supervised learning.

1.3.3.2 Ray casting

Although the ray casting technique was proposed roughly the same as the rasterization technique, ray casting used to be a relatively uncommon choice due to its high computational requirements. Compared to rasterization, which projects 3D models to 2D planes, ray casting works directly in 3D world space requiring floating point operations throughout rendering. Therefore, it is difficult to accomplish ray tracing just through software, such that the application of ray tracing was assumed to be non-competitive during the 1990s [11]. However, the development of hardware-enhanced the performance of ray casting dramatically. Furthermore, the benefits of ray tracing and realistic detail display draw people's attention.

Ray casting is mainly based on the pinhole model. The pinhole model uses an intrinsic matrix for the projection from a 3D object to a 2D image plane. To generalize the projection for arbitrary camera positions, an extrinsic matrix which consists of a rotation matrix and translation vector, can be used. Because of the depth ambiguity, the equations are non-injective. As a result, the models can do automatic differentiation and end-to-end optimization easily. To obtain camera matrices correctly, camera calibration algorithms are necessary to estimate camera parameters. In 2020, B. Mildenhall et al. introduced an advanced 3D representation method, named neural radiance field (NeRF), using ray casting to render an image. The usage of ray casting made it have an outstanding performance in terms of rendering photorealistic views.

1.4 Structure of this thesis

The rest of the report is organized as below

- Chapter 2 introduces some popular neural rendering methods and their features, and we also discuss the problem that we faced during this project and which approaches are potential solutions.
- Chapter 3 contains theories of the methods that we focus on in this project, including NeRF, mip-NeRF and NeRF++.
- Chapter 4 illustrates implementation details during our experiments. It includes data preparation, network adjustment and development of the new model.
- Chapter 5 describes three metrics firstly, then evaluate our experiment in visualized and numerical aspects respectively.
- Chapter 6 concludes the whole project and gives our outlook on the future of work.

2 Related work

In this section, we will first discuss the general approaches to rendering the target scene when the camera observes from an unobserved viewpoint. Then we will concentrate on the most often utilized solutions to our project's two major challenges: large-scale scenes and lack of camera parameters.

2.1 View synthesis and rendering

In practice, the quality of collected raw images is not always at a high standard, some of them are captured from terrible angles, and some even contain only a part of the target. When we try to use these images to generate a complete representation, this process can be seen as a synthesis operation by combining images from different views.

Light field sample interpolation techniques [35] can be easily used to reconstruct photorealistic inward and outward views with sampled representation for light fields. In addition, point cloud and triangle meshes are the two most frequently used representation methods in traditional 3D reconstruction. Then the geometric representation can be projected from input images into pixels on the photo, which is shot from the camera path via heuristic [6] or learning-based [23] methods. These algorithms have been refined and applied in a variety of actual settings including lengthy trajectories and large-scale sceneries. In 2010, Kopf et al. [32] proposed the Street Slide system. It combines the immersive 360° panoramas with the general multi-perspective strip panoramas so that it virtually improves the effect of visiting roads in a city, such as Google Street View. In 2011, Agarwal et al. [1] introduced the awesome project, Building Rome in a Day. In this system, the authors can reconstruct 3D city space using hundred thousand images from the internet within less than a day. They employed entire picture similarity and query expansion to match features in the distribution, then applied the Structure from Motion (SfM) approach to construct point cloud representation. Their excellent work demonstrates that it is feasible to create city-scale sceneries.

In recent years, introducing deep learning to traditional methods became a trend, such as Deepstereo [13] Deepview [14] and Stereo magnification [80]. End-to-end training can avoid the potential for failure in each complex stage of processing which often happens in traditional methods. This enhancement increased model adaptability and enabled high-quality outcomes on typically tough scenarios.

Another method is the volumetric-based representation method which attracted more attention in recent years. In these methods, feed-forward networks are usually used to learn a mapping from input images to an output volume, and global optimization is used to improve every generated scene. Among all these approaches, NeRF [46] is the most popular view synthesis scheme. NeRF pioneered the introduction of multilayer perceptrons (MLP) to differentiable volumetric rendering to construct a neural representation of the target and output more photorealistic results. Based on the original NeRF, more works are proposed to extend the performance of NeRF in several aspects. [41] applies NeRF on real-world unstructured images, [37] performed both view and time synthesis of dynamic scenes, and [47] introduces compact neural representations to implement real-time applications. Besides NeRF, Neural Sparse Voxel Fields (NSVF) [38] is another approach that defines a set of voxel-bounded implicit fields in a sparse voxel octree for free-viewpoint rendering.

This thesis will use NeRF as the benchmark method to achieve the building reconstruction effect. More information will be provided in Chapter 3. Although NeRF-based methods produced outstanding results on standard datasets, implementing NeRF in real-world situations remains difficult due to the high requirement for dense scene sampling, especially in large-scale scenes. Meanwhile, the lack of camera specifications is another disadvantage of using NeRF and producing reliable view synthesis results. The following two sections will briefly introduce current solutions for these two issues.

2.2 Neural radiance with unknown camera parameters

In most practical activities, we cannot get exact camera settings throughout the data gathering procedure. However, these required camera matrices can help us clarify the relationship between the world coordinate and the camera coordinate. For this reason, finding an appropriate strategy to overcome this limitation is the first problem we have to study. In general, there are three main approaches: (1) jointly optimize camera parameters during the training process [70], (2) using randomly initialized poses with the combination of Generative Adversarial Networks (GAN) [20] with NeRF [60], (3) using camera calibration software like COLMAP [59].

2.2.1 Trainable camera parameters

In 2021, the introduction of NeRF– [70] inspired us to create the camera parameters learnable to train networks with RGB images only. Assuming all photos are taken in a forward-facing setup with limited rotation and translation, all images should have the same intrinsic parameters. Therefore, the framework of NeRF– can be expressed as:

$$\Theta^*, \Pi^* = \underset{\Theta,\Pi}{\arg\min} \quad \mathcal{L}(\hat{I}, \hat{\Pi}|I), \tag{2.1}$$

where Π represents camera parameters, t represents input RGB images and Θ is the NeRF model.

2.2.2 NeRF with GAN

Generative Adversarial Networks (GAN) [20] has been used broadly for several years and have been proved to be excellent for generating photorealistic image synthesis with relatively high resolution [30]. However, for most prior GAN-based synthesis approaches, the generation process is always uncontrolled and does not explicitly model scenes' compositional nature. If we can manage the model at the object level, we can utilize the GAN concept to produce whatever object representation we require.

Generative Radiance Fields (GRAF) were trained with an unlabeled image dataset by bringing a multi-scale patch-based discriminator. Based on GRAF, GIRAFFE [48] was proposed to make the synthesis results editable so that the shape and appearance of the objects can be disentangled. However, these methods can only be used to generate simple scenes with a large amount of data input. To generate photorealistic images as done by NeRF, GNeRF [44] was proposed. GNeRF uses randomly initialized poses for complex scenes. Then feeds these photos and data into a two-stage end-to-end network. The first phase is a GAN network that can jointly optimize camera parameters and radiance field. Another stage refines them with additional photometric loss.

Considering the purpose of this project, GAN-based methods are not preferred choices because it has not been evaluated on large-scale scenes like enormous construction. Meanwhile, GAN was always famous for its difficulties in training stably and parameter adjustment. To make our experiments work efficiently and make the model more generic, the other two approaches seem more reasonable.

2.2.3 Camera calibration

The camera is a common instrument in both everyday life and scientific data collecting. A camera's primary function is to convert the 3D environment into a 2D picture. Camera calibration is the process that estimates the parameters of a lens and image sensor of an image or video camera. Then the geometric characteristics of the image process will be identified. As a result, when we just have RGB images without giving any camera details, camera calibration is essential for us to understand the link between the world where the object is located and the captured photo.

COLMAP [59] is the most popular open-source tool in the domain of camera calibration. Additionally, NSVF [38] labelled masked images from Tanks and Temples dataset [31] using a commercial software named Altizure. Besides that, Jeong et al. proposed Self-Calibrating Neural Radiance Fields (SCNeRF) [25]. This technique jointly learns geometry and constructs a geometry consistency for self-calibration using a unified end-to-end differentiable framework.

In this project, like most NeRF-based approaches, we will use the open-source implementation of COLMAP to generate the required dataset. More details will be discussed in Chapter 4.

2.3 Large-scale scene representation

For a long time, many academics' primary goal has been to produce a large-scale reconstruction. In 2004, Früh et al. [15] used 2D laser scanners to collect city streets views and matched laser scans with an aerial photograph or a Digital Surface Model to estimate global pose. Finally, an autonomous technique is used to generate several large-scale ground-based 3D city models. Then in 2006, a method named photo tourism [65] was proposed to help people browse large unstructured photos of a scene in 3D view. In 2008, a real-time 3D reconstruction method was developed for urban scenes. This method can generate representation with video and geo-registered coordinates input compared to previous methods. Recently, COLMAP [59] is another stable and robust 3D reconstruction software based on the Structure from Motion (SfM) algorithm. Two years later, the work from Zhu et al. [81] enabled SfM to process millions of images by distributed motion averaging. As for all of their methods, they share a similar pipeline: extract features from input images with methods like SIFT [40], then match these features between different images, and finally, jointly optimize 3D points and camera poses. However, all of these approaches generate 3D point cloud results. To get a complete 3D scene representation, an extra process is necessary to produce a dense point cloud or triangle mesh through a dense multi-view stereo algorithm [16]. These methods share a standard limitation: there are often many artifacts or holes with insufficient image elements.

The development of NeRF provided a new way to explore large-scale scene reconstruction. NeRF in the wild [42] took image appearance variations into account and jointly estimated transient objects. It made generating real-world models with NeRF possible. Afterwards, BungeeNeRF [73] developed a progressive learning paradigm to render city-scale scenes with images from Google Earth Studio. Block-NeRF [66] trained several NeRFs with different architecture separately to render the San Francisco streetscape. Urban Radiance Fields [56] used lidar to address exposure variation and leveraged predicted image segmentation to supervise densities on rays pointing at the sky. Recently, Chen et al. proposed a novel approach named TensoRF [9] that factorizes the scene tensor into multiple compact low-rank tensor components instead of representing the field using MLP only.

2.4 Summarization of NeRF-based approaches

Due to the remarkable performance of Neural Radiance Fields, we ultimately decided to investigate the performance of NeRF in our target building. As we discussed, the two most problematic issues are lack of camera parameters and restriction of large-scale scene reconstruction.

For camera parameters, because NeRF renders objects using the ray tracing approach, defining the ray function of each ray is an essential step. In order to determine the direction and origin of each ray precisely, positioning every camera in the world coordinate system is inevitable. However, camera calibration is necessary because these positions need to be represented by camera intrinsic and extrinsic matrices. For trainable methods, including NeRF– [70], the camera calibrations are not accurate enough, so these methods still cannot compete in solving real-world problems. Similarly, also with GAN [20], it is difficult to obtain reliable camera parameters. As a result, we choose to pre-process our raw images with COLMAP [59] as is commonly done, since it is easy to implement and the obtained camera parameters are sufficiently reliable.

For large-scale scene representation, NeRF-based methods are still not mature enough. The reconstruction pipeline with NeRF is much more complicated than other traditional methods. However, recent developments, such as Block-NeRF [66], Urban Radiance Fields [56] and BungeeNeRF [73] gave us confidence to reconstruct our target building with NeRF-like methods. Meanwhile, NeRF can output photorealistic images, therefore, we were and still are convinced that using NeRF to generate realistic landscapes will be a trend in the next years. Taking all of these factors into consideration, we decided to embark on developing NeRF-based methods for building scene reconstruction.

Methodology 3

As we discussed in Chapter 2, we will concentrate on neural radiance fields in this chapter. In general, the original NeRF is an experimental endeavor with perfect data and an experimental context. In addition, only a few of them investigated the performance of large-scale scene reconstruction. As a result, immediately processing our raw RGB images and building a 3D model is difficult. We have to separate our pipeline into several parts, and we concentrated on finding solutions to problems that we faced in each stage. The following problems are required to be overcome during our project:

1) The drone we used to take pictures of the target building cannot generate camera parameters automatically. So finding a method to generate parameters needed for NeRF is the first task we need to consider.
2) The background of raw images is exceptionally complex; this can lead to much noise in the rendering results.
3) In less contrived scenarios, there are many artifacts which may influence the final image quality.
4) Unbounded 360-degree panoramas always need an arbitrarily wide region of Euclidean space, but NeRF tends to confine 3D scene coordinates to a restricted domain. At the same time, the distance between the lens and the building's surface is relatively far, but models usually allocate more capacity to near content, so the details of the building may miss.

The following methods may overcome some of those issues:

1) Basic camera parameters and camera calibration techniques are beneficial for understanding the relationship between 3D models and 2D images. They are also necessary for calculating the correct ray function.
2) Mip-NeRF [3] traces cones instead of a ray so that objectionable aliasing artifacts are reduced, and mip-NeRF can also enhance the ability to reconstruct details.
3) NeRF++ [77] has two networks to train foreground and background scenes separately, so background influence may be eased. Because the view is near and far positions are not trained together, the fourth restriction can also get relief at the same time.

3.1 Camera model

As the foundation of the 3D graphic technique, camera parameters are the first thing that needs to be illustrated. The camera model can be seen as the pinhole perspective camera [22]. Figure 3.1 shows a simple working pinhole camera model. In this system, a barrier with a small aperture is located in the middle, with a 3D object and a photographic sensor on both sides. Each point on the 3D object emits several rays of light outwards. Because of the small holes, only one or a few can pass through the aperture and hit the image plane. Then the most straightforward one-to-one mapping between a 3D object and a 2D plane is established.

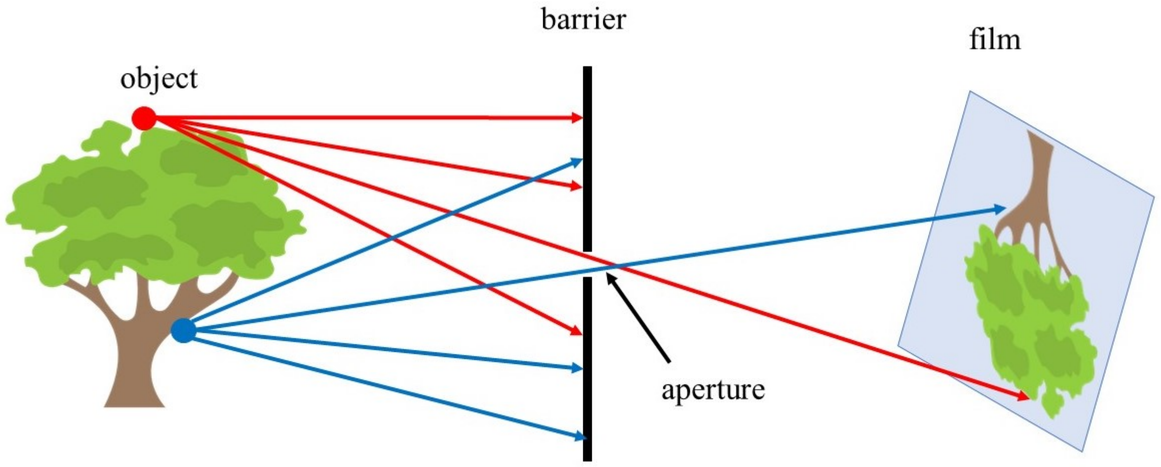

Figure 3.1: Pinhole camera model.

A more formal diagram of the pinhole camera is shown in figure 3.2. This schematic shows that the image plane is usually represented by Π, and O represents the pinhole. The distance between the image plane and the pinhole plane is the focal length f. To express the relationship between a point in the real world and a pixel on the image plane, we can first assume a point P on the object is $[x\ y\ z]^T$, and the corresponding point P' on the image plane Π' is $[x'\ y']^T$. Similarly, the pinhole can also be projected onto the image plane, giving a new point C'.

Then, we must define coordinate systems on different planes to represent the

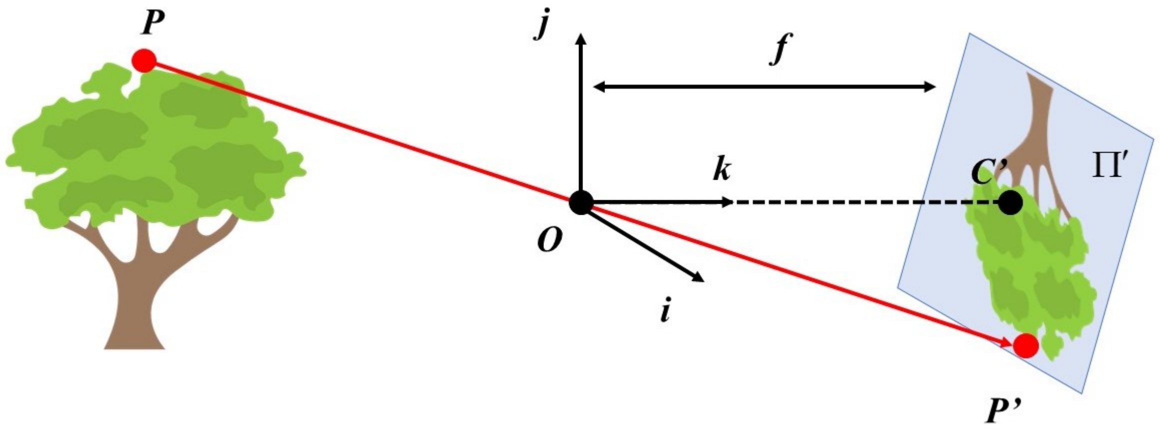

Figure 3.2: A formal construction of the pinhole camera model.

points. There are two 3D coordinate systems and two 2D coordinate systems in total.

The first is the world coordinate system, a 3D basic Cartesian coordinate system with arbitrary origins. It can represent the position of the object in the real world. Another 3D coordinate system is the camera coordinate system which is described with $[i\ j\ k]$ in figure 3.2 centred at the pinhole O so that the axis k is perpendicular to the image plane and points toward it. Understanding the theory of the camera coordinate system is crucial for figuring out mapping relations. The first 2D coordinate system is the image coordinate system that contains the projected points from the 3D camera coordinate system. Finally, the points in the image coordinate system will be discretized to integer values in the pixel coordinate.

Afterward, we need to figure out the complete transformation in this process. As shown in figure 3.3, this process can be divided into extrinsic and intrinsic components. The extrinsic parameters of a camera depend on its location and orientation without using internal parameters such as focal length, the field of view, etc. As for the intrinsic parameters, they reflect how the images are captured with focal length, aperture, field-of-view, resolution, etc.

