Fast generation of Neural Radiance Fields from monocular video sequences

Final Project Report

Author: Michael Ebenstein

Supervisor: Dr. Michael Spratling

Student ID: 1905801

December 2, 2022

Abstract

Over the last few years Neural Radiance Fields (NeRF) have achieved breakthrough results for the tasks of Novel View Synthesis (NVS) and Neural Scene Representation (NSR). This work looks at the main advancements made to NeRFs, specifically on speed improvements during the training process, covering literature up to January 2022. The various methods will be compared on general terms and with a focus on indoor scenes captured with a conventional smart phone, as this represents a real-world use case of NeRFs that goes against some simple assumptions made by existing methods. The main improvements will be evaluated in respect to ray sampling, input encoding, regularisation, network structure and task formulation. The underlying geometric structure represented by different NeRF models is investigated to show which 3D scene features enable fast convergence and which inhibit the optimization process. This analysis is then utilized to implement an architecture that extends the capabilities of a state-of-the-art model (instant-ngp) and is able to represent a challenging monocular RGB sequence within a couple of minutes. Experimental results with different architectures will be provided, with a specific focus on implementation considerations to best utilize GPUs for training.

Originality Avowal

I verify that I am the sole author of this report, except where explicitly stated to the contrary. I grant the right to King's College London to make paper and electronic copies of the submitted work for purposes of marking, plagiarism detection and archival, and to upload a copy of the work to Turnitin or another trusted plagiarism detection service. I confirm this report does not exceed 25,000 words.

Michael Ebenstein

December 2, 2022

Acknowledgements

I would like to thank my supervisor, Dr. Michael Spratling for his guidance throughout this project and for giving me the freedom to explore this area of research in my particular way. I would also like to thank Ayn Rand for her works, which inspire my ambition and diligence, and taught me how to think.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

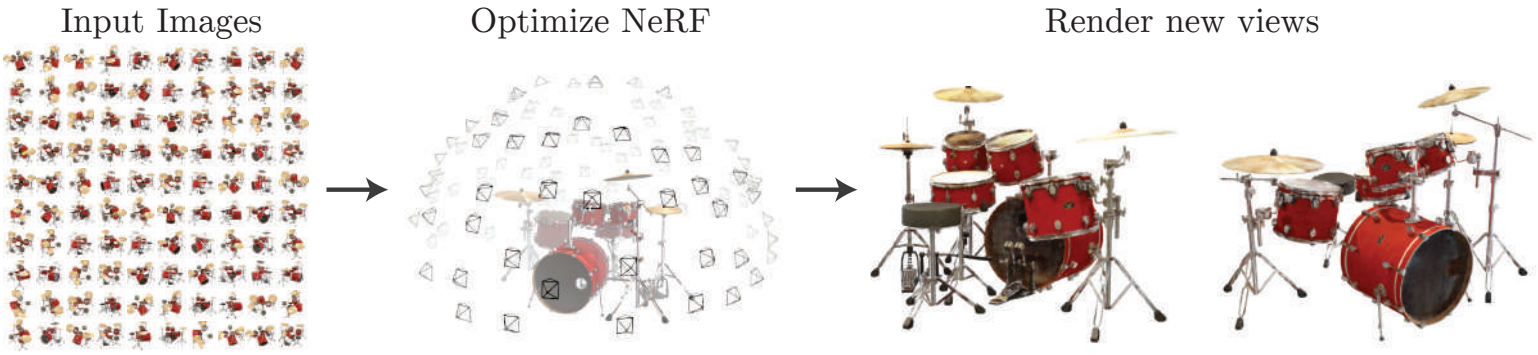

Figure 1.1: Overview of NeRFs capabilities. A set of images is taken from different poses centered on an object. The NeRF is able to represent the underlying scene and synthesize unobserved views. [27]

Representing real 3D scenes digitally has been an active focus of Computer Vision (CV) research over multiple decades. The applications for such a representation cover a wide range, including entertainment related use-cases like Visual Effects, Virtual/Augmented Reality and Video Games, to measurement, design and localization approaches for construction and surveying businesses. Having an accurate digital representation of a real scene via a CV approach saves time and resources, compared to recreating complex scenes manually [26, 52], and enables many downstream tasks that are infeasible otherwise. The rapid advancements of Machine Learning (ML) over the last decade lead to the development of neural scene representations, where a 3D scene is represented by the weights of a neural network. Those approaches often enable generalisable and memory efficient representations of complex 3D geometries. Most recently, an approach called NeRF uses neural networks as an encoding mechanism that allows for photo-realistic renderings of the scene from novel views, while only relying on a sparse set of RGB

frames and camera poses (see Figure 1.1). Since the seminal work by Mildenhall et. al. [27] in 2020, many advancements have made this method suitable to various tasks and constraint settings. Most importantly, the training time for a single scene has been reduce from 1-2 days on a V100 GPU in the original paper to a few seconds on a RTX3090 GPU in the most recent work [29]. The goal of this work is to look into the major insights that enabled this massive speed-up and to understand the trade-offs involved. This knowledge is then utilized to develop an architecture to solve a real-world application: autonomously generating a NeRF from an indoor scene captured on a smartphone camera within a few minutes. This task was chosen as it is not extensively covered in existing literature and achievable within the time and resource bounds for this work.

1.1 Report Structure

Essential background concepts relating to computer vision, 3d reconstruction and NeRFs are covered in chapter 2. This is followed by an extensive literature review in chapter 3, which compares various advancements to the NeRF architecture, with a special focus on train time reduction. Those improvements will be localised within different approaches to ray sampling, input encoding, regularisation, network structure and task formulation. The requirements and design considerations for the target architecture will be enumerated in chapter 4, followed by implementation details in chapter 5. Experiments and comparisons are covered in chapter 6. Finally, results and conclusions of this work are discussed in chapter 8.

Chapter 2

Background

Approaches to 3D scene representation can very widely depending on the requirements of the output and down-stream tasks. Often trade-offs are made between sparse and dense descriptors for geometric consistency, visual quality or performance considerations. The focus for this work will be on methods that only rely on monocular image data, as this data is easily accessible, abundant and can be generated by every person with a smartphone. Furthermore, an infinite sequence of images is theoretically guaranteed to contain all the information necessary to compute an exact, relative representation of an environment, while a representative finite sequence of images can be sufficient to approximate an environment to a desired degree [7].

2.1 3D reconstruction

Many tasks that require 3D scene understanding formulate this problem via Simultaneous Localization and Mapping (SLAM), which estimates the position of an agent and a representation of the environment simultaneously [8]. This framework is mostly used for real-time robotic tasks and focuses on topological and geometric understanding, but can applied to offline tasks as well. The two main challenges with SLAM are finding appropriate models to represent the environment and agent position (that allow for efficient computation) and the selection of features and correlation metrics to perform measurements. For example the ORB-SLAM3 system [3] utilizes ORB feature descriptors [39] to measure geometric relationships between frames and represents the tasks as a 7DoF pose graph optimization problem. Traditional approaches estimate three-dimensional structures using photogrammetric techniques like Structure from Motion (SfM), which relies on motion between consecutive frames to calculate disparities and

infer distances.

Video sequences contain many frames with redundant information or possible visual artifacts (like blur and camera noise) that can lead to inconsistent estimations. For this reason and to simplify problems that are discussed later, it is beneficial to reduce a video sequence to a set of key-frames that depict the essential features of the scene to be represented. Not only does this reduce the number of parameters that need to be optimized in SfM approaches, it also stabilizes the optimization. Park et al. [31] propose an algorithm that first removes frames that depict key-points with a short tracking lifetime, since those are deemed unreliable. Then starting from the first frame, successive key-frames are selected if the ratio between shared key-points falls below a threshold. Another simple approach is running a low-accuracy SfM pose-estimation and selecting frames in key-poses, which are easier to select because of the lower dimensionality (e.g. using pose centroids from k-means could be sufficient).

Many SLAM systems decide to encode the 3D scenes using sparse or dense point-clouds, which are easy to obtain from the descriptors and trackers, but can be challenging to deal with for large and detailed scenes. Depending on the purposes, scene understanding and visualization might be difficult or not sufficiently accurate when relying on point-clouds. Mesh based models are quite rare when it comes to 3D reconstruction, yet the majority of artificial created scenes are encoded as meshes. This is because meshes provide a data efficient way to describe large scenes and are optimized for visualization purposes. While approaches to convert point-clouds to meshes exist, they usually require expensive manual pre- and post-processing to get the desired results, owning to the prevalence of noise in the data from sensors and the complexity of 3D scenes. Another popular approach for encoding real 3D scenes are Signed Distance Functions (SDFs), which implicitly represents the scene in a voxel grid via the distance to the closest surface in each cell. SDFs can capture high-resolution features better than other grid based approaches and make it possible to convert the scene to a mesh via Marching Cubes, yet they usually depend on depth measurements alongside image data and are limited to a single resolution.

2.2 Neural Scene Representation

Because the previously discussed approaches are limited in the resolution and scope of what they can represent, recent efforts have focused on representing continuous 3D shapes implicitly via

deep neural networks [27]. Initial works investigated networks that map from spatial coordinates to SDFs or occupancy fields, which typically are limited to the ground truth synthetic 3D shapes used for training. Subsequent works utilized differentiable rendering methods to implicitly learn 3D shapes from images.

2.2.1 Neural Radiance Fields

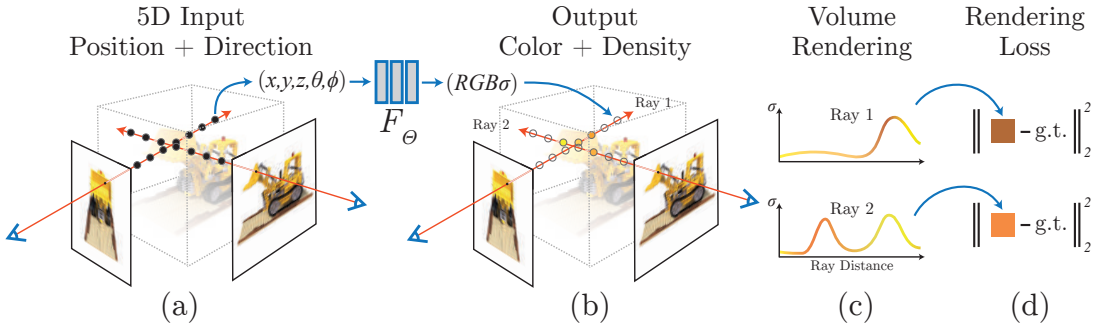

Figure 2.1: Diagram of the NeRF pipeline from the original paper [27]. (a) Points are sampled along rays that pass through associated image locations. (b) Each point yields a luminance and density value that proportionally contribute to the total ray luminance. (c) Color and volume density vary along the ray path. The color values are alpha-blended based on the point density according to equation 2.1. (d) The resulting color is used to establish a pixel-wise L2-loss between the predicted luminance values and the ground truth images.

The color for any given pixel in an image is determined by the properties of the light rays that hit the corresponding sensor element. Assuming the image is focused, the color can be describe by a single light ray from the scene, whose properties are determined by radiance properties of the scene point it reflected from (assuming static global luminance). Thus, any pixel can be described in terms of the scene points that are projected onto it. The main idea of NeRF[27] is to represent a scene by estimating the radiance properties of each point in space, which allows us to generate an representative image from any viewpoint in the scene via ray tracing. This radiance field is estimated with a MLP $F_\Theta : (\mathbf{x}, \mathbf{d}) \rightarrow (\mathbf{c}, \sigma)$ that predicts the radiance properties color $\mathbf{c} = (r, g, b)$ and volume density σ for each coordinate $\mathbf{x} = (x, y, z)$ in space. To allow for view-dependent effects, the radiance is additionally condition on the viewing direction $\mathbf{d} = (\theta, \phi)$. Earlier approaches like Sitzman *et al.* [41] use a similar concept to map from a 3D input to a emitted color, but the viewpoint in-variance limits the visual quality and scene complexity the network can represent. The MLP is designed to predict the density σ independent of the viewing direction, to ensure a globally consistent geometry is represented (see Figure 2.2 for more details on the exact network topology).

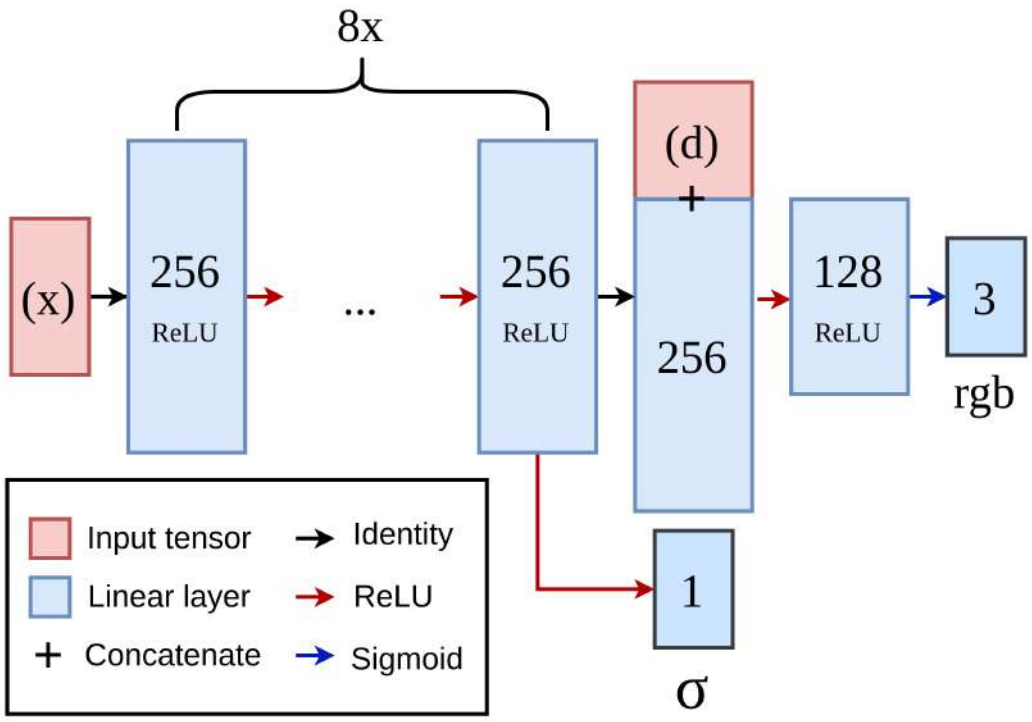

Figure 2.2: NeRF MLP overview. The network consists of 8 blocks of linear layers with ReLU activations that predict the point density σ and a feature vector of size 256. This vector is concatenated with the viewing direction d and passed through additional linear layers to yield the radiance values. To prevent vanishing gradients, sometimes the positional input is injected into the 4-th layer as well.

