Object and Scene Reconstruction using Neural Radiance Fields

by

Matthew Tancik

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Electrical Engineering and Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Spring 2023

Object and Scene Reconstruction using Neural Radiance Fields

Abstract

Object and Scene Reconstruction using Neural Radiance Fields

by

Matthew Tancik

Doctor of Philosophy in Electrical Engineering and Computer Science

University of California, Berkeley

Assistant Professor Angjoo Kanazawa, Co-chair

Associate Professor Ren Ng, Co-chair

This dissertation explores the synthesis of novel views of complex scenes through the optimization of a volumetric scene function using a sparse set of input views. Our approach represents the scene as a neural radiance field (NeRF), a field of densities and emitted radiance based on 5D coordinates encompassing spatial location (x, y, z) and viewing direction (θ, ϕ). NeRF enables the rendering of photorealistic novel views that surpass previous techniques, leading to numerous follow-ups and extensions in the computer vision and graphics communities. To enhance the representation of high-frequency details in NeRFs, we introduce a Fourier feature mapping technique that effectively learns high-frequency functions within low-dimensional problem domains, including NeRF. We demonstrate the benefits of leveraging learned initial weight parameters through standard meta-learning algorithms, resulting in accelerated convergence, stronger priors, and improved generalization for coordinate-based networks. In addition, we improve the scalability of NeRFs with a proposed method capable of representing arbitrarily large scenes. This method enables city-scale reconstructions using data captured under diverse environmental conditions. Finally, we present the Nerfstudio framework, a comprehensive suite of modular components and tools designed for the development and deployment of NeRF-based methods. This framework empowers researchers and practitioners with real-time visualization, streamlined data pipelines, and export capabilities, facilitating the democratization of NeRFs and extending their impact beyond research settings. With their potential to transform computer graphics, virtual reality, augmented reality, and other domains, NeRFs hold promise for revolutionizing the way we perceive and interact with digital worlds.

Contents

Contents **i**

List of Figures **iii**

List of Tables **v**

1 Introduction **1**
1.1 Representing the Plenoptic Fuction 1
1.2 Dissertation Overview 3

2 Neural Radiance Fields **6**
2.1 Related Work 8
2.2 Neural Radiance Field Scene Representation 10
2.3 Volume Rendering with Radiance Fields 11
2.4 Optimizing a Neural Radiance Field 11
2.5 Results 14
2.6 Discussion 20

3 Representing High Frequencies in Coordinate-Based Networks **21**
3.1 Related Work 23
3.2 Background and Notation 23
3.3 Fourier Features for a Tunable Stationary Neural Tangent Kernel 25
3.4 Stationary kernels 26
3.5 Manipulating the Fourier Feature Mapping 27
3.6 Experiments 30
3.7 Discussion 39

4 Initializing Coordinate-Based Networks **41**
4.1 Related Work 42
4.2 Overview 43
4.3 Implementation details 45
4.4 Results 47
4.5 Discussion 56

5 Scaling Neural Radiance Fields **57**
5.1 Related Work . . . 58
5.2 Background . . . 61
5.3 Method . . . 61
5.4 Results and Experiments . . . 66
5.5 Model Parameters / Optimization Details . . . 66
5.6 Limitations and Future Work . . . 71
5.7 Discussion . . . 72

6 Nerfstudio Framework **73**
6.1 Related Works . . . 76
6.2 Framework Design . . . 77
6.3 Core components . . . 78
6.4 Nerfacto Method . . . 81
6.5 Nerfstudio Dataset . . . 83
6.6 Experiments . . . 83
6.7 Open-source Contributions . . . 87
6.8 Discussion . . . 87

7 Conclusion **89**
7.1 Why NeRFs and the Role of Representation . . . 89
7.2 Limitations . . . 90
7.3 The Future of 3D . . . 91

Bibliography **92**

List of Figures

1.1 From NeRF to Nerfstudio 2

2.1 Neural Radiance Field problem setup. 7
2.2 Overview of Neural Radiance Field pipeline. 9
2.3 Visualization of view-dependent emitted radiance. 10
2.4 Ablation of view dependence and positional encoding. 12
2.5 Comparisons on test-set views of Blender synthetic scenes. 16
2.6 Comparisons on test-set views of real world scenes 17

3.1 Demonstration of improvements when using Fourier features across multiple tasks. . . 22
3.2 Fitting non-centered Gaussian functions with MLPs. 27
3.3 Visualization of the neural tangent kernel when using Fourier features. 28
3.4 Convergence plots when using Fourier feature mappings. 29
3.5 Effects of sweeping the standard deviation of the random Fourier features. 30
3.6 Results for the 2D image regression task. 32
3.7 Results on 3D shape occupancy task. 34
3.8 Results on 2D CT task. 35
3.9 Results on 3D MRI task. 36
3.10 Results on inverse rendering task. 37
3.11 Optimizing the Fourier feature mappings for specific target signals. 39
3.12 Frequency spreading in deeper networks. 39
3.13 NTK for networks of varying depths. 40

4.1 Illustrative example of meta-learned initializations. 42
4.2 Faster convergence result on 2D image regression. 49
4.3 CT reconstructions from a sparse set of views. 52
4.4 Single view reconstructions of ShapeNet. 53
4.5 Faster reconstruction of ShapeNet objects. 54
4.6 Reconstructions from the Phototourism dataset. 55

5.1 Large-scale scene reconstructions. 58
5.2 NeRF sampling and merging strategy. 62
5.3 Block-NeRF network architecture. 63

5.4 Using appearance codes to modify lighting and weather. 63
5.5 Altering exposure settings during inference. 64
5.6 Matching Block-NeRF with different appearances. 66
5.7 Ablations of the Block-NeRF model. 69

6.1 Overview of the Nerfstudio framework. 74
6.2 Nerfstudio pipeline components. 75
6.3 Real-world Nerfstudio dataset. 75
6.4 Representation of samples along a camera ray. 79
6.5 NeRF export options provided in Nerfstudio. 81
6.6 Nerfacto model architecture. 82
6.7 Visualization of unbounded samples contracted to a bounded space. 83
6.8 Nerfstudio ablation qualitative examples. 84
6.9 Screenshots of the provided real-time viewer. 86

List of Tables

2.1 Quantitative comparisons on synthetic and real images. 15
2.2 Ablation study of neural radiance fields. 19

3.1 2D image results (mean ± standard deviation of PSNR) 33
3.2 3D shape regression results. 33
3.3 2D CT results (mean ± standard deviation of PSNR). 35
3.4 3D MRI results (mean ± standard deviation of PSNR). 36
3.5 3D NeRF results (mean ± standard deviation of PSNR). 38

4.1 Comparison of different initialization methods on an image regression task. 50
4.2 PSNR comparison of four different dataset-specific learned initializations for image regression. 50
4.3 Comparison of initialization methods on a CT reconstruction task. 51
4.4 Metrics for single image ShapeNet reconstructions using a simple-NeRF model. 54
4.5 Reconstruction results on Phototourism data. 55

5.1 Ablations of different Block-NeRF components on a single intersection. 68
5.2 Comparison of different numbers of Block-NeRFs for reconstructing the Mission Bay dataset. 69
5.3 Effect of different NeRF overlaps in the 8 block scenario with 0.25M weights per block (2M weights in total). The default setting used for other experiments is marked *. . . . 70
5.4 Comparison of interpolation methods. 71

6.1 Average metrics on the MipNeRF360 dataset. 85
6.2 Average metrics for ablations on the Nerfstudio Dataset. 87

Acknowledgments

I am incredibly fortunate to have been surrounded by remarkable individuals during my time at Berkeley. While it is impossible to thank everyone who has played a crucial role in my success, I want to express my gratitude to those who have had a significant impact on my journey.

First and foremost, I would like to extend my deepest appreciation to my advisors, Ren Ng and Angjoo Kanazawa. When I was considering different universities, Ren's approach to research and his holistic perspective on life instantly captivated me. He taught me the importance of non-technical aspects in research, such as effective communication, problem framing, and maintaining a healthy work-life balance. Ren's graphics background provided a unique lens that complemented the machine learning perspective. My initial interaction with Angjoo during a remote meeting during the peak of the COVID-19 pandemic left a lasting impression. Her infectious energy breathed life into otherwise mundane virtual gatherings, and I knew immediately that I wanted to work with her. I greatly appreciate her encouragement of collaboration among students, as I have learned how rewarding it can be. Her dedication to staying at the forefront of emerging topics, rather than following established paths, is truly admirable. I consider myself incredibly fortunate to have had two exceptional advisors during my time at Berkeley.

My PhD experience can be divided into two distinct phases: pre-COVID and post-COVID. During the early years, I had the pleasure of working and socializing in the Soda office with an amazing group of individuals including Ben Mildenhall, Pratul Srinivasan, Utkarsh Singhal, Grace Kuo, Vivien Nguyen, and Cecilia Zhang. Among them, I want to highlight Ben and Pratul, with whom I worked on multiple joint projects, including NeRF, which has come to define my time at Berkeley. They were not only excellent mentors who patiently guided me as a young and inexperienced graduate student but also became great friends. They not only imparted invaluable research skills but also showed me how to derive joy from the process. During the latter half of my PhD, I had more interactions with the vision groups, particularly Angjoo's lab. I have had numerous enriching experiences, conversations, and collaborations with Ethan Weber, Justin Kerr, Evonne Ng, Ruilong Li, Brent Yi, Chun Min Kim, Alex Yu, Frederik Warburg, Ilija Radosavovic, Antonio Loquercio, Georgios Pavlakos, Shubham Goel, Vickie Ye, and Hang Gao. Working with Ethan has been particularly rewarding. He is not only a talented researcher but also a genuinely good person, and I am grateful for the opportunity to work with him in the lab and enjoy our time together outside of it. I want to express my gratitude to everyone who contributed their time and energy to the Nerfstudio project. Its success is a testament to the dedication and support of all those involved. I would like to give special recognition to Ethan and Evonne for their willingness to take the leap and invest their time in starting the project, even if it meant diverting attention from traditional research. Lastly, I want to thank all the undergraduate and master's students I had the privilege of working with during my PhD. It has been a rewarding experience, and I have thoroughly enjoyed watching your research and careers evolve over the years. I would also like to express my gratitude to Henrik Kretzschmar, Vincent Casser, Xinchen Yan, and Sabeek Pradhan for the stimulating intellectual discussions and collaborations during my time as an intern at Waymo.

I would also like to express my gratitude to the other members of my dissertation committee, Alyosha Efros and Jon Barron. Conversations with Alyosha about our shared love for California hikes have always been enjoyable. I consider myself privileged to have had the opportunity to work with Jon on multiple projects over the years. We have had many insightful research discussions, and Jon has consistently been a valuable source of life advice.

Prior to Berkeley, I had the opportunity to work in Ramesh Raskar's and Fredo Durand's labs at MIT during my undergraduate and master's studies. I am particularly grateful to Guy Satat and Zoya Bylinskii for their mentorship and for igniting my passion for research. Before working with them, I had never considered pursuing a PhD, but their guidance and encouragement opened new doors for me.

Beyond the lab, I have been fortunate to form friendships with many of the individuals mentioned above. It is through these friendships and others that I have been able to maintain my sanity throughout the years. I have cherished the adventures, skiing trips, backpacking excursions, and more that I have experienced with Ian Chesser, Julia Longmate, David Ye, Kenny Friedman, Alejandro Escontrela, Patience Stevens, and Stephen McCrory. I want to extend a special thank you to Suzie Petryk for being by my side through the highs and lows, and for being my enthusiastic adventure companion, even if it simply means watching goats graze on grass down the street.

Lastly, I would like to express my heartfelt gratitude to my family for their unwavering support throughout my PhD journey. To my mom, who always prioritizes my happiness and never hesitates to offer a vacation or understand when my school commitments take precedence. And to my father, who introduced me to technology through his old laptops and cameras when I was young. Although he is no longer with us to witness the completion of this dissertation, his influence was instrumental in shaping the person I am today.

Chapter 1

Introduction

The ability to replicate the world around us in three dimensions has been a longstanding pursuit of humanity. Throughout history, humans have employed various artistic techniques, from the earliest cave paintings to intricate Renaissance masterpieces, to capture and represent the world in visual form. However, this endeavor has always been time-consuming and required immense skill and craftsmanship.

The invention of the camera revolutionized the way we perceive and document the world. With the ability to capture and freeze moments in time onto a two-dimensional plane, photography brought about a new era of visual representation. However, despite the remarkable advancements in two-dimensional imaging technology, we are still faced with the challenge of faithfully reproducing the full three-dimensional complexity of the world around us.

Currently, the generation of three-dimensional content requires a significant amount of manual work. While there are technologies such as photogrammetry that can convert images into 3D models, the quality of the reconstructions often falls short of the desired level of realism. These methods still require substantial manual cleanup and refinement to achieve accurate representations.

The aim of this dissertation is to develop techniques that enables the replication of the three-dimensional world around us. By leveraging advancements in computer vision, graphics, and machine learning, we seek to overcome the limitations of current approaches and create novel methodologies for generating highly realistic and accurate 3D representations.

1.1 Representing the Plenoptic Fuction

The ultimate goal of this dissertation is to develop a representation for the plenoptic function, which, if achieved, would enable novel view synthesis. The plenoptic function encapsulates the complete information about the distribution of light rays passing through every point in a scene. In our context, we aim to represent a 5D version of the plenoptic function, considering both spatial location and viewing direction as dimensions.

To accomplish this, there are a few crucial aspects that need to be addressed: the representation of non-Lambertian effects, the representation of high-resolution details, and the ability to reconstruct

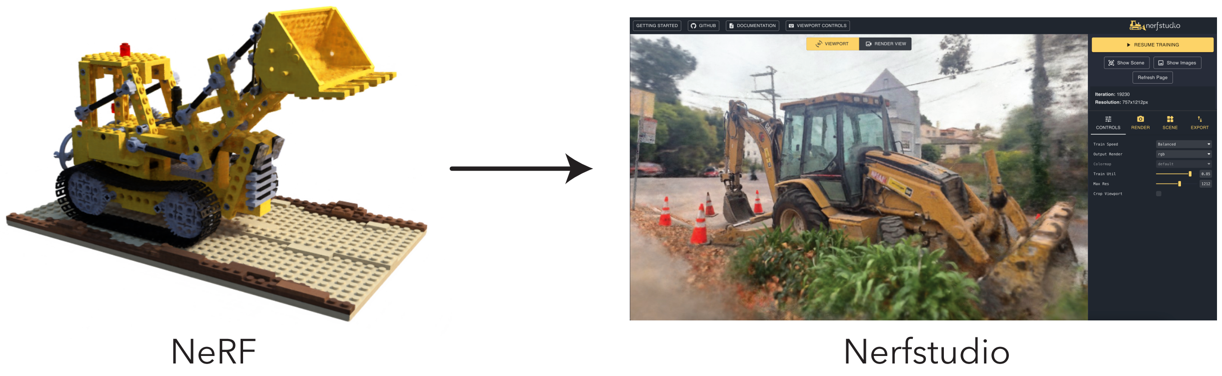


Figure 1.1: This dissertation introduces neural radiance fields (NeRF) as a technique for reconstructing 3D scenes from a collection of posed images. Through subsequent research efforts, we have further enhanced and expanded upon the NeRF framework to improve its functionality and accessibility. The culmination of these advancements is the development of Nerfstudio, a comprehensive framework that enables the reconstruction of real-world unbounded scenes.

from sparse views. These factors pose significant challenges in the field of 3D representation.