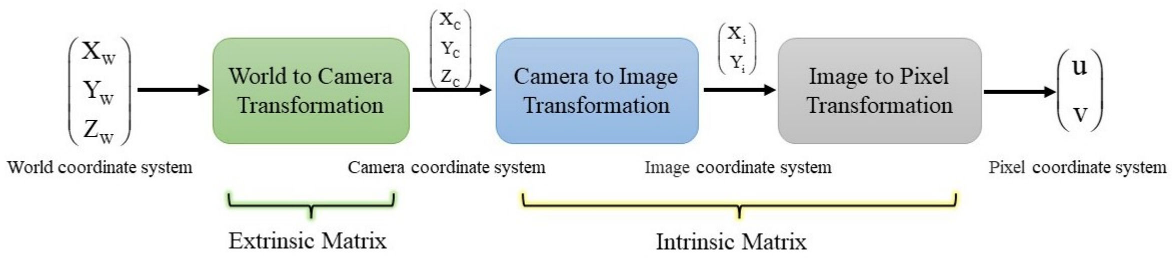

Figure 3.3: General transformation between each coordinate.

3.1.1 Extrinsic and intrinsic matrix decomposition

This section will detail why we decompose the process into two parts and how these matrices work together.

First of all, we assumed that the camera matrix is 3×4, which transforms 3D world coordinates into 2D image coordinates. We can denote the matrix as P, which can be expressed with the block form:

$$P = [M| - MC], \tag{3.1}$$

where M is an invertible 3×3 matrix, C is a column-vector that stores the camera's position in world coordinates. In practical projects, we tend to add an extra row to the bottom to preserve the z-coordinate so that the camera matrix will be a 4×4 matrix. If necessary, the third row can be dropped to get a 3×4 matrix. However, using only this camera matrix to transform has some vital drawbacks: it does not contain the camera's pose and internal geometry, and it also cannot help to get surface normals in camera coordinates to implement specular lighting.

As a result, decomposing the camera matrix into the product of two matrices became the most commonly-used solution. They are intrinsic matrix K and extrinsic matrix

$[R| - RC]$, respectively. So the camera matrix can be expressed as:

$$P = K[R| - RC], \tag{3.2}$$

the 3×3 intrinsic matrix K is an upper-triangular matrix that describes the camera's internal parameters like focal length. R is a 3×3 rotation matrix whose columns are the directions of the world axes in the camera's reference frame. And the vector C is the camera centre in the world coordinates, and the production of R and C gives the position of the world origin in camera coordinates. The following two sections will detail these matrices and explain the graphical significance.

3.1.2 The extrinsic matrix

The camera's location and heading direction in the world can be described with the extrinsic matrix. It consists of two components: a rotation matrix R, and a translation vector $\mathbf{t}$. By rotation and translation operations, we can convert from the world coordinate system to the camera coordinate system (and vice-versa). We can use a rigid transformation matrix: a 3×3 rotation matrix in the left block and a 3×1 translation column-vector in the right:

$$[R|\mathbf{t}] = \left[\begin{array}{ccc|c} r_{1,1} & r_{1,2} & r_{1,3} & t_1 \\ r_{2,1} & r_{2,2} & r_{2,3} & t_2 \\ r_{3,1} & r_{3,2} & r_{3,3} & t_3 \end{array}\right]. \tag{3.3}$$

As we mentioned before, adding an extra row to the bottom is a common operation. So that the matrix will be square and we can decompose the matrix into a rotation followed by a translation:

$$\begin{aligned} \left[\begin{array}{c|c} R & \mathbf{t} \\ \hline 0 & 1 \end{array}\right] &= \left[\begin{array}{c|c} I & \mathbf{t} \\ \hline 0 & 1 \end{array}\right] \times \left[\begin{array}{c|c} R & 0 \\ \hline 0 & 1 \end{array}\right] \\ &= \left[\begin{array}{ccc|c} 1 & 0 & 0 & t_1 \\ 0 & 1 & 0 & t_2 \\ 0 & 0 & 1 & t_3 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \times \left[\begin{array}{ccc|c} r_{1,1} & r_{1,2} & r_{1,3} & 0 \\ r_{2,1} & r_{2,2} & r_{2,3} & 0 \\ r_{3,1} & r_{3,2} & r_{3,3} & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right], \end{aligned} \tag{3.4}$$

the vector $\mathbf{t}$ can be interpreted as the position of the world origin in camera coordinates, and the columns of R represent the directions of the world axes in camera coordinates. The extrinsic matrix records the translation process from world to camera (w2c). However, as for most datasets that we explored, they commonly require a pose matrix to generate ray information. So we need to find out how to build the camera pose matrix, or camera to world matrix (c2w), from the extrinsic matrix.

For the pose matrix, we use a column vector C to represent the location of the camera centre in world coordinates and let R_C be the rotation matrix describing the camera's orientation for the world coordinate system. The pose matrix can be expressed with $[R_C|C]$. Similar to the extrinsic matrix, another row of (0, 0, 0, 1) is also added to the bottom. And the relationship between extrinsic matrix and pose matrix can be represented as:

$$
\begin{aligned}
\left[\begin{array}{c|c} R & \mathbf{t} \\ \hline \mathbf{0} & 1 \end{array}\right] &= \left[\begin{array}{c|c} R_C & C \\ \hline \mathbf{0} & 1 \end{array}\right]^{-1} \\
&= \left[\left[\begin{array}{c|c} I & C \\ \hline \mathbf{0} & 1 \end{array}\right]\left[\begin{array}{c|c} R_C & 0 \\ \hline \mathbf{0} & 1 \end{array}\right]\right]^{-1} && \text{composing rigid transform} \\
&= \left[\begin{array}{c|c} R_c & 0 \\ \hline \mathbf{0} & 1 \end{array}\right]^{-1}\left[\begin{array}{c|c} I & C \\ \hline \mathbf{0} & 1 \end{array}\right]^{-1} && \text{distributing the inverse} \\
&= \left[\begin{array}{c|c} R_C^T & 0 \\ \hline \mathbf{0} & 1 \end{array}\right]\left[\begin{array}{c|c} I & -C \\ \hline \mathbf{0} & 1 \end{array}\right] && \text{applying the inverse} \\
&= \left[\begin{array}{c|c} R_C^T & -R_C^T C \\ \hline \mathbf{0} & 1 \end{array}\right]. && \text{matrix multiplication}
\end{aligned}
\tag{3.5}
$$

We use the inverse because the inverse of a rotation matrix is its transpose, but inverting a translation matrix eliminates the translation vector. As a result, we can see that the link between the extrinsic matrix parameters and the camera's posture is simple:

$$
\begin{aligned}
R &= R_C^T, \\
t &= -RC.
\end{aligned}
\tag{3.6}
$$

3.1.3 The intrinsic matrix

As the camera coordinate has been determined, the transformation matrix that establishes the connection between 3D camera coordinates and 2D homogeneous image coordinates needs to be discussed. This matrix is the intrinsic matrix that can be parameterized as:

$$
K = \begin{pmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{3.7}
$$

where f_x and f_y are focal lengths, x_0 and y_0 are principal point offsets, and s is the axis skew.

The focal length is the distance between the pinhole and the image plane. In a real pinhole camera, both f_x and f_y have the same value.

For the principal point offset, it is the location of the principal point relative to the film's origin. The principal point is the intersection of the camera's principal axis with the film plane, as shown in figure 3.4.

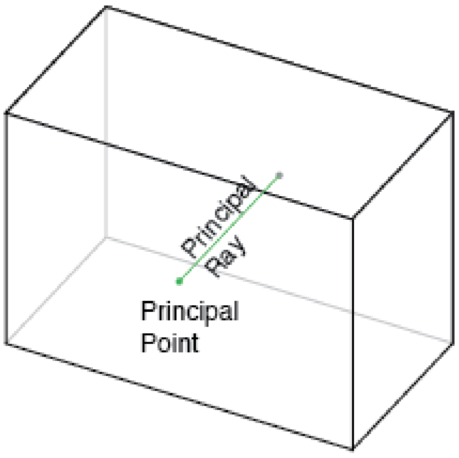

Figure 3.4: Principal axis [61].

The exact definition of principal point offset depends on which convention is used for the location of the origin; the illustration below assumes it is at the bottom-left of the film. If we increase x_0, it will reflect a rightward pinhole movement, which is equivalent to the leftward movement of the film. Figure 3.5 shows the coordinate of this system. As for the axis skew s, it leads to shear distortion in the projected image.

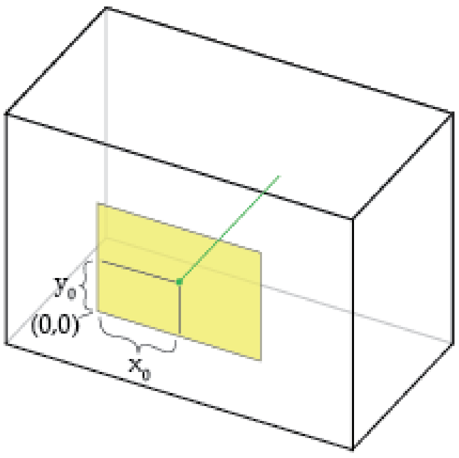

Figure 3.5: Principal point offset [61].

Occasionally, the 2D image plane is skewed instead of a rectangle. In this case, another transformation needs to be carried out to go from the rectangular plane to the skewed plane.

3.2 Neural radiance fields

In this section, we would like to introduce the neural radiance fields algorithm. In general, we will illustrate the NeRF in the following aspects: (1) how to predict volume density and radiance field with an MLP, (2) how to render photorealistic results using the traditional method, (3) the optimization of NeRF with positional encoding and hierarchical volume sampling. The general pipeline of neural radiance fields scene representation and rendering procedure can be found in figure 3.6.

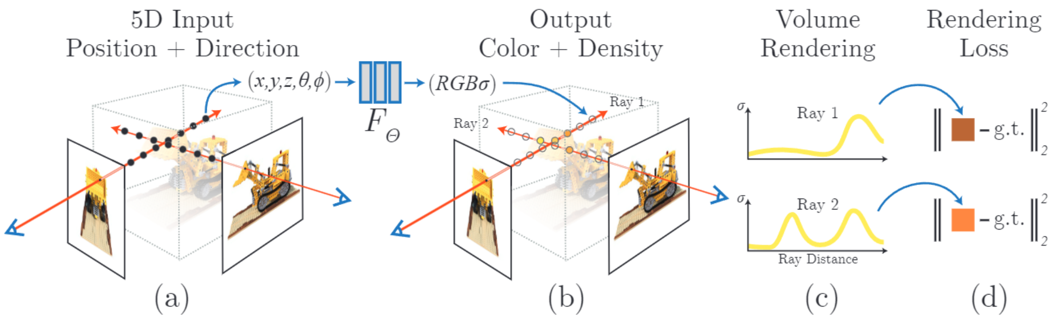

Figure 3.6: An overview of the NeRF's pipeline. The 5D input coordinates along camera rays, including position and view direction, are shown in (a). This information will be fed into a fully-connected deep network F_θ to generate the colour and volume density (b). Then volume rendering techniques will be used to composite these values into an image (c). Finally, because the rendering function is differentiable, the mean square error between the output picture and the ground truth is calculated as the loss value to improve the scene representation (d) [46]. The background of raw images is exceptionally complex; this can lead to much noise in the rendering results.

3.2.1 Neural radiance fields construction

As for the neural representation part, NeRF uses a fully-connected (non-convolutional) deep network, which requires a single continuous 5D coordinate including spatial location $\mathbf{x}$ = (x, y, z) and 2D viewing direction (θ, ϕ)), and output the volume density σ and view-dependent emitted radiance color $\mathbf{c} = (r, g, b)$ at that spatial location. The continuous 5D scene representation will be approximated by an MLP network $F_\theta : (\mathbf{x}, \mathbf{d}) \rightarrow (\mathbf{c}, \sigma)$ and the mapping from input values to their related density and colour output is set as weights θ which will be optimized.

As for the MLP, the whole framework of the network used for neural representation is based on the architecture from DeepSDF [51], which is shown in figure 3.7. This diagram shows that the MLP receives the positional encoded 3D coordinate $r(x)$ as the input value. The core component of the MLP is constructed with eight fully-connected layers, the ReLU function is used as an activation function and there are 256 channels on each layer. Similar to the network of DeepSDF, NeRF's MLP also introduces a skip connection that concatenates the input to the fifth layer's activation. Afterwards, the MLP will output the density σ and a 256-dimensional feature vector. Finally, the feature vector is concatenated with the camera ray's viewing direction and passed to one more fully-connected layer with the Sigmoid activation and 128 channels as the RGB layer. The output of the RGB layer is the view-dependent RGB colour.

Because free space and occluded regions that cannot reflect on the final results are also sampled repeatedly, and this causes much waste. To improve rendering efficiency,

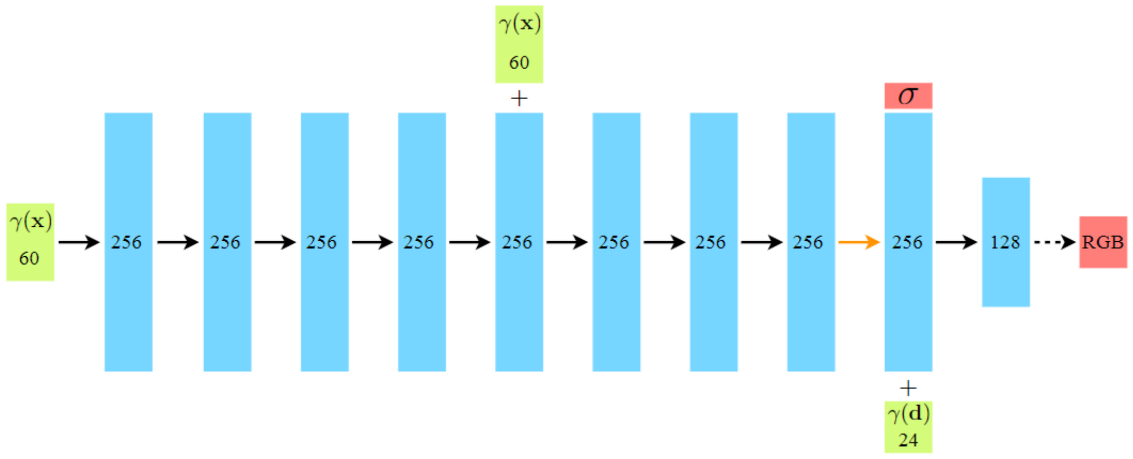

Figure 3.7: The fully-connected network architecture of NeRF. Green rectangles show input vectors, blue rectangles show intermediate hidden layers, and red rectangles show output vectors. All layers are fully-connected, black arrows represent ReLU activations, orange arrows indicate layers without activation, and dashed black arrows represent using the Sigmoid activation [46].

NeRF uses a hierarchical representation strategy. Compared to using a single network only to build the representation mode, NeRF optimizes two networks simultaneously: a "coarse" network and a "fine" network. For the training process, the "coarse" network is trained firstly with stratified sampled input. Then the output of the "coarse" network will be considered to create a more informed sampling of points along each ray. By introducing this process, more visible content will be trained.

Then calculates the total squared error between the RGB output and the ground truth images with:

$$L = \sum_{\mathbf{r}\in R}[\| \hat{C}_C(\mathbf{r}) - C(\mathbf{r}) \|_2^2 + \| \hat{C}_f(\mathbf{r}) - C(\mathbf{r}) \|_2^2], \tag{3.8}$$

where R is the ray in each batch, $C(\mathbf{r})$ is the ground truth image, and $\hat{C}(\mathbf{r})$ and $\hat{C_f}(\mathbf{r})$ are the output RGB values from coarse and fine network respectively.

In order to improve the photorealistic of the output images, NeRF makes good use of the input viewing direction to represent non-Lambertian effects. Lambertian reflectance defines an ideal diffusely reflecting surface. A Lambertian surface does not absorb incident light and has the same brightness in all directions of view when illuminated uniformly. However, in most situations in the real world, the surfaces of the objects are non-Lambertian. For this reason, representing non-Lambertian effects is critical for generating high-quality results. NeRF implements non-Lambertian effects by inputting view direction. As shown in figure 3.8, two visualized results are viewed from two different spatial locations. Furthermore, two viewing frames also captured details of the ship and water in both images. We can see the difference between the two views. And (c) demonstrates that NeRF is capable of predicting the changing specular appearance of these two 3D spots as well as generalization throughout the whole hemisphere of viewing directions.

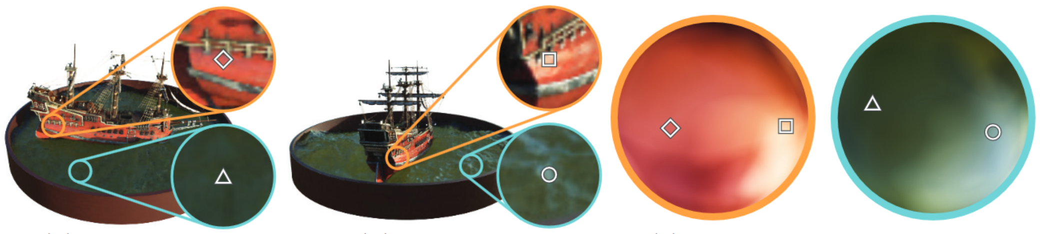
(a) View 1 (b) View 2 (c) Radiance Distributions

Figure 3.8: A visualization of view-dependent emitted radiance [46].

3.2.2 Volume rendering

When the neural radiance fields representation has been built successfully, we must find a solution to convert the volume density and directional emitted radiance to visualized images. As for NeRF-based methods, the classical ray tracing-based volume rendering method [27] [43] is used.

To get the rendering equation, we need to consider the ray absorption model first. We can assume that the participating medium has perfectly black particles that absorb all the light they intercept and do not scatter or emit any. In NeRF coordinate, the camera is located at the origin. The equation of this model is:

$$\frac{dI}{dt} = \rho(t)AI(t) = \tau(t)I(t), \tag{3.9}$$

where t indicates the location of the particle, $I(t)$ is the light intensity at t, ρ is the number of particles per unit volume, $A = \pi r^2$ is the projected area whose radius is r and $\tau(t)$ is called the extinction coefficient and represents the rate that light is occluded. Divide both sides of the equal sign by $I(t)$, we can get:

$$\frac{1}{I(t)}\frac{dI}{dt} = \tau(t), \tag{3.10}$$

then calculate integration on both side:

$$\int_0^s \frac{1}{I(t)}dI = \int_0^s \tau(t)dt, \tag{3.11}$$

where s is the distance variant along the direction of a ray. Then we can easily calculate:

$$ln\frac{I(s)}{I(0)} = \int_o^s \tau(t)dt \tag{3.12}$$

$$I(s) = I(0)e^{\int_0^s \tau(t)dt}. \tag{3.13}$$

As for the result, we define the $e^{\int_0^s -\tau(t)dt}$ as the accumulated transmittance of the medium within the distance s, which indicates the attenuation ratio of light intensity.

We use $T'(o,s)$ to represent this value.