To render the color for a pixel with corresponding coordinate o and viewing direction d, where $o - d$ equals the camera position, the radiance properties along the ray $r(t) = o + t * d$ are accumulated according to the volume rendering equation

$$\hat{C}(r) = \int_{t_n}^{t_f} T(t)\sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), \mathbf{d})\ dt, \text{where } T(t) = e^{-\int_{t_n}^{t} \sigma(\mathbf{r}(s))\ ds} \tag{2.1}$$

where t_n, t_f define the near and far bounds. In practice $n \geq 64$ random samples are taken within this integral to approximate the pixel color. The function $T(t)$ is often used for early-ray termination, as radiance values that contribute less than a threshold $\hat{t} < 1e - 4$ do not visually effect the final colors. The existing components also allow us to calculate the distance $\hat{d}$ between the observer and the scene point via [30]

$$\hat{d}(r) = \int_{t_n}^{t_f} T(t)\sigma(\mathbf{r}(t))t\ dt$$

The main task of NeRF is to learn a set of parameters Θ, such that the pixel-wise loss

between any reference frame pixel and it's corresponding predicted color is minimized

$$L = \left\| \hat{C}(r) - C_i(r) \right\|_2^2$$

The parameters Θ are optimized via backpropagation by predicting the colors for all pixels in each reference frame. By training the network for a long time, it is able to learn a globally consistent scene geometry and radiance, which enables us to render arbitrary views of the scene. See Figure 2.1 for an overview of the whole pipeline. A main component that allows the MLP to recover the scene geometry is that many of the points that are queried during training are used to render different target views. This requires the network to find a solution that is consistent across multiple views and thus leads the NeRF architecture to learn globally consistent geometries (other important aspects of the architecture that explain the properties of NeRF are discussed in chapter 3). Although neural networks are universal function approximators, they often struggle to represent high-frequency features and are biased toward low-frequency functions [27]. To counterweight this, the NeRF inputs are mapped to higher dimensional positional encodings using high frequency functions (sinusoidal encodings are used by default, other possible encodings are covered in subsection 3.4.1) Because of the implicit nature of the representation and large feature space of the architecture, it is quite expensive to optimize the network for a single scene. Many tricks are utilized to speed up training and inference of the network. In the original paper two networks are trained, one with a coarse resolution and one with a fine resolution to facilitate hierarchical volume sampling (more details in section 3.2. The networks are used to reduce the number of queries necessary by first querying the coarse network and then performing informed sampling via the fine network on regions of interest. The final optimization process for a single scenes took the authors around 1-2 days on a NVIDIA V100 GPU.

Chapter 3

NeRF advancements

In this chapter we will focus on advancements on specific aspects of the NeRF architecture that lead to improved training and rendering times. Common advancements relating to dynamic or large scenes and specific visual feature improvements will not be covered explicitly.

3.1 Network Architecture

Foreground and Background split

A single MLP can only represent a finite amount of information and thus larger or complex scenes are challenging to encode. Most work on NeRF is focused on representing single objects that are captured on equidistant points on a hemisphere. For real world applications we often want to represent large scenes that are captured from a variety of different viewpoints, which requires the network to trade-off the quality between different parts of the scene. Scenes that cover a large range of depth values, especially when captured with 360° cameras, force the network to allocate the same information resolution to background and foreground information to achieve good results, resulting in an overall loss of image quality. In NeRf++ [51] this issue is resolved by splitting the scene into foreground and background, defined by a unit-sphere, and training two separate networks to focus on either of them. Since NeRF integrates over Euclidean depth, the values are only well-approximated for close ranges and the quality drastically deteriorates on large depth ranges. This means that we can model the foreground well with existing models, but a different approach is necessary for the background regions. The authors propose an inverted sphere parameterization $(x', y', z', 1/r)$ that represents each query point as 4-tuple of consisting of a point on the unit sphere (x', y', z') and a distance

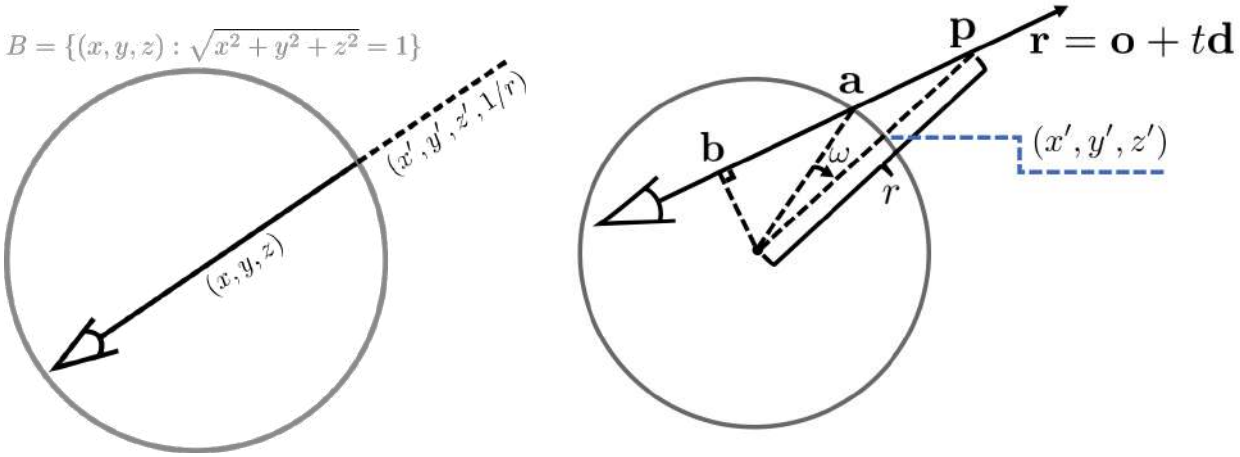

Figure 3.1: (left) Different parameterizations are used inside and outside the unit foreground sphere. [51]
(right) Outside the inverse sphere parameterizations is used by calculating the point (x', y', z') that multiplied by r equals the query point p along the ray $\mathbf{r} = \mathbf{o} + t\mathbf{d}$.

r. The model receives the inverse of the distance r, meaning that all values are bounded $x', y', z' \in [-1, 1], 1/r \in [0, 1]$, which leads to increased numerical stability and has the side-effect of decreasing the information resolution of points that are farther away. As illustrated in Figure 3.1, the inverse parameters are calculated as follows:

$$t_b = -\frac{\sum_i (o * d)_i}{\sum_i d_i^2} \tag{3.1}$$

$$b = o + t_b * d \tag{3.2}$$

$$\tag{3.3}$$

$$t_a = t_b + \frac{\sqrt{1 - |b|^2}}{|d|} \tag{3.4}$$

$$a = o + t_a * d \tag{3.5}$$

$$\tag{3.6}$$

$$\omega = \arcsin b - \arcsin\left(|b| * t\right) \tag{3.7}$$

$$k = o \times a \tag{3.8}$$

$$v = a * \cos\omega + (\hat{k} \times a) * \sin\omega + \tag{3.9}$$

$$\hat{k} * (1 - \cos\omega) * \sum_i (\hat{k} * a)_i \tag{3.10}$$

$$(x', y', z') = \hat{v} \tag{3.11}$$

$$1/r = \frac{|d|}{t} * \sqrt{1 - (|b| * t)^2} + t_b \tag{3.12}$$

, where ˆ indicates a unit vector and v is the vector a rotated by ω degrees using the Rodrigues rotation formula. While this approach does not directly improve performance, it does not penalize it either, since the two networks train on separate parts of the scene, unlike the two hierarchical networks in NeRF, but the increased numerical stability and specialization of each network can lead to faster overall convergence.

Scene partitioning

KiloNeRF [36] aims to directly improve the performance of training and inference by employing thousands of small MLPs that are associated with a scene voxel. Points are queried via the MLP assigned to the voxel they fall into. Since each MLP is only responsible for a small part of the scene, each network requires far less parameters to represent it. This means that each network itself is exponentially easier to optimize than the large MLP, but also that the optimization problem itself is easier. By monitoring the use of each network one can also prune the final representation by removing networks that only represent empty space. One drawback of this method is that it only achieves state-of-the-art results by employing a separate NeRF as a trainer network, since otherwise artifacts are generated in empty space and between voxel boundaries. MegaNeRF [43] combines the two previous approaches and partitions each scene into multiple NeRFs, but uses a foreground and background network for each part using inverted sphere parameterization. The networks are partition regularly along a 2D plane and again are used by the query points they are closest to. One difference to the sphere parameterization of NeRf++ is that here they use a bounding ellipsoid to more tightly enclose the foreground content. In DeRF [35] the computational complexity of the problem is also reduced by employing a number of smaller MLPs. But instead of partitioning them in regular intervals as in previous approaches, a parameterized Voronoi Diagram is learned to assign each network to a region in space with roughly the same information content. Yang et al. [48] propose Recursive-NeRF

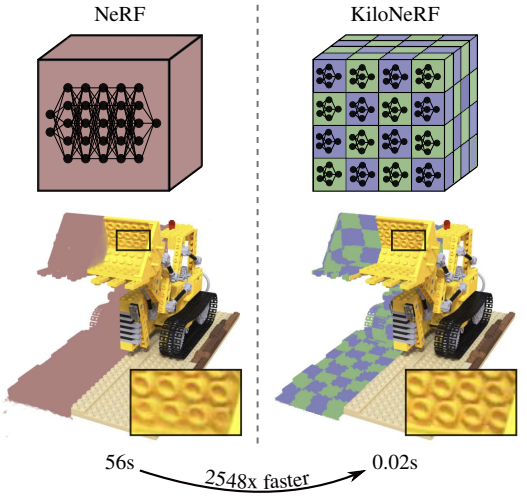

Figure 3.2: (left) Usually a single MLP is responsible for representing the whole scene. (right) In KiloNerf each MLP is responsible for a small section of the scene. [36]

where a recursive structure of MLPs are utilized to reduce the computation cost and focus each module on specific scene aspects. They propose MLP-modules that return color c, density σ, uncertainty δ and a feature vector y. The outputs of each module are recorded and periodically k cluster centers of the δ values are determined via K-means to create k sub-modules. All further predictions that are above a threshold of uncertainty δ will be assigned to one of the k sub-modules and further processed, using the previous outputs and the feature vector as an input. Thus, more complex scene objects go deeper through the architecture and simple objects or empty spaces should terminate early.

3.2 Ray sampling

Since the whole NeRF architecture is based around ray sampling, the way rays are sampled has a major influence on the models rendering quality and training time. The original NeRF architecture reduces the number of ray samples by using a *coarse* and *fine* NeRF network, which requires two separate networks to be trained, where the *coarse* network accounts for 25% of the computation cost. In NeuSample [9], the authors replace the coarse network with a *sample field network* that returns $N = 192$ sample points for a given ray and thus only requires one forward pass per ray. Lindell et al. [23] propose that one can transform a MLP which is sampled to approximate an integral into a different network, that shares the same parameters but has a vastly different architecture. This second *Grad* network can be fitted to the signal and be transformed back into the original network, which now represents a definite integral over the signal space. Thus, an integral does not need to be numerically approximated, but can be definitely calculated using two evaluations of the network. Because the volume rendering formula includes a nested

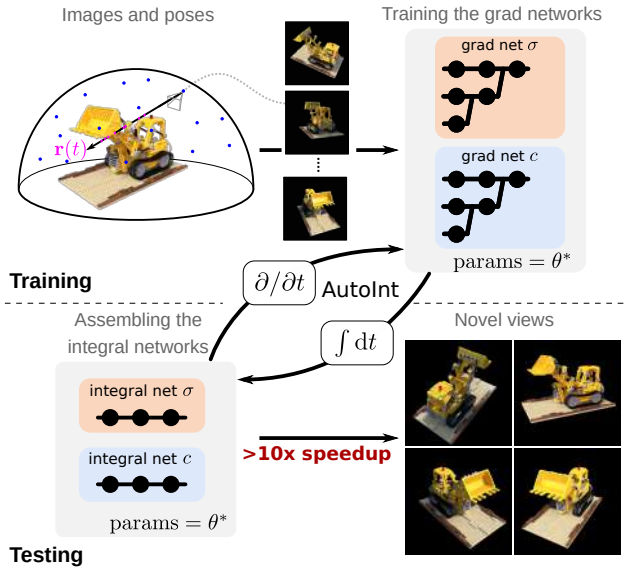

Figure 3.3: Illustration of the AutoInt architecture. (top) A MLP is trained on a set of images using conventional methods. (bottom) The trained parameters are converted to an integral network during test, which can evaluate ray integrals much faster. [23]

integral, two evaluations are not sufficient, but the rays can instead be piece-wise approximated by integrals, and thus AutoInt requires a smaller number of samples along each ray than NeRF. Since many network evaluations are wasted on querying points in empty space, VaxNerf [20] utilizes a visual hull to prevent queries outside objects. Creating visual hulls is easy and efficient and also guaranteed to only undercut spaces and thus never removes important points of the scenes. This drastically reduces the number of query points and removes the need for a *coarse* network, enabling small scenes to be photo-realistically represented in 30 minutes. The only caveat for my purposes is that VaxNerf only focuses on scenes with single objects and thus more advanced modifications might be necessary for scenes with many objects and difficult foreground/background separations. Lin et al. [22] utilize a CNN to construct a 3D cost volume, which is used by a 3D CNN to predict a range of depth values for each pixel. This enables the architecture to only sample rays around the surface of objects while only evaluating the CNNs once per image. Training time is also reduced by the fact that CNNs are less expensive to train and compute when compared to an additional MLP.

3.3 Regularization

The default NeRF architecture requires 10-100 images per scene to accurately represent the scene geometry and allow for realistic renderings from unobserved viewpoints. Yet many indoor scenes can be sufficiently captured using a much smaller number of images and still provide enough information to infer the scene geometry. Not only would this reduce the optimization complexity and should lead to a reduce training time, it also removes scene ambiguities that would be introduced by camera distortions/artifacts and other capture inaccuracies. Jain et al. [15] observe that training a NeRF representation with very few samples leads to degenerate solutions at unobserved viewpoints because of the lack of geometric constrains. Although this certainly is a problem, Zhan et al. [51] show that the NeRF architecture in general avoids highly degenerate solutions and favours consistent and accurate geometries. They argue that in theory the network could represent each scene with arbitrarily degenerate geometries, such as a unit sphere, and still yield color values that would perfectly match the input views. But this would require the network to encode color as a high-frequency function and due to the limited representational capacity of the MLP, solutions that are low-frequency are much preferred. Also the injection of the view direction close to the end of the network acts as a smoothness prior on the color features and prevents high-frequency representations based on the viewing direction, which would be required to counterweight degenerate geometries. Thus, NeRF in general is

likely to converge to smooth geometries, but further constraints are necessary to achieve high geometric accuracy when using a small set of input views.