Firstly, representing non-Lambertian effects requires the ability to capture and represent different colors and intensities of light as they vary with viewing angles. Lambertian surfaces, which exhibit uniform reflectance regardless of the viewing angle, fail to capture the intricate interplay of light and materials, resulting in less realistic reconstructions. Traditional photogrammetry methods typically make a Lambertain assumption leading to less photorealistic reconstructions, particularly for shiny or transparent objects. Our objective is to develop methods that can accurately model and represent these non-Lambertian effects, enabling us to achieve more faithful and visually compelling 3D reconstructions.

Secondly, achieving high-resolution representations is crucial for capturing fine-grained details and preserving the complexity of real-world scenes. However, handling high-resolution data poses significant memory and computational challenges. Storing data for every viewing direction of every spatial location in a naïve implementation would quickly become memory-intensive, imposing a significant burden on memory resources.

Finally, the task of wide baseline reconstruction from a sparse set of viewpoints presents a formidable challenge. The Nyquist theorem guides us on the necessary sampling rate for signal reconstruction, but achieving such a rate is infeasible in our scenario. To be practically applicable, it becomes imperative to devise methods capable of generating these scenes from tens to hundreds of photos.

To address these challenges, in Chapter 2, we propose a solution that mitigates the memory issue by employing a neural network trained to map a 5D coordinate to both color and density values. This approach enables us to represent a “cloud” of densities and view-dependent colors, which can then be rendered into an image using established volumetric rendering techniques. We refer to this representation as a neural radiance field (NeRF). By representing the scene as a continuous volume, NeRF allows for more efficient optimization compared to surface-based representations. By leveraging volumetric supervision, NeRF exhibits surprising effectiveness in reconstructing scenes from sparse view sets. The incorporation of loss gradients across multiple regions along each camera ray contributes to improved accuracy and convergence during the rendering process. This stands in contrast to surface-based representations, which primarily focus on gradient updates near an object's surface.

By leveraging the power of NeRFs, we aim to overcome the limitations of previous approaches, capturing the richness of non-Lambertian effects, preserving high-resolution details, and wide baseline reconstruction from sparse viewpoints.

Leveraging Technologies

An often overlooked aspect is the ability to leverage accessible technologies and advancements in the field. In the case of this research, we harnessed the power of deep learning and neural networks when developing our representation. By framing our problem within the context of machine learning, we were able to tap into the capabilities offered by modern auto-differentiation libraries. This familiar setup played a crucial role in the widespread adoption of NeRFs within the computer vision community.

Throughout this dissertation, we keep this lesson in mind, emphasizing the importance of abstraction. We recognize that providing accessible and user-friendly frameworks is key to enabling the wider adoption and development of these methods. It is with this motivation that we introduce the Nerfstudio framework in Chapter 6. The goal of Nerfstudio is to streamline and simplify the process of developing NeRF-related methods, making them more accessible to computer vision practitioners. By abstracting away implementation complexities, researchers can focus on pushing the boundaries of the field and exploring novel applications of NeRFs without being hindered by technical barriers.

1.2 Dissertation Overview

This dissertation delves into the development and applications of neural radiance fields (NeRFs), tracing its journey from motivation and development to the creation of Nerfstudio, a comprehensive framework crafted to streamline NeRF usage. In addition to these key contributions, this work delves into the theoretical aspects of NeRFs, providing a deeper understanding, and explores various advancements aimed at extending the functionality and capabilities of NeRFs. Through these endeavors, this dissertation presents a holistic approach to NeRF research, encompassing theory, development, and advancements, with the overarching goal of advancing the field of neural rendering and view synthesis. It is important to emphasize that the full appreciation of many results requires experiencing them through videos. For videos and supplementary results, please visit .

Chapter 2

This chapter the key novel method of this dissertation, NeRFs, for synthesizing new views of complex scenes. The approach involves optimizing a volumetric scene function using a sparse set of input views. The scene is represented by a fully-connected deep network that takes a single 5D coordinate (including spatial location and viewing direction) as input and produces the volume density and view-dependent emitted radiance at that location. To synthesize new views, the method queries 5D coordinates along camera rays and uses classic volume rendering techniques to project the output colors and densities into an image. Because volume rendering is inherently differentiable, the method requires only a set of images with known camera poses as input to optimize the representation. We explain how to effectively optimize neural radiance fields to render photorealistic novel views of scenes with complex geometry and appearance, and demonstrate results that surpass previous work in the field of neural rendering and view synthesis.

Chapter 3

This chapter explores the application of a simple Fourier feature mapping to enhance the representation of high-frequency details in NeRFs. Specifically, the aim is to enable multilayer perceptrons (MLPs) to effectively learn high-frequency functions in low-dimensional problem domains, which are referred to as coordinate-based neural networks. Through insights drawn from the neural tangent kernel (NTK) literature, the chapter highlights the inherent slow convergence of standard MLPs when it comes to high-frequency signal components, owing to their spectral bias. To overcome this limitation, the chapter proposes the utilization of a Fourier feature mapping, which transforms the effective NTK into a stationary kernel with an adjustable bandwidth. Furthermore, an approach for selecting problem-specific Fourier features is suggested, greatly improving the performance of MLPs in low-dimensional regression tasks that are relevant to the fields of computer vision and graphics.

Chapter 4

In the context of optimizing coordinate-based networks, the traditional approach of initializing weights randomly for each new signal proves to be inefficient. To address this challenge, this chapter proposes a solution that involves applying standard meta-learning algorithms to learn the initial weight parameters of fully-connected networks based on the specific class of signals being represented. For instance, this approach can be tailored for signals such as images of faces or 3D models of chairs. By incorporating minor modifications into the implementation process, utilizing learned initial weights offers several advantages. It facilitates faster convergence during optimization, acts as a strong prior over the signal class being modeled, and enhances generalization even when only partial observations of a given signal are available. The chapter investigates the benefits of this technique across various tasks, encompassing 2D image representation, CT scan reconstruction, and the recovery of 3D shapes and scenes from 2D image observations.

Chapter 5

This chapter details the exploration of scaling NeRFs for large environments. The proposed Block-NeRF method decomposes city-scale scenes into multiple NeRFs that are individually trained. This decomposition allows for scaling rendering to arbitrarily large environments while decoupling rendering time from scene size. To ensure robustness to data captured over long periods and under varying environmental conditions, several architectural changes are adopted, such as incorporating appearance embeddings, learned pose refinement, and controllable exposure to each individual NeRF. The chapter also introduces a process for aligning the appearance between adjacent NeRFs, enabling seamless combination. The research resulted in the creation of a grid of Block-NeRFs using 2.8 million images representing a neighborhood in San Francisco.

Chapter 6

This chapter introduces Nerfstudio, a modular PyTorch framework designed to facilitate the development and deployment of NeRF-based methods. Nerfstudio provides plug-and-play components that make it easy for researchers and practitioners to integrate NeRF into their projects. Its modular design supports real-time visualization tools, streamlined pipelines for importing captured in-the-wild data, and tools for exporting to video, point cloud, and mesh representations. Nerfstudio's modularity also allows the development of Nerfacto, a method that balances speed and quality by combining components from recent papers while remaining flexible for future modifications. All associated code and data are publicly available with open-source licensing, encouraging community-driven development.

Chapter 7

In this concluding chapter, we delve into the insights gleaned from the preceding chapters. We examine the remaining limitations in scene reconstructions and offer perspectives on the future directions that this research can be taken.

Chapter 2

Neural Radiance Fields

In this chapter, we address the long-standing problem of novel view synthesis given a set of captured images. We introduce a novel approach that directly optimizes the parameters of a continuous 5D volumetric scene representation to minimize the error between rendered images and captured images of the scene.

We represent a static scene as a continuous 5D function that outputs the radiance emitted in each direction (θ, ϕ) at each point (x, y, z) in space, and a density at each point which acts like a differential opacity controlling how much radiance is accumulated by a ray passing through (x, y, z). Our method optimizes a deep fully-connected neural network without any convolutional layers (often referred to as a multilayer perceptron or MLP) to represent this function by regressing from a single 5D coordinate (x, y, z, θ, ϕ) to a single volume density and view-dependent RGB color. To render this *neural radiance field* (NeRF) from a particular viewpoint we: 1) march camera rays through the scene to generate a sampled set of 3D points, 2) use those points and their corresponding 2D viewing directions as input to the neural network to produce an output set of colors and densities, and 3) use classical volume rendering techniques to accumulate those colors and densities into a 2D image. Because this process is naturally differentiable, we can use gradient descent to optimize this model by minimizing the error between each observed image and the corresponding views rendered from our representation. Minimizing this error across multiple views encourages the network to predict a coherent model of the scene by assigning high volume densities and accurate colors to the locations that contain the true underlying scene content. Figure 2.2 visualizes this overall pipeline.

We find that the basic implementation of optimizing a neural radiance field representation for a complex scene does not converge to a sufficiently high-resolution representation and is inefficient in the required number of samples per camera ray. We address these issues by transforming input 5D coordinates with a positional encoding that enables the MLP to represent higher frequency functions, and we propose a hierarchical sampling procedure to reduce the number of queries required to adequately sample this high-frequency scene representation.

Our approach inherits the benefits of volumetric representations: both can represent complex real-world geometry and appearance and are well suited for gradient-based optimization using

This chapter is based on joint work published at ECCV 2020 [123]

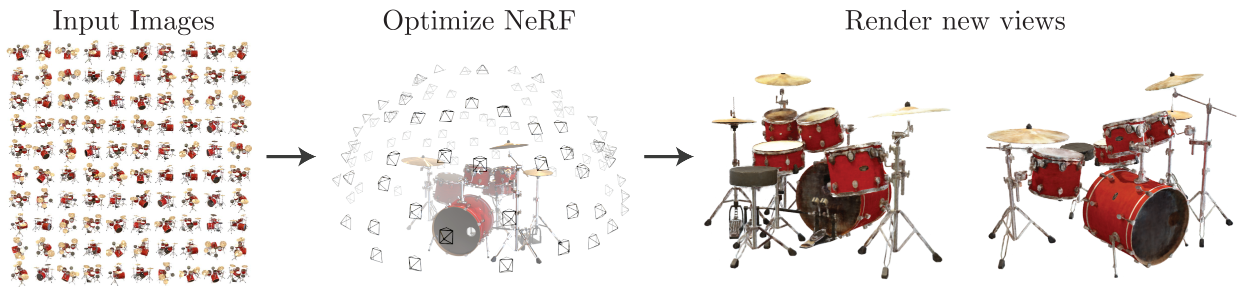


Figure 2.1: We present a method that optimizes a continuous 5D neural radiance field representation (volume density and view-dependent color at any continuous location) of a scene from a set of input images. We use techniques from volume rendering to accumulate samples of this scene representation along rays to render the scene from any viewpoint. Here, we visualize the set of 100 input views of the synthetic *Drums* scene randomly captured on a surrounding hemisphere, and we show two novel views rendered from our optimized NeRF representation.

projected images. Crucially, our method overcomes the prohibitive storage costs of *discretized* voxel grids when modeling complex scenes at high-resolutions by utilizing a compact neural network representation, requiring only a few megabytes of memory instead of gigabytes. In summary, our technical contributions are:

- An approach for representing continuous scenes with complex geometry and materials as 5D neural radiance fields, parameterized as basic MLP networks.
- A differentiable rendering procedure based on classical volume rendering techniques, which we use to optimize these representations from standard RGB images. This includes a hierarchical sampling strategy to allocate the MLP's capacity towards space with visible scene content.
- A positional encoding to map each input 5D coordinate into a higher dimensional space, which enables us to successfully optimize neural radiance fields to represent high-frequency scene content.

We demonstrate that our resulting neural radiance field method quantitatively and qualitatively outperforms state-of-the-art view synthesis methods, including works that fit neural 3D representations to scenes as well as works that train deep convolutional networks to predict sampled volumetric representations. As far as we know, this paper presents the first continuous neural scene representation that is able to render high-resolution photorealistic novel views of real objects and scenes from RGB images captured in natural settings.

2.1 Related Work

A promising recent direction in computer vision is encoding objects and scenes in the weights of an MLP that directly maps from a 3D spatial location to an implicit representation of the shape, such as the signed distance [31] at that location. However, these methods have so far been unable to reproduce realistic scenes with complex geometry with the same fidelity as techniques that represent scenes using discrete representations such as triangle meshes or voxel grids. In this section, we review these two lines of work and contrast them with our approach, which enhances the capabilities of neural scene representations to produce state-of-the-art results for rendering complex realistic scenes.

A similar approach of using MLPs to map from low-dimensional coordinates to colors has also been used for representing other graphics functions such as images [184], textured materials [65, 135, 152, 151], and indirect illumination values [159].

Neural 3D shape representations Recent work has investigated the implicit representation of continuous 3D shapes as level sets by optimizing deep networks that map xyz coordinates to signed distance functions [76, 137] or occupancy fields [54, 115]. However, these models are limited by their requirement of access to ground truth 3D geometry, typically obtained from synthetic 3D shape datasets such as ShapeNet [21]. Subsequent work has relaxed this requirement of ground truth 3D shapes by formulating differentiable rendering functions that allow neural implicit shape representations to be optimized using only 2D images. Niemeyer *et al.* [130] represent surfaces as 3D occupancy fields and use a numerical method to find the surface intersection for each ray, then calculate an exact derivative using implicit differentiation. Each ray intersection location is provided as the input to a neural 3D texture field that predicts a diffuse color for that point. Sitzmann *et al.* [177] use a less direct neural 3D representation that simply outputs a feature vector and RGB color at each continuous 3D coordinate, and propose a differentiable rendering function consisting of a recurrent neural network that marches along each ray to decide where the surface is located.

Though these techniques can potentially represent complicated and high-resolution geometry, they have so far been limited to simple shapes with low geometric complexity, resulting in oversmoothed renderings. We show that an alternate strategy of optimizing networks to encode 5D radiance fields (3D volumes with 2D view-dependent appearance) can represent higher-resolution geometry and appearance to render photorealistic novel views of complex scenes.