After figuring out the absorption model, we will turn to discuss the emission model. The medium can add extra light to the ray by emission or reflection of external illumination. In this part, we will assume that the particles in the absorption model are transparent and we will add them together in the next part. The differentiable equation of this model is:

$$\frac{dI}{dt} = -c(t)\rho(t)A = -c(t)\tau(t) = -g(t), \tag{3.14}$$

where the intensity C represents the diffuse reflection per projected region. This function is equivalent to:

$$dI = -c(t)\rho(t)Adt = -c(t)\tau(t)dt = -g(t)dt. \tag{3.15}$$

After the integration, we will get:

$$\int_o^s dI = \int_0^s -g(t)dt, \tag{3.16}$$

$$I(s) = I(0) - \int_0^s g(t)dt, \tag{3.17}$$

the term $g(t)$ is called the source term.

For the next step, we need to compute the summation of the previous two models:

$$\frac{dI}{dt} = \tau(t)I(t) - g(t), \tag{3.18}$$

$$\frac{dI}{dt} - \tau(t)I(t) = -g(t), \tag{3.19}$$

$$\left(\frac{dI}{dt} - \tau(t)I(t)\right)e^{\int_t^0 \tau(t)dt} = -g(t)e^{\int_t^0 \tau(t)dt}, \tag{3.20}$$

$$\frac{d}{dt}\left(I(t)e^{\int_t^0 \tau(t)dt}\right) = -g(t)e^{\int_t^0 \tau(t)dt}, \tag{3.21}$$

$$d\left(I(t)e^{\int_t^0 \tau(t)dt}\right) = -g(t)e^{\int_t^0 \tau(t)dt}dt, \tag{3.22}$$

$$\int_\infty^0 d\left(I(t)e^{\int_t^0 \tau(t)dt}\right) = \int_\infty^0 -g(t)e^{\int_t^o \tau(t)dt}dt, \tag{3.23}$$

$$I(0)e^{\int_0^0 \tau(t)dt} - I(\infty)e^{\int_\infty^0 \tau(t)dt} = \int_\infty^0 -g(t)e^{\int_t^0 \tau(t)dt}dt, \tag{3.24}$$

$$I(0) = I(\infty)e^{\int_\infty^0 \tau(t)dt} + \int_0^\infty g(t)e^{\int_t^0 \tau(t)dt}dt, \tag{3.25}$$

$$I(0) = I(\infty)e^{\int_o^\infty -\tau(t)dt} + \int_0^\infty g(t)e^{\int_0^t -\tau(t)dt}dt, \tag{3.26}$$

$$I(0) = I(\infty)T'(0,\infty) + \int_0^\infty g(t)T'(0,t)dt. \tag{3.27}$$

On the right side of the equation, the first term represents the light intensity from the coordinate origin to infinity, which can be called background light which is considered 0 in NeRF. As a result, this function can be simplified as follows:

$$I(0) = \int_0^\infty g(t)T'(0,t)dt = \int_0^\infty T'(0,t)\tau(t)c(t)dt. \tag{3.28}$$

In practice, not all positions between 0 and ∞ have media on them, the object always has a boundary, so we can use near bound t_n and far bound t_f to define the region. So we can get:

$$I(0) = \int_{t_n}^{t_f} T'(t_n,t)\tau(t)c(t)dt, \tag{3.29}$$

then we introduce another term $\sigma(t) = \tau(t)$ which we named volume density, and let $T(t) = e^{\int_{t_n}^{t} -\sigma(t)dt}$, then the previous function can be expressed as:

$$I(0) = \int_{t_n}^{t_f} T(t)\sigma(t)c(t)dt. \tag{3.30}$$

In NeRF algorithm, we usually use $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ to represent rays, then the $\sigma(\mathbf{r}(t))$ refers to the volume density at position t on this ray of $\mathbf{r}$, $c(\mathbf{r}(t), \mathbf{d})$ is the Light intensity of camera ray within the near and far bounds. Finally, because we mainly focus on the ray along each typical light, for a fixed view, we can ignore the position of the camera location and add parameter $\mathbf{r}$ to represent the ray variant. So the final expression of getting the expected colour C is:

$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), \mathbf{d})dt, \text{ where } T(t) = \exp(-\int_{t_n}^{t} \sigma(\mathbf{r}(s))ds). \tag{3.31}$$

The process of estimating the continuous integration is quadrature. However, the MLP can only be queried at a fixed discrete set of locations, and deterministic quadrature may influence the representation's resolution. As a result, a stratified sampling approach is used to partition $[t_n, t_f]$ into N evenly-spaced bins and then draw one sample uniformly at random from within each bin:

$$t_i \sim \mathcal{U}\left[t_n + \frac{i-1}{N}(t_f - t_n),\ t_n + \frac{i}{N}(t_f - t_n)\right]. \tag{3.32}$$

The expression of using these sampled results to estimate $C(\mathbf{r})$ with the quadrature rule is shown as:

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i(1 - \exp(-\sigma_i\delta_i))\mathbf{c}_i, \text{ where } T_i = \exp(-\sum_{j=1}^{i-1} \sigma_j\delta_j), \tag{3.33}$$

where $\delta_i = t_{i+1} - t_i$ is the distance between adjacent samples. The final function is differentiable and can use alpha values $\alpha_i = 1 - \exp(-\sigma_i\delta_i)$ to implement traditional alpha compositing, which is the procedure of combining an image with a background to generate a novel view.

Considering the usage of hierarchical volume sampling, the colour from the "coarse" network will change into a weighted sum of all sampled colours c_i along the ray:

$$\hat{C}_c(\mathbf{r}) = \sum_{i=1}^{N_c} w_i c_i, \text{ where } w_i = T_i(1 - \exp(-\sigma_i \delta_i)), \tag{3.34}$$

where N_c represents a set of sampled locations.

3.2.3 Positional encoding

To generate images with higher resolution, two required optimization methods need to be used. One is hierarchical volume sampling, and another is a positional encoding (PE) of the input coordinates, which helps the network to represent high-frequency functions.

Even though neural networks F_θ are universal function approximators, NeRF still cannot achieve good performance when the model needs to reconstruct high-frequency variation in colour and geometry. According to Rahaman et al. [54], deep neural networks are biased towards learning lower frequency functions. Their research also shows that mapping the inputs to a higher dimensional space with high-frequency functions can improve the ability to fit data with a high-frequency variation.

To solve this problem, NeRF reformulates the network as a composition of two functions $F_\theta = F'_\theta \circ \gamma$, where γ is a mapping from $\mathbb{R}$ into a higher dimensional space $\mathbb{R}^{2L}$. The encoding function is similar to the positional encoding in the popular Transformer [68], as shown as:

$$\gamma(p) = (\sin(2^0 \pi p),\ \cos(2^0 \pi p),\ \dots\ ,\ \sin(s^{L-1} \pi p),\ \cos(2^{L-1} \pi p)). \tag{3.35}$$

3.3 Mip-NeRF

In this section, we will give a brief introduction to a popular NeRF-based method that we have studied in this project, the name of the method is mip-NeRF [3]. Compared to the original NeRF, the most significant change of mip-NeRF is tracing conical frustums instead of rays. Figure 3.9 shows the difference between these two methods clearly. To achieve this goal, mip-NeRF developed new algorithms to implement cone tracing and integrated positional encoding (IPE) to handle the conical frustum situation. In the following sections, we will detail these improvements and explain why these operations can benefit the rendering results.

3.3.1 Cone tracing

This section will illustrate the mathematical explanation of casting cones and featuring conical frustums along the cone. Similar to NeRF, which renders one pixel each time, mip-NeRF also processes an individual pixel of interest at a time. The camera's centre of projection is the starting point $\mathbf{o}$ of the ray, which is also the cone's apex, and the cone's direction $\mathbf{d}$ passes through the centre and is vertical to the circular surface. Then we define the radius of the cross-section of the cone at $\mathbf{o} + t\mathbf{d}$ as $\dot{r}$. The value of $\dot{r}$ is

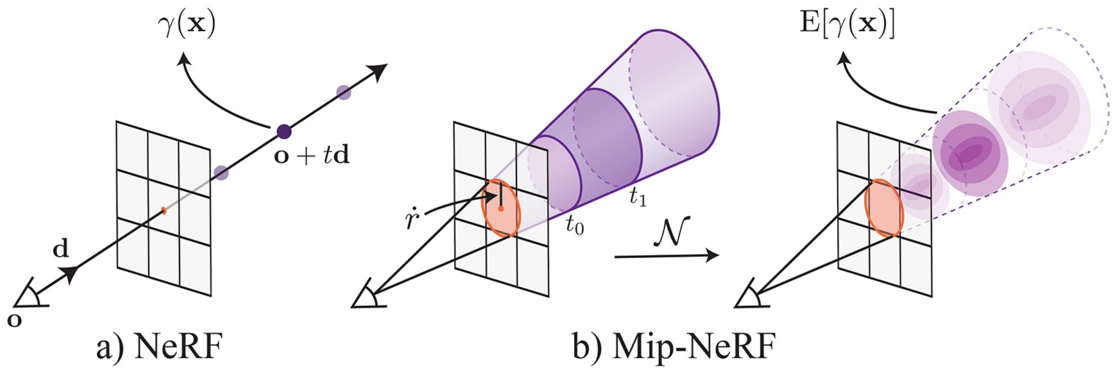

Figure 3.9: A comparison of NeRF and mip-NeRF. NeRF (a) traces points x along rays, and mip-NeRF (b) traces conical frustums defined by each camera pixel [3].

assigned as the width of the pixel in world coordinates scaled by $2/\sqrt{12}$. As in NeRF, we also need to calculate the position $\mathbf{x}$ within the region $[t_0, t_1]$:

$$\mathrm{F}(\mathbf{x}, \mathbf{o}, \mathbf{d}, \dot{r}, t_0, t_1) = \mathbb{1}\left\{\left(t_0 < \frac{\mathbf{d}^{\mathrm{T}}(\mathbf{x}-\mathbf{o})}{\|\mathbf{d}\|_2^2} < t_1\right) \wedge \left(\frac{\mathbf{d}^{\mathrm{T}}(\mathbf{x}-\mathbf{o})}{\|\mathbf{d}\|_2\|\mathbf{x}-\mathbf{o}\|_2} > \frac{1}{\sqrt{1+(\dot{r}/\|\mathbf{d}\|_2)^2}}\right)\right\}, \tag{3.36}$$

where $\mathbb{1}$ is an indicator function: $\mathrm{F}(\mathbf{x}, \cdot) = 1$ iff $\mathbf{x}$ is inside the conical frustum defined by $(\mathbf{o},\ \mathbf{d},\ \dot{r},\ t_0,\ t_1)$.

3.3.2 Integrated positional encoding

As we have discussed in the NeRF section, featured representation can enhance the training results by representing high-frequency coefficients better. Similarly, mip-NeRF also requires positional encoding for each conical frustum so that the size and form of the conical frustums will also be considered instead of just their centroids. Considering the complexity and efficiency of the implementation, mip-NeRF shows the expected positional encoding of all coordinates inside the conical frustum:

$$\gamma^*(\mathbf{o}, \mathbf{d}, \dot{r}, t_0, t_1) = \frac{\int \gamma(\mathbf{x})\mathrm{F}(\mathbf{x}, \mathbf{o}, \mathbf{d}, \dot{r}, t_0, t_1)d\mathbf{x}}{\int \mathrm{F}(\mathbf{x}, \mathbf{o}, \mathbf{d}, \dot{r}, t_0, t_1)d\mathbf{x}}. \tag{3.37}$$

Nonetheless, the numerator does not have a closed-form solution. In order to efficiently approximate the required feature, which we shall refer to as an ”integrated positional encoding”, mip-NeRF consequently approximates the conical frustum with a multivariate Gaussian (IPE).

The first step of characterizing a Gaussian model is to find the expression of the mean and covariance of $\mathrm{F}(\mathbf{x}, \cdot)$. Because the section of the cone is circular and conical frustums are symmetric around the axis of the cone, a Gaussian can be characterized by three parameters: the mean distance along the ray μ_t, the variance along the ray σ_t^2, and the variance perpendicular to the ray σ_r^2:

$$\mu_t = t_\mu + \frac{2t_\mu t_\delta^2}{3t_\mu^2 + t_\delta^2}, \tag{3.38}$$

$$\sigma_t^2 = \frac{t_\delta^2}{3} - \frac{4t_\delta^4(12t_\mu^2 - t_\delta^2)}{15(3t_\mu^2 + t_\delta^2)^2}, \tag{3.39}$$

$$\sigma_r^2 = \dot{r}^2 \left(\frac{t_\mu^2}{4} + \frac{5t_\delta^2}{12} - \frac{4t_\sigma^4}{15(3t_\mu^2 + t_\delta^2)} \right), \tag{3.40}$$

these quantities are parameterized with respect to a midpoint $t_\mu = (t_0 + t_1)/2$ and a half-width $t_\delta = (t_1 - t_0)/2$. Then convert the coordinate of the conical frustum into the world coordinate system:

$$\boldsymbol{\mu} = \mathbf{o} + \mu_t \mathbf{d}, \quad \boldsymbol{\Sigma} = \sigma_t^2(\mathbf{d}\mathbf{d}^{\mathrm{T}}) + \sigma_r^2 \left(\mathbf{I} - \frac{\mathbf{d}\mathbf{d}^{\mathrm{T}}}{\|\mathbf{d}\|_2^2} \right), \tag{3.41}$$

which is the final multivariate Gaussian.

The next step is to calculate the expectation of a positional encoded coordinate distributed according to the aforementioned Gaussian as the IPE. The positional encoding equation 3.35 can be rewritten as a Fourier feature [55]:

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 0 & 2 & 0 & 0 & & 2^{L-1} & 0 & 0 \\ 0 & 1 & 0 & 0 & 2 & 0 & \dots & 0 & 2^{L-1} & 0 \\ 0 & 0 & 1 & 0 & 0 & 2 & & 0 & 0 & 2^{L-1} \end{bmatrix}^T, \; \gamma(\mathbf{X}) = \begin{bmatrix} \sin(\mathbf{Px}) \\ \cos(\mathbf{Px}) \end{bmatrix}. \tag{3.42}$$

Because of this reparameterization, we can then identify an IPE closed-form expression. Using the notion that a linear transformation of a variable's covariance is a linear transformation of the variable's covariance ($\mathrm{Cov}[\mathbf{Ax}, \mathbf{By}] = \mathbf{A}\mathrm{Cov}[\mathbf{x}, \mathbf{y}]\mathbf{B}^{\mathrm{T}}$), We can determine the conical frustum Gaussian's mean and covariance as:

$$\boldsymbol{\mu}_\gamma = \mathbf{P}\boldsymbol{\mu}, \quad \boldsymbol{\Sigma}_\gamma = \mathbf{P}\boldsymbol{\Sigma}\mathbf{P}^T \tag{3.43}$$

Finally, the expectation of the lifted multivariate Gaussian can be obtained as the IPE feature, modulated by the sine and the cosine of position. These expectations have simple closed-form expressions which are attenuated by a Gaussian function of the variance:

$$\mathrm{E}_{x \sim \mathcal{N}(\mu, \sigma^2)}[\sin(x)] = \sin(\mu)\exp(-(1/2)\sigma^2) \tag{3.44}$$

$$\mathrm{E}_{x \sim \mathcal{N}(\mu, \sigma^2)}[\cos(x)] = \cos(\mu)\exp(-(1/2)\sigma^2). \tag{3.45}$$

With this, we can compute our final IPE feature as the expected sines and cosines of the mean and the diagonal of the covariance matrix:

$$\begin{aligned} \gamma(\boldsymbol{\mu}, \boldsymbol{\Sigma}) &= \mathrm{E}_{\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}_\gamma, \boldsymbol{\Sigma}_\gamma)}[\gamma(\mathbf{x})] \\ &= \begin{bmatrix} \sin(\boldsymbol{\mu}_\gamma) \circ \exp(-(1/2)\mathrm{diag}(\boldsymbol{\Sigma}_\gamma)) \\ \cos(\boldsymbol{\mu}_\gamma) \circ \exp(-(1/2)\mathrm{diag}(\boldsymbol{\Sigma}_\gamma)) \end{bmatrix}, \end{aligned} \tag{3.46}$$

where $\circ$ represents element-wise multiplication, and the diagonal of $\boldsymbol{\Sigma}_\gamma$ can be computed by:

$$\mathrm{diag}(\boldsymbol{\Sigma}_\gamma) = \left[\mathrm{diag}(\boldsymbol{\Sigma}), 4\mathrm{diag}(\boldsymbol{\Sigma}), \; \dots \;, 4^{L-1}\mathrm{diag}(\boldsymbol{\Sigma}) \right]^{\mathrm{T}}. \tag{3.47}$$

This vector depends on just the diagonal of the 3D position's covariance $\mathbf{\Sigma}$, which can be computed as:

$$\operatorname{diag}(\mathbf{\Sigma}) = \sigma_t^2(\mathbf{d} \circ \mathbf{d}) + \sigma_r^2 \left(\mathbf{1} - \frac{\mathbf{d} \circ \mathbf{d}}{\|\mathbf{d}\|_2^2} \right). \tag{3.48}$$

As we showed previously, we can conclude that PE retains all frequencies up to a manually adjusted hyperparameter. At the same time, IPE preserves consistent frequencies over an interval and gently "removes" frequencies that change over an interval. Due to the sine and cosine scaling, IPE features can encode the size and shape of a volume representation smoothly to develop an anti-aliased effect. Meanwhile, the hyperparameter L can also be removed by setting it to a substantial value.

3.4 NeRF++

Mip-NeRF is a very impressive method that provides an approach to avoiding aliasing artifacts. However, the limitation of applying NeRF-liked models to real-world problems is still existing. All experiments by authors of the mip-NeRF still concentrate on the standard datasets which are built in NeRF project. For 360-degree tests, these methods only implement reconstructing simple objects with masked backgrounds. Mip-NeRF is rarely used in realistic scenarios reconstruction, especially in representing unbounded large-scale scenes. An evaluation [2] shows that mip-NeRF still struggles with rendering 360-degree views in the real world. According to this study, mip-NeRF mainly has three aspects of issues:

1) **Parameterization.** Unbounded 360-degree scenes usually occupy an arbitrarily large Euclidean region, but NeRF and mip-NeRF can only process scenes within a bounded area. For this reason, an ideal 3D scene's optimal parameterization should be able to give local material more capacity and distant stuff less.

2) **Efficiency.** Many details need to be represented when we try to generate a large-scale scene like a cityscape. However, training a large MLP along each ray with high information content is expensive.

3) **Ambiguity.** Capturing large-scale scenes usually needs to place the sensor at a distance from the target. As a result, the content of unbounded scenes may lie at a significant distance region. Only a tiny part of the rays can observe these details, which will cause the inherent ambiguity of reconstructing the 3D scene from 2D images.

In this section, we will focus on an improved method based on NeRF named NeRF++ [77]. Overall, this approach used an extra network to represent distant objects which can contribute to solving the first problem we mentioned.