Unobserved views

In the DietNeRF model [15], the authors sample unobserved viewpoints during training and instead of using a visual loss, as for the input views, they apply a CLIP [34] model to the renderings and ensure that the predicted semantics labels are similar to the ones observed in the input views. Yu et al. [49] proposed PixelNeRF, which aims to constrain the problem further by not only relying on local and individual pixel features for each input view, but by utilizing semantic image features and sharing them between the views for improved geometric consistency. This is done by associating a semantic embedding with each pixel per input view. When rendering a ray, the sample points are projected back into the input views and the appropriate embeddings are retrieved and added to the MLP for context information. The network is evaluated once for each available input view and the resulting tuples are averaged to yield the final color and density for each point. Although this approach is complex and increases the training time, it is generalizable (see the following section) and thus offers a trade-off between geometric consistency and per-scene training time. IBRNet [46] use a similar approach where for each query point the projected feature embeddings in nearby source views are retrieved and fed through a Ray Transformer to retrieve the densities. This ensure that semantic contexts are shared among views and the transformer encodes smooth density distributions along the rays. A separate MLP is used to retrieve the colors for each ray.

Depth conditioning

Other papers [5, 37, 47] make use of depth values that are estimated during the calculation of image poses via SfM or monocular depth estimation networks to shrink the problem space. The depth values are used to constrain the depth maps generated by NeRF and thus ensure a view consistent geometry. In RegNerf [30] the authors use image independent regularization metrics to constrain NeRF. For geometric consistency they assume that depth maps are piecewise smooth and thus add a depth smoothness loss to the model. Although most issues get resolved by representing an accurate geometry, they further add a color regularization loss for unobserved view, where they maximize the log-likelihoods given by a RealNVP [6] normalizing flow model and thus encourage realistic image creation.

3.4 Task formulation

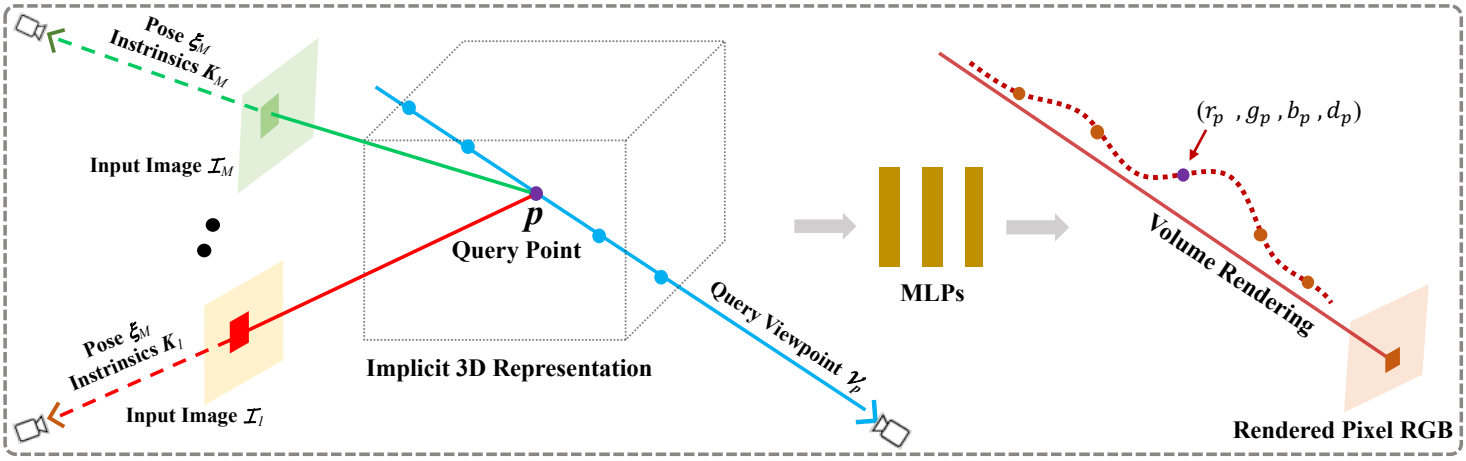

Figure 3.4: Diagram of the GRF pipeline [42]. Each query point p is projected into the input frames and according image features are retrieved. Those features are then used within the MLP to predict the radiance and density and thus is conditioned on the input views.

A significant bottleneck of NeRF is that each scene is learned from scratch and weights cannot be re-purposed, although most scenes share essential characteristics that the model should be able to reuse. Trevithick et al. [42] introduced the General Radiance Field (GRF) that learns a general representation of similar scenes and is only conditioned on the explicit scene geometry and content during rendering of each frame. This is done by making each query point contingent on the input image features in addition to location and view angle. A query point p is projected into an input image I_m and the image features associated with the projected point are retrieved (see Figure 3.5). Instead of using raw RGB values, more robust and global features are extracted from each image via a CNN encoder-decoder and associated with each pixel. The CNN itself receives the RGB values stacked with the image view-point to allow the network to reason about global structures and relative positions. Since the number of input views is variable per scene, the input features are combined using an attention mechanism. To allow the network to distinguish between different query points that lead to the same projected feature vectors, the initial features are concatenated with the query point p and transformed by a separate MLP. This combined with the attention mechanism allows the architecture to weight the influences of different views and handle ambiguities arising from occlusions. The complexity of the architecture and addition of multiple neural networks significantly increases the training time to achieve state-of-the-art results. But since this model is generalized, only a short amount of time is necessary to fine-tine the network on each scene to achieve comparable results to scene specific networks. For this reason, GRF enables a drastic time improvement

for the purposes of this project as the majority of computations can be done before the actual application of the model.

PixelNeRF [49] offers a similar approach for generalized representations. Instead of an encoder-decoder, a ResNet34 is utilized [12] for the pixel-wise features and instead of retrieving the features closest to the projected query point, bi-linear interpolation is used between the closest feature vectors. When conditioning the rendering on a single input view, the retrieved image features are added as a residual to the first few layers (denoted f_1) instead of as a network input, which reduces the network complexity. To incorporate multiple views, f_1 is evaluated on each view independently and the outputs are combined via an averaging operation to be fed through the last 2 layers of the full network, which yields a single color and density value. Since this method can represents scenes accurately from very few input views and fewer, less complex networks are utilized, shorter training and rendering times are to be expected in comparison to GRF. IBRNet [46] allows for generalized representations, but as mentioned before utilizes a Ray Transformer to query each sample point. Another difference is the way re-projected image features are aggregated. Each feature is concatenated with the mean and variance of all current features and transformed by an MLP to yield features that are conditioned on global and local structures alongside a weight. The weighted average of those features is again transformed by a different MLP to yield density features for the Ray Transformer. In parallel, the transformed features are used to predict weights for the associated pixel values of the input views, which are averaged to yield the final color. Since previous approaches are quite expensive to train and slow to render, NeuralMVS [38] promises to achieve significant performance improvements and quality renderings with little fine-tuning by incorporating Multi-View Stereo methods. Here input view features are aggregated by extracting the mean and variance using barycentric weights. Besides reducing the feature complexity, another big time improvement is made by predicting depth piece-wise via an LSTM, which allows to drastically reduce the number of queries made per ray (to 18 in NeuralMVS) as the depth is used to guide the sampling points. The final query point is considered to be the object surface is re-projected to the input views, whose color and feature values transformed like in IBRNet, while only utilizing MLPs. Thus only surface points require expensive evaluations, while intermittent query points are reduced and in-expensive to compute.

3.4.1 Input Encodings

Choosing appropriate input encodings is an essential aspect of developing a neural network system and is related to feature engineering in data science [13]. It defines how the network "sees" the data and thus how complex the relations it has to learn are, which in turn heavily influences how quickly a network converges to the desired state. The goal of an input encoding should be to reduce the problem complexity as much as possible, while providing enough information to enable the network to navigate the feature space appropriately.

Sinusoidal Encoding (PE) Mildenhall et. al. [27] noticed that the MLP struggles to endistinguish between the different query points with sufficient accuracy and instead map the coordinates x to a representation with higher dimensionality and frequencies. They chose a sinusoidal encoding that maps a 1-dimensional variable p to an N-dimensional space via

$$\gamma(p) = (\sin(2^0\pi p), \cos(2^0\pi p), \ldots, \sin(2^{L-1}\pi p), \cos(2^{L-1}\pi p))$$

, where $N = 2 * L$. $L = 10$ is chosen to encode each coordinate value, and $L = 4$ is used for the viewing direction. This change alone lead to an 8% improvement of the image accuracy. This input encoding was used as it proved successful for other architectures, most notably Transformers [45].

Integrated Positional Encoding Barron et al. [2] point out that the NeRFs sampling strategy limits the representation to a single scale and does not allow the network to reason about scale differences. This means the representation quality significantly degrades when training on multi-scale input images and NeRF in general struggles with accurately depicting visual features at unseen scales. The lack of scale information also means that ambiguities between different scale images cannot be resolved and the learned geometry might be inconsistent. The authors propose mip-NeRF, where instead of casting a single ray for each pixel, they cast a cone and integrate over the conic volume. In practice the only difference the positional encoding for the query points. They propose a new positional encoding *"integrated positional encoding"* (IPE), that represents a given conic frustum as a Gaussian of expected positions within it and allows the network to reason about the size and volume of the sample cones. Besides significantly improving the multi-scale accuracy of the network, this features also removes the need for a *coarse* network, since the single mip-NeRF model can be queried at different scales to guide the cone sampling and thus leads to faster training times.

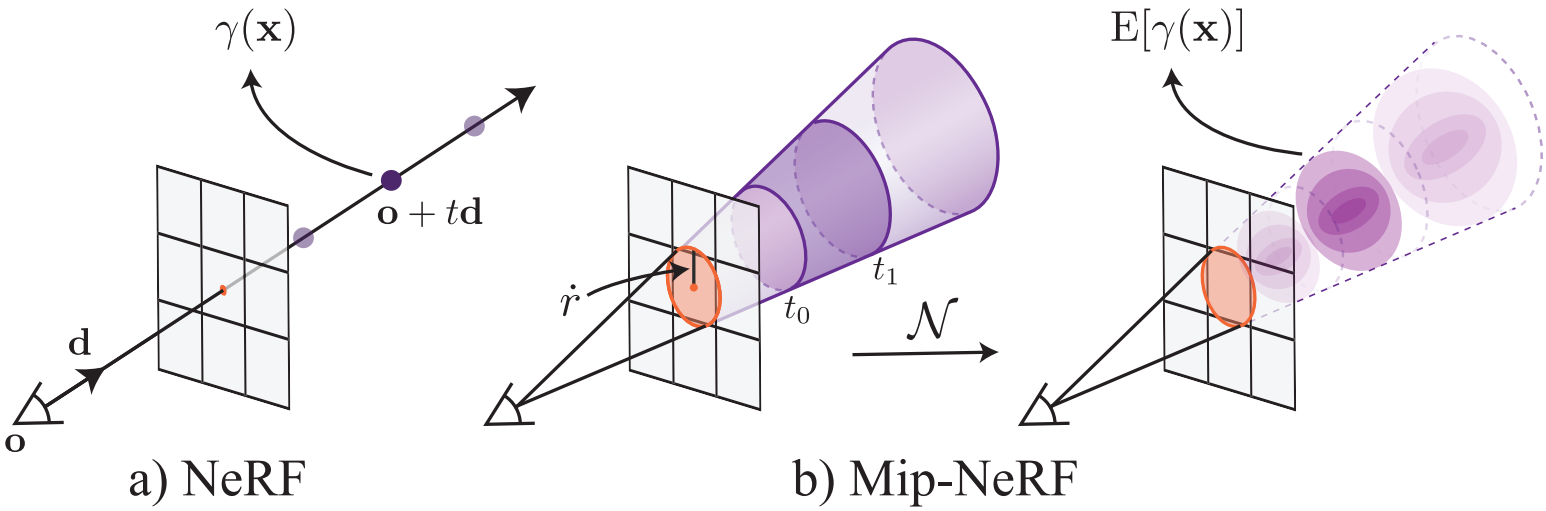

Figure 3.5: Comparison of the NeRF input encoding (a) and the IPE. [2]

Multi-resolution Hash Encoding Müller et. al. [29] extend existing ideas used for input encodings and network architecture to implement a Multi-resolution Hash Encoding (MHE) (also referred to as instant-ngp throughout this work). Initially NeRF predicted radiance in a structure-less setting, by only relying on continuous query coordinates and viewing direction. It quickly became apparent that combining NeRF with traditional structured approaches can drastically reduce the optimization complexity and lead to a much faster convergence. KiloNerf [36] achieved great speedups by incorporating a spatial structure into the architecture, which reduced computational complexity of each evaluation step and allowed each part to specialize on a sub-set of the scene. Similar approaches have been implemented with respect to input encodings [4, 17, 24, 33], where learned embedding vectors are associated with spatial coordinates and interpolated according to the relative position of the query point x. This has a similar effect to KiloNeRF, as local scene information can be separated and specialized, while the MLP needs less trainable parameters.

The trainable parameters are associated with voxel corners in a regular voxel grid over a specified volume. The grid resolution provides an easy mechanism to trade-off computational complexity and memory consumption. Early implementations relied on a single dense grid, which is easy to implement, but has a larger memory footprint and computational cost than multi-resolution grids, which allow to separate global and local information better encoding redundant information. The MHE stores the parameters at the voxel corners of an L-dimensional grid. The grid resolutions follow an geometric progression $N_l = \lfloor N_{min} * b^l \rfloor$, where $N_{min} = 16$ and $b = 1.38$. Each level needs to store parameters for $t_l = N_l^3$ coordinates.