View synthesis and image-based rendering Given a dense sampling of views, photorealistic novel views can be reconstructed by simple light field sample interpolation techniques [93, 28, 32]. For novel view synthesis with sparser view sampling, the computer vision and graphics communities have made significant progress by predicting traditional geometry and appearance representations from observed images. One popular class of approaches uses mesh-based representations of scenes with either diffuse [201] or view-dependent [17, 33, 209] appearance. Differentiable rasterizers [24, 56, 106, 109] or pathtracers [95, 132] can directly optimize mesh representations to reproduce a set of input images using gradient descent. However, gradient-based mesh optimization based on

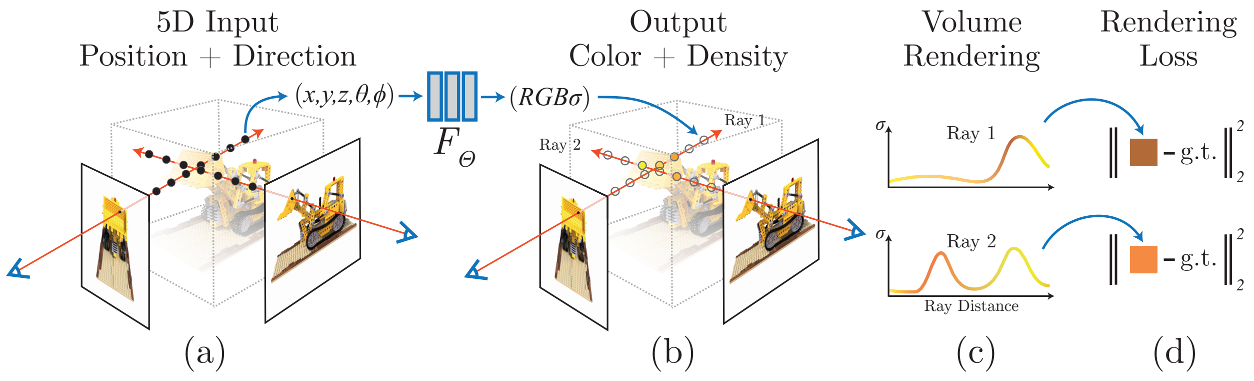


Figure 2.2: An overview of our neural radiance field scene representation and differentiable rendering procedure. We synthesize images by sampling 5D coordinates (location and viewing direction) along camera rays (a), feeding those locations into an MLP to produce a color and volume density (b), and using volume rendering techniques to composite these values into an image (c). This rendering function is differentiable, so we can optimize our scene representation by minimizing the residual between synthesized and ground truth observed images (d).

image reprojection is often difficult, likely because of local minima or poor conditioning of the loss landscape. Furthermore, this strategy requires a template mesh with fixed topology to be provided as an initialization before optimization [95], which is typically unavailable for unconstrained real-world scenes.

Another class of methods use volumetric representations to address the task of high-quality photorealistic view synthesis from a set of input RGB images. Volumetric approaches are able to realistically represent complex shapes and materials, are well-suited for gradient-based optimization, and tend to produce less visually distracting artifacts than mesh-based methods. Early volumetric approaches used observed images to directly color voxel grids [90, 173, 189]. More recently, several methods [44, 66, 79, 119, 141, 183, 198, 230] have used large datasets of multiple scenes to train deep networks that predict a sampled volumetric representation from a set of input images, and then use either alpha-compositing [144] or learned compositing along rays to render novel views at test time. Other works have optimized a combination of convolutional networks (CNNs) and sampled voxel grids for each specific scene, such that the CNN can compensate for discretization artifacts from low resolution voxel grids [178] or allow the predicted voxel grids to vary based on input time or animation controls [108]. While these volumetric techniques have achieved impressive results for novel view synthesis, their ability to scale to higher resolution imagery is fundamentally limited by poor time and space complexity due to their discrete sampling — rendering higher resolution images requires a finer sampling of 3D space. We circumvent this problem by instead encoding a *continuous* volume within the parameters of a deep fully-connected neural network, which not only produces significantly higher quality renderings than prior volumetric approaches, but also requires just a fraction of the storage cost of those *sampled* volumetric representations.

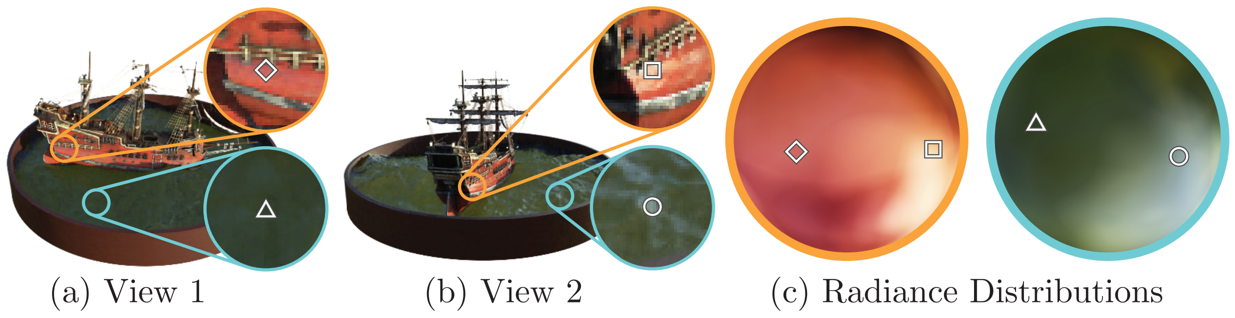


Figure 2.3: A visualization of view-dependent emitted radiance. Our neural radiance field representation outputs RGB color as a 5D function of both spatial position $\mathbf{x}$ and viewing direction $\mathbf{d}$. Here, we visualize example directional color distributions for two spatial locations in our neural representation of the *Ship* scene. In (a) and (b), we show the appearance of two fixed 3D points from two different camera positions: one on the side of the ship (orange insets) and one on the surface of the water (blue insets). Our method predicts the changing specular appearance of these two 3D points, and in (c) we show how this behavior generalizes continuously across the whole hemisphere of viewing directions.

2.2 Neural Radiance Field Scene Representation

We represent a continuous scene as a 5D vector-valued function whose input is a 3D location $\mathbf{x} = (x, y, z)$ and 2D viewing direction (θ, ϕ), and whose output is an emitted color $\mathbf{c} = (r, g, b)$ and volume density σ. In practice, we express direction as a 3D Cartesian unit vector $\mathbf{d}$. We approximate this continuous 5D scene representation with an MLP network $F_\Theta : (\mathbf{x}, \mathbf{d}) \to (\mathbf{c}, \sigma)$ and optimize its weights Θ to map from each input 5D coordinate to its corresponding volume density and directional emitted color.

We encourage the representation to be multiview consistent by restricting the network to predict the volume density σ as a function of only the location $\mathbf{x}$, while allowing the RGB color $\mathbf{c}$ to be predicted as a function of both location and viewing direction. To accomplish this, the MLP F_Θ first processes the input 3D coordinate $\mathbf{x}$ with 8 fully-connected layers (using ReLU activations and 256 channels per layer), and outputs σ and a 256-dimensional feature vector. This feature vector is then concatenated with the camera ray's viewing direction and passed to one additional fully-connected layer (using a ReLU activation and 128 channels) that output the view-dependent RGB color.

See Fig. 2.3 for an example of how our method uses the input viewing direction to represent non-Lambertian effects. As shown in Fig. 2.4, a model trained without view dependence (only $\mathbf{x}$ as input) has difficulty representing specularities.

2.3 Volume Rendering with Radiance Fields

Our 5D neural radiance field represents a scene as the volume density and directional emitted radiance at any point in space. We render the color of any ray passing through the scene using principles from classical volume rendering [78]. The volume density $\sigma(\mathbf{x})$ can be interpreted as the differential probability of a ray terminating at an infinitesimal particle at location $\mathbf{x}$. The expected color $C(\mathbf{r})$ of camera ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ with near and far bounds t_n and t_f is:

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

The function $T(t)$ denotes the accumulated transmittance along the ray from t_n to t, *i.e.*, the probability that the ray travels from t_n to t without hitting any other particle. Rendering a view from our continuous neural radiance field requires estimating this integral $C(\mathbf{r})$ for a camera ray traced through each pixel of the desired virtual camera.

We numerically estimate this continuous integral using quadrature. Deterministic quadrature, which is typically used for rendering discretized voxel grids, would effectively limit our representation's resolution because the MLP would only be queried at a fixed discrete set of locations. Instead, we use a stratified sampling approach where we partition $[t_n, t_f]$ into N evenly-spaced bins and then draw one sample uniformly at random from within each bin:

$$t_i \sim \mathcal{U}\left[t_n + \frac{i-1}{N}(t_f - t_n),\ t_n + \frac{i}{N}(t_f - t_n)\right]. \tag{2.2}$$

Although we use a discrete set of samples to estimate the integral, stratified sampling enables us to represent a continuous scene representation because it results in the MLP being evaluated at continuous positions over the course of optimization. We use these samples to estimate $C(\mathbf{r})$ with the quadrature rule discussed in the volume rendering review by Max [113]:

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

where $\delta_i = t_{i+1} - t_i$ is the distance between adjacent samples. This function for calculating $\hat{C}(\mathbf{r})$ from the set of $(\mathbf{c}_i, \sigma_i)$ values is trivially differentiable and reduces to traditional alpha compositing with alpha values $\alpha_i = 1 - \exp(-\sigma_i\delta_i)$.

2.4 Optimizing a Neural Radiance Field

In the previous section we have described the core components necessary for modeling a scene as a neural radiance field and rendering novel views from this representation. However, we observe that these components are not sufficient for achieving state-of-the-art quality, as demonstrated in Section 2.5). We introduce two improvements to enable representing high-resolution complex

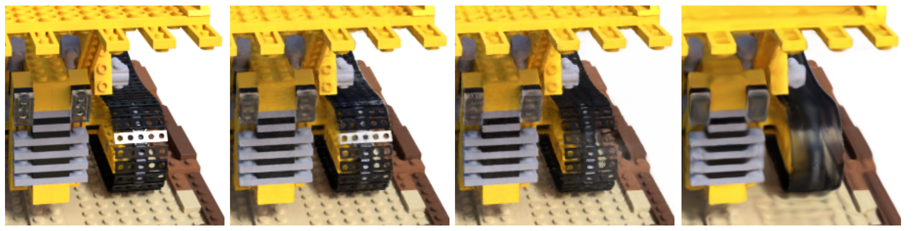


Figure 2.4: Here we visualize how our full model benefits from representing view-dependent emitted radiance and from passing our input coordinates through a high-frequency positional encoding. Removing view dependence prevents the model from recreating the specular reflection on the bulldozer tread. Removing the positional encoding drastically decreases the model's ability to represent high frequency geometry and texture, resulting in an oversmoothed appearance.

scenes. The first is a positional encoding of the input coordinates that assists the MLP in representing high-frequency functions, and the second is a hierarchical sampling procedure that allows us to efficiently sample this high-frequency representation.

Positional encoding

Despite the fact that neural networks are universal function approximators [67], we found that having the network F_Θ directly operate on $xyz\theta\phi$ input coordinates results in renderings that perform poorly at representing high-frequency variation in color and geometry. This is consistent with recent work by Rahaman *et al.* [148], which shows that deep networks are biased towards learning lower frequency functions. They additionally show that mapping the inputs to a higher dimensional space using high frequency functions before passing them to the network enables better fitting of data that contains high frequency variation.

We leverage these findings in the context of neural scene representations, and show that reformulating F_Θ as a composition of two functions $F_\Theta = F'_\Theta \circ \gamma$, one learned and one not, significantly improves performance (see Fig. 2.4 and Table 2.2). Here γ is a mapping from $\mathbb{R}$ into a higher dimensional space $\mathbb{R}^{2L}$, and F'_Θ is still simply a regular MLP. Formally, the encoding function we use is:

$$\gamma(p) = \left(\sin(2^0 \pi p), \quad \cos(2^0 \pi p), \quad \cdots, \quad \sin\big(2^{L-1}\pi p\big), \quad \cos\big(2^{L-1}\pi p\big) \right). \tag{2.4}$$

This function $\gamma(\cdot)$ is applied separately to each of the three coordinate values in $\mathbf{x}$ (which are normalized to lie in $[-1, 1]$) and to the three components of the Cartesian viewing direction unit vector $\mathbf{d}$ (which by construction lie in $[-1, 1]$). In our experiments, we set $L = 10$ for $\gamma(\mathbf{x})$ and $L = 4$ for $\gamma(\mathbf{d})$.

A similar mapping is used in the popular Transformer architecture [199], where it is referred to as a *positional encoding*. However, Transformers use it for a different goal of providing the discrete positions of tokens in a sequence as input to an architecture that does not contain any notion of order. In contrast, we use these functions to map continuous input coordinates into a higher dimensional space to enable our MLP to more easily approximate a higher frequency function. Concurrent work on a related problem of modeling 3D protein structure from projections [229] also utilizes a similar input coordinate mapping.

We investigate positional encoding further in chapter 3.

Hierarchical volume sampling

Our rendering strategy of densely evaluating the neural radiance field network at N query points along each camera ray is inefficient: free space and occluded regions that do not contribute to the rendered image are still sampled repeatedly. We draw inspiration from early work in volume rendering [92] and propose a hierarchical representation that increases rendering efficiency by allocating samples proportionally to their expected effect on the final rendering.

Instead of just using a single network to represent the scene, we simultaneously optimize two networks: one "coarse" and one "fine". We first sample a set of N_c locations using stratified sampling, and evaluate the "coarse" network at these locations as described in Eqns. 2.2 and 2.3. Given the output of this "coarse" network, we then produce a more informed sampling of points along each ray where samples are biased towards the relevant parts of the volume. To do this, we first rewrite the alpha composited color from the coarse network $\hat{C}_c(\mathbf{r})$ in Eqn. 2.3 as a weighted sum of all sampled colors c_i along the ray:

$$\hat{C}_c(\mathbf{r}) = \sum_{i=1}^{N_c} w_i c_i \,, \qquad w_i = T_i(1 - \exp(-\sigma_i \delta_i)) \,. \tag{2.5}$$

Normalizing these weights as $\hat{w}_i = {}^{w_i}/_{\sum_{j=1}^{N_c} w_j}$ produces a piecewise-constant PDF along the ray. We sample a second set of N_f locations from this distribution using inverse transform sampling, evaluate our "fine" network at the union of the first and second set of samples, and compute the final rendered color of the ray $\hat{C}_f(\mathbf{r})$ using Eqn. 2.3 but using all $N_c + N_f$ samples. This procedure allocates more samples to regions we expect to contain visible content. This addresses a similar goal as importance sampling, but we use the sampled values as a nonuniform discretization of the whole integration domain rather than treating each sample as an independent probabilistic estimate of the entire integral.

Implementation details

We optimize a separate neural continuous volume representation network for each scene. This requires only a dataset of captured RGB images of the scene, the corresponding camera poses and intrinsic parameters, and scene bounds (we use ground truth camera poses, intrinsics, and bounds for synthetic data, and use the COLMAP structure-from-motion package [170] to estimate these parameters for real data). At each optimization iteration, we randomly sample a batch of camera rays from the set of all pixels in the dataset, and then follow the hierarchical sampling described in Sec. 2.4 to query N_c samples from the coarse network and $N_c + N_f$ samples from the fine network. We then use the volume rendering procedure described in Sec. 2.3 to render the color of each ray from both sets of samples. Our loss is simply the total squared error between the rendered and true pixel colors for both the coarse and fine renderings:

$$\mathcal{L} = \sum_{\mathbf{r}\in\mathcal{R}} \left[\left\| \hat{C}_c(\mathbf{r}) - C(\mathbf{r}) \right\|_2^2 + \left\| \hat{C}_f(\mathbf{r}) - C(\mathbf{r}) \right\|_2^2 \right] \tag{2.6}$$

where $\mathcal{R}$ is the set of rays in each batch, and $C(\mathbf{r})$, $\hat{C}_c(\mathbf{r})$, and $\hat{C}_f(\mathbf{r})$ are the ground truth, coarse volume predicted, and fine volume predicted RGB colors for ray $\mathbf{r}$ respectively. Note that even though the final rendering comes from $\hat{C}_f(\mathbf{r})$, we also minimize the loss of $\hat{C}_c(\mathbf{r})$ so that the weight distribution from the coarse network can be used to allocate samples in the fine network.