To overcome the limitation of large dynamic depth range caused by complex background, NeRF++ represents foreground and background separately. Firstly, an inverted sphere parameterization is used to facilitate free view synthesis. This sphere partitions the scene into two components, the volume of the inner unit sphere contains the foreground and all the cameras. In contrast, the outer volume represents the remainder of the environment.

Specifically, we can define the 3D sphere with $\{(x, y, z) : \sqrt{x^2 + y^2 + z^2} = 1\}$, for the 3D coordinate (x, y, z) that makes the radius $r = \sqrt{x^2 + y^2 + z^2} > 1$ are located outside the sphere. On the contrary, it is located inside the sphere. For the outer volume, we can reparameterize the coordinate by the quadruple $(x', y', z', 1/r)$, where $x'^2 + y'^2 + z'^2 = 1$ and (x', y', z') is a unit vector along the same direction as (x, y, z) representing a direction on the sphere, which is shown in figure 3.10.

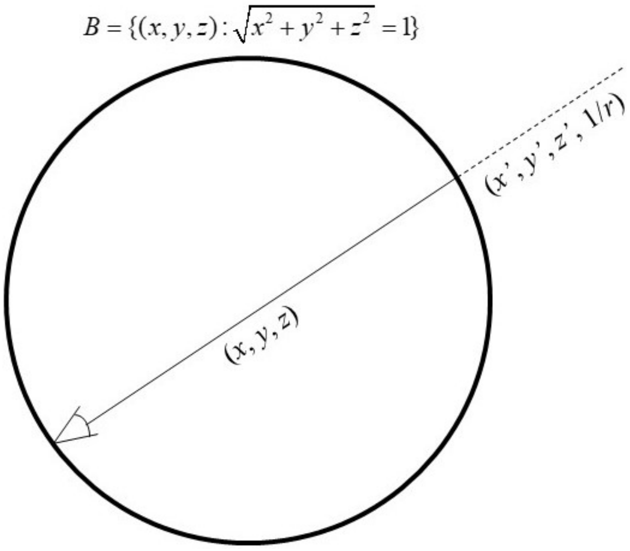

Figure 3.10: NeRF++ applies different parameterizations for scene contents inside and outside the unit sphere.

However, the outer volume cannot represent the unlimited distance from the origin, all the numbers in the re-parametrized quadruple are bounded. This is mainly because farther objects will also fade into the background in real situations. As for the ray tracing operation, the inner NeRF does not need re-parameterization, so this part is identical to the original NeRF, but the outer NeRF should leverage the 4D bounded volume after an inverted sphere parameterization. In particular, the ray $\mathbf{r} = \mathbf{o} + t\mathbf{d}$ is divided into two segments also by the unit sphere. In this way, the rendering equation 3.31 can be rewritten as:

$$\begin{aligned}\mathbf{C}(\mathbf{r}) = & \underbrace{\int_{t=0}^{t'} \sigma(\mathbf{o} + t\mathbf{d}) \cdot \mathbf{c}(\mathbf{o} + t\mathbf{d}, \mathbf{d}) \cdot e^{-\int_{s=0}^{t} \sigma(\mathbf{o}+s\mathbf{d})ds} dt}_{(i)} \\ & + \underbrace{e^{-\int_{s=0}^{t'} \sigma(\mathbf{o}+s\mathbf{d})ds}}_{(ii)} \cdot \underbrace{\int_{t=t'}^{\infty} \sigma(\mathbf{o} + t\mathbf{d}) \cdot \mathbf{c}(\mathbf{o} + t\mathbf{d}, \mathbf{d}) \cdot e^{-\int_{s=t'}^{t} \sigma(\mathbf{o}+s\mathbf{d})ds} dt}_{(iii)} .\end{aligned} \tag{3.49}$$

Terms (i) and (ii) are computed in Euclidean space, while term (iii) is computed in inverted sphere space with $\frac{1}{r}$ as the integration variable. In other words, we use $\sigma_{in}(\mathbf{o}+t\mathbf{d})$, $\mathbf{c}_{in}(\mathbf{o}+t\mathbf{d},\mathbf{d})$ in (i) and (ii), and $\sigma_{out}(x',y',z',1/r)$, $\mathbf{c}_{out}(x',y',z',1/r,\mathbf{d})$ in (iii).

To compute term (iii) for the ray, some extra operations are required to evaluate σ_{out}, $\mathbf{c}_{out}$ at any $1/r$ by computing (x',y',z') corresponding to a given $1/r$. As shown in figure 3.11, the intersection point $\mathbf{a}$ of the ray and the sphere is $\mathbf{a}=\mathbf{o}+t_a\mathbf{d}$ which can be obtained by solving $|\mathbf{o}+t_a\mathbf{d}|=1$. And the the midpoint of the chord aligning with the ray be the point $\mathbf{b}=\mathbf{o}+t_b\mathbf{d}$ which is computed by solving $\mathbf{d}^T(\mathbf{o}+t_b\mathbf{d})=0$. Then to get (x',y',z') given $1/r$, we can rotate the vector $\mathbf{a}$ along the unit vector $\mathbf{k}=\mathbf{b}\times\mathbf{d}$, which can be defined by a cross product of two arbitrary non-zero vectors, by the angle $\omega=\arcsin|\mathbf{b}|-\arcsin(|\mathbf{b}|\cdot\frac{1}{r})$. We will use Rodrigues formula for the rotated vector $\mathbf{P}$:

$$\mathbf{P}=\mathbf{a}\cos(\omega)+(\mathbf{k}\times\mathbf{a})\sin(\omega)+\mathbf{k}(\mathbf{k}\cdot\mathbf{a})(1-\cos(\omega)). \tag{3.50}$$

Then we can sample a finite number of points from the interval $[0,1]$ to compute term (iii).

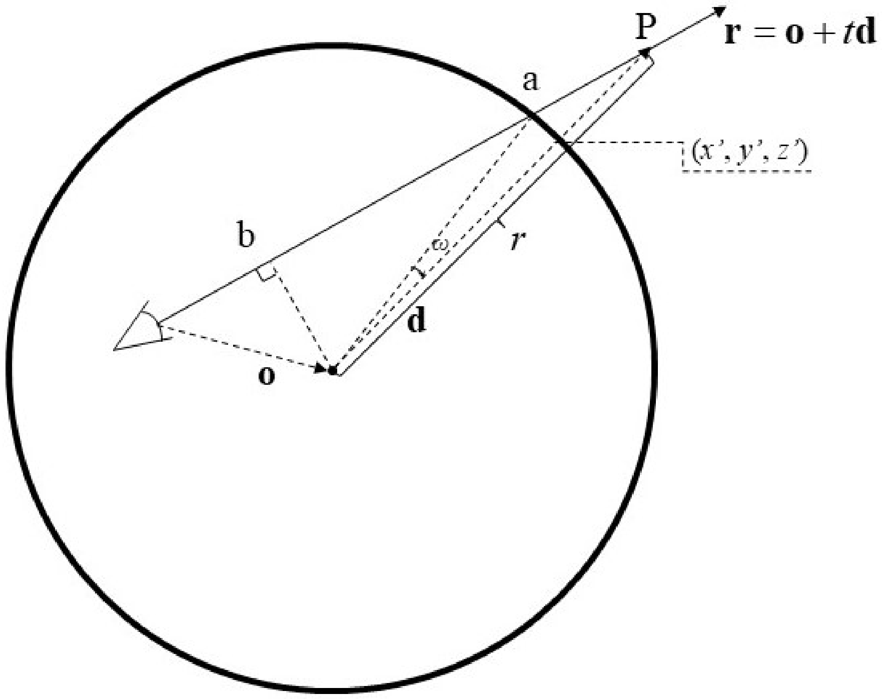

Figure 3.11: The derivation of (x',y',z') for point $\mathbf{P}$ with known $1/r$ in the inverted sphere parameterization.

According to the experiments of NeRF++, it can implement 360-degree reconstruction simply. In the next section, we will discuss more details and show the test results in Chapter 5. Moreover, mip-NeRF will be introduced into the NeRF++ to explore the performance of using different ray casting geometric shapes.

4 Implementation details

In earlier chapters, we discovered the fundamental principles of rendering with a built neural radiance representation. We aim to offer a complete introduction to our efforts and experiments throughout this project, as well as explain why we altered the model and how these changes help to high-quality reconstruction of the structure. To be more specific, we will focus on the following four aspects: the preparation of the dataset from raw images, parameters for the camera path dataset, network adjustment and the implementation of 360-degree reconstruction with NeRF++ and mip-NeRF.

4.1 Data collection and camera parameters estimation

In the beginning, it is essential for me to introduce what data should be included in the dataset and how we can obtain such a dataset when we have RGB images only.

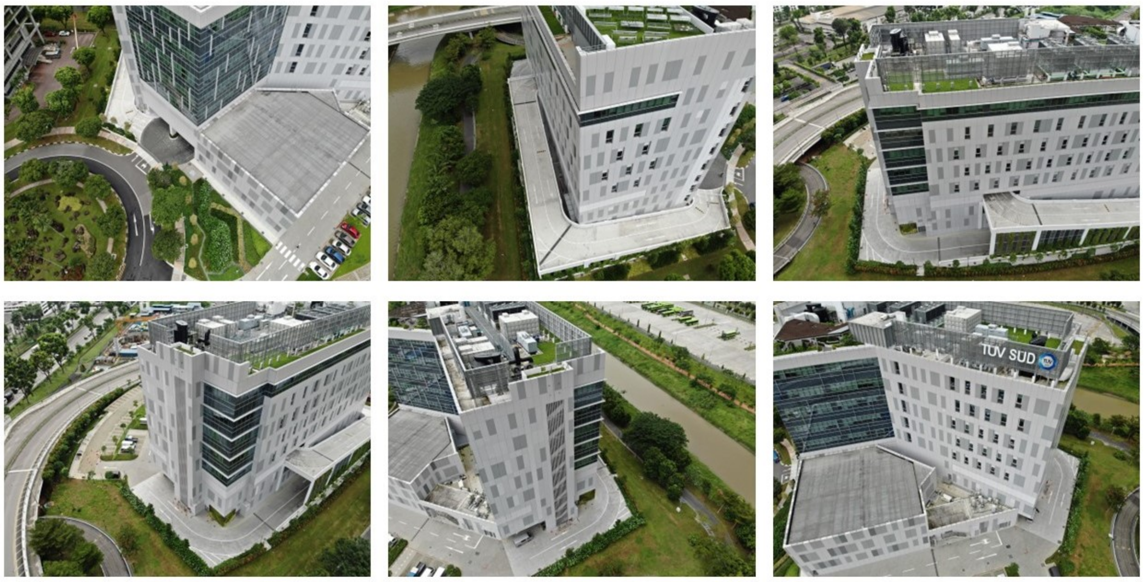

Figure 4.1: Examples of raw images from our dataset.

For this project, all the images are provided by TÜV SÜD in Singapore. They collected image information using DJI ZH20T unmanned aerial vehicle (UAV). The Resolution of the captured raw images is 4056×3040. When they collect information, they let the drone fly to a height between 70 and 80 meters and take bird's-eye photos of their office building. The drone flies slowly around the building to capture 360-degree information about the building. To ensure that too much detail is not lost the distance between the camera and the building is controlled within a limited region so that not all of these images can include the complete building. Figure 4.1 shows some example images of our dataset. From these pictures, we can notice that the background of these images is very complicated which will be very challenging for NeRF to distinguish the object from the environment.

4.1.1 Commonly-used datasets for NeRF

In this section, we will give a brief introduction to the datasets used by NeRF-like methods. The structure and data format of these datasets provide guidance on how we may pre-process our raw images.

1) **NeRF-synthetic.** NeRF creates this dataset which has eight objects, six of them are rendered from viewpoints sampled on the upper hemisphere and two of them are sampled on an entire sphere. All of these objects are generated by the open-source 3D graphic software Blender. There are 100 images in the train set and 200 images in the test set, all of them are 800×800 pixels. And it uses three JSON files to store transformation parameters for train, test, and validation sets.

2) **LLFF.** Another widely-used dataset is using the script from Local Light Field Fusion algorithm [45]. Images in this dataset are collected in the real world with cellphones and downsample these images with $\frac{1}{4}$ and $\frac{1}{8}$. Because this dataset is captured from the real world instead of modelling software, COLMAP [59] is used to estimate camera parameters and the information is stored in a NPY file. LLFF dataset mainly contains images within a minimal angle of view.

3) **Tanks and Temples.** This dataset consists of 360-degree images that predict camera poses with COLMAP. To run a NeRF-based model with this dataset, this dataset is usually constructed with corresponding TXT files to store pose and intrinsic matrices with respect to each image. In this project, we will use a similar method to pre-process our building images because of its impressive performance on 360-degree scenes.

4.1.2 Pose and intrinsic matrices estimation with COLMAP

In this section, we will show the pipeline of using COLMAP to estimate pose parameters and introduce how to use their open-source software to generate the data structure that we need.

COLMAP was developed based on incremental Structure-from-motion (SfM) which is a strategy for reconstructing 3D scenes with unordered images. The pipeline of incremental SfM is shown in figure 4.2. From this diagram, we can find that it consists of two sections, the first component is correspondence search which aims to find the overlap region between input images, and the second step is an incremental reconstruction which implements image registration and reconstruction. The first step starts by extracting invariant features from each image and then using the extracted features to find the overlap area between different images by matching the similarity between features. However, this matching process is not always reliable. Additional geometric

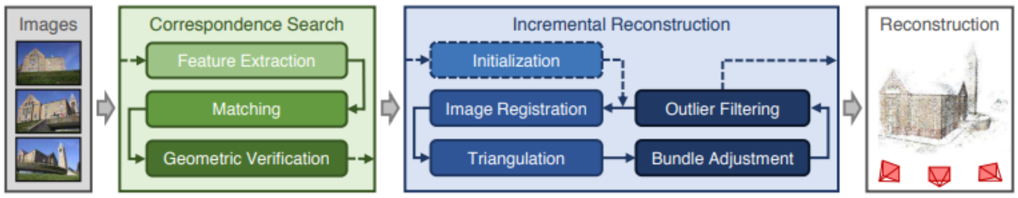

Figure 4.2: Pipeline of incremental Structure-from-Motion [59].

verification is necessary. SfM verifies the matches by attempting to estimate a transformation that leverages projective geometry to map feature points between pictures. Depending on the spatial configuration of an image pair, different mappings describe their geometric relation. This procedure outputs a scene graph with images as nodes and verified pairs of images as edges. For the incremental reconstruction stage, SfM needs to choose an initial pair at the beginning. Then by solving the Perspective-n-Point (PnP) problem [12], camera pose and intrinsic parameters can be predicted using RANSAC and a minimal pose solver [17]. And SfM has another crucial step that can improve the initial camera pose through increased redundancy, named triangulation. Up to now, all actions have been helpful for our project.

The COLMAP introduced a new geometric verification strategy, view selection method, and triangulation approach. In general, these improvements contribute to the robustness and efficiency of the camera parameters estimation results. Meanwhile, COLMAP developed mature software which is easy to install and use with customized image input. They offer both pre-built binaries and building-from-source options. After the program runs, we will get three binary files that contain sparse models, and the information is split into three files about cameras, images, and 3D points. Then the information on the world-to-camera matrices and intrinsic camera matrices can be obtained. As we introduced in Section 3.1.2, the pose matrix is the inverse of the W2C matrix, so we can quickly get pose matrices. Then the final step is storing these parameters with the needed structures. If all these processes run in order, the desired dataset is built successfully. As for the NeRF++, because we need to make sure that all cameras are inside a sphere whose radius is scaled to a smaller value than the difference between the centre of the camera positions and each camera, we need to normalize the camera position to ensure that the unit sphere covers the majority of the foreground scene content.

The dataset that we used in our experiment is built using the COLMAP. We finally chose 84 images, 70 of them are training sets and 14 of them are test sets. In each folder, three subfolders are used to save RGB images, pose matrices and intrinsic matrices, respectively. As for the RGB images, we downsample the original images to 1034×767 to improve efficiency and robustness. In the following sections, we will name our dataset with TÜV SÜD dataset.

4.2 Camera path adjustment

We have obtained all the required parameters for training and test operations. However, when we finish training the model, there is still one more step required to generate the 360-degree rendered results. We can imagine that we locate another series of cameras surrounding the object that are taking photos of the target. We need to ensure that we can observe the entire building from each camera and by combining output pictures, we can get a continuous video of the building. To determine the location and pose of these cameras, we need to set values of their poses and intrinsic matrices and put them in another subfolder.

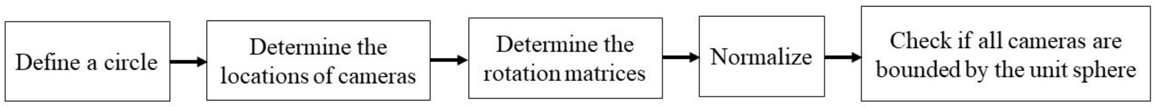
Define a circle
Determine the locations of cameras
Determine the rotation matrices
Normalize
Check if all cameras are bounded by the unit sphere

Figure 4.3: Pipeline of defining camera path.

Figure 4.3 shows the pipeline of generating the camera path dataset. The first stage is defining a circle where cameras locate on with setting sin and cos values to determine the direction of the coordinates. Ideally, there should be an axis with a fixed value so the camera will rotate around this axis. However, because our raw images are bird-eye images whose rays are not parallel to the ground, so we need to set all three axes with trigonometric functions and determine suitable coefficients for these axes, to make sure that all cameras are at a similar altitude. In the next step, specific locations of these cameras will be evenly spaced on the circle. According to our experiment, $[r\text{sin}(t), 0.87r\text{cos}(t), -0.5r\text{cos}(t) - 10]$ is the final setting for the circle that can render the building perfectly, where r is the radius of the circle and t is the location of these cameras. Afterwards, we need to define the rotation matrices of cameras to make sure that all cameras can capture nearly the whole building accurately. The fourth step is still normalization, and we also need to check if all cameras are bounded by the unit sphere as the final step. If everything is good, we need to save the generated pose matrices and intrinsic matrices to TXT files.

Choosing the right parameters for a suitable camera path is not easy, we can use Open3D [79] to visualize these cameras and their poses. It can help us have a more precise observation of the effect of different parameters on camera pose. Figure 4.4 shows a visualized example during our experiment. We can see the direction of these cameras and the difference between the two sets of camera paths intuitively.