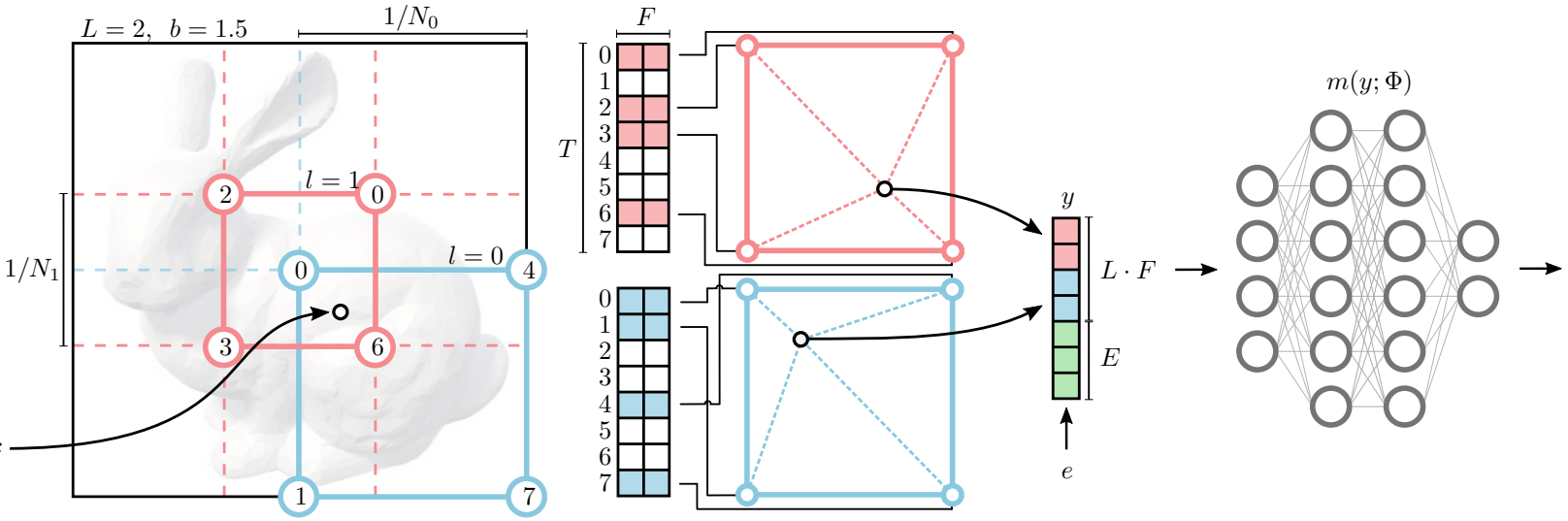

Figure 3.6: Illustration of the MHE [29]. For each query point the surrounding voxel corners for each level are calculated (1) and the parameters retrieved using a spatial hashing function (2), interpolated according to the relative position within the voxel (3) and concatenated to form the final input (4) that is fed to the network (5).

l	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
N_l	16	22	30	42	58	80	110	152	210	290	400	553	763	1053	1453	2005
t_l as 2^k	12	13	15	16	18	19	20	22	23	25	26	27	29	30	32	33

Table 3.1: This table shows the resolution of each grid along with the number of unique voxel corners, shown as the closest power of 2.

As the number of voxel corners increases quickly, the authors limit the number of stored parameters per level to 2^{14}. This makes the it easier to experiment with different grid resolutions and limits the memory footprint, while not impacting the quality of the model too much. This is because high grid densities will likely store redundant information, yet they are necessary to allow for highly localized information. This insight if often used to represent scenes using octrees or other structures that are adapted to the scene geometry. Since NeRF starts without any knowledge of the scene geometry, it is not possible select an a prior structure that assign a sufficient number of parameters to high-frequency scene areas and few parameters to low-frequency areas or empty space. Some approaches periodically evaluate the network during training to extract an approximate scene geometry and adapt the architecture accordingly. For example MegaNerf [43] uses the information to assign models to approximately equal areas of information. A main component of MHE is the spatial hashing function which allows the encoding to quickly retrieve the voxel any point x belongs to.

$$h(x) = \left(\bigoplus_{i=1}^{d} x_i \pi_i \right) \bmod T$$

which maps any coordinate to a array index via the bit-wise XOR operation with 3 prime numbers π_i. The network has to learn to disambiguate hash collisions and thus is has the freedom to interpret the data in a way that optimizes information entropy.

Chapter 4

Design & Specification

This section will layout the project specifications and show various design iterations that happened throughout this project.

4.1 Specification

No.	Requirement	Description	Specification
0	Represent a video sequence as a NeRF	The final software must be able to represent a small indoor scene with multiple objects as captured by a normal camera (e.g. phone camera). This representation must be able to accurately depict the scene to near photo-realistic quality from any location within the view frustum of the video sequence.	Given a MKV or MP4 video sequence, the model must be able to represent the the scene such that it can return a rendering of the scene given any position p within the view frustum. The representation has to minimize the distance between the renderings and the frames within the video sequence.
1	Estimate camera parameters	Given a video sequence, the model must be able to estimate the shared intrinsic parameters of the camera model and estimate extrinsic parameters for each frame.	The model can assume minimal camera distortions and should rely on a pinhole model to estimate the mentioned parameters.

2	Select key-frames	A given scene should be well represented from a small number of frames that cover the majority of scene features. The model should select the smallest number of frames sufficient to represent the scene. This helps reduce scene inconsistencies and simplifies the model convergence.	Between 1-32 frames should be selected to represent a small indoor scene. The input videos are expected to move slowly through the space and thus frames within a small time-interval are usually safe to be pruned. Furthermore, scene content that is located near the image centers is prioritized over other features, which do not necessarily need to be captured by the key-frames.
3	Minimal training time	The training time for a given scene should be as low as possible, while still achieving sufficient visual quality (see requirement 5).	No scene should take longer than 1h to train on the available hardware (RTX 3080m)
4	Reasonable rendering time	The rendering time for a view of the scene should be low enough to enable scene captures within a couple of minutes.	A minimum of 0.5 FPS is required.
5	Sufficient visual quality	The resulting scene renderings should be close to photo-realistic from views near the input frames.	The reference quality should be above 19 PSNR.
6	Ease of creating	An end-user should be able to create a representation from their phone without doing more than recording a video	Besides only requiring a video sequence as an input, the model should directly receive the video input from the users phone.
7	Ease of viewing	An end-user should be able to view the scene from any location by just supplying a view coordinate, view angle and image size. Optionally it should be possible to view the representation on a phone.	A representation on the phone should not require an active internet connection and render images within a minute.

To train a NeRF on a video sequence in a reasonable amount of time requires combining many of the aforementioned advancements. One big challenge is that user-friendly video inputs will have various characteristics that are difficult to deal with:

- Blurry images: This can be caused by motion blur and de-focus blur, both of which are common for normal smartphone users
- Bad lighting: Many scene that users might want to capture do not have appropriate lighting conditions, which can lead to grainy images and poor contrasts. Advanced users can adjust the exposure time of their camera, but they have to chose between better image quality and added motion blur or capture complexity by having to move the camera more deliberate.
- Concave scene captures: When recording a scene, especially indoors, it is often easier to capture large parts of it without moving, by just panning the camera. Most literature on NeRF focuses on scenes (both virtual and real) captured from a set of convex poses that all point to the scene centre (see Figure 4.1 for an illustration of the different pose sets). Convex poses are preferable since they require larger movement between frames, which provides more depth information, and ensures that there is a substantive overlap of the content captured from each pose. This not only helps SfM algorithms to estimate better poses and find matching descriptors, but also constrains NeRF by providing more overlapping query points. If scenes are not captured with precision equipment or virtual cameras, it is quite difficult to position the camera in such a way. It requires the camera operator to keep the camera pointed to a central location, which is made more difficult by the necessary larger movements through the scene. Concave captures on the other hand are much easier for a user to perform, since they only require them to pan slow enough to capture some non-blurry frames. In contrast to convex captures, this method provides only little overlap between image frames (and usually only on the image margins), and reveals minimal depth information. As this work is not focusing on 360° video captures, it is assumed that the input sequences will contain a mix of concave and convex poses of varying magnitude.

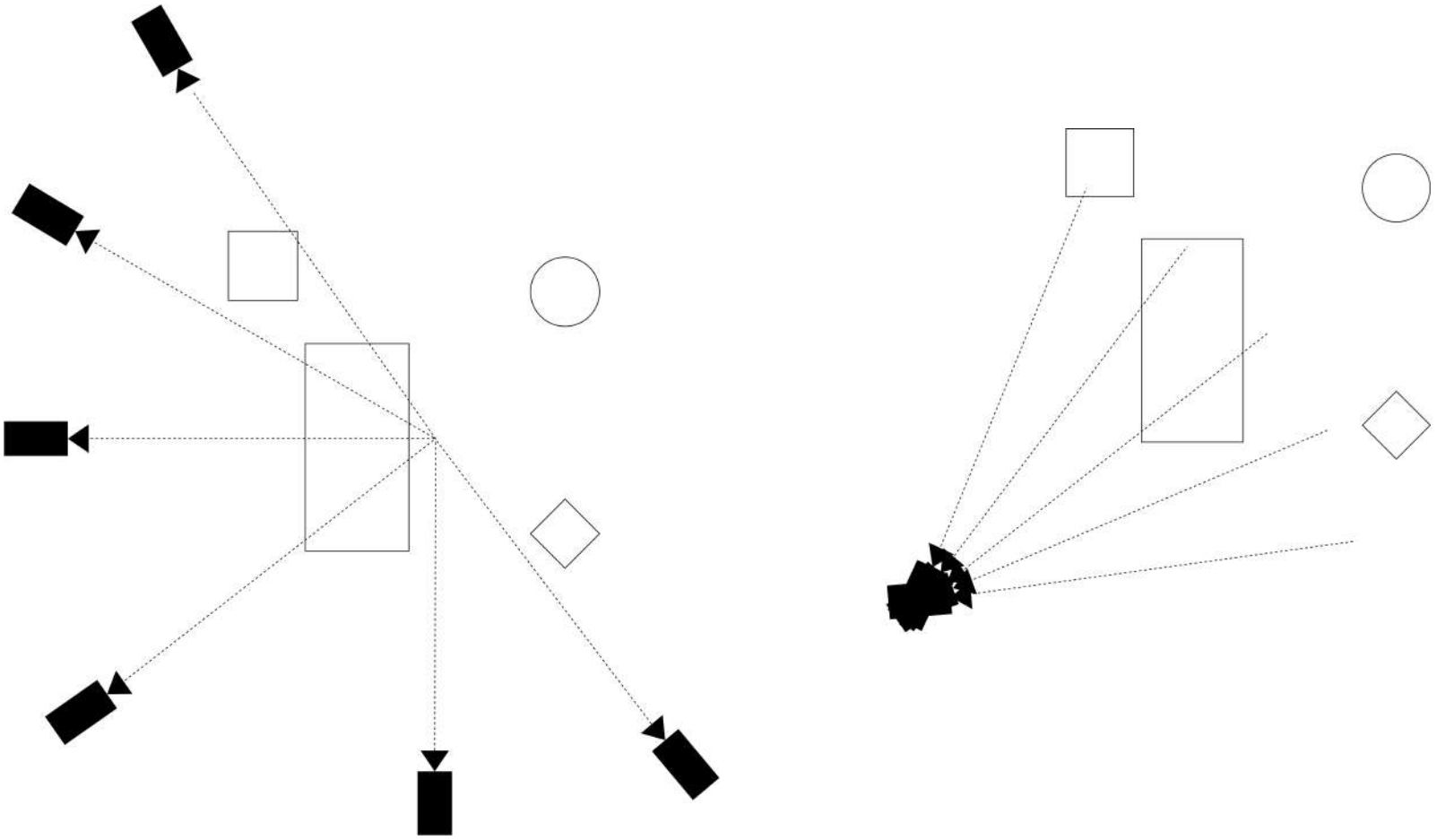

Figure 4.1: (left) Diagram of a scene captured from a convex set of poses. (right) The same scene captured from a concave set of poses.

4.2 Design

Initial design

The proposed architecture visualized in Figure 4.2 is designed to address all the requirements stated above. Non-general NeRF models currently require many hours of training to converge to a reasonable quality and thus are not suitable for this project. To achieve fast training times, a general NeRF will be utilized and combined with efficient ray sampling methods discussed above. For faster and more accurate convergence, additional optimization constraints will be added.

Key-frame selection will first focus on removing bad frames that likely lead to inconsistencies and sampling a small number of frames using methods discussed above. The estimation of

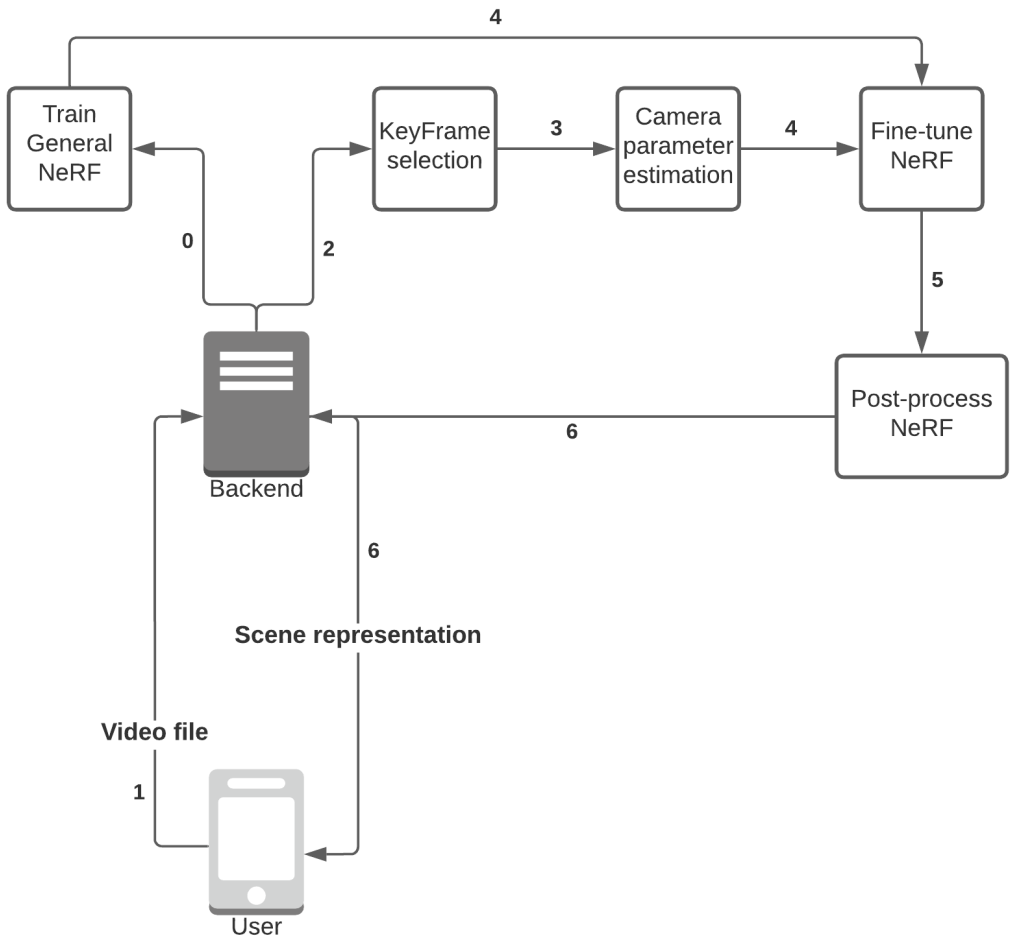

Figure 4.2: Overview of the architecture. 0) A general NeRF will be pre-trained on many indoor scenes. 1) A user records a video via an App that sends the file to a back-end server. 2) The server selects a couple of key-frames that depict the scene. 3) The key-frames are used to estimate the camera parameters using SfM. 4) The pre-trained general NeRF will be fine-tuned for a short period to increase the image quality. 5) The resulting NeRF will be post-processed to reduce the render time for each frame. 6) The final model will be send back to the user for their purposes.

camera parameters will be done via COLMAP, as this is an easy state-of-the-art method that is utilized by most researchers. Post-processing of the NeRF will be limited to simple sampling optimizations like hulling and depth-guided sampling, depending on the final architecture, but will not cover rasterisation approaches or any methods that require a drastic amount of processing or memory. For the end-user a simple Flutter based application will be implemented that just uploads a video file to a Flask back-end server that does the processing. Phone-based visualization methods will only be explored once all other requirements are implemented.