In our experiments, we use a batch size of 4096 rays, each sampled at $N_c = 64$ coordinates in the coarse volume and $N_f = 128$ additional coordinates in the fine volume. We use the Adam optimizer [86] with a learning rate that begins at 5×10^{-4} and decays exponentially to 5×10^{-5} over the course of optimization (other Adam hyperparameters are left at default values of $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-7}$). The optimization for a single scene typically take around 100–300k iterations to converge on a single NVIDIA V100 GPU (about 1–2 days).

2.5 Results

We quantitatively (Tables 2.1) and qualitatively (Figs. 2.5 and 2.6) show that our method outperforms prior work, and provide extensive ablation studies to validate our design choices (Table 2.2).

Datasets

Synthetic renderings of objects We first show experimental results on two datasets of synthetic renderings of objects (Table 2.1, "Diffuse Synthetic $360°$" and "Realistic Synthetic $360°$"). The DeepVoxels [178] dataset contains four Lambertian objects with simple geometry. Each object is rendered at 512×512 pixels from viewpoints sampled on the upper hemisphere (479 as input and 1000 for testing). We additionally generate our own dataset containing pathtraced images of eight objects that exhibit complicated geometry and realistic non-Lambertian materials. Six are rendered from viewpoints sampled on the upper hemisphere, and two are rendered from viewpoints sampled on a full sphere. We render 100 views of each scene as input and 200 for testing, all at 800×800 pixels.

Real images of complex scenes We show results on complex real-world scenes captured with roughly forward-facing images (Table 2.1, "Real Forward-Facing"). This dataset consists of 8 scenes captured with a handheld cellphone (5 taken from the LLFF paper and 3 that we capture), captured with 20 to 62 images, and hold out $1/8$ of these for the test set. All images are 1008×756 pixels.

	Diffuse Synthetic 360° [178]			Realistic Synthetic 360°			Real Forward-Facing [119]		
Method	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
SRN [177]	33.20	0.963	0.073	22.26	0.846	0.170	22.84	0.668	0.378
NV [108]	29.62	0.929	0.099	26.05	0.893	0.160	-	-	-
LLFF [119]	34.38	0.985	0.048	24.88	0.911	0.114	24.13	0.798	**0.212**
Ours	**40.15**	**0.991**	**0.023**	**31.01**	**0.947**	**0.081**	**26.50**	**0.811**	0.250

Table 2.1: Our method quantitatively outperforms prior work on datasets of both synthetic and real images. We report PSNR/SSIM (higher is better) and LPIPS [228] (lower is better). The DeepVoxels [178] dataset consists of 4 diffuse objects with simple geometry. Our realistic synthetic dataset consists of pathtraced renderings of 8 geometrically complex objects with complex non-Lambertian materials. The real dataset consists of handheld forward-facing captures of 8 real-world scenes (NV cannot be evaluated on this data because it only reconstructs objects inside a bounded volume). Though LLFF achieves slightly better LPIPS, we urge readers to view our supplementary video where our method achieves better multiview consistency and produces fewer artifacts than all baselines.

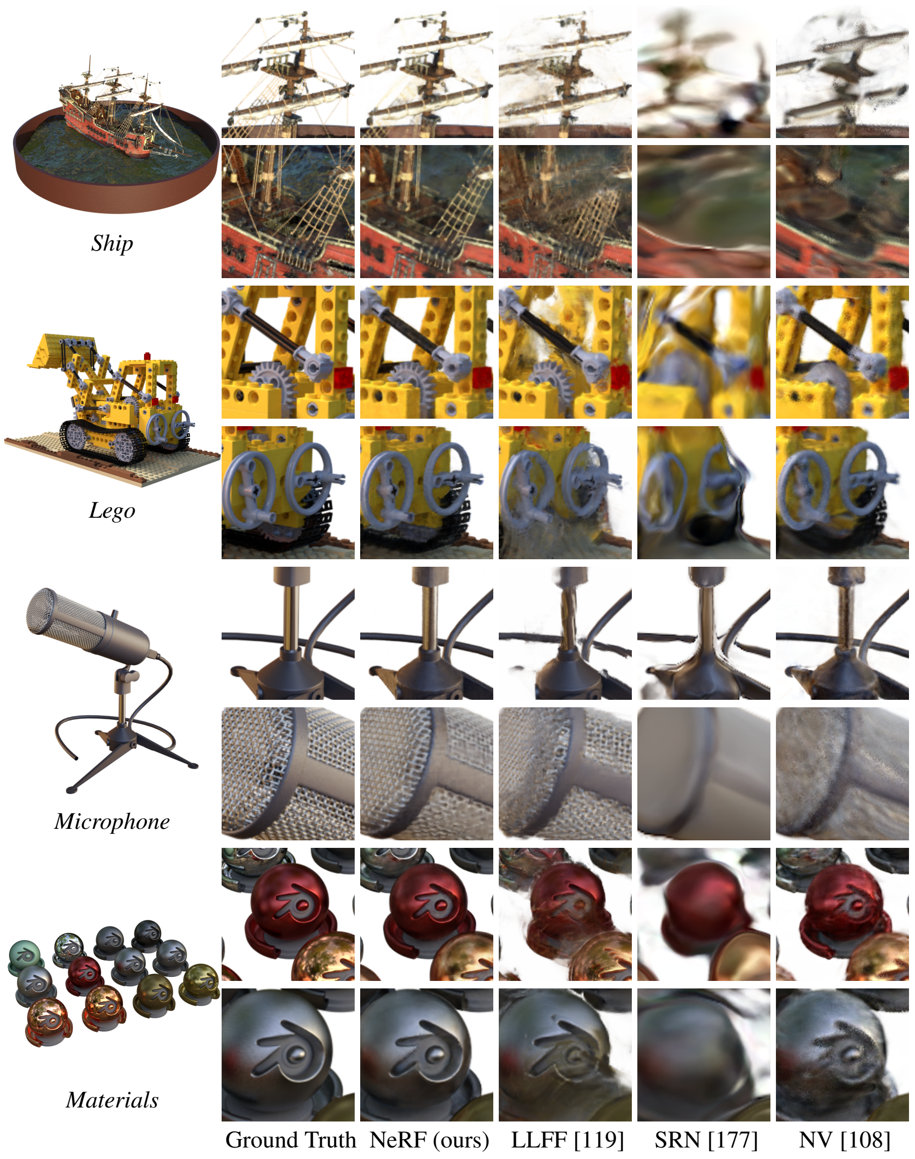


Figure 2.5: Comparisons on test-set views for scenes from our new synthetic dataset generated with a physically-based renderer. Our method is able to recover fine details in both geometry and appearance, such as *Ship*'s rigging, *Lego*'s gear and treads, *Microphone*'s shiny stand and mesh grille, and *Material*'s non-Lambertian reflectance. LLFF exhibits banding artifacts on the *Microphone* stand and *Material*'s object edges and ghosting artifacts in *Ship*'s mast and inside the *Lego* object. SRN produces blurry and distorted renderings in every case. Neural Volumes cannot capture the details on the *Microphone*'s grille or *Lego*'s gears, and it completely fails to recover the geometry of *Ship*'s rigging.

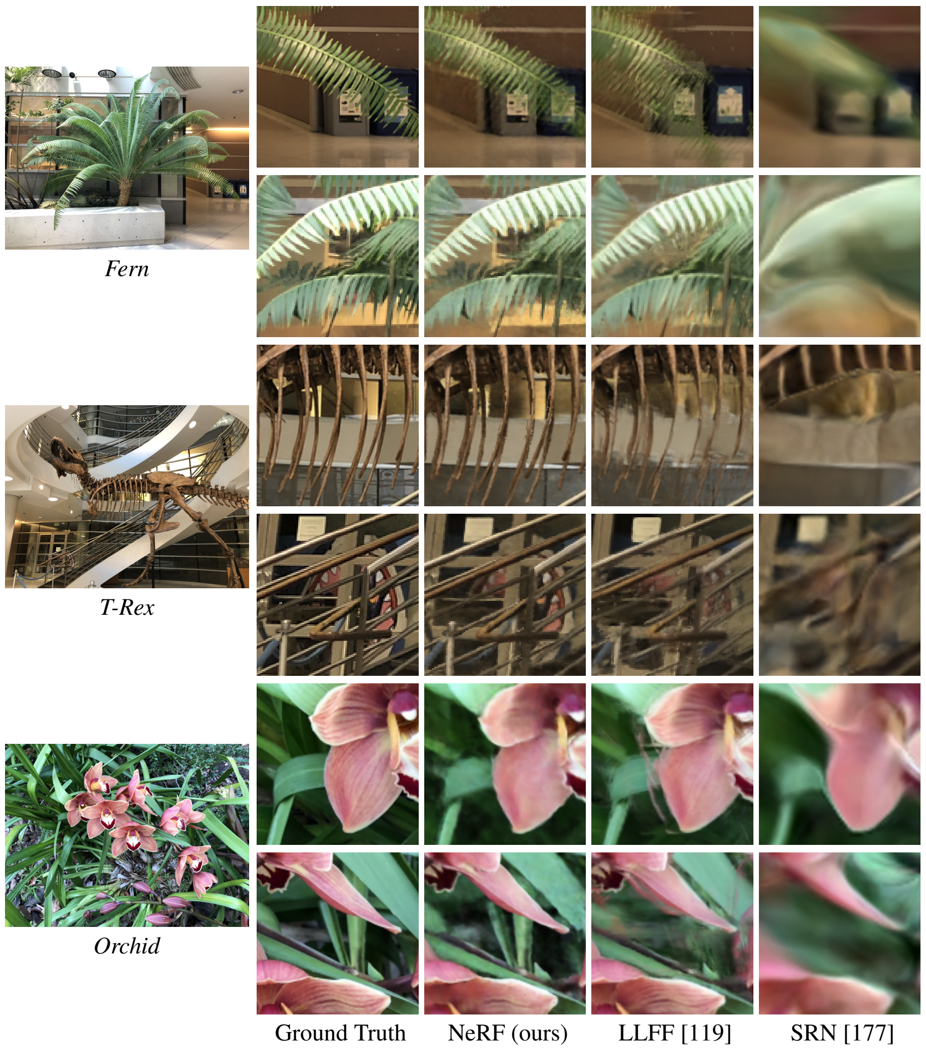


Figure 2.6: Comparisons on test-set views of real world scenes. LLFF is specifically designed for this use case (forward-facing captures of real scenes). Our method is able to represent fine geometry more consistently across rendered views than LLFF, as shown in *Fern*'s leaves and the skeleton ribs and railing in *T-rex*. Our method also correctly reconstructs partially occluded regions that LLFF struggles to render cleanly, such as the yellow shelves behind the leaves in the bottom *Fern* crop and green leaves in the background of the bottom *Orchid* crop. Blending between multiples renderings can also cause repeated edges in LLFF, as seen in the top *Orchid* crop. SRN captures the low-frequency geometry and color variation in each scene but is unable to reproduce any fine detail.

Comparisons

To evaluate our model we compare against current top-performing techniques for view synthesis, detailed below. All methods use the same set of input views to train a separate network for each scene except Local Light Field Fusion [119], which trains a single 3D convolutional network on a large dataset, then uses the same trained network to process input images of new scenes at test time.

Neural Volumes (NV) [108] synthesizes novel views of objects that lie entirely within a bounded volume in front of a distinct background (which must be separately captured without the object of interest). It optimizes a deep 3D convolutional network to predict a discretized RGBα voxel grid with 128^3 samples as well as a 3D warp grid with 32^3 samples. The algorithm renders novel views by marching camera rays through the warped voxel grid.

Scene Representation Networks (SRN) [177] represent a continuous scene as an opaque surface, implicitly defined by a MLP that maps each (x, y, z) coordinate to a feature vector. They train a recurrent neural network to march along a ray through the scene representation by using the feature vector at any 3D coordinate to predict the next step size along the ray. The feature vector from the final step is decoded into a single color for that point on the surface. Note that SRN is a better-performing followup to DeepVoxels [178] by the same authors, which is why we do not include comparisons to DeepVoxels.

Local Light Field Fusion (LLFF) [119] LLFF is designed for producing photorealistic novel views for well-sampled forward facing scenes. It uses a trained 3D convolutional network to directly predict a discretized frustum-sampled RGBα grid (multiplane image or MPI [230]) for each input view, then renders novel views by alpha compositing and blending nearby MPIs into the novel viewpoint.

Discussion

We thoroughly outperform both baselines that also optimize a separate network per scene (NV and SRN) in all scenarios. Furthermore, we produce qualitatively and quantitatively superior renderings compared to LLFF (across all except one metric) while using only their input images as our entire training set.

The SRN method produces heavily smoothed geometry and texture, and its representational power for view synthesis is limited by selecting only a single depth and color per camera ray. The NV baseline is able to capture reasonably detailed volumetric geometry and appearance, but its use of an underlying explicit 128^3 voxel grid prevents it from scaling to represent fine details at high resolutions. LLFF specifically provides a “sampling guideline” to not exceed 64 pixels of disparity between input views, so it frequently fails to estimate correct geometry in the synthetic datasets which contain up to 400-500 pixels of disparity between views. Additionally, LLFF blends between different scene representations for rendering different views, resulting in perceptually-distracting inconsistency.

	Input	#Im.	L	(N_c , N_f)	PSNR↑	SSIM↑	LPIPS↓
1) No PE, VD, H	xyz	100	-	(256, -)	26.67	0.906	0.136
2) No Pos. Encoding	$xyz\theta\phi$	100	-	(64, 128)	28.77	0.924	0.108
3) No View Dependence	xyz	100	10	(64, 128)	27.66	0.925	0.117
4) No Hierarchical	$xyz\theta\phi$	100	10	(256, -)	30.06	0.938	0.109
5) Far Fewer Images	$xyz\theta\phi$	25	10	(64, 128)	27.78	0.925	0.107
6) Fewer Images	$xyz\theta\phi$	50	10	(64, 128)	29.79	0.940	0.096
7) Fewer Frequencies	$xyz\theta\phi$	100	5	(64, 128)	30.59	0.944	0.088
8) More Frequencies	$xyz\theta\phi$	100	15	(64, 128)	30.81	0.946	0.096
9) Complete Model	$xyz\theta\phi$	100	10	(64, 128)	$\mathbf{31.01}$	$\mathbf{0.947}$	$\mathbf{0.081}$

Table 2.2: An ablation study of our model. Metrics are averaged over the 8 scenes from our realistic synthetic dataset. See Sec. 2.5 for detailed descriptions.

The biggest practical tradeoffs between these methods are time versus space. All compared single scene methods take at least 12 hours to train per scene. In contrast, LLFF can process a small input dataset in under 10 minutes. However, LLFF produces a large 3D voxel grid for every input image, resulting in enormous storage requirements (over 15GB for one "Realistic Synthetic" scene). Our method requires only 5 MB for the network weights (a relative compression of $3000\times$ compared to LLFF), which is even less memory than the *input images alone* for a single scene from any of our datasets.