4.3 Implementations and improvements for large 360-degree scenes

When all data is prepared, we can start to study the implementation procedure of the NeRF++ program and seek improvement methods to improve our rendering effect. In this section, we will introduce how the NeRF++ algorithm is implemented, how we

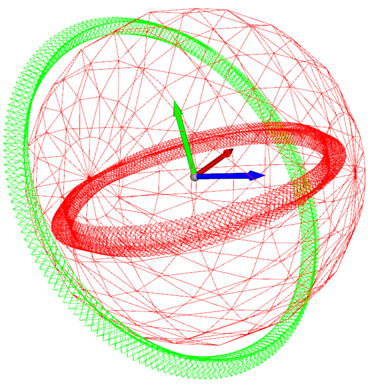

Figure 4.4: An example of visualizing camera path.

fine-tuned the network and a combination of NeRF++ and mip-NeRF to improve the performance.

4.3.1 NeRF++ implementation

Figure 4.5 shows the pipeline of NeRF++; the first step is loading the dataset and generating the origin of rays, the direction of rays, and RGB values with given pose matrices and images. Then defining the depth parameters of foreground and background respectively so that the ray functions can be expressed. Then these parameters will be input to MLP in order and we can get density and colour results, respectively. Finally, calculating the weighted summation of RGB values together will get the final RGB data of both the inner and outer spheres. Then we composite the foreground and background together and get the RGB information of an entire image. These results can compute the loss value to optimize the network during training or convert to output PNG images during testing.

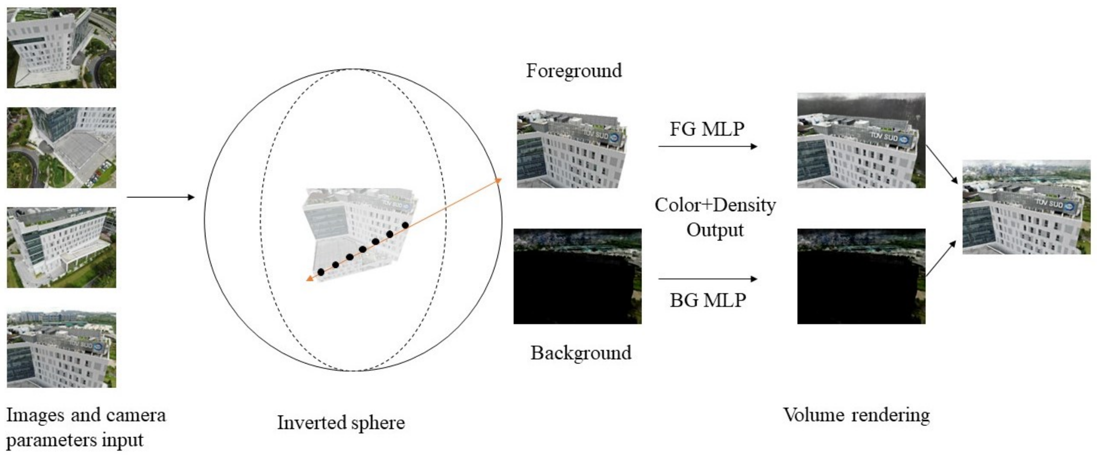

Figure 4.5: Pipeline of NeRF++.

4.3.2 Fine-tuned network

As shown in figure 3.7, there is a Sigmoid activation function at the end of the RGB layer. Following equation shows the expression of the Sigmoid function:

$$\sigma(x) = \frac{1}{1+e^{-x}}. \tag{4.1}$$

As an activation function, the Sigmoid function can limit the output range, so the data is not easy to disperse in the process of passing, but the gradient may be too small at the time of convergence. In our experiment, we tried to remove this Sigmoid function and achieved a better result.

4.3.3 Combining NeRF++ with mip-NeRF

NeRF++ uses the original NeRF model to process both foreground and background. As mip-NeRF became a popular alternative algorithm for NeRF, replacing the NeRF network in the NeRF++ with mip-NeRF is expected to be a hopeful method to improve performance. In the following sections, we will call this model mip-NeRF++.

As we introduced before, the most significant difference between mip-NeRF and NeRF is that mip-NeRF traces cones instead of rays. To express the cone, the radius value needs to be defined in advance with the width of the pixel in world coordinates scaled by $2/\sqrt{12}$. Then the multivariate Gaussian expression of conical frustums can be derived, and input the result to IPE instead of PE to encode a 3D position and its neighbour region. In our project, we only replace the foreground network's NeRF with mip-NeRF and continue to use NeRF to represent the background because the foreground is the object we need to reconstruct, so we want to concentrate more on the foreground. Meanwhile, mip-NeRF has the same number of samples on both the "coarse" and "fine" networks, but NeRF uses different values, so these parameters need to be set separately.

To be more specific, the first step of updating NeRF to mip-NeRF is to add algorithms to calculate the mean and covariance of the Gaussian function. In another work, we need to use equation 3.41 instead the simple ray function $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$. The next improvement is implementing positional encoding to approximate the mean and the covariance. Afterwards, inputting these encoded parameters into the network. The rest of the steps are the same as for NeRF++.

With changing previous components, we have a new framework with the combination of mip-NeRF and NeRF++. Experimental results and discussions will be presented in the next section.

Results and discussions 5

Section 4 introduced the most critical steps in changing 2D images to a 3D representation. In this section, we will introduce three widely recognized metric methods in the area of neural reconstruction at the beginning. Then we will show visualized rendering results and compare the performance between different approaches. Finally, we will discuss these findings and weigh the benefits and drawbacks of each strategy.

5.1 Metrics

Although visualized results can clearly show our model's performance on large-scale scene representation, we still need to calculate numerical results to evaluate our model quantitatively. There are three commonly-used metrics: PSNR, SSIM [69] and LPIPS [78].

5.1.1 PSNR

The Peak Signal-to-Noise Ratio (PSNR) is the most basic and straightforward metric in the field of neural rendering. PSNR is a statistic of the ratio between the maximum possible power of a signal and the power of corrupting noise. The logarithmic representation of the PSNR is as follows:

$$\mathrm{PSNR} = 10 \cdot \log_{10}\left(\frac{\mathrm{MAX}_I^2}{\mathrm{MSE}}\right) = 20 \cdot \log_{10}\left(\frac{\mathrm{MAX}_I}{\sqrt{\mathrm{MSE}}}\right), \tag{5.1}$$

where

$$\mathrm{MSE} = \frac{1}{mn}\sum_{0}^{m-1}\sum_{0}^{n-1}\|f(i,j) - g(i,j)\|^2, \tag{5.2}$$

f represents the matrix data of our original image, g is the matrix data of our rendered images, m indicates the number of rows of pixels of the images and i is the index of that row. Similarly, n and j show the columns and MAX_I is the maximum possible pixel value of the image.

5.1.2 SSIM

Structural Similarity Index (SSIM) is another commonly-used method to quantify the visibility of errors between a distorted image and a reference image. SSIM is a perception-based model that considers image degradation as a perceived change in structural information while incorporating critical perceptual phenomena, including luminance masking and contrast masking terms.

With knowing two windows of target images $\mathbf{x}$ and $\mathbf{y}$, and there averages μ_x and μ_y, the variance of σ_x^2 and σ_y^2, covariance σ_{xy}, $C_1 = (k_1L)^2$ and $C_2 = (k_2L)^2$ are two variables to stabilize the division with weak denominator, L is the dynamic range of the pixel-values, $k_1 = 0.01$ and $k_2 = 0.03$. The expression of SSIM is based on three comparison measurements: luminance, contrast and structure:

$$l(\mathbf{x},\mathbf{y}) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y 2 + C_1}, \quad c(\mathbf{x},\mathbf{y}) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2}, \quad s(\mathbf{x},\mathbf{y}) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3}, \tag{5.3}$$

where $C_3 = C_2/2$. Then, SSIM can be computed by a weighted combination of those comparative measures:

$$\mathrm{SSIM}(\mathbf{x},\mathbf{y}) = [l(\mathbf{x},\mathbf{y})]^\alpha \cdot [c(\mathbf{x},\mathbf{y})]^\beta \cdot [s(\mathbf{x},\mathbf{y})]^\gamma, \tag{5.4}$$

where α, β and γ are parameters used to adjust the relative importance of the three components. If $\alpha = \beta = \gamma = 1$, we can get:

$$\mathrm{SSIM}(\mathbf{x},\mathbf{y}) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{5.5}$$

5.1.3 LPIPS

The Learned Perceptual Image Patch Similarity (LPIPS) is another metric which can be used to judge the perceptual similarity between two images. LPIPS aims to compute the similarity between the activations of two image patches for some pre-defined and more similar image patches that will bring lower LPIPS.

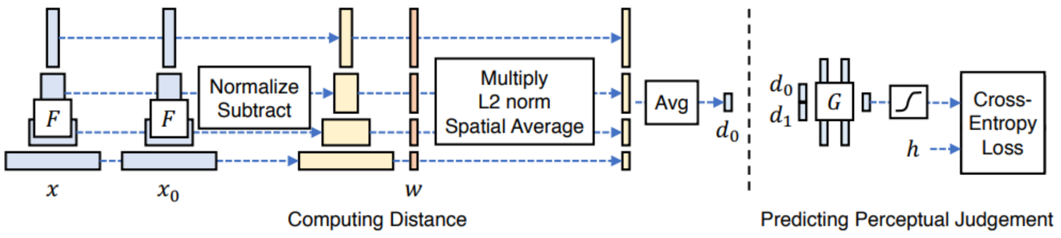

Figure 5.1: Pipeline of calculating LPIPS [78].

As shown in figure 5.1, a neural network F (VGG [62], AlexNet [33] or SqueezeNet [24]) is used to extract features from two input patches x, x_0. Then we unit-normalize the output features of each layer in the channel dimension to $\hat{y}^l, \hat{y}_0^l \in \mathbb{R}^{H_l \times W_l \times C_l}$. Then the l_2 distance is calculated by multiplying the weights $w_l \in \mathbb{R}^{C_l}$ and averaging the distances. The expression of calculating the distance is shown below:

$$d(x, x_0) = \sum_l \frac{1}{H_l W_l} \sum_{h,w} \| w_l \odot (\hat{y}_{hw}^l - \hat{y}_{0hw}^l) \|_2^2, \tag{5.6}$$

where h is a perceptual judgment trained by a small network G and $w_l = 1 \forall l$ is equivalent to computing cosine distance.

5.2 Reconstruction results of the TÜV SÜD dataset

For our experiments, we set the learning rate to a fixed value 5×10^{-4} and use the Adam optimizer to optimize the network. Optimizing a scene typically takes around 250-300k iterations to converge on two NVIDIA RTX A6000 GPUs to accelerate training.

5.2.1 Rendered results based on the test set

When we finish training and acquire checkpoints, we must first render photos using test postures, such that the output images have the same view as the test images.

The first pair of images (figure 5.2 and figure 5.3) show a comparison between results from NeRF and NeRF++ respectively using the same dataset. We can observe that the image on the left-hand side is extremely blurred and we can only barely make out the exterior of the building without any detailed display. On the other hand, the result of NeRF++ is much better, we can recognize the surface of the building, and many details like windows and letters are also represented. As for the numerical result, we can also find that the value of PSNR increased significantly from 12.83 to 16.83.

Figure 5.4 shows the results of training the NeRF++ model with our customized building dataset. The top two rows display the entire rendered images, the middle two rows only show the predicted foreground scenes without background information, and the last two lines are the ground truth images. As for the rendered results, we can find that for each angle of the scene, the majority part of the building can be represented accurately. However, they are still quite blurred when we try to observe details, such as letters and glasses. For the background landscape, we can find that our results are also not detailed. Buildings in the distance cannot be represented because of the lack of depth information in the training dataset, i.e., choosing images depends more on the coverage of buildings instead of the background. We find that most of the background vanishes for the six images without background, and the building is highlighted.

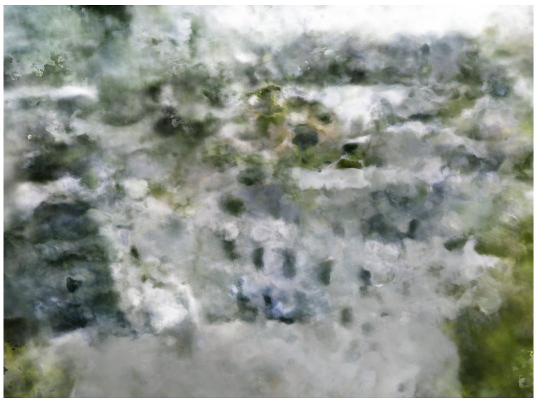
Figure 5.2: NeRF result (PSNR: 12.83).

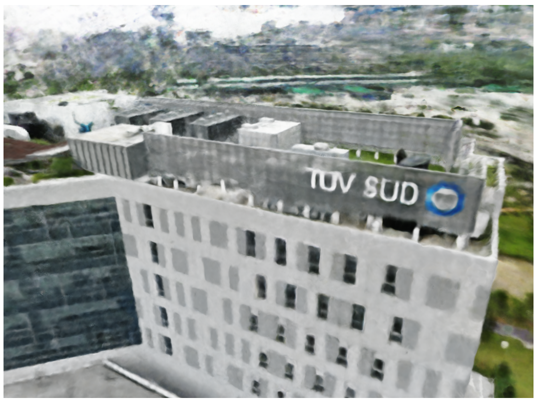

Figure 5.3: NeRF++ result (PSNR: 16.83).

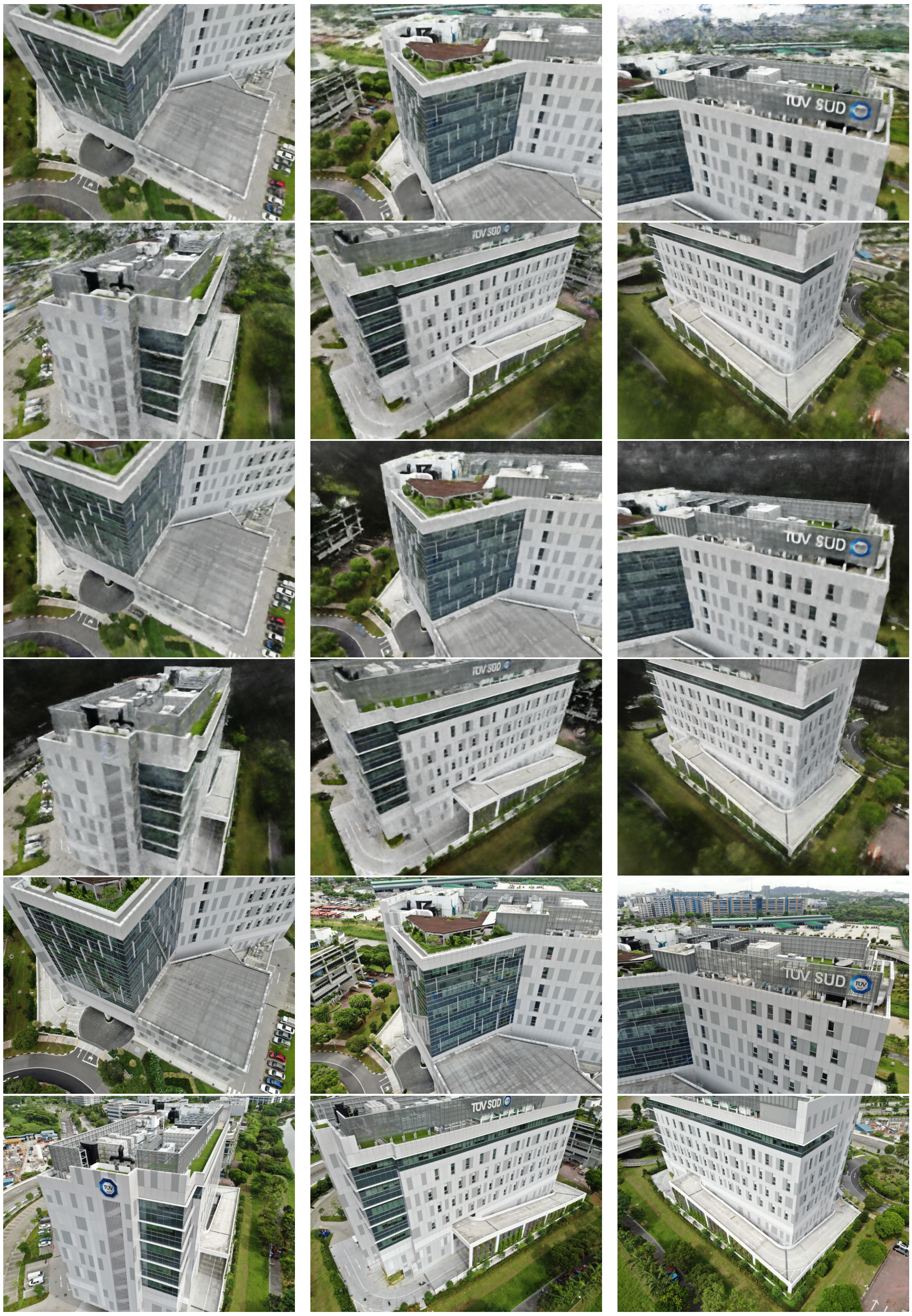

Figure 5.4: Rendered results with test set camera settings using NeRF++.

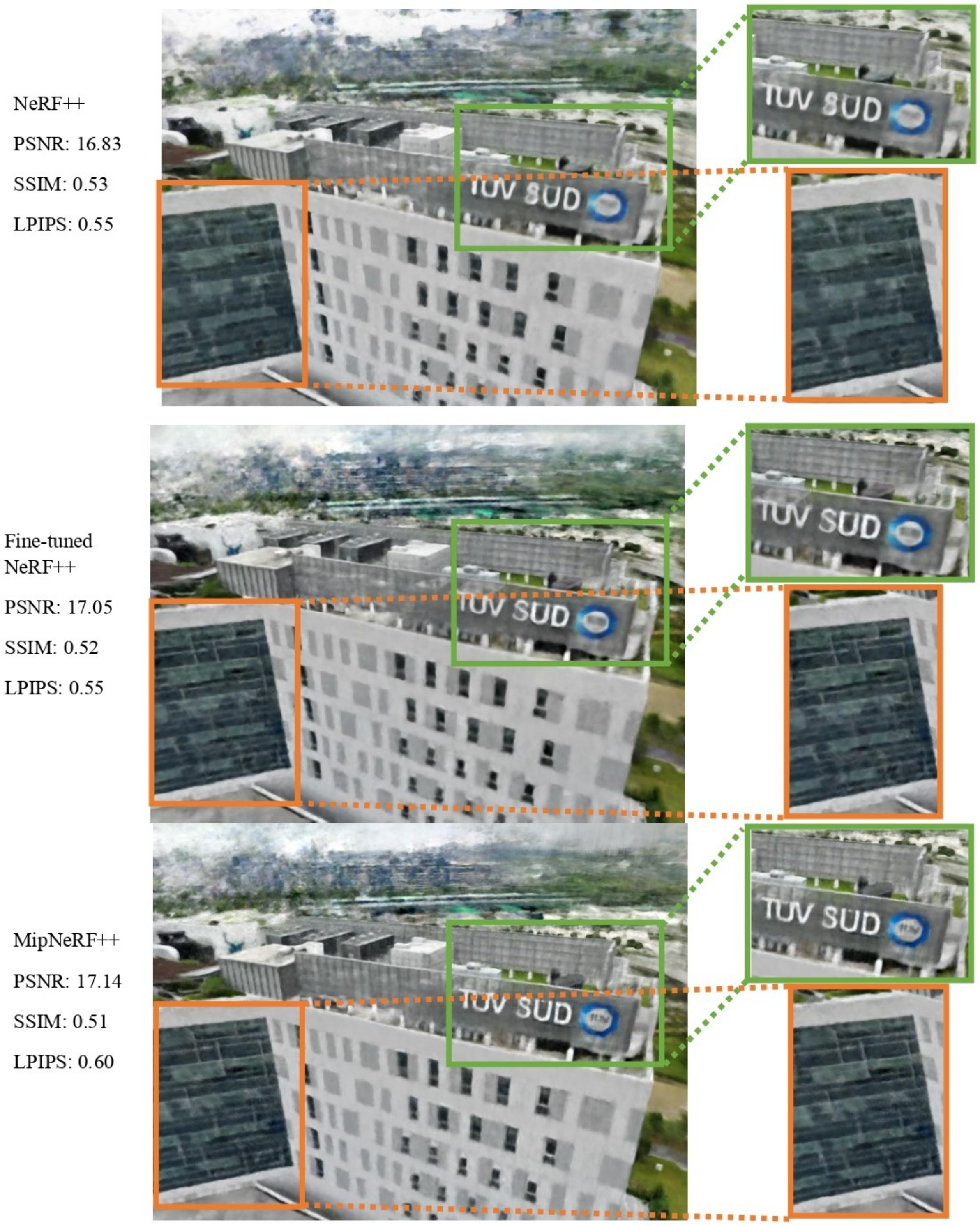

Figure 5.5: Comparison of rendering from the same view with using NeRF++, fine-tuned NeRF++ and mip-NeRF++ respectively. Fine-tuned NeRF++ and replacing NeRF with mip-NeRF can improve the ability to represent details.