Final design

During the project run-time, a couple of drastic changes were made to the original design which were driven by significant advancements made with different architectures as well as time-constraints. The key change came after instant-ngp [29] was released in January 2022 and proved that scenes can be learned from scratch within a few seconds. Not only does this architecture solve major challenges my design would have to face, it also allowed for rapid experimentation and design iterations, whereas training a generalisable NeRF would require days of training. After initial experiments it became apparent that this architecture does not work out-of-the-box on the data domain this project targets. It was decided to focus on analysing this new architecture, understanding what it's weaknesses are and how it can be adapted to work with phone videos. This means that the pre-training and fine-tuning step are merged to a single training processes, all other parts being equal to the original design. Due to time-constraints the back-end and front-end interfaces were not implemented and instead a pipeline that converts MP4 videos to a NeRF was created.

Chapter 5

Implementation

Most components in this work were implemented using Python[44] and PyTorch[32] since many existing works rely on those frameworks and other frameworks such as Tensorflow or JAX only provide an advantage when working with specialized hardware like TPUs. All experiments were run using a laptop with a RTX 3080m GPU with 16GB of memory. NeRF requires a huge amount of computation, so efficient implementations are paramount to achieve state-of-the-art results [29]. The following aspects were emphasized in the implementation to maximize efficiency

- Half-precision floating-point compatibility: using 16 bit floats vastly reduces the memory footprint and computation cost. An alternative is using BFloat16 [18], but the reduced accuracy limits the representation quality. Instead, all values are normalized to fit within the bounds of 16 bit floats.
- TorchScript Just-in-time compatibility: PyTorch provides a JiT mechanism that allows one to implement very efficient operations. One big advantage is that GPU operation can be merged into a single kernel invocation, which vastly reduces the operation latency. Furthermore, PyTorch provides a concurrency mechanism for TorchScript modules that enables implementing parallel training mechanisms.
- Data pre-loading: Data allocation operations and data transfers are the largest bottlenecks in machine learning pipelines [14]. Pre-loading data into memory cache optimized operations can vastly speed up the training process [29].

This work heavily relies on instant-ngp, which is implemented in C++ and CUDA to achieve optimal efficiency. While this implementation is very well suited for using the model in it's cur-

rent form, it is not suitable for research purposes. Working with C++ not only requires more difficult implementations and re-compilation (which slow down the development process), but also restricts access to existing machine learning frameworks that are targeted for Python. Especially when experimenting with custom loss functions, implementations in CUDA require explicit implementations of gradients for the backpropagation step, which raises the implementation difficulty and possible bug rates. The instant-ngp implementation relies on a library [28] by the same authors that provides fast implementations of various input encodings and fully-fused neural networks (a MLP where all operations are done within a single GPU kernel invocation), while also providing PyTorch bindings. Using this library makes it possible to experiment with instant-ngp without loosing access to the existing infrastructure or incurring too heavy performance losses. An already existing implementation (torch-ngp [1]) that combines instant-ngp with PyTorch was adapted for this project, since it also provides crucial early ray termination mechanisms.

Torch-ngp does not support gradients for the depth values, which is paramount for the regularizations that are applied in this work. For a given prediction $\{\sigma_i, r_i, g_i, b_i\}$ with accumulated values up to k of n samples $\{\Sigma_k, R_k, G_k B_k\}$, ray distance t_i and loss gradients $\{\sigma_i^\nabla, r_i^\nabla, g_i^\nabla, b_i^\nabla\}$, the gradient is defined as:

$$
\begin{aligned}
\nabla\sigma_i = t_i * (& \\
& r_i^\nabla * (T(t_i) * r_i - (R_n - R_{t_i})) + \\
& g_i^\nabla * (T(t_i) * g_i - (G_n - G_{t_i})) + \\
& b_i^\nabla * (T(t_i) * b_i - (B_n - B_{t_i})) + \\
& \sigma_i^\nabla * (T(t_i) * (1 - e^{-\sigma_i * t_i}) - (\Sigma_n - \Sigma_{t_i})) \\
&)
\end{aligned}
\tag{5.1}
$$

This gradient operation was implemented by adapting the existing CUDA code and PyTorch bindings. See listing **??** for details on the actual code implementation.

5.1 Parallelisation

This sections outlines some experiments that were done in an effort to implement the initially proposed design shown in chapter 4. As the base network architecture, KiloNerf with the MiP-

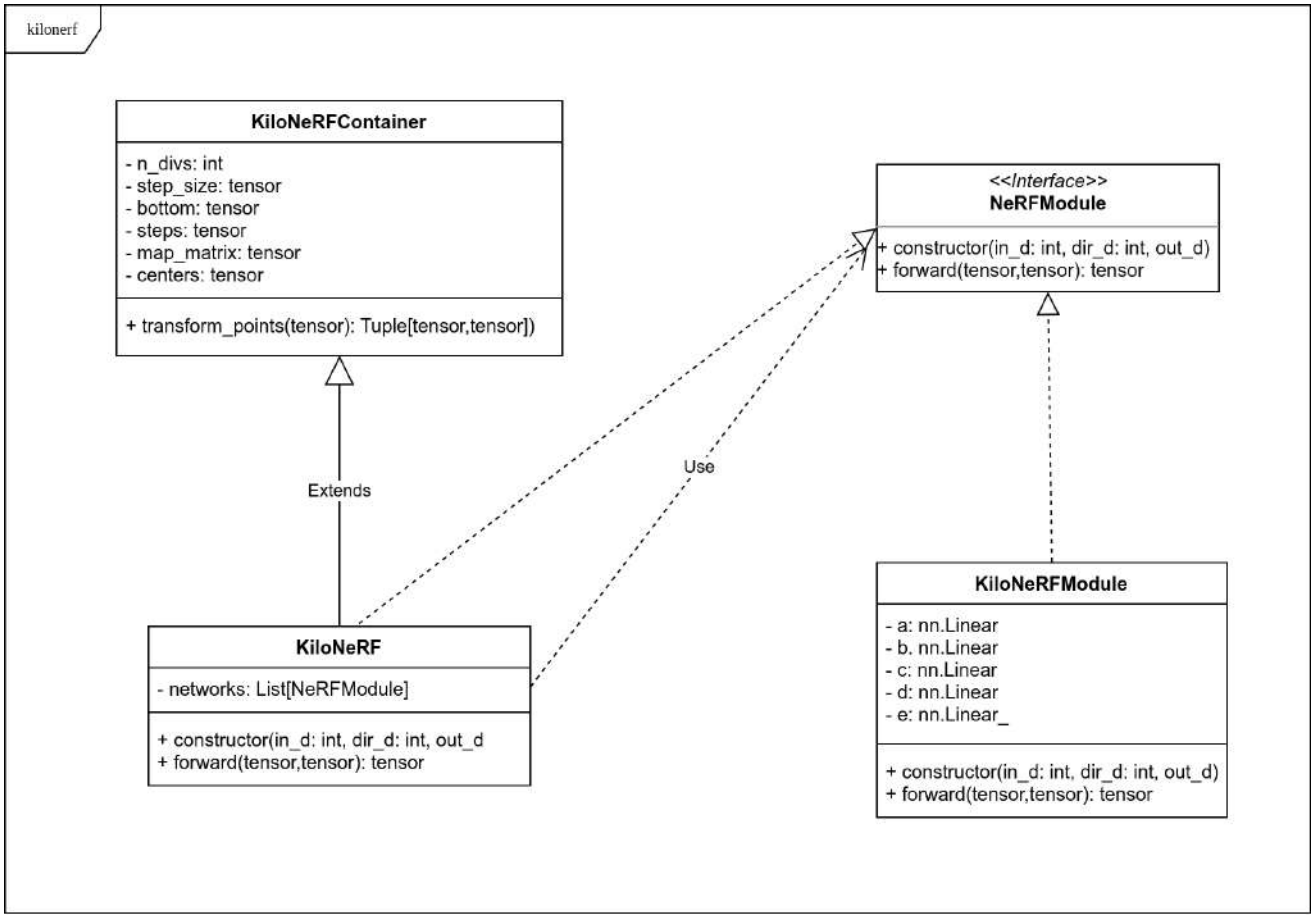

Figure 5.1: Class diagram the KiloNeRF implementation. A container class (KiloNeRFContainer) is implemented that takes care of the spatial distribution of the modules and provides a method to assign query coordinates to the appropriate module. To interact with any NeRF model, a simple interface (NeRFModule) is implemented that maps a set of encoded points and view directions to luminance and density values. This interface is implemented according to the minimal architecture specified by KiloNeRF [36]. Finally, the same interface is implemented by the global class (KiloNeRF) that extends the container and uses a set of modules to execute queries on the scene.

NeRF encoding was implemented in PyTorch (see Figure 5.1). Figure 5.2 shows the comparison in performance between running the NeRF modules concurrently or sequentially on an 100x100 image. It is clear the the concurrent implementation is 40% faster during forward passes, but the during the backward pass the sequential architecture is significantly faster, which is probably to some details in the auto-grad library. The original KiloNerf implementation was done in CUDA, which removes many PyTorch overheads and reduces the backpropagation bottlenecks. For the same reasons as mentioned above, working in CUDA would be infeasible for this project. It was also attempted to use the FullyFused MLP designed for instant-ngp here, but as the module is not compatible with TorchScript the overall performance is worse than our implementation in pure PyTorch. Also compared to the sequential implementation,

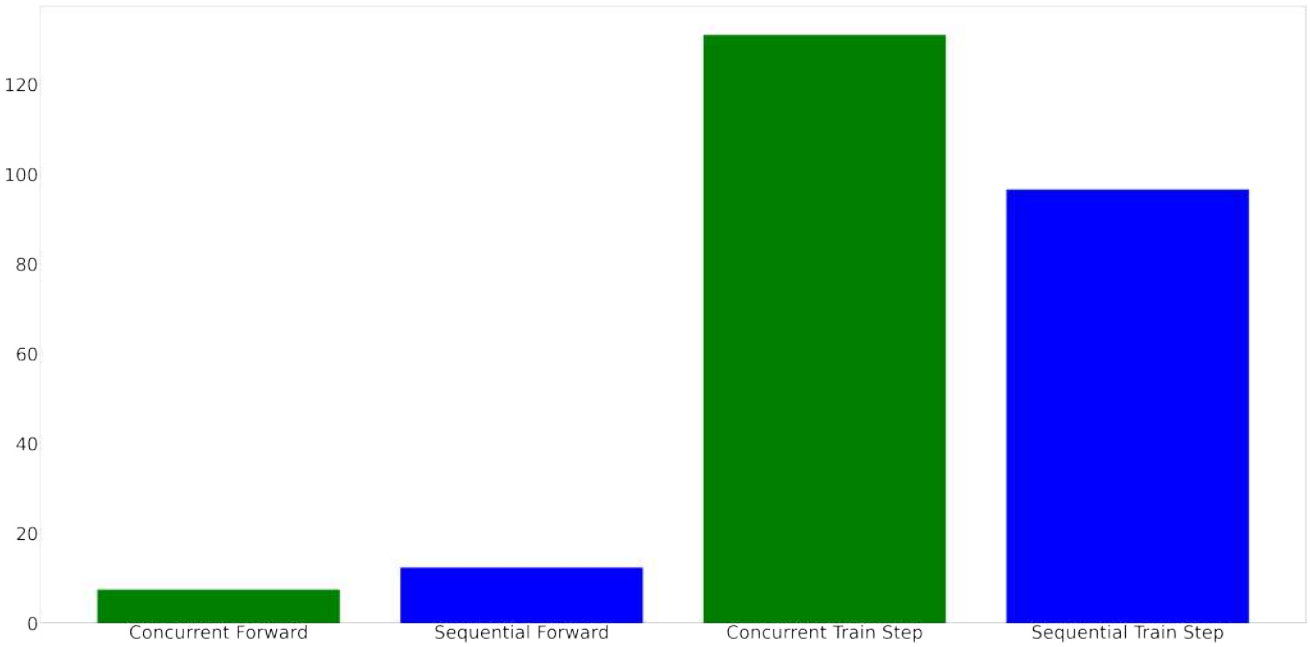

Figure 5.2: Visualization of the performance (in milliseconds) of different KiloNerf implementations.

the added overhead of the Python bindings make it less efficient than our implementation.

5.2 MIP-NeRF

The MIP-NeRF encoding was ported from the original JAX implementation [2] to PyTorch with additional optimizations for this particular use-case. Comparing the two implementations by encoding all rays from a single viewpoint shows that the PyTorch implementation significantly outperforms the existing implementation on both GPU and CPU by a factor of 72 and 148 respectively (see Figure 5.3).