Ablation studies

We validate our algorithm's design choices and parameters with an extensive ablation study in Table 2.2. We present results on our "Realistic Synthetic $360°$" scenes. Row 9 shows our complete model as a point of reference. Row 1 shows a minimalist version of our model without positional encoding (PE), view-dependence (VD), or hierarchical sampling (H). In rows 2–4 we remove these three components one at a time from the full model, observing that positional encoding (row 2) and view-dependence (row 3) provide the largest quantitative benefit followed by hierarchical sampling (row 4). Rows 5–6 show how our performance decreases as the number of input images is reduced. Note that our method's performance using only 25 input images still exceeds NV, SRN, and LLFF across all metrics when they are provided with 100 images. In rows 7–8 we validate our choice of the maximum frequency L used in our positional encoding for $\mathbf{x}$ (the maximum frequency used for $\mathbf{d}$ is scaled proportionally). Only using 5 frequencies reduces performance, but increasing the number of frequencies from 10 to 15 does not improve performance. We believe the benefit of increasing L is limited once 2^L exceeds the maximum frequency present in the sampled input images (roughly 1024 in our data).

2.6 Discussion

This chapter addresses deficiencies of prior work that uses MLPs to represent objects and scenes as continuous functions. We demonstrate that representing scenes as 5D neural radiance fields (an MLP that outputs volume density and view-dependent emitted radiance as a function of 3D location and 2D viewing direction) produces better renderings than the previously-dominant approach of training deep convolutional networks to output discretized voxel representations.

Although we have proposed a hierarchical sampling strategy to make rendering more sample-efficient (for both training and testing), there is still much more progress to be made in investigating techniques to efficiently optimize and render neural radiance fields. Another direction for future work is interpretability: sampled representations such as voxel grids and meshes admit reasoning about the expected quality of rendered views and failure modes, but it is unclear how to analyze these issues when we encode scenes in the weights of a deep neural network. We believe that this work makes progress towards a graphics pipeline based on real world imagery, where complex scenes could be composed of neural radiance fields optimized from images of actual objects and scenes.

Chapter 3

Representing High Frequencies in Coordinate-Based Networks

One of the fundamental aspects that facilitated NeRF's ability to capture high-frequency details, as elaborated in Chapter 2, was the positional encoding of network inputs. This technique is not exclusive to NeRF and can be extended to any "coordinate-based" MLP which takes low-dimensional coordinates as inputs (typically points in $\mathbb{R}^3$) and are trained to output a representation of shape, density, and/or color at each input location (see Figure 3.1). This strategy is compelling since coordinate-based MLPs are amenable to gradient-based optimization and machine learning, and can be orders of magnitude more compact than grid-sampled representations. Coordinate-based MLPs have been used to represent images [128, 184] (referred to as "compositional pattern producing networks"), volume density [123], occupancy [115], and signed distance [137], and have achieved state-of-the-art results across a variety of tasks such as shape representation [26, 34, 53, 55, 76, 118, 137], texture synthesis [64, 135], shape inference from images [104, 105], and novel view synthesis [123, 130, 169, 177].

We leverage recent progress in modeling the behavior of deep networks using kernel regression with a neural tangent kernel (NTK) [71] to theoretically and experimentally show that standard MLPs are poorly suited for these low-dimensional coordinate-based vision and graphics tasks. In particular, MLPs have difficulty learning high frequency functions, a phenomenon referred to in the literature as "spectral bias" [10, 148]. NTK theory suggests that this is because standard coordinate-based MLPs correspond to kernels with a rapid frequency falloff, which effectively prevents them from being able to represent the high-frequency content present in natural images and scenes.

A few recent works [123, 229] have experimentally found that a heuristic sinusoidal mapping of input coordinates (called a "positional encoding") allows MLPs to represent higher frequency content. We observe that this is a special case of Fourier features [150]: mapping input coordinates $\mathbf{v}$ to $\gamma(\mathbf{v}) = \left[a_1 \cos(2\pi \mathbf{b}_1^{\mathrm{T}} \mathbf{v}), a_1 \sin(2\pi \mathbf{b}_1^{\mathrm{T}} \mathbf{v}), \ldots, a_m \cos(2\pi \mathbf{b}_m^{\mathrm{T}} \mathbf{v}), a_m \sin(2\pi \mathbf{b}_m^{\mathrm{T}} \mathbf{v})\right]^{\mathrm{T}}$ before passing them into an MLP. We show that this mapping transforms the NTK into a stationary (shift-invariant)

This chapter is based on joint work published at NeurIPS 2020 [193]

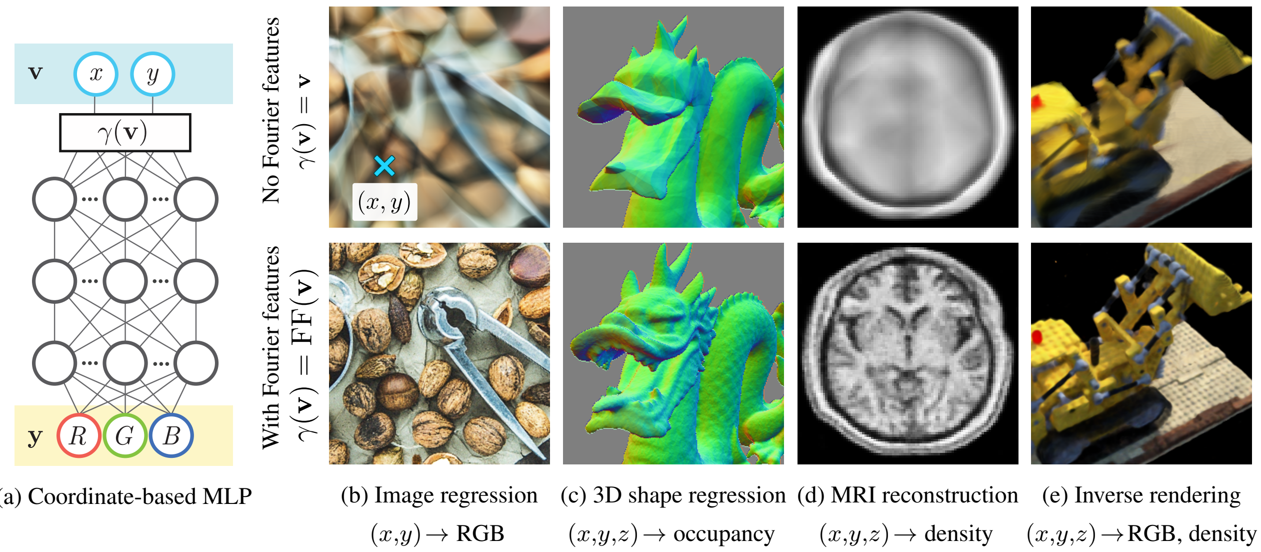


Figure 3.1: Fourier features improve the results of coordinate-based MLPs for a variety of high-frequency low-dimensional regression tasks, both with direct (b, c) and indirect (d, e) supervision. We visualize an example MLP (a) for an image regression task (b), where the input to the network is a pixel coordinate and the output is that pixel's color. Passing coordinates directly into the network (top) produces blurry images, whereas preprocessing the input with a Fourier feature mapping (bottom) enables the MLP to represent higher frequency details.

kernel and enables tuning the NTK's spectrum by modifying the frequency vectors $\mathbf{b}_j$, thereby controlling the range of frequencies that can be learned by the corresponding MLP. We show that the simple strategy of setting $a_j = 1$ and randomly sampling $\mathbf{b}_j$ from an isotropic distribution achieves good performance, and that the scale (standard deviation) of this distribution matters much more than its specific shape. We train MLPs with this Fourier feature input mapping across a range of tasks relevant to the computer vision and graphics communities. As highlighted in Figure 3.1, our proposed mapping dramatically improves the performance of coordinate-based MLPs. In summary, we make the following contributions:

- We leverage NTK theory and simple experiments to show that a Fourier feature mapping can be used to overcome the spectral bias of coordinate-based MLPs towards low frequencies by allowing them to learn much higher frequencies (Section 3.3).
- We demonstrate that a random Fourier feature mapping with an appropriately chosen scale can dramatically improve the performance of coordinate-based MLPs across many low-dimensional tasks in computer vision and graphics (Section 3.5).

3.1 Related Work

Our work is motivated by the widespread use of coordinate-based MLPs to represent a variety of visual signals, including images [184] and 3D scenes [115, 123, 137]. In particular, our analysis is intended to clarify experimental results demonstrating that an input mapping of coordinates (which they called a "positional encoding") using sinusoids with logarithmically-spaced axis-aligned frequencies improves the performance of coordinate-based MLPs on the tasks of novel view synthesis from 2D images [123] and protein structure modeling from cryo-electron microscopy [229]. We analyze this technique to show that it corresponds to a modification of the MLP's NTK, and we show that other non-axis-aligned frequency distributions can outperform this positional encoding.

Prior works in natural language processing and time series analysis [81, 199, 211] have used a similar positional encoding to represent time or 1D position. In particular, Xu *et al.* [211] use random Fourier features (RFF) [150] to approximate stationary kernels with a sinusoidal input mapping and propose techniques to tune the mapping parameters. Our work extends this by directly explaining such mappings as a modification of the resulting network's NTK. Additionally, we address the embedding of multidimensional coordinates, which is necessary for vision and graphics tasks.

To analyze the effects of applying a Fourier feature mapping to input coordinates before passing them through an MLP, we rely on recent theoretical work that models neural networks in the limits of infinite width and infinitesimal learning rate as kernel regression using the NTK [7, 12, 39, 71, 91]. In particular, we use the analyses from Lee *et al.* [91] and Arora *et al.* [7], which show that the outputs of a network throughout gradient descent remain close to those of a linear dynamical system whose convergence rate is governed by the eigenvalues of the NTK matrix [7, 10, 12, 91, 213]. Analysis of the NTK's eigendecomposition shows that its eigenvalue spectrum decays rapidly as a function of frequency, which explains the widely-observed "spectral bias" of deep networks towards learning low-frequency functions [10, 11, 148].

We leverage this analysis to consider the implications of adding a Fourier feature mapping before the network, and we show that this mapping has a significant effect on the NTK's eigenvalue spectrum and on the corresponding network's convergence properties in practice.

3.2 Background and Notation

To lay the foundation for our theoretical analysis, we first review classic kernel regression and its connection to recent results that analyze the training dynamics and generalization behavior of deep fully-connected networks. In later sections, we use these tools to analyze the effects of training coordinate-based MLPs with Fourier feature mappings.

Kernel regression. Kernel regression is a classic nonlinear regression algorithm [202]. Given a training dataset $(\mathbf{X}, \mathbf{y}) = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, where $\mathbf{x}_i$ are input points and $y_i = f(\mathbf{x}_i)$ are the corresponding scalar output labels, kernel regression constructs an estimate $\hat{f}$ of the underlying function at any point $\mathbf{x}$ as:

$$\hat{f}(\mathbf{x}) = \sum_{i=1}^{n} \left(\mathbf{K}^{-1}\mathbf{y}\right)_i k(\mathbf{x}_i, \mathbf{x}) \,, \tag{3.1}$$

where $\mathbf{K}$ is an $n \times n$ kernel (Gram) matrix with entries $\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ and k is a symmetric positive semidefinite (PSD) kernel function which represents the "similarity" between two input vectors. Intuitively, the kernel regression estimate at any point $\mathbf{x}$ can be thought of as a weighted sum of training labels y_i using the similarity between the corresponding $\mathbf{x}_i$ and $\mathbf{x}$.

Approximating deep networks with kernel regression. Let f be a fully-connected deep network with weights θ initialized from a Gaussian distribution $\mathcal{N}$. Theory proposed by Jacot *et al.* [71] and extended by others [7, 10, 91] shows that when the width of the layers in f tends to infinity and the learning rate for SGD tends to zero, the function $f(\mathbf{x}; \theta)$ converges over the course of training to the kernel regression solution using the *neural tangent kernel* (NTK), defined as:

$$k_{\mathrm{NTK}}(\mathbf{x}_i, \mathbf{x}_j) = \mathbb{E}_{\theta\sim\mathcal{N}} \left\langle \frac{\partial f(\mathbf{x}_i; \theta)}{\partial \theta}, \frac{\partial f(\mathbf{x}_j; \theta)}{\partial \theta} \right\rangle . \tag{3.2}$$

When the inputs are restricted to a hypersphere, the NTK for an MLP can be written as a dot product kernel (a kernel in the form $h_{\mathrm{NTK}}(\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j)$ for a scalar function $h_{\mathrm{NTK}} : \mathbb{R} \to \mathbb{R}$).

Prior work [7, 10, 71, 91] shows that an NTK linear system model can be used to approximate the dynamics of a deep network during training. We consider a network trained with an L2 loss and a learning rate η, where the network's weights are initialized such that the output of the network at initialization is close to zero. Under asymptotic conditions stated in Lee *et al.* [91], the network's output for any data $\mathbf{X}_{\text{test}}$ after t training iterations can be approximated as:

$$\hat{\mathbf{y}}^{(t)} \approx \mathbf{K}_{\text{test}}\mathbf{K}^{-1}\left(\mathbf{I} - e^{-\eta\mathbf{K}t}\right)\mathbf{y} \,, \tag{3.3}$$

where $\hat{\mathbf{y}}^{(t)} = f(\mathbf{X}_{\text{test}}; \theta)$ are the network's predictions on input points $\mathbf{X}_{\text{test}}$ at training iteration t, $\mathbf{K}$ is the NTK matrix between all pairs of training points in $\mathbf{X}$, and $\mathbf{K}_{\text{test}}$ is the NTK matrix between all points in $\mathbf{X}_{\text{test}}$ and all points in the training dataset $\mathbf{X}$.

Spectral bias when training neural networks. Let us consider the training error $\hat{\mathbf{y}}_{\text{train}}^{(t)} - \mathbf{y}$, where $\hat{\mathbf{y}}_{\text{train}}^{(t)}$ are the network's predictions on the training dataset at iteration t. Since the NTK matrix $\mathbf{K}$ must be PSD, we can take its eigendecomposition $\mathbf{K} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{\mathrm{T}}$, where $\mathbf{Q}$ is orthogonal and $\mathbf{\Lambda}$ is a diagonal matrix whose entries are the eigenvalues $\lambda_i \geq 0$ of $\mathbf{K}$. Then, since $e^{-\eta\mathbf{K}t} = \mathbf{Q}e^{-\eta\mathbf{\Lambda}t}\mathbf{Q}^{\mathrm{T}}$:

$$\mathbf{Q}^{\mathrm{T}}(\hat{\mathbf{y}}_{\text{train}}^{(t)} - \mathbf{y}) \approx \mathbf{Q}^{\mathrm{T}}\left(\left(\mathbf{I} - e^{-\eta\mathbf{K}t}\right)\mathbf{y} - \mathbf{y}\right) = -e^{-\eta\mathbf{\Lambda}t}\mathbf{Q}^{\mathrm{T}}\mathbf{y} \,. \tag{3.4}$$

This means that if we consider training convergence in the eigenbasis of the NTK, the i^{th} component of the absolute error $|\mathbf{Q}^{\mathrm{T}}(\hat{\mathbf{y}}_{\text{train}}^{(t)} - \mathbf{y})|_i$ will decay approximately exponentially at the rate $\eta\lambda_i$. In other words, components of the target function that correspond to kernel eigenvectors with larger eigenvalues will be learned faster. For a conventional MLP, the eigenvalues of the NTK decay rapidly [11, 12, 61]. This results in extremely slow convergence to the high frequency components of the target function, to the point where standard MLPs are effectively unable to learn these components, as visualized in Figure 3.1. Next, we describe a technique to address this slow convergence by using a Fourier feature mapping of input coordinates before passing them to the MLP.