Furthermore, we have made improvements by removing the Sigmoid activation function and mip-NeRF, figure 5.5 shows the results. The first image is the result from NeRF++. We can find that the logo and letters on the building are not transparent and the glass is blurred. And the second image is the result from NeRF++ without Sigmoid activation, an improvement in the region of the letters can be clearly noticed. The last one is from the framework with mip-NeRF, the overall appearance of the picture is sharper. Meanwhile, focusing on the letter "Ü", the rendered results from NeRF++ models are highly blurred. However, when mip-NeRF is introduced to NeRF++, these two points can be distinguished. And the word on the logo is also vaguely distinguishable as another two are entirely vague.

In summary, we can conclude that our improvement can benefit rendering the target building with given raw images in visualized results. In the next section, we will provide numerical results to analyse our results quantitatively.

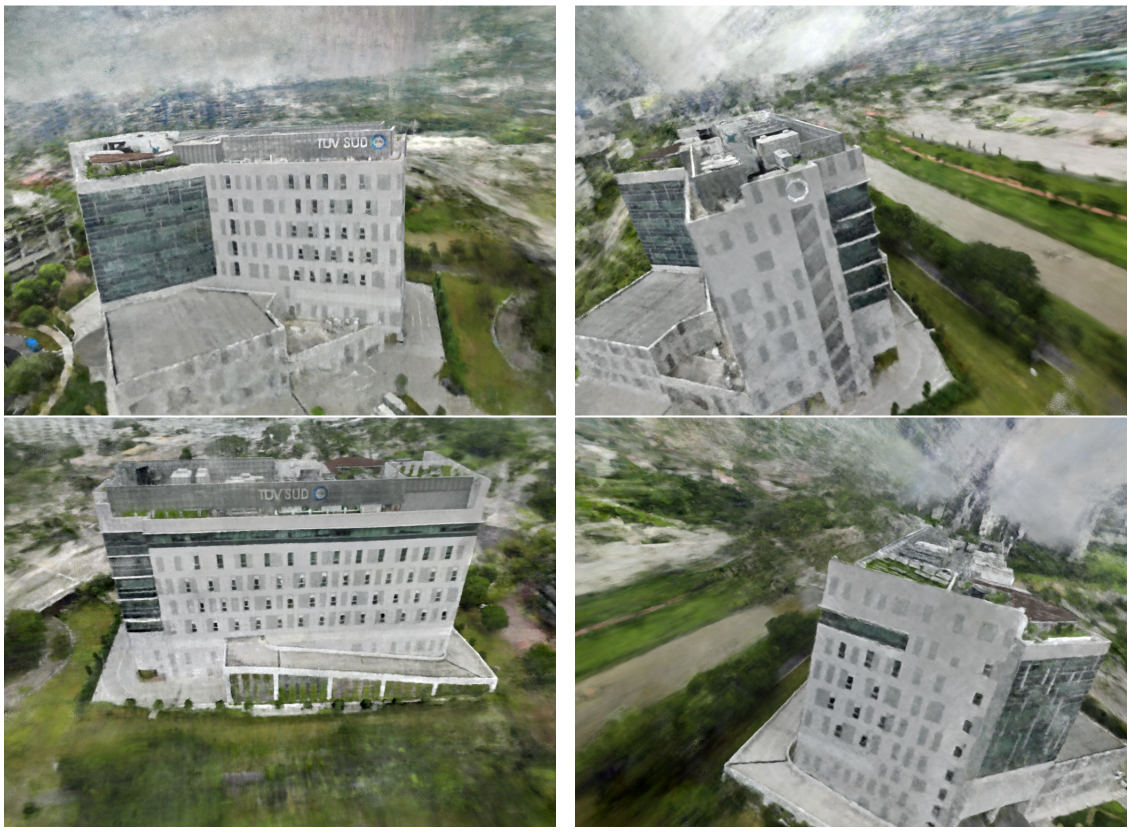

Figure 5.6: Rendered results with camera path using mip-NeRF++.

5.2.2 Rendered results based on camera path

After verifying our model on the test set, we can render the final images with our camera path. For our experiments, we locate 200 cameras in total on a circle around the building, combining images of these cameras, we can obtain a video to help us view the representation from 360-degree views. Figure 5.6 shows 4 examples of the RGB results and figure 5.7 presents results without background. More output images can be found in the appendix.

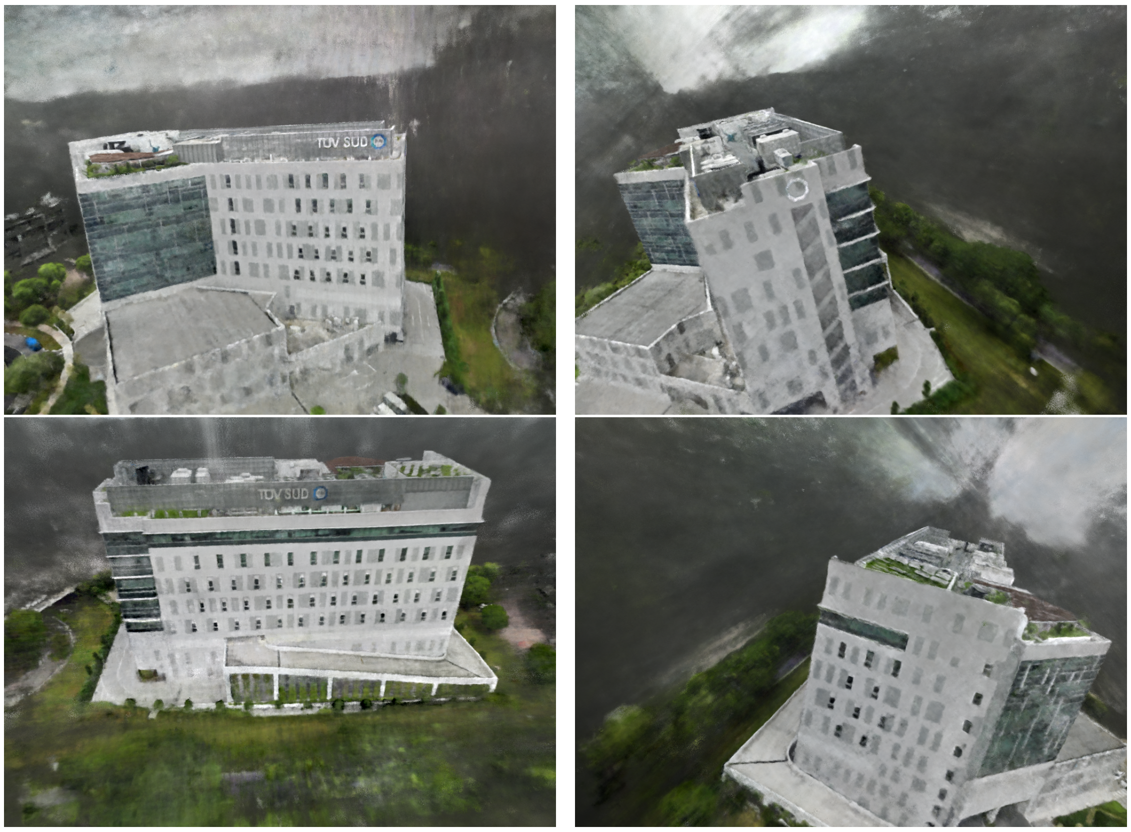

Figure 5.7: Rendered results with camera path using mip-NeRF++ with foreground network only.

5.3 Evaluation and discussion

5.3.1 Results with the TÜV SÜD datasset

Table 5.1 shows numerical results of testing NeRF, NeRF++, fine-tuned NeRF++ and mip-NeRF++ respectively. We can observe that each boost has an improvement on the PSNR results, but the values of SSIM and LPIPS have no greater change or even a tendency to get worse.

	PSNR↑	SSIM↑	LPIPS↓
NeRF	11.86	0.33	0.71
NeRF++	19.44	**0.58**	**0.52**
Fine-tuned NeRF++	19.59	**0.58**	**0.52**
Mip-NeRF++	**19.74**	0.57	0.57

Table 5.1: PSNR, SSIM and LPIPS results of all three main implementations.

First of all, it shows clearly that NeRF++ improves the performance of 360-degree large-scale scene reconstruction distinctly. In general, two facts may cause this terrible consequence: (1) the amount of input images is not large enough, NeRF cannot represent all details using only one network, (2) the outdoor 360 scene has an extremely complicated background with extensive depth range, significant artifacts or severe resolution loss may be caused. NeRF++ uses a different network to process foreground and background separately can contribute to avoiding such an unbounded problem.

Combining the visualized results and numerical results, we can tentatively conclude that the changing of traced ray can benefit the visualized results to some extent. Compared to ray-tracing, cone-tracing can help to restore more details, including generating more identifiable letters and logos and a more contrasted glass effect. From a geometric point of view, casting a ray requires positional encoding features extracted by sampling points. Because of the inadequate expressiveness due to being too narrow per ray, the detailed shape and size of the object may be ignored by each ray. Consequently, the same confusing point-sampled feature may be produced by two different cameras imaging the same spot at different scales. When we switch to tracing cones, we get a continuous space of scales. The intersection of two cones may explicitly model the volume of each sampled conical frustum. This can be easily considered as a simulation of mipmapping, which uses a series of images with progressively lower resolution to speed up rendering and reduces image jaggies. Sections from the top to the bottom of the cone represent more information and construct a prefiltered radiance field.

Besides the overall analysis, we also evaluate the building's front, back and side, respectively. To better display the model's ability to represent details, we have taken the areas of these images that contain more detail as we did in figure 5.5. Numerical results of these areas are also calculated and are shown in table 5.2, 5.3, and 5.4.

Table 5.2: Numerical results of the front scene, segmentation 1 is the logo view and segmentation 2 is the glass view.

	Full image			Segmentation 1			Segmentation 2		
	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
NeRF++	16.83	**0.53**	**0.55**	**17.38**	**0.48**	**0.50**	19.29	0.51	0.57
Fine-tuned NeRF++	17.05	0.52	**0.55**	16.97	**0.48**	**0.50**	19.72	**0.52**	**0.55**
Mip-NeRF++	**17.14**	0.51	0.60	16.70	0.45	0.56	**19.93**	0.51	0.64

Table 5.3: Numerical results of the back scene, segmentation 1 is the logo view and segmentation 2 is the slope view.

	Full image			Segmentation 1			Segmentation 2		
	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
NeRF++	20.54	**0.65**	**0.49**	20.73	**0.65**	**0.46**	**20.04**	**0.56**	0.55
Fine-tuned NeRF++	20.53	**0.65**	**0.49**	20.53	**0.65**	**0.46**	19.46	0.55	**0.48**
Mip-NeRF++	**21.08**	0.64	0.53	**21.29**	**0.65**	0.50	19.86	0.54	0.50

Table 5.4: Numerical results of the side scene, segmentation 1 is the roof view and segmentation 2 is the glass view.

	Full image			Segmentation 1			Segmentation 2		
	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
NeRF++	23.04	**0.69**	0.51	20.61	0.58	**0.52**	21.87	0.59	0.49
Fine-tuned NeRF++	23.04	**0.69**	**0.50**	**20.70**	**0.59**	**0.52**	22.13	**0.60**	**0.48**
Mip-NeRF++	**23.13**	0.68	0.57	20.69	0.56	0.59	**22.28**	0.59	0.54

From this detailed evaluation, the previous inference is proved to be a common phenomenon. All of the PSNR results of these full images show that introducing the idea of mip-NeRF to NeRF++ can benefit the result. Because PSNR is computed by comparing the maximum possible intensity levels of the image and the mean square error between the rendered images and the ground truth images, higher PSNR represents that the reconstructed model has better quality in direct comparison for each pixel. However, the major limitation of PSNR is that it does not consider the visual characteristics of the human eye, so it cannot always reflect human perception. Therefore, SSIM and LPIPS are introduced as references. According to SSIM and LPIPS results, which evaluate the differences between the two images from more aspects. The poor performance of our mip-NeRF++ indicates that our model is not a perfect improvement. Considering the performance of visualized and numerical results together, our model still has its strength in detail reconstruction but also has weakness in an all-around arrangement. Consequently, our model is still a competitive approach for tasks that require restoring more details.

5.3.2 Results with the Tanks and Temples dataset

In addition to the TÜV SÜD dataset, we also tested our model on the Tanks and Temples dataset [31]. Table 5.5 shows the results of testing on the train scene with setting the batch size to 1024. All three metrics on this public dataset show that our model has a noticeable improvement compared to previous approaches. Combining with the visualized results as demonstrated in figure 5.8, 5.9, 5.10 and 5.11, we can conclude that in some more miniature practical scenes, our model is still competitive. However, the fine-tuned results are not a good improvement for this scenario.

Table 5.5: PSNR, SSIM and LPIPS results of the train scene with three main implementations.

	PSNR↑	SSIM↑	LPIPS↓
NeRF++	16.77	0.50	0.58
Fine-tuned NeRF++	16.83	0.50	0.59
Mip-NeRF++	**17.41**	**0.52**	**0.57**

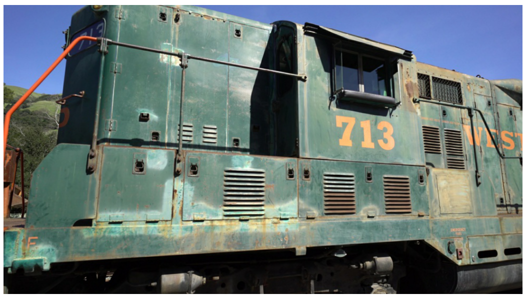

Figure 5.8: Ground truth.

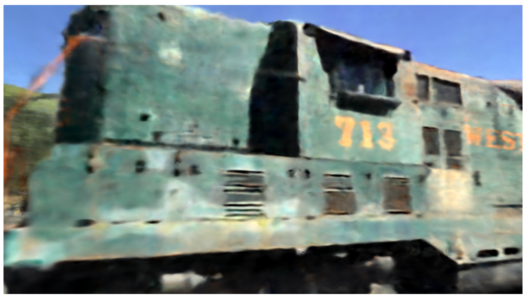

Figure 5.9: NeRF++.

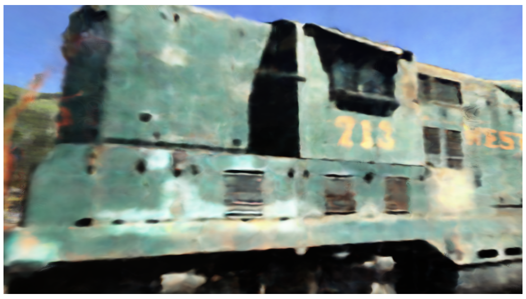

Figure 5.10: Fine-tuned NeRF++.

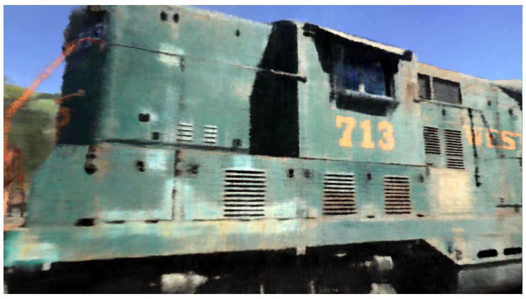

Figure 5.11: Mip-NeRF++.

Conclusions and future works 6

6.1 Conclusion

This project accomplished 360-degree large-scale scene reconstruction of a building using neural radiance fields-based approaches. Experiments include data preparation, camera calibration, network training, and rendering. During this project, we discovered the limitation of NeRF, And then we found NeRF++, which can solve our problems perfectly and have a good match for our goals. As a result, we chose NeRF++ as our starting point and proposed several methods to improve performance in response to our needs. The first improvement happens on the network, we found that removing the Sigmoid function can benefit the resolution of the rendered image. Furthermore, we also embedded mip-NeRF into the NeRF++ model to improve the anti-aliasing ability of the foreground network. Eventually, visualized generated results show the effect of improvement intuitively and numerical results also help us recognize the impact of different models from different aspects. We found that our model is only partially ahead in reproducing large buildings, but its improvement is more notable in a smaller scene from the Tanks and Temples dataset. We may conclude from these results that our model is worth examining in some practical scenarios. Additionally, the position of hypothetical cameras used to observe the reconstructed model from surrounding perspectives are defined by setting camera paths. For different objects, they require additional parameters to compute the final pose matrices. This mainly depends on the size of the thing, the distance between the lens and the target, and the relationship between the target and world coordinates. With plenty of experiments, the most suitable camera parameters are obtained to get the best view of the whole building.

According to these results that we received from experiments, we have several important findings as follows:

1) NeRF++ is a powerful model for large scene representation and is still efficient in super large-range reconstruction.

2) Removing the Sigmoid function from the RGB layer is worth to be tried to improve the training process.