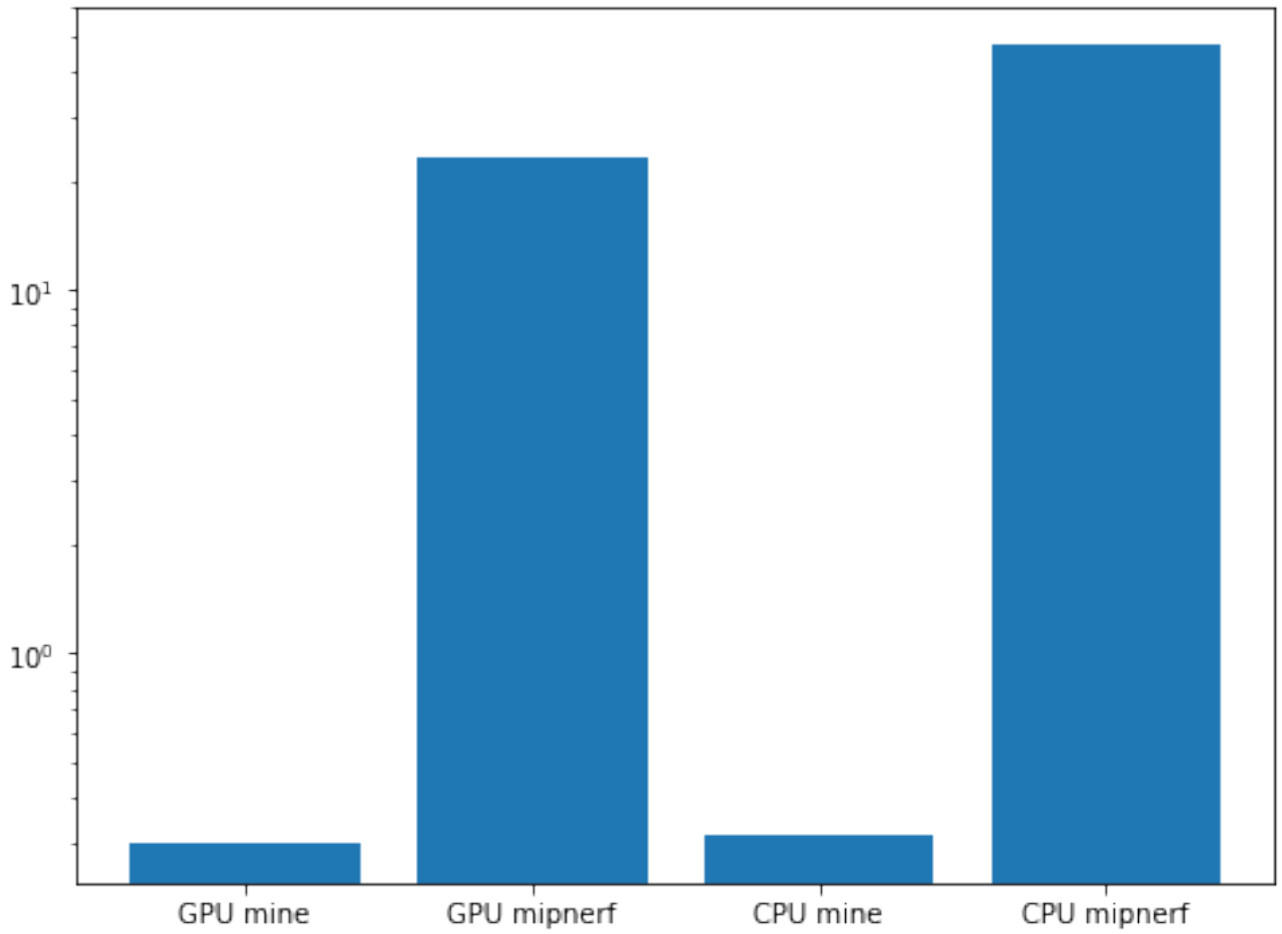

Figure 5.3: Comparison of performance (in milliseconds) between the official MIP-nerf encoding implementation in JAX and my optimized implementation in PyTorch. The rays of a single viewpoint were encoded with 128 sample points between a distance of 2 and 6, with random permutations.

Chapter 6

Experiments

All of the requirements listed in the design by themselves make it difficult for NeRF to learn the scene geometry accurately. This is made worse by considering that the SfM pose estimation is also made less accurate by all of those factors and thus further complicates the optimization problem the architecture has to solve. It will be necessary to add regularization terms to the NeRF architecture to alleviate the impact of the aforementioned characteristics. To prevent negative impacts from the start, pre-processing steps are applied as discussed in the next section.

Evaluation protocol

For all experiments in this section an instant-ngp model is trained with the following configuration:

Parameter	n
density layers	2
radiance Layers	3
MLP width	64
grid levels	32
level scale	1.48692392112

All other parameters that are not mentioned are kept the same as the original implementation. The number grid resolution was chosen to allow for a sufficient scene resolution while keeping the training time below 5 minutes. Each experiment is trained for around 600 iterations (usually 60 epochs) as test scenes are able to converge within 500 iterations. Camera poses were estimated using COLMAP [40] with the exhaustive sparse matching strategy.

Evaluation metrics

To assess the qualities of the scene representations, various different metrics will be used:

- Peak signal-to-noise ratio (PSNR): is a metric to estimate the visual quality of lossy image reconstructions for human perception. It measures the ratio between the maximum signal value in the target image (usually 255) and the maximum introduced noise, measured via the mean squared error (MSE) between the target image and the reconstructed image. It is measured in a logarithmic scale (in dB) and a higher value indicates a more accurate reconstruction.

- Structural similarity index measure (SSIM): is a metric that takes local texture into account by measuring the similarity between mean adjusted windows over the image matrix. The mean of all window indices is taken as the global similarity index.

- Depth map: to assess the quality of the underlying geometric representation, depth maps can provide insight into the range of depth values, local and global consistency and anomalies. If ground-truth depth maps are available, heat-maps that highlight the major divergences will be utilized.

- Point cloud: as shown in Figure 6.2, NeRF can synthesize novel views accurately if the underlying geometric is consistent across viewpoints. To evaluate how compatible the learned geometry is with other viewpoints, the depth maps from training views are used to generate pointclouds which are rotated to correspond to a test view. This transformation reviles the geometric structure of the scene, especially around edges and quickly reveals degenerate representations.

6.1 Data pre-processing

Given an input video, instant-ngp samples frames in a fixed interval to yield 50-100 frames and uses those as training data. This is a very minimal approach that does work well with purposefully captured scenes, but could be improved for more complex settings.

Deblurring

The default NeRF architecture is unable to learn an accurate scene representation when trained on blurry images. Deblur-NeRF [25] describes this problem in detail and proposes solutions that make it possible to recover a high-fidelity scene representation. Due to the implementation

	PSNR	SSIM
default	26.02	0.8489
+ deblurring	25.11	0.8309
+ filtering	25.97	0.8443
+ both	25.33	0.8327

Table 6.1: Image quality of a test pose from the kitchen sequence after 60 epochs of training, split up by various omponents.

complexity of this approach, project time constraints and the lack of available source code, a simpler solution for this problem was devised (it would be worthwhile to investigate the added benefit of implementing this approach in a future work). Instead this work relays on pre-trained networks that are tasked to reverse blurring artefacts. Different available architectures were briefly evaluated [21, 50], of which DeblurGANv2 was chosen as it provides a simple way to use the existing models while delivering results sufficient for this work.

Frame selection

Instead of simply sampling every n-th video frame, we sample the frames with the lowest blurriness within an n-interval that are above a minimal threshold b_{min}. Given a set of frames $I = I_0, \ldots, I_n$ and a interval length f the resulting set of train image indices $F = F_0, \ldots, F_{\lceil n/f \rceil}$ is selected according to the following equations:

$$\nabla^2 I_i = I_i * \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$var(X) = \frac{\sum (X_{i,j} - \bar{X})^2}{X - 1}$$

$$F_k = \underset{j \in \{x | k*f \leq x(k+1)*f\}}{\operatorname{argmax}} var(\nabla^2 I_j)$$

The blurriness is measured by looking at the 2nd-order change in luminance across the image. A low variance implies low-frequency image features, common for blurry images, whereas a high-frequency implies a high information density. Any frame F_k is only used if $var(I_{F_k}) \geq b_{min}$. For evaluation purposes 3 different sequences will be used throughout this chapter, two custom sequences of a desk (each purposefully captured to in a concave or convex manner respectively) and one reference RGBD sequence [10] of a kitchen.

Applying the deblurring model to the video sequences (as shown in Figure 6.1) has great

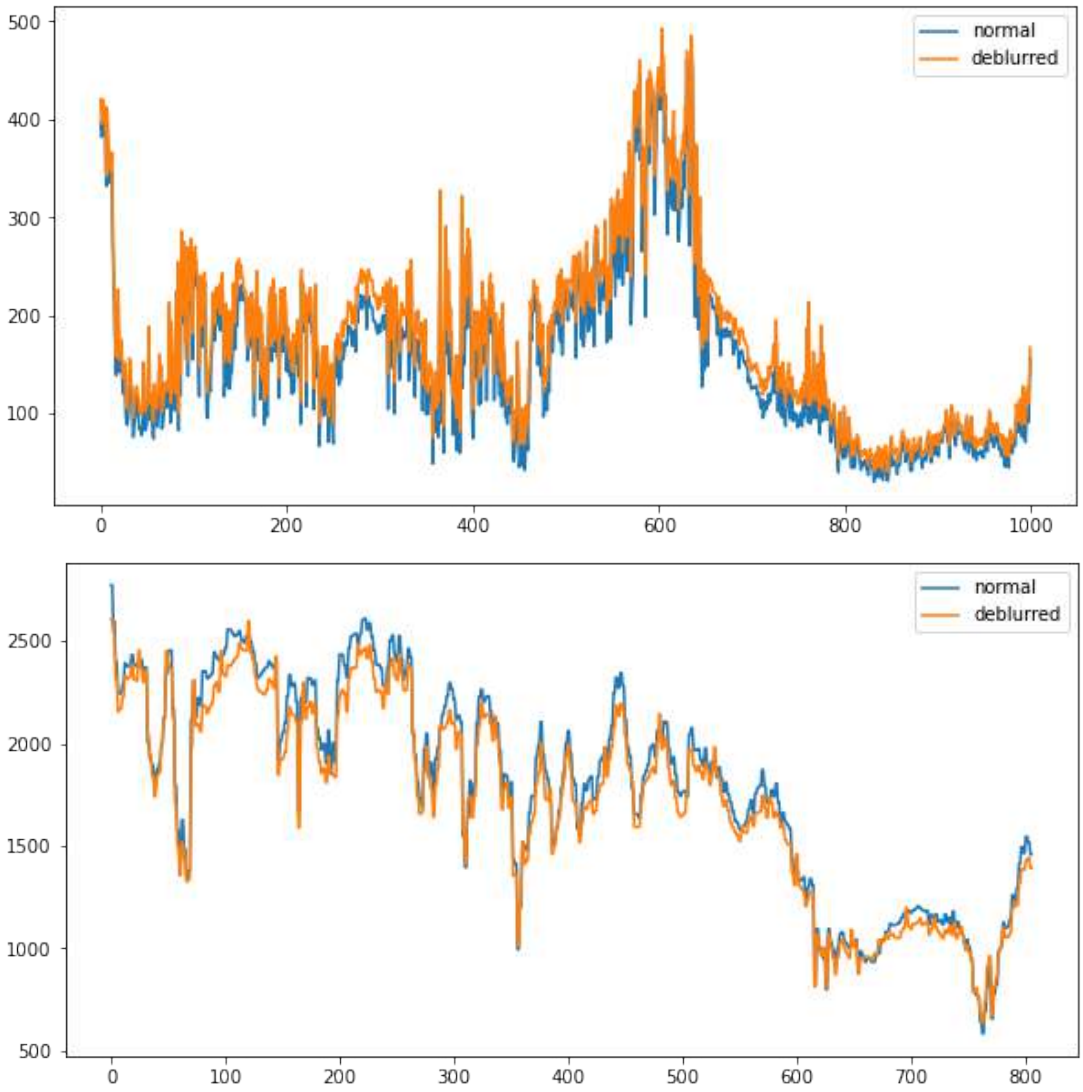

Figure 6.1: (top) Variance of Laplacian sharpness value throughout the kitchen sequence. (bottom) Sharpness values for the convex video sequence.

benefits for low resolution parts of the input frames. For the kitchen sequence the sharpness was improved by 15.4% on average and in some cases by up to 63.6%. In general greater improvements can be observed in frames with lower sharpness, which means that especially longer sequences of blurry frames could yield at least some frames that reach the sharpness threshold and thus contribute to the scene representation. The unmodified kitchen sequence contains a stretch of over 100 frames that falls below the chosen threshold of 80 (chosen visually, can be further optimized), and thus over 3 seconds of footage are not usable. Deblurring the sequence reduces this to 58 frames and thus reduces the gap by almost 50%. On the other had it is apparent that frames with low blurriness do not benefit much from this deblurring method, very sharp images even degrade in quality throughout this process. For the convex

sequence the sharpness was reduced by 3.4% on average and by 9.8% in the worst case. This is probably due to the pre-trained model introducing image artefacts into high-fidelity image regions, as it was only trained on blurry images and is not optimized to retain the image quality in sharp images. Since the reduction in sharpness is minimal considering the high image quality of the test sequence, the process probably would not be affected much by using the degraded frames. This downside can easily be overcome by selecting the sharpest frames from both the original video sequence and the pre-processed one. Applying those modifications does not yield any improvements for the scenes tested here (see Table 6.1). In some cases it has negative effects, which are probably due to the deblurring model introducing high-frequency information that is more difficult to represent. For the reasons mentioned here it might be worthwhile to experiment with other debluring methods in the future.

6.2 Analysis of instant-ngp

When training on a scene it is most important that the network learns an accurate scene geometry [30]. The default NeRF produces very accurate and multi-view consistent depth maps (see Figure 6.2), with only minimal floating artefacts. While instant-ngp achieves comparable visual results, the underlying scene geometry is often degenerate (see Figure 6.3). The point-clouds indicate that a smooth geometry is learned for some prominent scene elements, yet many sample points terminate before the actual surface is reached. For poses included in the train set the overall geometry is consistent with the ground truth data, but significant geometric errors can be observed in views rendered from test poses. Furthermore, the scene geometry appears distorted in favour of the particular viewpoint that is chosen.