3.3 Fourier Features for a Tunable Stationary Neural Tangent Kernel

Machine learning analysis typically addresses the case in which inputs are high dimensional points (*e.g.* the pixels of an image reshaped into a vector) and training examples are sparsely distributed. In contrast, in this work we consider *low-dimensional regression* tasks, wherein inputs are assumed to be dense coordinates in a subset of $\mathbb{R}^d$ for small values of d (*e.g.* pixel coordinates). This setting has two significant implications when viewing deep networks through the lens of kernel regression:

1. We would like the composed NTK to be shift-invariant over the input domain, since the training points are distributed with uniform density. In problems where the inputs are normalized to the surface of a hypersphere (common in machine learning), a dot product kernel (such as the regular NTK) corresponds to spherical convolution. However, inputs in our setting are dense in Euclidean space. A Fourier feature mapping of input coordinates makes the composed NTK stationary (shift-invariant), acting as a convolution kernel over the input domain (see Section 3.4 for additional discussion on stationary kernels).

2. We would like to control the bandwidth of the NTK to improve training speed and generalization. As we see from Eqn. 3.4, a "wider" kernel with a slower spectral falloff achieves faster training convergence for high frequency components. However, we know from signal processing that reconstructing a signal using a kernel whose spectrum is *too* wide causes high frequency aliasing artifacts. We show in Section 3.5 that a Fourier feature input mapping can be tuned to lie between these "underfitting' and "overfitting" extremes, enabling both fast convergence and low test error.

Fourier features and the composed neural tangent kernel. Fourier feature mappings have been used in many applications since their introduction in the seminal work of Rahimi and Recht [150], which used random Fourier features to approximate an arbitrary stationary kernel function by applying Bochner's theorem. Extending this technique, we use a Fourier feature mapping γ to featurize input coordinates before passing them through a coordinate-based MLP, and investigate the theoretical and practical effect this has on convergence speed and generalization. The function γ maps input points $\mathbf{v} \in [0,1)^d$ to the surface of a higher dimensional hypersphere with a set of sinusoids:

$$\gamma(\mathbf{v}) = \left[a_1 \cos(2\pi \mathbf{b}_1^{\mathrm{T}} \mathbf{v}), a_1 \sin(2\pi \mathbf{b}_1^{\mathrm{T}} \mathbf{v}), \ldots, a_m \cos(2\pi \mathbf{b}_m^{\mathrm{T}} \mathbf{v}), a_m \sin(2\pi \mathbf{b}_m^{\mathrm{T}} \mathbf{v})\right]^{\mathrm{T}}. \tag{3.5}$$

Because $\cos(\alpha - \beta) = \cos\alpha \cos\beta + \sin\alpha \sin\beta$, the kernel function induced by this mapping is:

$$k_\gamma(\mathbf{v}_1, \mathbf{v}_2) = \gamma(\mathbf{v}_1)^{\mathrm{T}} \gamma(\mathbf{v}_2) = \sum_{j=1}^{m} a_j^2 \cos\big(2\pi \mathbf{b}_j^{\mathrm{T}} (\mathbf{v}_1 - \mathbf{v}_2)\big) = h_\gamma(\mathbf{v}_1 - \mathbf{v}_2), \tag{3.6}$$

$$\text{where } h_\gamma(\mathbf{v}_\Delta) \triangleq \sum_{j=1}^{m} a_j^2 \cos(2\pi \mathbf{b}_j^{\mathrm{T}} \mathbf{v}_\Delta). \tag{3.7}$$

Note that this kernel is stationary (a function of only the difference between points). We can think of the mapping as a Fourier approximation of a kernel function: $\mathbf{b}_j$ are the Fourier basis frequencies used to approximate the kernel, and a_j^2 are the corresponding Fourier series coefficients.

After computing the Fourier features for our input points, we pass them through an MLP to get $f(\gamma(\mathbf{v});\theta)$. As discussed previously, the result of training a network can be approximated by kernel regression using the kernel $h_{\text{NTK}}(\mathbf{x}_i^{\text{T}}\mathbf{x}_j)$. In our case, $\mathbf{x}_i = \gamma(\mathbf{v}_i)$ so the composed kernel becomes:

$$h_{\text{NTK}}(\mathbf{x}_i^{\text{T}}\mathbf{x}_j) = h_{\text{NTK}}\Big(\gamma(\mathbf{v}_i)^{\text{T}}\gamma(\mathbf{v}_j)\Big) = h_{\text{NTK}}(h_\gamma(\mathbf{v}_i - \mathbf{v}_j)). \tag{3.8}$$

Thus, training a network on these embedded input points corresponds to kernel regression with the *stationary* composed NTK function $h_{\text{NTK}} \circ h_\gamma$. The MLP function approximates a convolution of the composed NTK with a weighted Dirac delta at each input training point $\mathbf{v}_i$:

$$\hat{f} = (h_{\text{NTK}} \circ h_\gamma) * \sum_{i=1}^{n} w_i \delta_{\mathbf{v}_i} \tag{3.9}$$

where $\mathbf{w} = \mathbf{K}^{-1}\mathbf{y}$ (from Eqn. 3.1). This allows us to draw analogies to signal processing, where the composed NTK acts similarly to a reconstruction filter. In the next section, we show that the frequency decay of the composed NTK determines the behavior of the reconstructed signal.

3.4 Stationary kernels

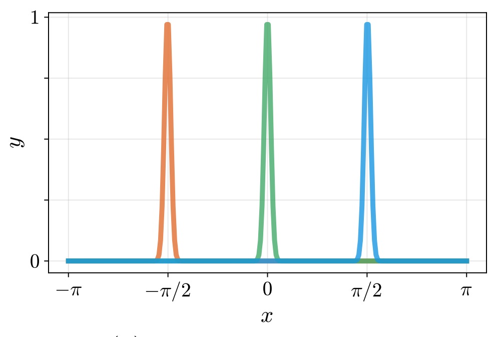
One of the primary benefits of our Fourier feature mapping is that it results in a *stationary* composed NTK function. In this section, we offer some intuition for why stationarity is desirable for our low-dimensional graphics and imaging problems.

First, let us consider the implications of using an MLP applied directly to a low-dimensional input (without any Fourier feature mapping). In this setting, the NTK is a function of the dot product between its inputs and of their norms [10, 12, 14, 71]. This makes the NTK *rotation*-invariant, but not *translation*-invariant. For our graphics and imaging applications, we want to be able to model an object or scene equally well regardless of its location, so translation-invariance or *stationarity* is a crucial property. We can then add approximate rotation invariance back by using an isotropic frequency sampling distribution.

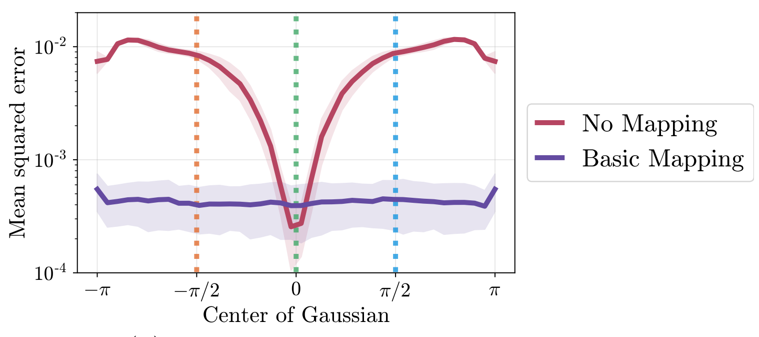
This aligns with standard practice in signal processing, in which $k(\mathbf{u},\mathbf{v}) = \tilde{h}(\mathbf{u}-\mathbf{v}) = \tilde{h}(\mathbf{v}-\mathbf{u})$ (*e.g.* the Gaussian or radial basis function kernel, or the sinc reconstruction filter kernel). This Euclidean notion of similarity based on difference vectors is better suited to the low-dimensional regime, in which we expect (and can afford) dense and nearly uniform sampling. Regression with a stationary kernel corresponds to reconstruction with a convolution filter: new predictions are sums of training points, weighted by a function of Euclidean distance.

One of the most important features of our sinusoidal input mapping is that it translates between these two regimes. If $\mathbf{u},\mathbf{v} \in \mathbb{R}^d$ for small d, γ is our Fourier feature embedding function, and k is a dot product kernel function, then $k(\gamma(\mathbf{u}),\gamma(\mathbf{v})) = h(\gamma(\mathbf{u})^{\text{T}}\gamma(\mathbf{v})) = \tilde{h}(\mathbf{u}-\mathbf{v})$. In words, our sinusoidal input mapping transforms a dot product kernel into a stationary one, making it better suited to the low-dimensional regime.

This effect is illustrated in a simple 1D example in Figure 3.2, which shows that the benefits of a stationary composed NTK indeed appear in the MLP setting with a basic Fourier featurization (using a single frequency). We train MLPs with and without this basic Fourier embedding to learn a set of shifted 1D Gaussian probability density functions. The plain MLP successfully fits a zero-centered function but struggles to fit shifted functions, while the MLP with basic Fourier embedding exhibits stationary behavior, with good performance regardless of shifts.



(a) Example target signals



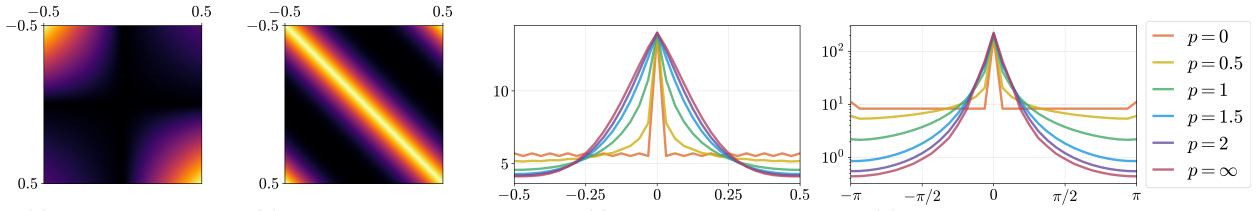
(b) Reconstruction accuracy

Figure 3.2: A plain coordinate-based MLP can learn a centered function (in this case a Gaussian density) but struggles to model shifts of the same function. Adding a basic Fourier embedding (with a single frequency) enables the MLP to fit the target function equally well regardless of shifts. The NTK corresponding to the plain MLP is based on dot products between inputs, whereas the NTK corresponding to the NTK with Fourier embedding is based on Euclidean distances between inputs, making it shift-invariant. In this experiment we train an MLP (4 layers, 256 channels, ReLU activation) for 500 iterations using the Adam [86] optimizer with a learning rate of 10^{-4}. We report mean and standard deviation performance over 20 random network initializations.

3.5 Manipulating the Fourier Feature Mapping

Preprocessing the inputs to a coordinate-based MLP with a Fourier feature mapping creates a composed NTK that is not only stationary but also *tunable*. By manipulating the settings of the a_j and $\mathbf{b}_j$ parameters in Eqn. 3.5, it is possible to dramatically change both the rate of convergence and the generalization behavior of the resulting network. In this section, we investigate the effects of the Fourier feature mapping in the setting of 1D function regression.

We train MLPs to learn signals f defined on the interval $[0, 1)$. We sample cn linearly spaced points on the interval, using every c^{th} point as the training set and the remaining points as the test set. Since our composed kernel function is stationary, evaluating it at linearly spaced points on a periodic domain makes the resulting kernel matrix circulant: it represents a convolution and is diagonalizable by the Fourier transform. Thus, we can compute the eigenvalues of the composed NTK matrix by



(a) No mapping NTK (b) Basic mapping NTK (c) NTK spatial (d) NTK Fourier spectrum

Figure 3.3: Adding a Fourier feature mapping can improve the poor conditioning of a coordinate-based MLP's neural tangent kernel (NTK). (a) We visualize the NTK function $k_{\text{NTK}}(x_i, x_j)$ (Eqn. 3.2) for a 4-layer ReLU MLP with one scalar input. This kernel is not shift-invariant and does not have a strong diagonal, making it poorly suited for kernel regression in low-dimensional problems. (b) A basic input mapping $\gamma(v) = [\cos 2\pi v, \sin 2\pi v]^{\mathrm{T}}$ makes the composed NTK $k_{\text{NTK}}(\gamma(v_i), \gamma(v_j))$ shift-invariant (stationary). (c) A Fourier feature input mapping (Eqn. 3.5) can be used to tune the composed kernel's width, where we set $a_j = 1/j^p$ and $b_j = j$ for $j = 1, \ldots, n/2$. (d) Higher frequency mappings (lower p) result in composed kernels with wider spectra, which enables faster convergence for high-frequency components (see Figure 3.4).

simply taking the Fourier transform of a single row. All experiments are implemented in JAX [15] and the NTK functions are calculated automatically using the Neural Tangents library [133].

Visualizing the composed NTK. We first visualize how modifying the Fourier feature mapping changes the composed NTK. We set $b_j = j$ (full Fourier basis in 1D) and $a_j = 1/j^p$ for $j = 1, \ldots, n/2$. We use $p = \infty$ to denote the mapping $\gamma(v) = [\cos 2\pi v, \sin 2\pi v]^{\mathrm{T}}$ that simply wraps $[0, 1)$ around the unit circle (this is referred to as the "basic" mapping in later experiments). Figure 3.3 demonstrates the effect of varying p on the composed NTK. By construction, lower p values result in a slower falloff in the frequency domain and a correspondingly narrower kernel in the spatial domain.

Effects of Fourier features on network convergence. We generate ground truth 1D functions by sampling cn values from a family with parameter α as follows: we sample a standard i.i.d. Gaussian vector of length cn, scale its i^{th} entry by $1/i^\alpha$, then return the real component of its inverse Fourier transform. We will refer to this as a "$1/f^\alpha$ noise" signal.

In Figure 3.4, we train MLPs (4 layers, 1024 channels, ReLU activations) to fit a bandlimited $1/f^1$ noise signal ($c = 8, n = 32$) using Fourier feature mappings with different p values. Figures 3.4b and 3.4d show that the NTK linear dynamics model accurately predict the effects of modifying the Fourier feature mapping parameters. Separating different frequency components of the training error in Figure 3.4c reveals that networks with narrower NTK spectra converge faster for low frequency components but essentially never converge for high frequency components, whereas networks with wider NTK spectra successfully converge across all components. The Fourier feature mapping $p = 1$ has adequate power across frequencies present in the target signal (so the network converges rapidly during training) but limited power in higher frequencies (preventing overfitting or aliasing).