3) Introducing mip-NeRF to NeRF++ to change the way of ray casting is controversial. On the one hand, visualized results indicate that mip-NeRF helps the model generate more precise details. PSNR values also show that the improved model performs better on the pixel level. On the other hand, SSIM and LPIPS values are not improved compared to the original model. It means that our rendered results are still defective for human habits. However, this attempt is still a potential method to obtain better large-scale reconstruction results.

6.2 Contributions

In this section, we will summarize our efforts to achieve this effect:

1) Constructing the TÜV SÜD dataset with given raw images. Data are pre-processed by COLMAP and required camera parameters are estimated.

2) Successfully implemented NeRF++ with the constructed dataset and fine-tuned the network to improve the performance.

3) Setting camera path parameters against our data so that we can observe the entire building from 360-degree view in the distance.

4) We combined mip-NeRF with NeRF++ and came up with mip-NeRF++ that can achieve a better result in some assessment methods. This improvement provides a new option for realistic large-scale scene reconstruction.

6.3 Future work

NeRF-based methods, being one of the most popular neural rendering approaches, have enormous potential to be applied to additional domains and achieve greater outcomes. In terms of large-scale 360-degree scene reconstruction, the following approaches are still worth investigating for future study.

For large-scale scene reconstruction, the first method worth trying is parameterizing rays before positional encoding as done in [2]. A parameterization of a 3D scene in terms of disparity can allocate more capacity to nearby content and less to distant range. In practice, a Jacobian function can be used as the smooth coordinate transformation to map parameters. This process is equivalent to the classic Extended Kalman filter [28]. However, implementing this process is problematic because it needs to calculate the Jacobian matrix in each step. However, using PyTorch to calculate consumes too many resources, so the training speed is deficient. As a result, we have to give up this idea in this project. In future work, with the update of PyTorch or using another framework such as JAX may have the ability to overcome this limitation.

The second vital improvement may happen in training and rendering speed. Too slow training is a common problem for all original NeRF-based methods. A typical NeRF-based model takes more than one day to train and render when trained on reasonably high-performance GPUs. If we do not downsample the raw images, it will consume more time and resources. A typical NeRF-based model takes more than one day to train and render when trained on reasonably high-performance GPUs. To solve this problem, a caching system can be developed to store learned radiance values and then use them in explicit spatial data structures during rendering. In addition, another less compactness method introduces a dual network that takes an orthogonal direction by learning how to reduce the number of required sample points best.

The third possible effort is about data collection. For this project, the original dataset only contains RGB images without any camera parameters. So we have to use COLMAP to estimate all the required parameters. However, results from COLMAP are not always precise, so the offset of rays may cause ambiguity. In this region, if we pursue developing a commercial 3D reconstruction model, we should change the way of collecting data. For instance, we can calibrate the camera in advance so that the intrinsic matrix can be obtained directly. Pose matrices can also be more accurately estimated by using GPS information. Aside from camera sensors, lidar sensors are helpful for measuring distances to surfaces, leading to more accurate ray expressions.

A Rendered examples

A.1 Rendered entire images using NeRF++

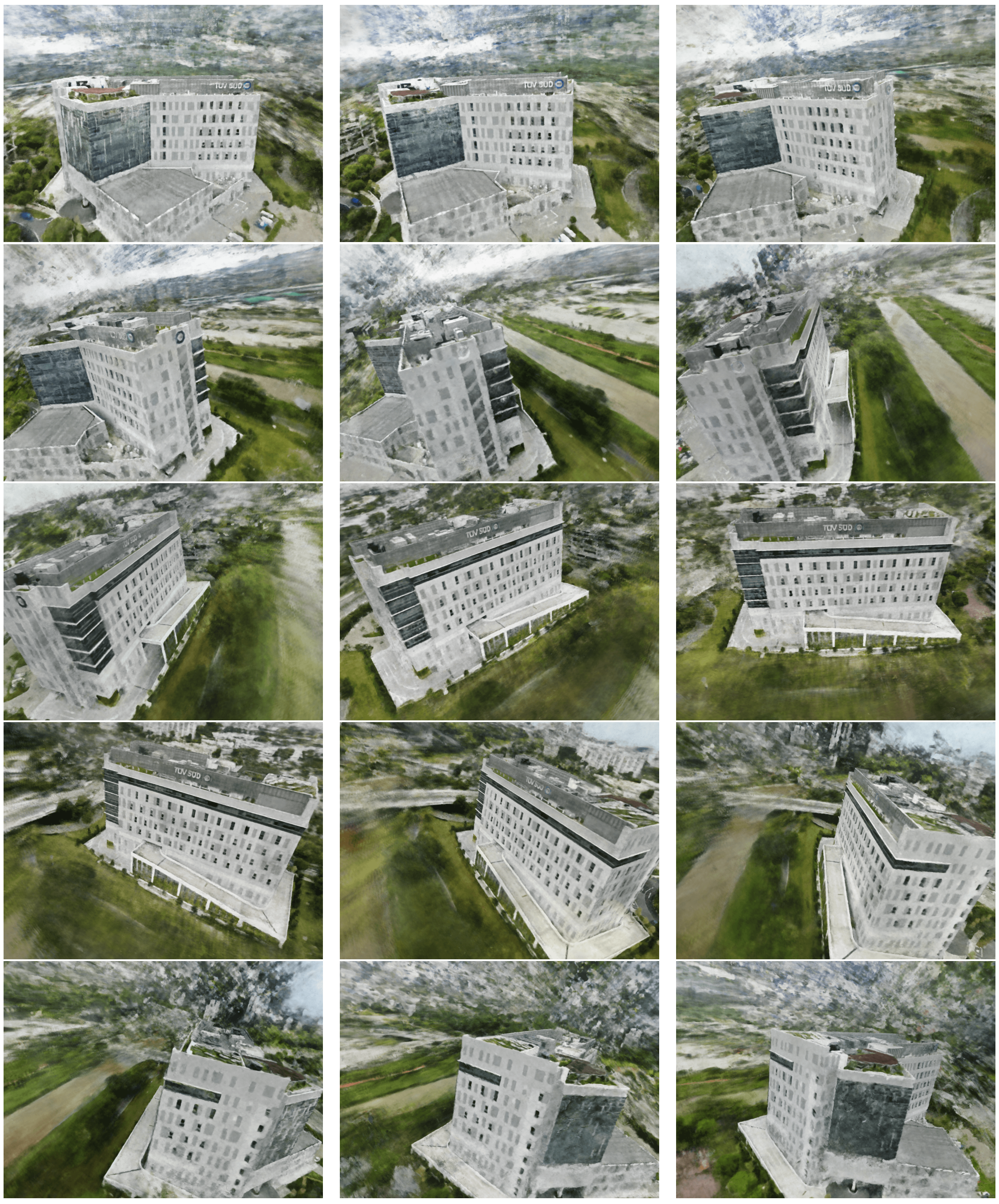

Figure A.1: Examples of rendered images with NeRF++

A.2 Rendered images with foreground only using NeRF++.

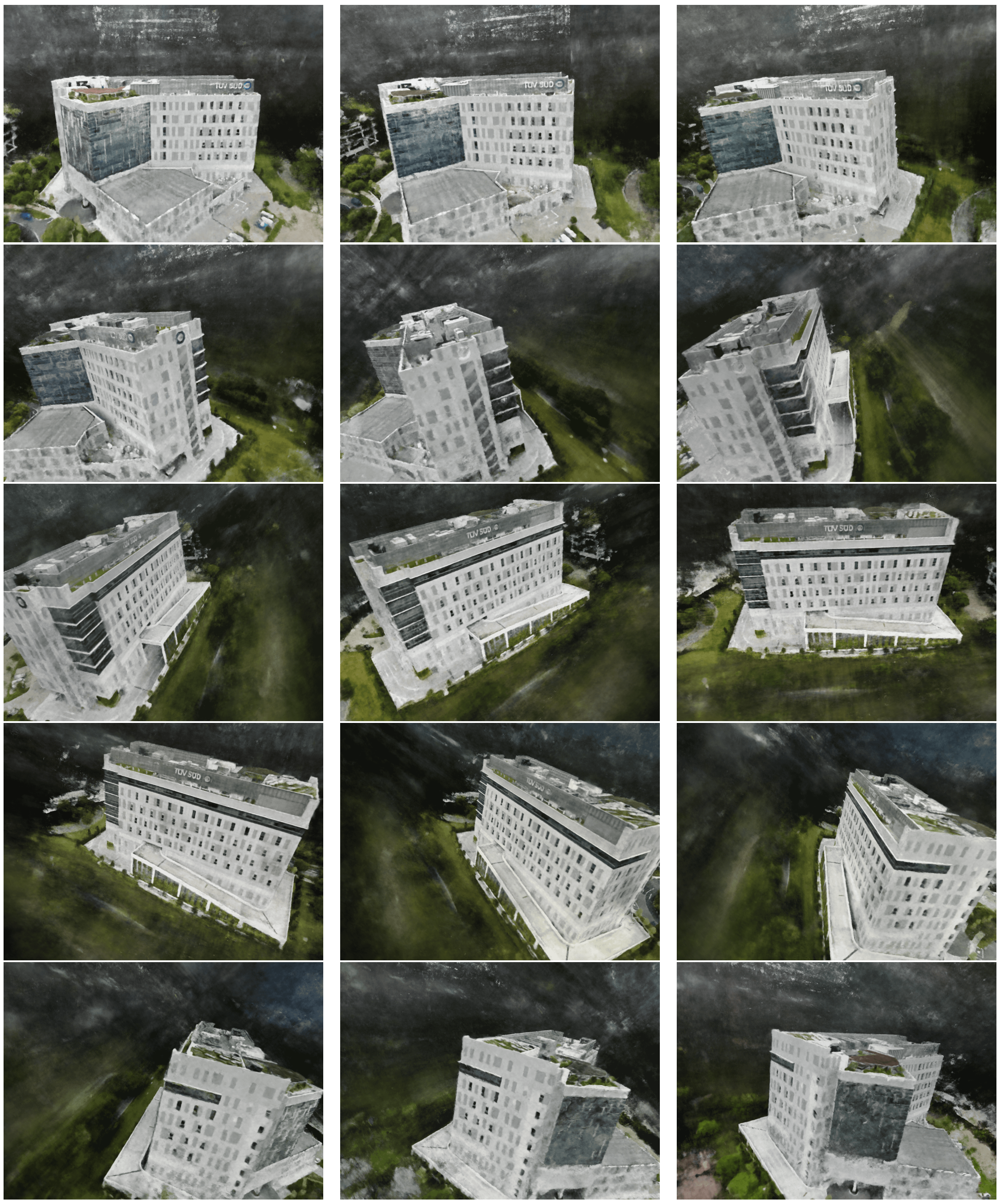

Figure A.2: Examples of rendered foreground images with NeRF++.

A.3 Rendered entire images using fine-tuned NeRF++

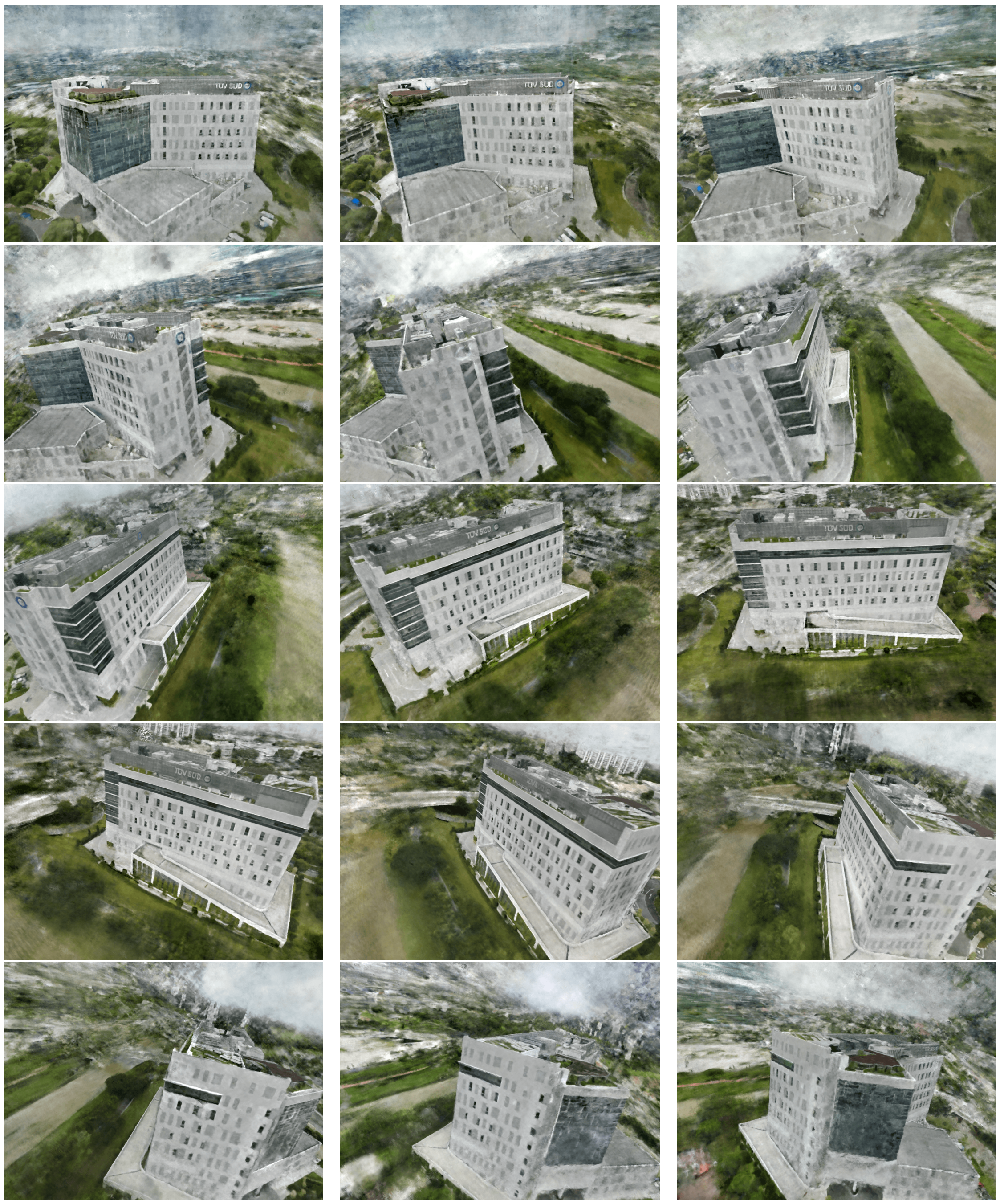
Figure A.3: Examples of rendered images with fine-tuned NeRF++

A.4 Rendered images with foreground only using fine-tuned NeRF++

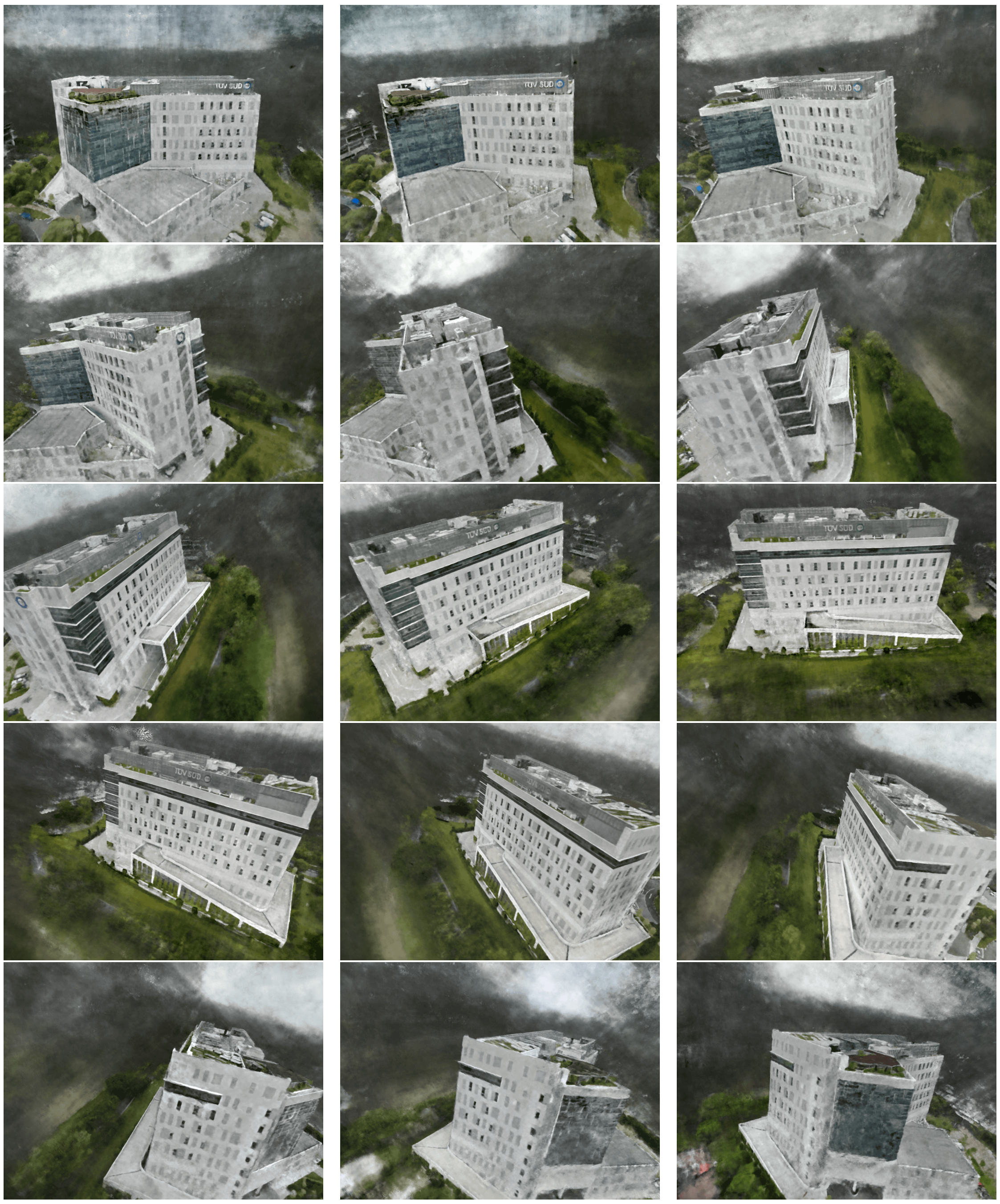

Figure A.4: Examples of rendered foreground images with fine-tuned NeRF++.