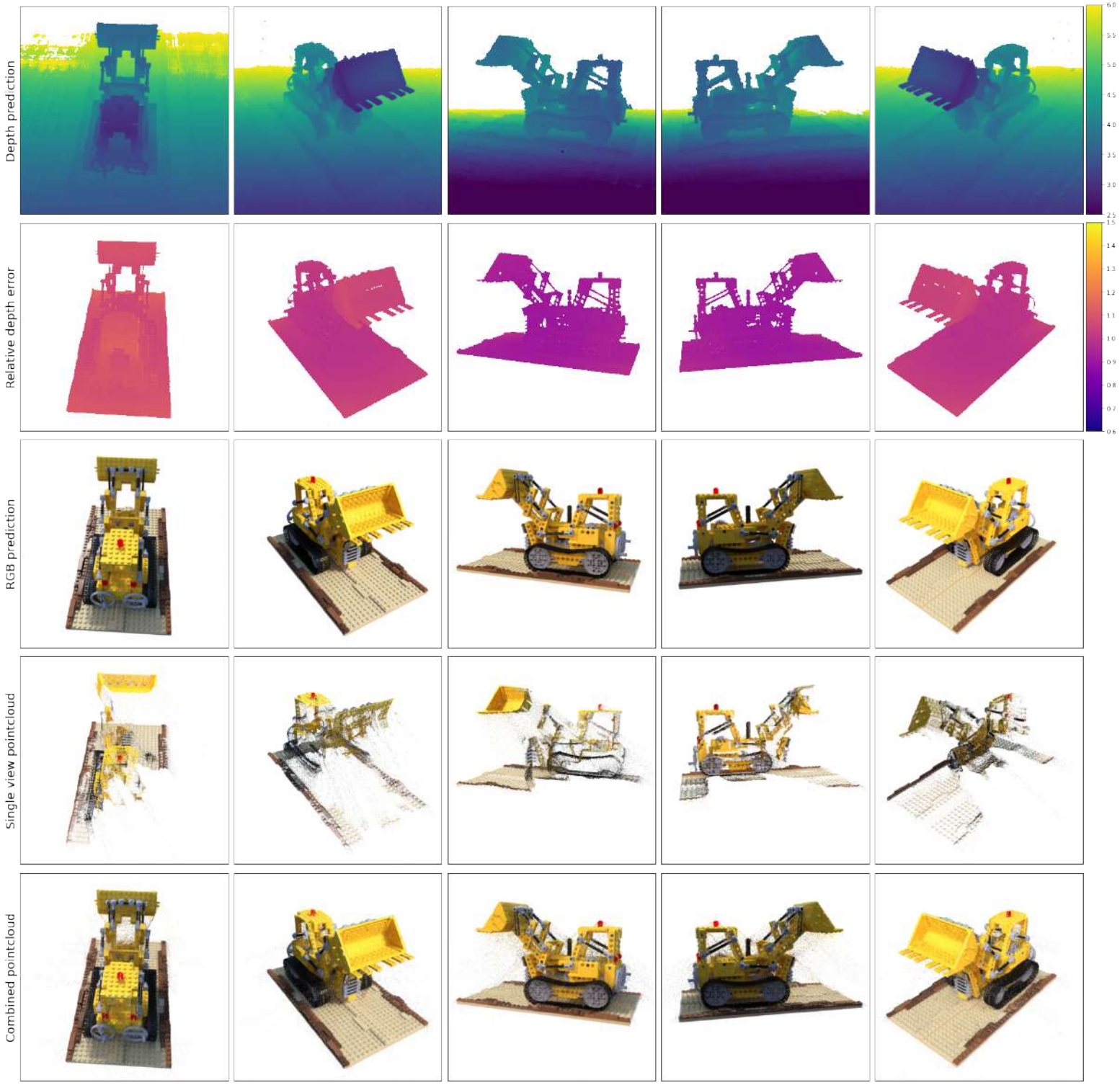

Figure 6.2: Analysis of the learned scene geometry by the original NeRF model. The first row shows the depth maps that NeRF has learned to represent the test scene and the second row shows the relative error to the ground truth depth maps. The third row shows the predicted rendering of the scene from various viewpoints. The fourth row shows a point-cloud generated via the predicted depth maps from the pose of the previous column. The last row shows the pointclouds from all the poses combined.

6.3 Regularization techniques

The geometric errors are making it hard to synthesize novel views, which is only getting worse for scenes with complex geometries and noisy input data. To overcome this, regularization techniques are utilized which enforce specific properties upon the learned representation. Those properties either ensure that the final result exhibits desired characteristics (comparable to variable auto-encoders, where a normal relationship between latent variables is enforced) or that aim to simplify the optimization process by constraining the search space. Instant-ngp already applies local geometric constraints by assigning geometrically related points to a set of shared parameters. Still, the achieved geometric representation is not sufficient for real world scenes and requires more regularization. From initial test it is obvious that recognizable geometries can be learned from convex scenes, but concave scenes require much more help as they learned geometry is strongly degenerate (see the first rows in Figure 6.4 and Figure 6.5 respectively)

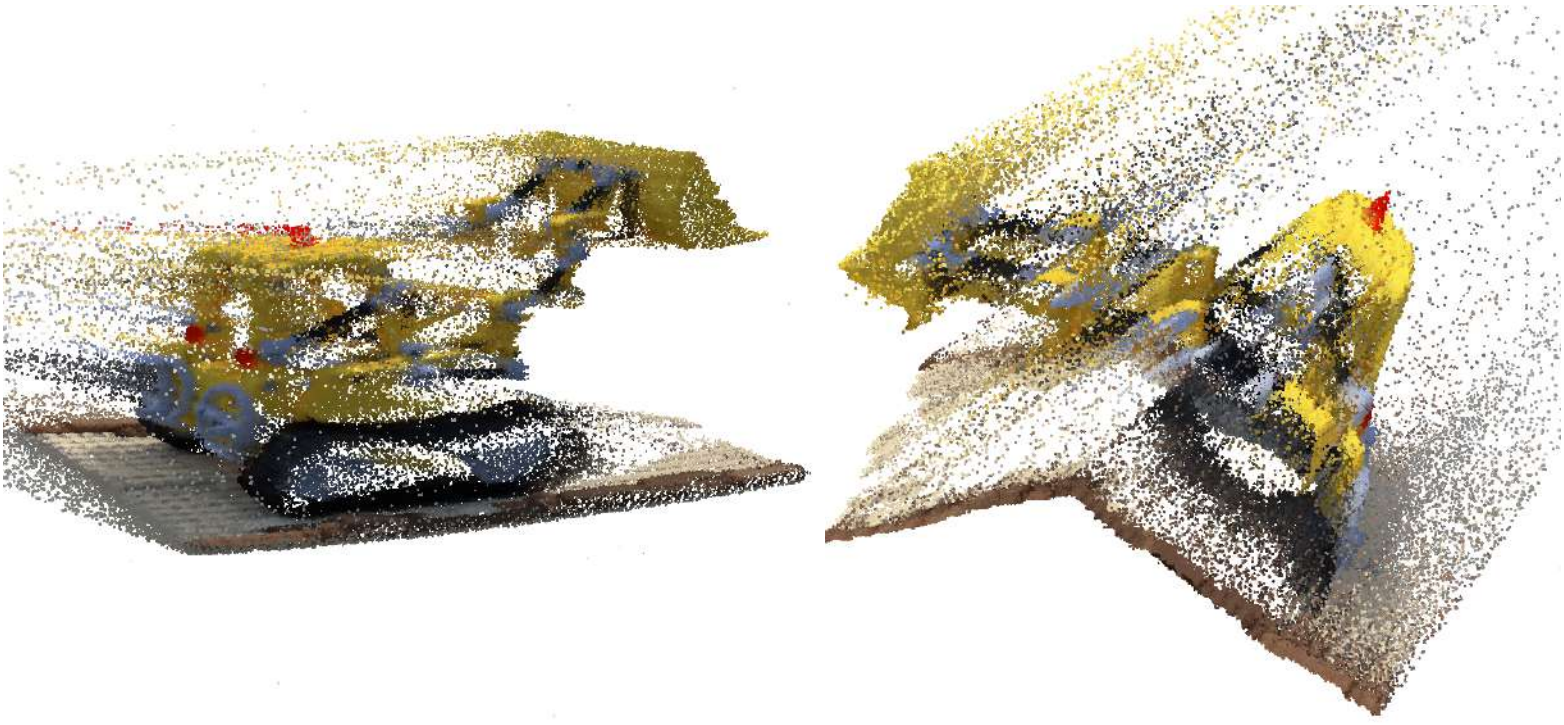

Figure 6.3: (left) Point-cloud generated from a train-set view. (right) Point-cloud generated from a test-set view. Both sampled from a network trained for 60 epochs.

Sampling

Most state-of-the-art methods sample a random set of n points on each image during the training process (n is usually around 2^{12}). This is done because faster convergence can be achieved via this sampling approach, since many points in the training set are mostly redundant. Furthermore, most GPUs do not have enough memory to sample large images in a single pass, especially during training where the gradients take up a lot of memory and thus only chunks of an image can be optimized at once. Working with image patches as mini-batches risks that only local feature consistency is achieved and global features are not captured properly. Sampling random points ensure that each mini-batch optimizes a more globally consistent representation. When dealing with real-world data on the other hand, random sampling tends to result in floating artefacts and bad representations for novel views (see Figure 6.5). Furthermore, many regularization metrics require continuous image patches and thus cannot be applied. A simple trade-off is presented by sampling n random $m \times m$ patches, where n has to be scaled so that $n \times m\times$ queries fit into GPU memory. Smaller values of m allow for more concurrent samples from distinct image regions and thus leads to a faster convergence on the training set, while larger values of m provide more local information to be used by regularization algorithms. For the purposes of this work $m = 3$ has been chosen as it is the smallest size required for the regularization terms applied below. The second row in Figure 6.4 and Figure 6.5 show that sampling by 3 patches increases the training accuracy slightly, but already results in a better geometric representation due to the added local consistency and thus enables novel view synthesis with less visual artefacts.

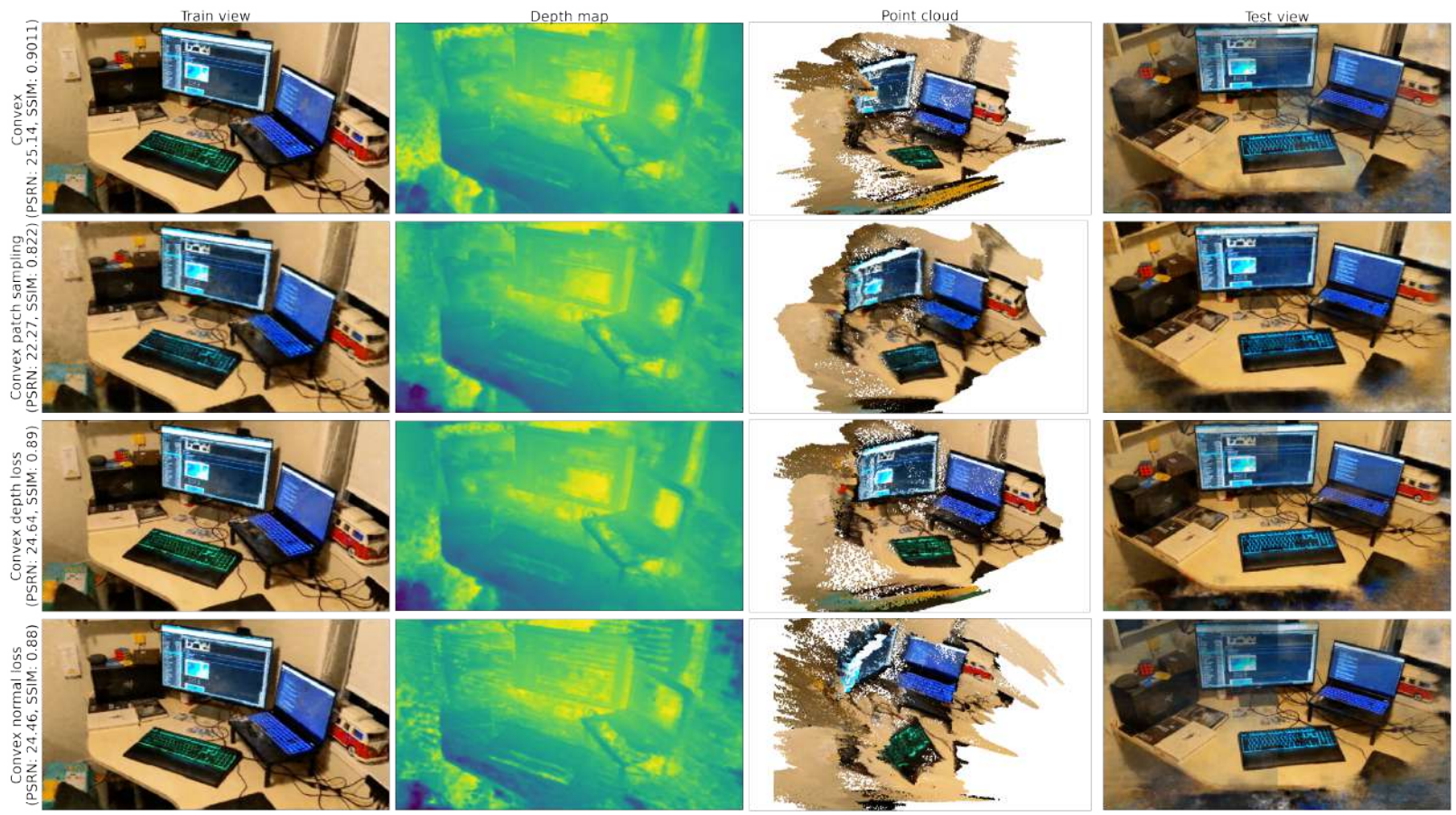

Figure 6.4: Comparison of instant-ngp representations (trained for 60 epochs) of a convex scene using different regularization techniques.

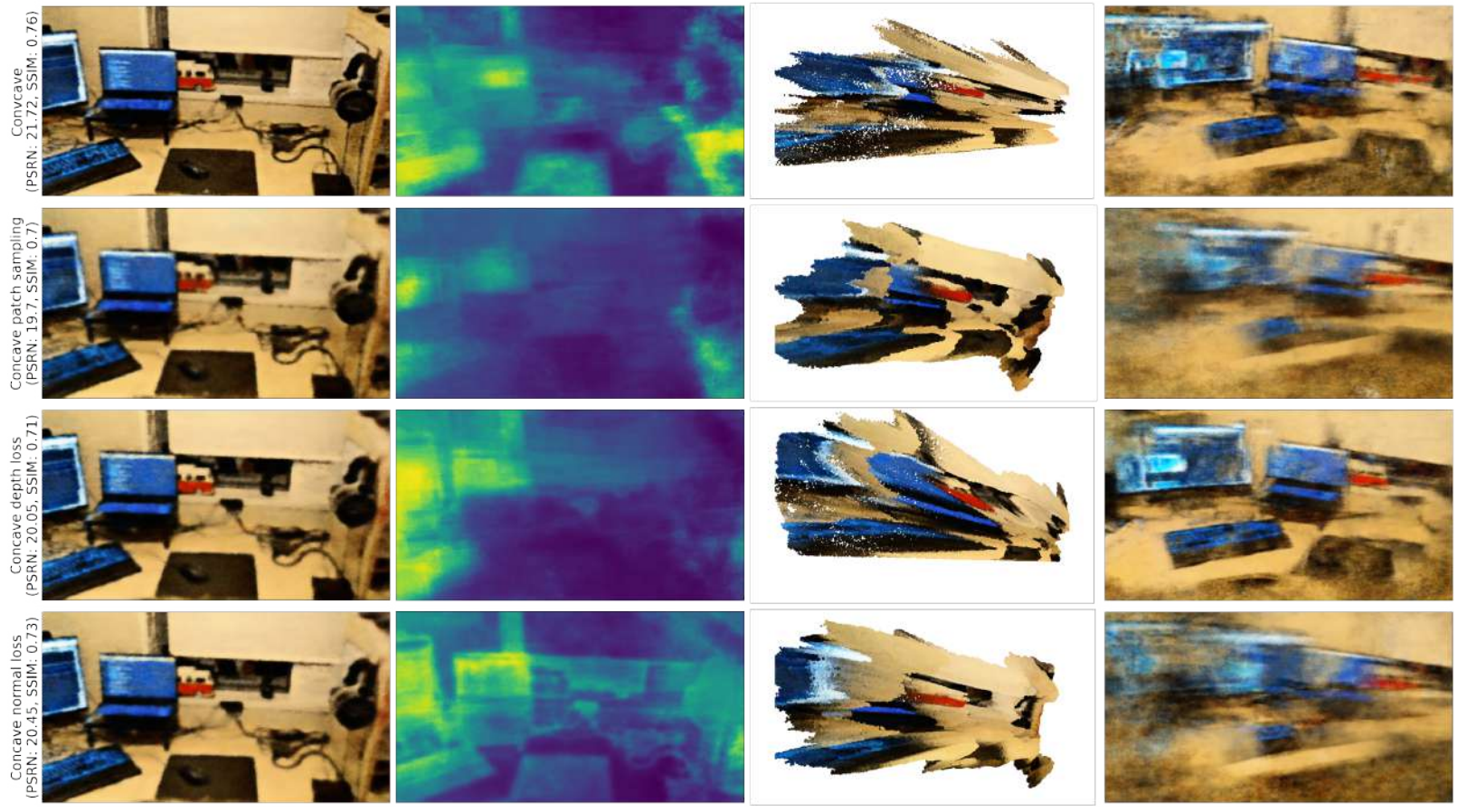

Figure 6.5: Comparison of instant-ngp representations (trained for 120 epochs due to the smaller sequence length) of a concave scene using different regularization techniques.