Tuning Fourier features in practice. Eqn. 3.3 allows us to estimate a trained network's theoretical

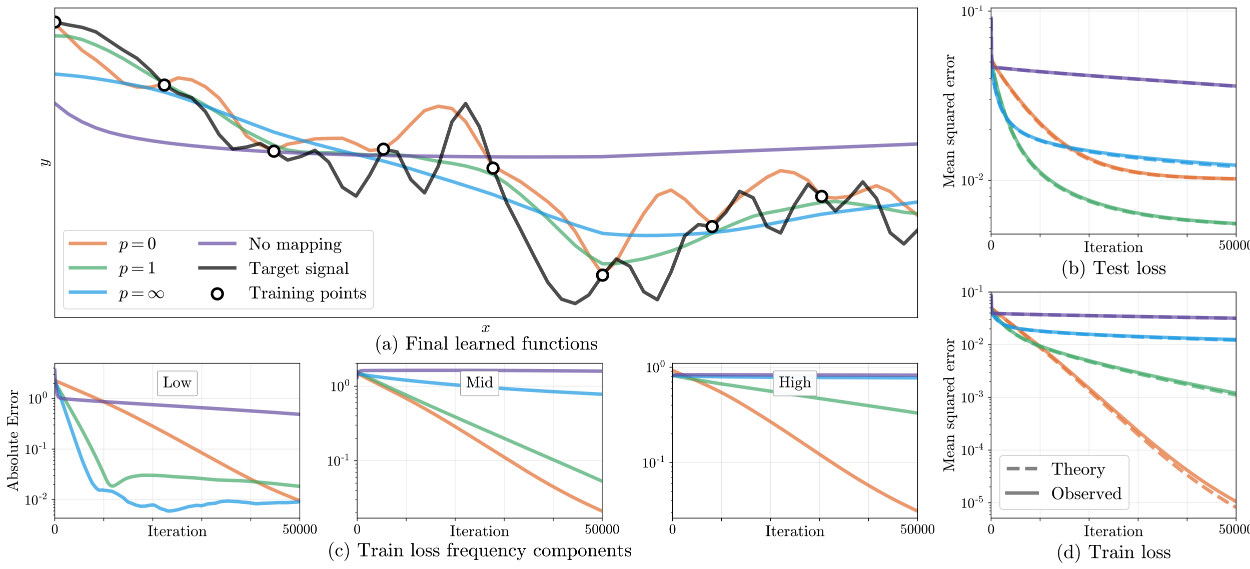
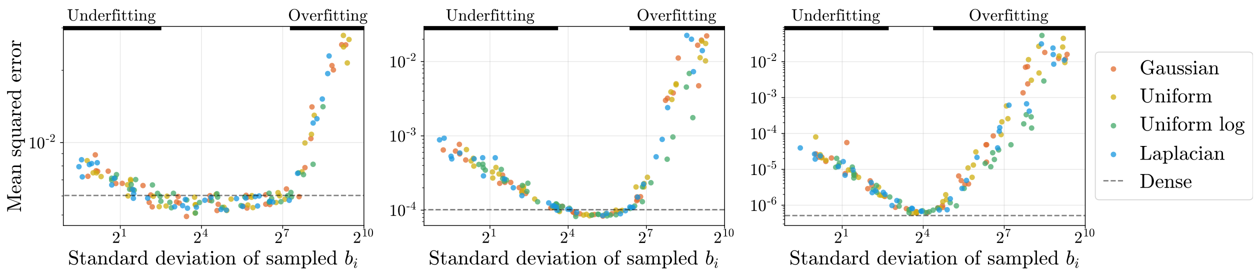


Figure 3.4: Combining a network with a Fourier feature mapping has dramatic effects on convergence and generalization. Here we train a network on 32 sampled points from a 1D function (a) using mappings shown in Fig. 3.3. A mapping with a smaller p value yields a composed NTK with more power in higher frequencies, enabling the corresponding network to learn a higher frequency function. The theoretical and experimental training loss improves monotonically with higher frequency kernels (d), but the test-set loss is lowest at $p = 1$ and falls as the network starts to overfit (b). As predicted by Eqn. 3.4, we see roughly log-linear convergence of the training loss frequency components (c). Higher frequency kernels result in faster convergence for high frequency loss components, thereby overcoming the "spectral bias" observed when training networks with no input mapping.

loss on a validation set using the composed kernel. For small 1D problems, we can minimize this loss with gradient-based optimization to choose mapping parameters a_j (given a dense sampling of b_j). In this carefully controlled setting (1D signals, small training dataset, gradient descent with small learning rate, very wide networks), we find that this optimized mapping also achieves the best performance when training networks. Please refer to section 3.6 for details and experiments.

In real-world problems, especially in multiple dimensions, it is not feasible to use a feature mapping that densely samples Fourier basis functions; the number of Fourier basis functions scales with the number of training data points, which grows exponentially with dimension. Instead, we sample a set of random Fourier features [150] from a parametric distribution. We find that the exact sampling distribution family is much less important than the distribution's scale (standard deviation).

Figure 3.5 demonstrates this point using hyperparameter sweeps for a variety of sampling distributions. In each subfigure, we draw 1D target signals ($c = 2, n = 1024$) from a fixed $1/f^\alpha$ distribution and train networks to learn them. We use random Fourier feature mappings (of length



(a) Data sampled from $1/f^{0.5}$ (b) Data sampled from $1/f^{1.0}$ (c) Data sampled from $1/f^{1.5}$

Figure 3.5: We find that a sparse random sampling of Fourier features can perform as well as a dense set of features and that the width of the distribution matters more than the shape. Here, we generate random 1D signals from $1/f^\alpha$ noise and report the test-set accuracy of different trained models that use a sparse set (16 out of 1024) of random Fourier features sampled from different distributions. Each subplot represents a different family of 1D signals. Each dot represents a trained network, where the color indicates which Fourier feature sampling distribution is used. We plot the test error of each model versus the empirical standard deviation of its sampled frequencies. The best models using sparsely sampled features are able to match the performance of a model trained with dense Fourier features (dashed lines with error bars). All sampling distributions trace out the same curve, exhibiting underfitting (slow convergence) when the standard deviation of sampled frequencies is too low and overfitting when it is too high. This implies that the precise shape of the distribution used to sample frequencies does not have a significant impact on performance.

16) sampled from different distribution families (Gaussian, uniform, uniform in log space, and Laplacian) and sweep over each distribution's scale. Perhaps surprisingly, the standard deviation of the sampled frequencies alone is enough to predict test set performance, regardless of the underlying distribution's shape. We also observe that passing this sparse sampling of Fourier features through an MLP matches the performance of using a dense set of Fourier features with the same MLP, suggesting a strategy for scaling to higher dimensions. We proceed with a Gaussian distribution for our higher-dimensional experiments in Section 3.6 and treat the scale as a hyperparameter to tune on a validation dataset.

3.6 Experiments

We validate the benefits of using Fourier feature mappings for coordinate-based MLPs with experiments on a variety of regression tasks relevant to the computer vision and graphics communities.

Compared mappings

We compare the performance of coordinate-based MLPs with no input mapping and with the following Fourier feature mappings ($\cos, \sin$ are applied elementwise):

Basic: $\gamma(\mathbf{v}) = [\cos(2\pi\mathbf{v}v), \sin(2\pi\mathbf{v})]^{\mathrm{T}}$. Simply wraps input coordinates around the circle.

Positional encoding: $\gamma(\mathbf{v}) = \left[\ldots, \cos(2\pi\sigma^{j/m}\mathbf{v}), \sin(2\pi\sigma^{j/m}\mathbf{v}), \ldots\right]^{\mathrm{T}}$ for $j = 0, \ldots, m-1$. Uses log-linear spaced frequencies for each dimension, where the scale σ is chosen for each task and dataset by a hyperparameter sweep. This is a generalization of the "positional encoding" used by prior work [123, 199, 229]. Note that this mapping is deterministic and only contains on-axis frequencies, making it naturally biased towards data that has more frequency content along the axes.

Gaussian: $\gamma(\mathbf{v}) = [\cos(2\pi\mathbf{B}\mathbf{v}), \sin(2\pi\mathbf{B}\mathbf{v})]^{\mathrm{T}}$, where each entry in $\mathbf{B} \in \mathbb{R}^{m \times d}$ is sampled from $\mathcal{N}(0, \sigma^2)$, and σ is chosen for each task and dataset with a hyperparameter sweep. In the absence of any strong prior on the frequency spectrum of the signal, we use an isotropic Gaussian distribution.

Our experiments show that all of the Fourier feature mappings improve the performance of coordinate-based MLPs over using no mapping and that the Gaussian RFF mapping performs best.

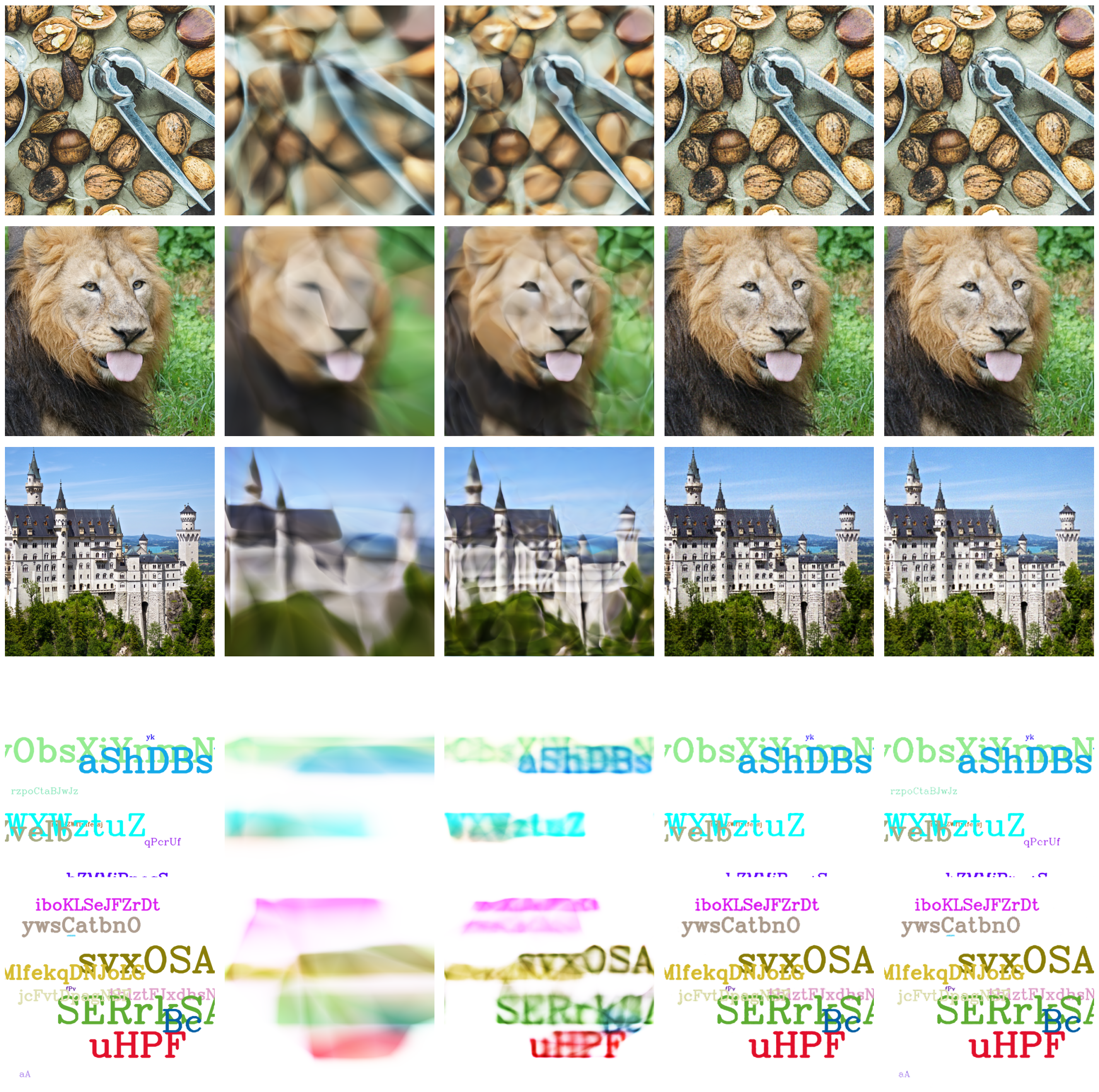
Tasks

We conduct experiments with direct regression, where supervision labels are in the same space as the network outputs, as well as indirect regression, where the network outputs are passed through a forward model to produce observations in the same space as the supervision labels. For each task and dataset, we tune Fourier feature scales on a held-out set of signals. For each target signal, we train an MLP on a training subset of the signal and compute error over the remaining test subset. All tasks (except 3D shape regression) use L2 loss and a ReLU MLP with 4 layers and 256 channels. The 3D shape regression task uses cross-entropy loss and a ReLU MLP with 8 layers and 256 channels. We apply a sigmoid activation to the output for each task (except the view synthesis density prediction). We use 256 frequencies for the feature mapping in all experiments.

2D image

The 2D image regression tasks presented in the main text all use 512×512 resolution images. A subsampled grid of 256×256 pixels is used as training data, and an offset grid of 256×256 pixels is used for testing. We use two image datasets: *Natural* and *Text*, each consisting of 32 images. The *Natural* images are generated by taking center crops of randomly sampled images from the Div2K dataset [4]. The *Text* images are generated by placing random strings of text with random sizes and colors on a white background (examples can be seen in Figure 3.6). For each dataset we perform a hyperparameter sweep over feature mapping scales on 16 images. We find that scales $\sigma_g = 10$ and $\sigma_p = 6$ work best for the *Natural* dataset and $\sigma_g = 14$ and $\sigma_p = 5$ work best for the *Text* dataset. In Table 3.1, we report model performance using the optimal mapping scale on the remaining 16 images.

Each model (MLP with 4 layers, 256 channels, ReLU activation, sigmoid output) is trained for 2000 iterations using the Adam [86] optimizer with default settings ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$). Learning rates are manually tuned for each dataset and method. For *Natural* images a learning rate of 10^{-3} is used for the Gaussian RFF and the positional encoding, and a learning rate of 10^{-2} is used for the basic mapping and "no mapping" methods. For the *Text* images a learning rate of 10^{-3} is used for all methods.



(a) Ground Truth (b) No mapping (c) Basic (d) Positional enc. (e) Gaussian

Figure 3.6: Results for the 2D image regression task, for three images from our *Natural* dataset (top) and two images from our *Text* dataset (bottom).

3D shape

We evaluate the 3D shape regression task (similar to Occupancy Networks [115]) on four complex triangle meshes commonly used in computer graphics applications (*Dragon*, *Armadillo*, *Buddha*, and *Lucy*, shown in Figure 3.7), each containing hundreds of thousands of vertices. We train one coordinate-based MLP network to represent a single mesh rather than trying to generalize one network to encode multiple objects, since our goal is to demonstrate that a network with no mapping

	Natural	Text
No mapping	19.32 ± 2.48	18.40 ± 2.23
Basic	21.71 ± 2.71	20.48 ± 1.96
Positional enc.	24.95 ± 3.72	27.57 ± 3.07
Gaussian	$\mathbf{25.57 \pm 4.19}$	$\mathbf{30.47 \pm 2.11}$

Table 3.1: 2D image results (mean $\pm$ standard deviation of PSNR)

or the low frequency "basic" mapping cannot accurately represent even a *single* shape, let alone a whole class of objects.