A.5 Rendered entire images using mip-NeRF++

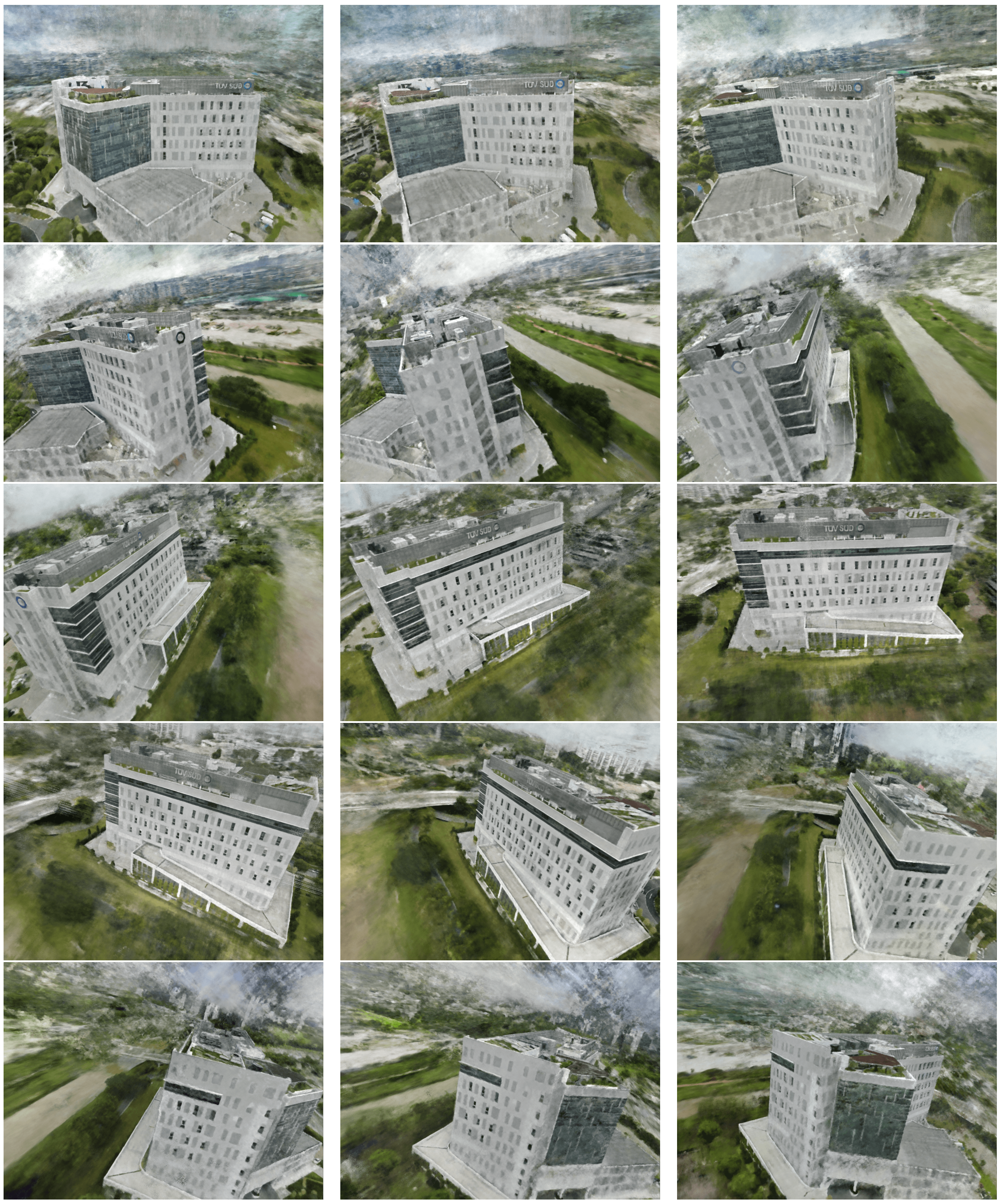
Figure A.5: Examples of rendered images with mip-NeRF++.

A.6 Rendered images with foreground only using mip-NeRF++

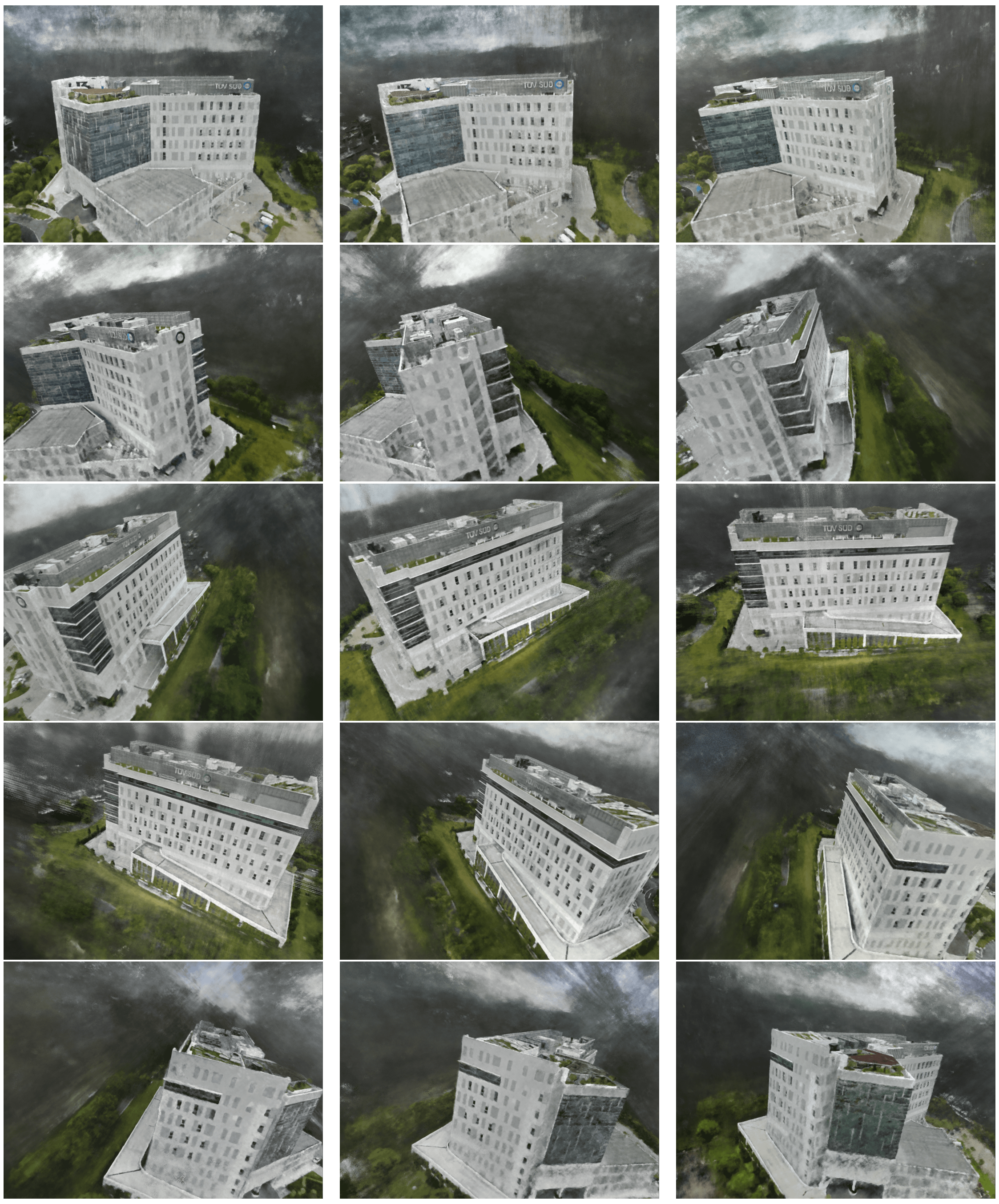

Figure A.6: Examples of rendered foreground images with mip-NeRF++.

Bibliography

[1] Sameer Agarwal et al. “Building rome in a day”. In: *Communications of the ACM* 54.10 (2011), pp. 105–112.

[2] Jonathan T Barron et al. “Mip-nerf 360: Unbounded anti-aliased neural radiance fields”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 5470–5479.

[3] Jonathan T Barron et al. “Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 5855–5864.

[4] Mario Botsch, Andreas Wiratanaya, and Leif Kobbelt. “Efficient high quality rendering of point sampled geometry.” In: *Rendering Techniques* 2002.13th (2002), p. 2.

[5] Michael Broxton et al. “Immersive light field video with a layered mesh representation”. In: *ACM Transactions on Graphics (TOG)* 39.4 (2020), pp. 86–1.

[6] Chris Buehler et al. “Unstructured lumigraph rendering”. In: *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. 2001, pp. 425–432.

[7] Giang Bui et al. “Point-based rendering enhancement via deep learning”. In: *The Visual Computer* 34.6 (2018), pp. 829–841.

[8] Andrei Burov, Matthias Nießner, and Justus Thies. “Dynamic surface function networks for clothed human bodies”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 10754–10764.

[9] Anpei Chen et al. “TensoRF: Tensorial Radiance Fields”. In: *arXiv preprint arXiv:2203.09517* (2022).

[10] Ulrich Clarenz, Martin Rumpf, and Alexandru C Telea. “Finite elements on point based surfaces”. In: *PBG*. 2004, pp. 201–211.

[11] Tomáš Davidovič et al. “3D rasterization: a bridge between rasterization and ray casting”. In: *Proceedings of Graphics Interface 2012*. 2012, pp. 201–208.

[12] Martin A Fischler and Robert C Bolles. “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography”. In: *Communications of the ACM* 24.6 (1981), pp. 381–395.

[13] John Flynn et al. *Deepstereo: learning to predict new views from real world imagery*. US Patent 9,916,679. Mar. 2018.

[14] John Flynn et al. “Deepview: View synthesis with learned gradient descent”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 2367–2376.

[15] Christian Früh and Avideh Zakhor. “An automated method for large-scale, ground-based city model acquisition”. In: *International Journal of Computer Vision* 60.1 (2004), pp. 5–24.

[16] Yasutaka Furukawa and Jean Ponce. “Accurate, dense, and robust multi-view stereopsis (pmvs)”. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Vol. 2. 2007, p. 3.

[17] Xiao-Shan Gao et al. "Complete solution classification for the perspective-three-point problem". In: *IEEE transactions on pattern analysis and machine intelligence* 25.8 (2003), pp. 930–943.

[18] Kyle Genova et al. "Learning shape templates with structured implicit functions". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision.* 2019, pp. 7154–7164.

[19] Kyle Genova et al. "Local deep implicit functions for 3d shape". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 2020, pp. 4857–4866.

[20] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems* 27 (2014).

[21] Jeffrey P Grossman and William J Dally. "Point sample rendering". In: *Eurographics Workshop on Rendering Techniques.* Springer. 1998, pp. 181–192.

[22] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision.* Cambridge university press, 2003.

[23] Peter Hedman et al. "Deep blending for free-viewpoint image-based rendering". In: *ACM Transactions on Graphics (TOG)* 37.6 (2018), pp. 1–15.

[24] Forrest N Iandola et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and¡ 0.5 MB model size". In: *arXiv preprint arXiv:1602.07360* (2016).

[25] Yoonwoo Jeong et al. "Self-calibrating neural radiance fields". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision.* 2021, pp. 5846–5854.

[26] Chiyu "Max" Jiang et al. "Local Implicit Grid Representations for 3D Scenes". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).* June 2020.

[27] James T Kajiya and Brian P Von Herzen. "Ray tracing volume densities". In: *ACM SIGGRAPH computer graphics* 18.3 (1984), pp. 165–174.

[28] Rudolph Emil Kalman. "A new approach to linear filtering and prediction problems". In: (1960).

[29] Angjoo Kanazawa et al. "Learning category-specific mesh reconstruction from image collections". In: *Proceedings of the European Conference on Computer Vision (ECCV).* 2018, pp. 371–386.

[30] Tero Karras et al. "Analyzing and improving the image quality of stylegan". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition.* 2020, pp. 8110–8119.

[31] Arno Knapitsch et al. "Tanks and Temples: Benchmarking Large-Scale Scene Reconstruction". In: *ACM Transactions on Graphics* 36.4 (2017).

[32] Johannes Kopf et al. "Street slide: browsing street level imagery". In: *ACM Transactions on Graphics (TOG)* 29.4 (2010), pp. 1–8.

[33] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012).

[34] Samuli Laine and Tero Karras. "Efficient sparse voxel octrees–analysis, extensions, and implementation". In: *NVIDIA Corporation* 2.6 (2010).

[35] Marc Levoy and Pat Hanrahan. “Light field rendering”. In: *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. 1996, pp. 31–42.

[36] Marc Levoy and Turner Whitted. “The use of points as a display primitive”. In: (1985).

[37] Zhengqi Li et al. “Neural scene flow fields for space-time view synthesis of dynamic scenes”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 6498–6508.

[38] Lingjie Liu et al. “Neural sparse voxel fields”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 15651–15663.

[39] Shichen Liu et al. “Soft rasterizer: Differentiable rendering for unsupervised single-view mesh reconstruction”. In: *arXiv preprint arXiv:1901.05567* (2019).

[40] David G Lowe. “Distinctive image features from scale-invariant keypoints”. In: *International journal of computer vision* 60.2 (2004), pp. 91–110.

[41] Ricardo Martin-Brualla et al. “NeRF in the Wild: Neural Radiance Fields for Unconstrained Photo Collections”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2021, pp. 7210–7219.

[42] Ricardo Martin-Brualla et al. “Nerf in the wild: Neural radiance fields for unconstrained photo collections”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 7210–7219.

[43] Nelson Max. “Optical models for direct volume rendering”. In: *IEEE Transactions on Visualization and Computer Graphics* 1.2 (1995), pp. 99–108.

[44] Quan Meng et al. “Gnerf: Gan-based neural radiance field without posed camera”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 6351–6361.

[45] Ben Mildenhall et al. “Local light field fusion: Practical view synthesis with prescriptive sampling guidelines”. In: *ACM Transactions on Graphics (TOG)* 38.4 (2019), pp. 1–14.

[46] Ben Mildenhall et al. “Nerf: Representing scenes as neural radiance fields for view synthesis”. In: *European conference on computer vision*. Springer. 2020, pp. 405–421.

[47] Thomas Neff et al. “DONeRF: Towards Real-Time Rendering of Compact Neural Radiance Fields using Depth Oracle Networks”. In: *Computer Graphics Forum*. Vol. 40. 4. Wiley Online Library. 2021, pp. 45–59.

[48] Michael Niemeyer and Andreas Geiger. “Giraffe: Representing scenes as compositional generative neural feature fields”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 11453–11464.

[49] Hitoshi Nishimura. “Object modeling by distribution function and a method of image generation.” In: *Trans Inst Electron Commun Eng Japan* 68 (1985), p. 718.

[50] Michael Oechsle et al. “Texture fields: Learning texture representations in function space”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 4531–4540.

[51] Jeong Joon Park et al. “Deepsdf: Learning continuous signed distance functions for shape representation”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 165–174.

[52] Mark Pauly and Markus Gross. “Spectral processing of point-sampled geometry”. In: *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. 2001, pp. 379–386.
[53] Francesco Pittaluga et al. “Revealing scenes by inverting structure from motion reconstructions”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 145–154.
[54] Nasim Rahaman et al. “On the spectral bias of neural networks”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 5301–5310.
[55] Ali Rahimi and Benjamin Recht. “Random features for large-scale kernel machines”. In: *Advances in neural information processing systems* 20 (2007).
[56] Konstantinos Rematas et al. “Urban Radiance Fields”. In: *CVPR* (2022).
[57] Darius Rückert, Linus Franke, and Marc Stamminger. “Adop: Approximate differentiable one-pixel point rendering”. In: *arXiv preprint arXiv:2110.06635* (2021).
[58] Shunsuke Saito et al. “Pifu: Pixel-aligned implicit function for high-resolution clothed human digitization”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 2304–2314.
[59] Johannes L Schonberger and Jan-Michael Frahm. “Structure-from-motion revisited”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 4104–4113.
[60] Katja Schwarz et al. “Graf: Generative radiance fields for 3d-aware image synthesis”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 20154–20166.
[61] Kyle Simek. *Dissecting the Camera Matrix, Part 3: The Intrinsic Matrix*. [EB/OL]. https://ksimek.github.io/2013/08/13/intrinsic/ Accessed August 13, 2013.
[62] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014).
[63] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. “Scene representation networks: Continuous 3d-structure-aware neural scene representations”. In: *Advances in Neural Information Processing Systems* 32 (2019).
[64] Vincent Sitzmann et al. “Deepvoxels: Learning persistent 3d feature embeddings”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 2437–2446.
[65] Noah Snavely, Steven M Seitz, and Richard Szeliski. “Photo tourism: exploring photo collections in 3D”. In: *ACM siggraph 2006 papers*. 2006, pp. 835–846.
[66] Matthew Tancik et al. “Block-nerf: Scalable large scene neural view synthesis”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 8248–8258.
[67] Greg Turk and James F O’brien. “Modelling with implicit surfaces that interpolate”. In: *ACM Transactions on Graphics (TOG)* 21.4 (2002), pp. 855–873.
[68] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).
[69] Zhou Wang et al. “Image quality assessment: from error visibility to structural similarity”. In: *IEEE transactions on image processing* 13.4 (2004), pp. 600–612.

[70] Zirui Wang et al. "NeRF–: Neural radiance fields without known camera parameters". In: *arXiv preprint arXiv:2102.07064* (2021).

[71] Suttisak Wizadwongsa et al. "Nex: Real-time view synthesis with neural basis expansion". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 8534–8543.

[72] Jainhua Wu and Leif Kobbelt. "Optimized sub-sampling of point sets for surface splatting". In: *Computer Graphics Forum*. Vol. 23. 3. Wiley Online Library. 2004, pp. 643–652.

[73] Yuanbo Xiangli et al. "BungeeNeRF: Progressive Neural Radiance Field for Extreme Multi-scale Scene Rendering". In: *The European Conference on Computer Vision (ECCV)*. 2022.

[74] Yusheng Xu, Xiaohua Tong, and Uwe Stilla. "Voxel-based representation of 3D point clouds: Methods, applications, and its potential use in the construction industry". In: *Automation in Construction* 126 (2021), p. 103675.

[75] Lior Yariv et al. "Volume rendering of neural implicit surfaces". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 4805–4815.

[76] Jae Shin Yoon et al. "Novel view synthesis of dynamic scenes with globally coherent depths from a monocular camera". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 5336–5345.

[77] Kai Zhang et al. "Nerf++: Analyzing and improving neural radiance fields". In: *arXiv preprint arXiv:2010.07492* (2020).

[78] Richard Zhang et al. "The unreasonable effectiveness of deep features as a perceptual metric". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 586–595.

[79] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. "Open3D: A Modern Library for 3D Data Processing". In: *arXiv:1801.09847* (2018).

[80] Tinghui Zhou et al. "Stereo magnification: Learning view synthesis using multiplane images". In: *arXiv preprint arXiv:1805.09817* (2018).

[81] Siyu Zhu et al. "Very large-scale global sfm by distributed motion averaging". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 4568–4577.

[82] Matthias Zwicker et al. "Surface splatting". In: *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. 2001, pp. 371–378.