L2 Depth loss

Previous works applied geometric regularization by ensuring the produced depth maps are piece-wise smooth [30]. The loss is defined as in terms of the depth matrix D:

$$\sum_{i,j=1} (D_{ij} - D_{i+1j})^2 + (D_{ij} - D_{ij+1})^2$$

and reduces the distance between neighboring depth values. The depth loss was added added to the existing radiance loss with a factor of 0.01. The third row in Figure 6.4 and Figure 6.5 shows that this loss leads noticeable improvement in the train accuracy, while still showing improved novel views. Especially for concave scenes the geometric accuracy is greatly improved and radiance ambiguities are reduced.

3D geometry smoothness loss (3DGS)

Jia et. al. [16] proposed a loss that not only takes depth smoothness into account, but also focuses on explicit geometric properties. The previous loss tends to over-smooth the depth values, particularly around edges (we refer to the referenced work for more details on the shortcomings of the L2 loss). Given a matrix of $3x3$ depth values, each point is projected to 3D world space using the camera intrinsic. The normal vectors are calculated via the cross product between two neighboring vectors going from the central point in the patch two any of the outer points. The normal vector for the central point is estimated as the average of all 8 normal vectors. The loss is calculated as

$$\mathcal{L}_{3DGS} = \frac{1}{N_p} \sum_p \underbrace{|e^{-\partial_x I}| \S_x A + |e^{-\partial_y I}| \S_y A}_{\mathcal{C}^1, \mathcal{C}^2 \; smoothness} + \underbrace{|e^{-\partial_x I}| \partial_x D^{-1} + |e^{-\partial_y I}| \partial_y D^{-1}}_{\mathcal{C}^0 \; smoothness},$$

where $\S(\overrightarrow{n_1}, \overrightarrow{n_2}) = 1 - (\frac{\overrightarrow{n_1} \cdot \overrightarrow{n_2}}{\|\overrightarrow{n_1}\|_2 \|\overrightarrow{n_2}\|_2})^2$ is the sine distance function, A is the matrix of normal vectors, D^-1 is the disparity map and I is the color image. This loss slightly increases the image quality during training, but performs worse than the depth smoothness loss for novel view synthesis. It is notable that the depth maps generated with this loss do exhibit the expected characteristics and articulate edges far better than other methods mentioned here. Yet this does not lead to a better NeRF geometry. See Listing **??** for implementation details.

Chapter 7

Professional and Ethical Issues

This project was executed in accordance to the British Computer Society Code of Conduct. All third party sources and influences were referenced and attributed.

Ethical concerns were considered during all steps of this project. Video sequences that contain private information might be problematic as the scene representations will also includes those details. The video sequences used for this work were selected to not include any sensitive or private information. Privacy can be kept by blurring information in the video files, which will probably lead to a worse reconstruction quality. Future works could investigate if instead the learned representation can be altered to blur out the radiance values without destroying the geometric information.

Chapter 8

Conclusion

This worked showed how the latest research can be used to represent real-world scenes from phone videos and what the limitations of it are. The capabilities are far from general and require further regularization techniques. Of the requirements laid out in chapter 4, all except requirement 6 and 7 were achieved. The processing pipeline only requires an MP4 file as an input, estimates the camera parameters using COLMAP, applies a key-frame selection algorithm, is able to converge on a scene within a few minutes and can render views with up to 20 FPS (assuming an 800x800 image), while having a train accuracy of over 19 PSNR and a similar quality for test-views in concave scenes. The number of selected key-frames is higher than the one specified, but since the chosen architecture is sufficiently easier to train the specification is not applicable here. Due to the design changes caused by rapid advancements in this field, the general implementation complexity of this method and training time requirements, only a few novel experiments could be implemented within the project bounds. The next section will outline what aspects future works should investigate further.

Possible future work

Besides regularizing training views, some existing works[30] also apply regularization to random unseen views during training, which likely increases the accuracy for test-view syntheses. Since the biggest factor in representation quality is the quality of input images and poses, another way to alleviate the optimization hurdles is to assist users in capturing the scenes via an smartphone app. The most important factors an ideal user would have to consider are:

- Move slow enough to reduce motion blur
- Capture a set of mostly concave poses

Certain camera mechanisms can be used simplify this process Reducing the exposure time can allow the user to move at natural speeds without risking motion blur, but this reduces the luminance of the scene which might become problematic in low-light situations. Furthermore, the final representation would probably not look as intended. Since requiring additional lighting is unfeasible, existing models for low-light image enhancement could be utilized to retain the same lighting conditions by biasing it on luminance values obtain with standard exposure time[11]. A given exposure time limits the maximum speed the camera can move at before motion blur artefacts appear. The assistant could further visualize the blurriness metric and guide the user to be within a safe range of motion. As mentioned in chapter 4, capturing concave scenes can be quite difficult without additional equipment. Again, a visual assistant could help by keeping track of a selected scene centre using standard computer vision methods. Assistants like this are not uncommon and today are built into most smartphones for the purpose of 360° photo captures.

The problems encountered through this work share large commonalities with monocular depth estimation and many of the approaches used there could improve this optimization process. It seems that most approaches struggle to generate depth maps that are consistent within object boundaries. A popular approach used for depth estimation is relying on semantic segmentation to identify objects and constraining the network to learn continuous depth values within each object boundary. [19] Self-supervised approaches often rely on relative pose predictions between two frames and then enforce geometric consistency by projecting the pixels between the two poses via the depth maps to ensure multi-view consistency. Since NeRF already has poses available, similar losses could easily be applied without needing to train a pose-estimation network that likely introduces noise into the system.

References

[1] ashawkey. torch-ngp, 2022.

[2] J. T. Barron, B. Mildenhall, M. Tancik, P. Hedman, R. Martin-Brualla, and P. P. Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields, 2021.

[3] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. M. Montiel, and J. D. Tardós. ORB-SLAM3: an accurate open-source library for visual, visual-inertial and multi-map SLAM. *CoRR*, abs/2007.11898, 2020.

[4] R. Chabra, J. E. Lenssen, E. Ilg, T. Schmidt, J. Straub, S. Lovegrove, and R. Newcombe. Deep local shapes: Learning local sdf priors for detailed 3d reconstruction. In A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, editors, *Computer Vision – ECCV 2020*, pages 608–625, Cham, 2020. Springer International Publishing.

[5] K. Deng, A. Liu, J.-Y. Zhu, and D. Ramanan. Depth-supervised nerf: Fewer views and faster training for free, 2021.

[6] L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using real nvp, 2017.

[7] M. Dissanayake, P. Newman, S. Clark, H. Durrant-Whyte, and M. Csorba. A solution to the simultaneous localization and map building (slam) problem. *IEEE Transactions on Robotics and Automation*, 17(3):229–241, 2001.

[8] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *IEEE Robotics Automation Magazine*, 13(2):99–110, 2006.

[9] J. Fang, L. Xie, X. Wang, X. Zhang, W. Liu, and Q. Tian. Neusample: Neural sample field for efficient view synthesis, 2021.

[10] B. Glocker, S. Izadi, J. Shotton, and A. Criminisi. Real-time rgb-d camera relocalization. In *2013 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 173–179, 2013.

[11] C. Guo, C. Li, J. Guo, C. C. Loy, J. Hou, S. Kwong, and R. Cong. Zero-reference deep curve estimation for low-light image enhancement. 2020.

[12] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015.

[13] J. Heaton. An empirical analysis of feature engineering for predictive modeling. *SoutheastCon 2016*, Mar 2016.

[14] A. Isenko, R. Mayer, J. Jedele, and H.-A. Jacobsen. Where Is My Training Bottleneck? Hidden Trade-Offs in Deep Learning Preprocessing Pipelines. *arXiv:2202.08679 [cs]*, Feb. 2022. arXiv: 2202.08679.

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

[16] S. Jia, X. Pei, W. Yao, and S. C. Wong. Self-supervised depth estimation leveraging global perception and geometric smoothness using on-board videos, 2021.

[17] C. M. Jiang, A. Sud, A. Makadia, J. Huang, M. Nießner, and T. Funkhouser. Local implicit grid representations for 3d scenes. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[18] D. D. Kalamkar, D. Mudigere, N. Mellempudi, D. Das, K. Banerjee, S. Avancha, D. T. Vooturi, N. Jammalamadaka, J. Huang, H. Yuen, J. Yang, J. Park, A. Heinecke, E. Georganas, S. Srinivasan, A. Kundu, M. Smelyanskiy, B. Kaul, and P. Dubey. A study of BFLOAT16 for deep learning training. *CoRR*, abs/1905.12322, 2019.

[19] M. Klingner, J.-A. Termöhlen, J. Mikolajczyk, and T. Fingscheidt. Self-supervised monocular depth estimation: Solving the dynamic object problem by semantic guidance, 2020.

[20] N. Kondo, Y. Ikeda, A. Tagliasacchi, Y. Matsuo, Y. Ochiai, and S. S. Gu. Vaxnerf: Revisiting the classic for voxel-accelerated neural radiance field, 2021.

[21] O. Kupyn, T. Martyniuk, J. Wu, and Z. Wang. Deblurgan-v2: Deblurring (orders-of-magnitude) faster and better, 2019.

[22] H. Lin, S. Peng, Z. Xu, H. Bao, and X. Zhou. Efficient neural radiance fields with learned depth-guided sampling, 2021.

[23] D. B. Lindell, J. N. P. Martel, and G. Wetzstein. Autoint: Automatic integration for fast neural volume rendering, 2021.

[24] L. Liu, J. Gu, K. Z. Lin, T.-S. Chua, and C. Theobalt. Neural sparse voxel fields. *NeurIPS*, 2020.

[25] L. Ma, X. Li, J. Liao, Q. Zhang, X. Wang, J. Wang, and P. V. Sander. Deblur-nerf: Neural radiance fields from blurry images, 2021.

[26] Mario, Briyce, J. Rushton, and B. Comp. How much does 3d cad modeling design cost amp; what are prices for freelance design service firms?

[27] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. Nerf: Representing scenes as neural radiance fields for view synthesis, 2020.

[28] T. Müller. Tiny CUDA neural network framework, 2021. https://github.com/nvlabs/tiny-cuda-nn.

[29] T. Müller, A. Evans, C. Schied, and A. Keller. Instant neural graphics primitives with a multiresolution hash encoding. *arXiv:2201.05989*, Jan. 2022.

[30] M. Niemeyer, J. T. Barron, B. Mildenhall, M. S. M. Sajjadi, A. Geiger, and N. Radwan. Regnerf: Regularizing neural radiance fields for view synthesis from sparse inputs. *arXiv.org*, 2021.

[31] M.-G. Park and K.-J. Yoon. Optimal key-frame selection for video-based structure-from-motion. *Electronics Letters*, 47:1367–1369, 12 2011.

[32] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[33] S. Peng, M. Niemeyer, L. Mescheder, M. Pollefeys, and A. Geiger. Convolutional occupancy networks. In *European Conference on Computer Vision (ECCV)*, 2020.

[34] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. Learning transferable visual models from natural language supervision, 2021.

[35] D. Rebain, W. Jiang, S. Yazdani, K. Li, K. M. Yi, and A. Tagliasacchi. Derf: Decomposed radiance fields, 2020.

[36] C. Reiser, S. Peng, Y. Liao, and A. Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps, 2021.

[37] B. Roessle, J. T. Barron, B. Mildenhall, P. P. Srinivasan, and M. Nießner. Dense depth priors for neural radiance fields from sparse input views, 2021.

[38] R. A. Rosu and S. Behnke. Neuralmvs: Bridging multi-view stereo and novel view synthesis, 2021.

[39] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In *2011 International Conference on Computer Vision*, pages 2564–2571, 2011.

[40] J. L. Schönberger and J.-M. Frahm. Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[41] V. Sitzmann, M. Zollhöfer, and G. Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. *CoRR*, abs/1906.01618, 2019.

[42] A. Trevithick and B. Yang. Grf: Learning a general radiance field for 3d representation and rendering, 2021.

[43] H. Turki, D. Ramanan, and M. Satyanarayanan. Mega-nerf: Scalable construction of large-scale nerfs for virtual fly-throughs, 2021.

[44] G. Van Rossum and F. L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[45] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[46] Q. Wang, Z. Wang, K. Genova, P. Srinivasan, H. Zhou, J. T. Barron, R. Martin-Brualla, N. Snavely, and T. Funkhouser. Ibrnet: Learning multi-view image-based rendering, 2021.

[47] Y. Wei, S. Liu, Y. Rao, W. Zhao, J. Lu, and J. Zhou. Nerfingmvs: Guided optimization of neural radiance fields for indoor multi-view stereo. In *ICCV*, 2021.

[48] G.-W. Yang, W.-Y. Zhou, H.-Y. Peng, D. Liang, T.-J. Mu, and S.-M. Hu. Recursive-nerf: An efficient and dynamically growing nerf, 2021.

[49] A. Yu, V. Ye, M. Tancik, and A. Kanazawa. pixelnerf: Neural radiance fields from one or few images, 2021.

[50] S. W. Zamir, A. Arora, S. H. Khan, M. Hayat, F. S. Khan, and M. Yang. Restormer: Efficient transformer for high-resolution image restoration. *CoRR*, abs/2111.09881, 2021.

[51] K. Zhang, G. Riegler, N. Snavely, and V. Koltun. Nerf++: Analyzing and improving neural radiance fields. *arXiv:2010.07492*, 2020.

[52] Y. Zhang, G. M. Gibson, R. Hay, R. W. Bowman, M. J. Padgett, and M. P. Edgar. A fast 3d reconstruction system with a low-cost camera accessory. *Scientific Reports*, 5(1):10909, Jun 2015.