We use a network with 8 layers of 256 channels each and a ReLU nonlinearity between each layer. We apply a sigmoid activation to the output. Our batch size is 32^3 points, and we use the Adam optimizer [86] with a learning rate starting at 5×10^{-4} and exponentially decaying by a factor of 0.01 over the course of 10000 total training iterations. At each training iteration, we sample a batch of 3D points uniformly at random from the bounding box of the mesh, and then calculate ground truth labels (using the point-in-mesh method implemented in the Trimesh library [117], which relies on the Embree kernel for acceleration [203]). We use cross-entropy loss to train the network to match these classification labels (0 for points outside the mesh, 1 for points inside).

The meshes are scaled to fit inside the unit cube $[0, 1]^3$ such that the centroid of the mesh is $(0.5, 0.5, 0.5)$. We use the *Lucy* statue mesh as a validation object to find optimal scale values for the positional encoding and Gaussian feature mapping. As described in the caption for Table 3.2, we calculate error on both a uniformly random test set and a test set that is close to the mesh surface (randomly chosen mesh vertices that have been perturbed by a random Gaussian vector with standard deviation 0.01) in order to illustrate that Fourier feature mappings provide a large benefit in resolving fine surface details. Both test sets have 64^3 points.

	Uniform points	Boundary points
No mapping	0.959 ± 0.006	0.864 ± 0.014
Basic	0.966 ± 0.007	0.892 ± 0.017
Positional enc.	0.987 ± 0.005	0.960 ± 0.011
Gaussian	$\mathbf{0.988 \pm 0.007}$	$\mathbf{0.973 \pm 0.010}$

Table 3.2: 3D shape results (mean $\pm$ standard deviation of intersection-over-union). *Uniform points* is an "easy" test set where points are sampled uniformly at random from the bounding box of the ground truth mesh, while *Boundary points* is a "hard" test set where points are sampled near the boundary of the ground truth mesh.

In Figure 3.7, we visualize additional results on all four meshes mentioned above (including the validation mesh *Lucy*). We render normal maps, which are computed by taking the cross product of the numerical horizontal and vertical derivatives of the depth map. The original depth map is

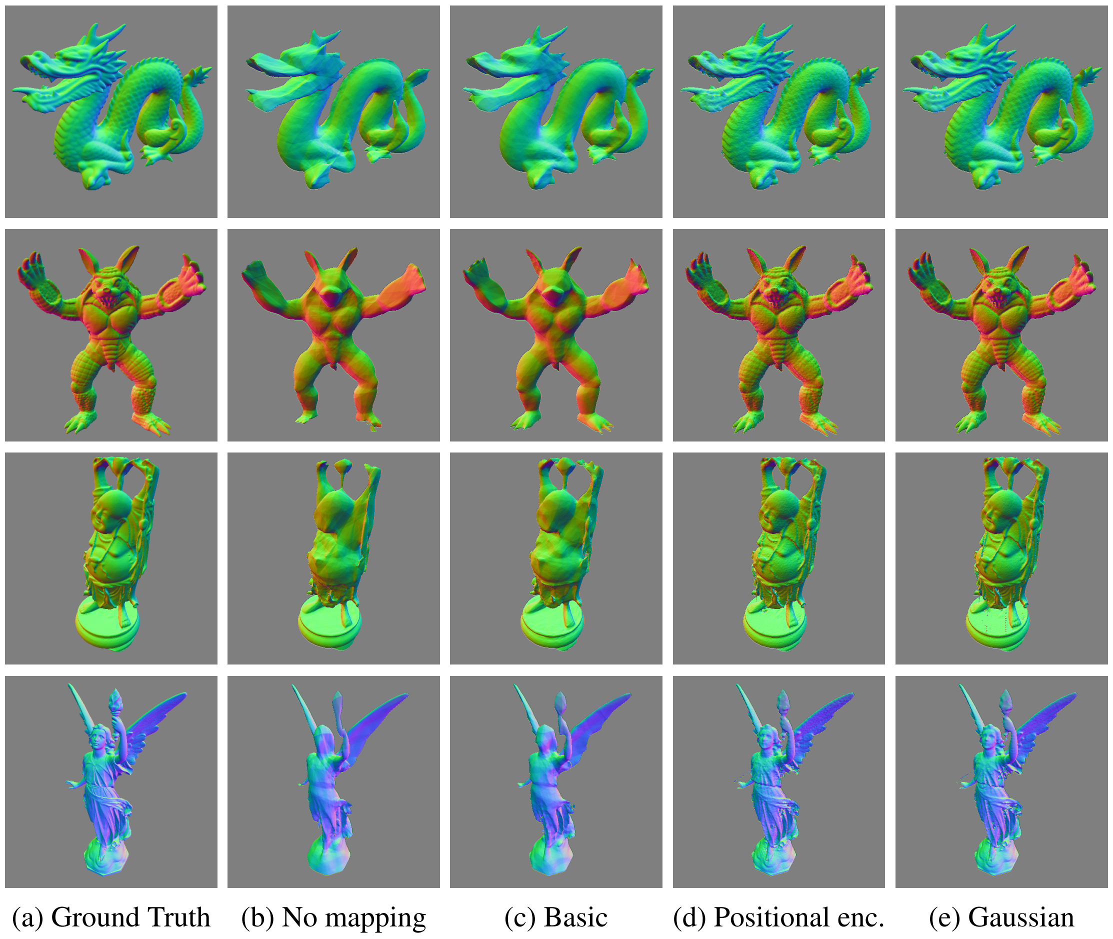


Figure 3.7: Results for the 3D shape occupancy task [115].

generated by intersecting camera rays with the first 0.5 isosurface of the network. We select the Fourier feature scales for (d) and (e) by doing a hyperparameter search based on validation loss for the *Lucy* mesh in the last row and report test loss over the other three meshes (Table 3.2). Note that the weights for each trained MLP are only 2MB, while the triangle mesh files for the objects shown are 61MB, 7MB, 79MB, and 32MB respectively.

2D CT

In computed tomography (CT), we observe measurements that are integral projections (integrals along parallel lines) of a density field. We construct a 2D CT task by using ground truth 512×512 resolution images, and computing 20 synthetic integral projections at evenly-spaced angles. For each of these images, the supervision data is the set of integral projections, and the test PSNR is evaluated over the original image.

We use two datasets for our 2D CT task: randomized Shepp-Logan phantoms [175], and the

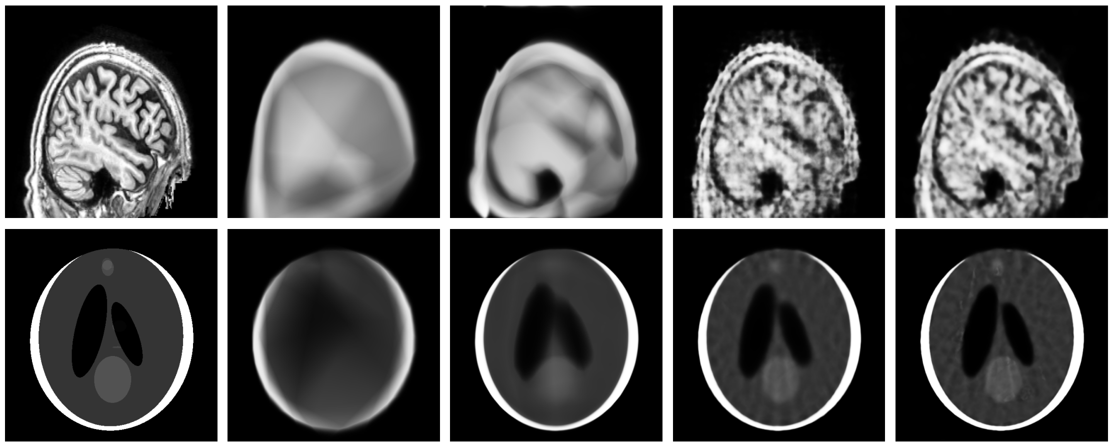

(a) Ground Truth (b) No mapping (c) Basic (d) Positional enc. (e) Gaussian

Figure 3.8: Results for the 2D CT task.

ATLAS brain dataset [99]. For each dataset, we perform a hyperparameter sweep over mapping scales on 8 examples. We found that scales $\sigma_g = 4$ and $\sigma_p = 3$ work best for the *Shepp* dataset and $\sigma_g = 5$ and $\sigma_p = 5$ work best for the *ATLAS* dataset. In Table 3.3, we report model performance using the optimal mapping scale on a distinct set of 8 images.

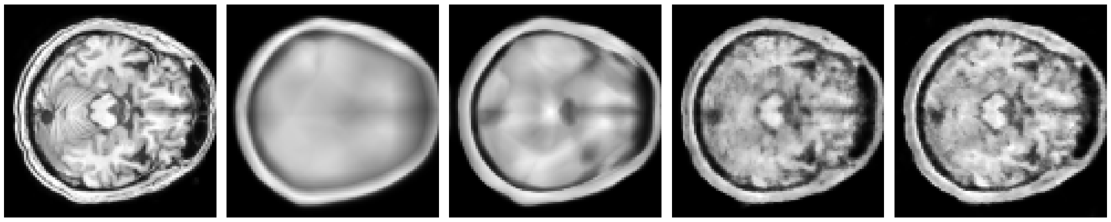
	Shepp	ATLAS
No mapping	16.75 ± 3.64	15.44 ± 1.28
Basic	23.31 ± 4.66	16.95 ± 0.72
Positional enc.	26.89 ± 1.46	19.55 ± 1.09
Gaussian	$\mathbf{28.33 \pm 1.15}$	$\mathbf{19.88 \pm 1.23}$

Table 3.3: 2D CT results (mean $\pm$ standard deviation of PSNR).

Each model (MLP with 4 layers, 256 channels, ReLU activation, sigmoid output) is trained for 1000 iterations using the Adam [86] optimizer with default settings ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$). The learning rate is manually tuned for each method. Gaussian RFF and positional encoding use a learning rate of 10^{-3}, and the basic and "no mapping" method use a learning rate of 10^{-2}.

3D MRI

In magnetic resonance imaging (MRI), we observe measurements that are Fourier coefficients of the atomic response to radio waves under a magnetic field. We construct a toy 3D MRI task by using ground truth $96 \times 96 \times 96$ resolution volumes and randomly sampling $\sim 13\%$ of the Fourier coefficients for each volume from an isotropic Gaussian. For each of these volumes, the supervision



(a) Ground Truth (b) No mapping (c) Basic (d) Positional enc. (e) Gaussian

Figure 3.9: Results for the 3D MRI task.

data is the set of sampled Fourier coefficients, and the test PSNR is evaluated over the original volume.

We use the ATLAS brain dataset [99] for our 3D MRI experiments. We perform a hyperparameter sweep over mapping scales on 6 examples. We find that scales $\sigma_g = 5$ and $\sigma_p = 4$ perform best. In Table 3.4, we report model performance using the optimal mapping scale on a distinct set of 6 images. Each model (MLP with 4 layers, 256 channels, ReLU activation, sigmoid output) is trained for 1000 iterations using the Adam [86] optimizer with default settings ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$). We use a manually-tuned learning rate of 2×10^{-3} for each method. Results are visualized in Figure 3.9.

	ATLAS
No mapping	26.14 ± 1.45
Basic	28.58 ± 2.45
Positional enc.	32.23 ± 3.08
Gaussian	$\mathbf{34.51 \pm 2.72}$

Table 3.4: 3D MRI results (mean ± standard deviation of PSNR).

3D inverse rendering for view synthesis

In this task we use the "tiny NeRF" simplified version of the view synthesis method NeRF [123] where hierarchical sampling and view dependence have been removed. The model is trained to predict the color and volume density at an input 3D point. Volumetric rendering is used to render novel viewpoints of the object. The loss is calculated between the rendered views and ground truth renders. In our experiments we use the NeRF *Lego* dataset of 120 images downsampled to 400×400 pixel resolution. The dataset is split into 100 training images, 7 validation images, and 13 test images. The reconstruction quality on the validation images is used to determine the best mapping scale; for this scene we find $\sigma_g = 6.05$ and $\sigma_p = 1.27$ perform best.

The model (MLP with 4 layers, 256 channels, ReLU activation, sigmoid on RGB output) is trained for 5×10^5 iterations using the Adam [86] optimizer with default settings ($\beta_1 = 0.9$,

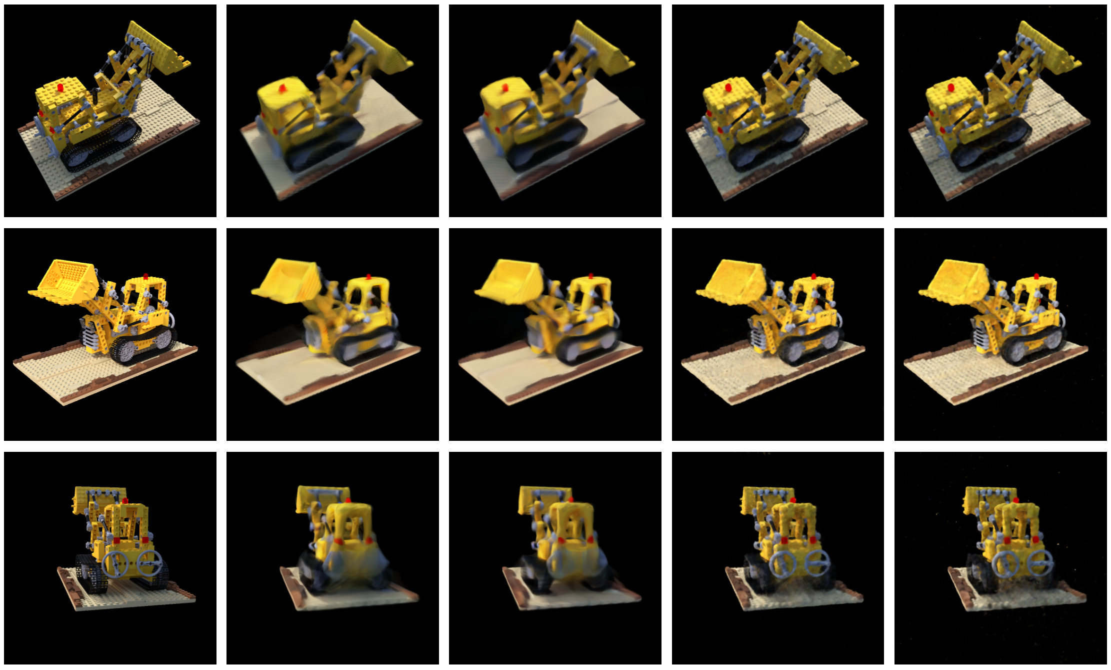

(a) Ground Truth (b) No mapping (c) Basic (d) Positional enc. (e) Gaussian

Figure 3.10: Results for the inverse rendering task [123].

$\beta_2 = 0.999$, $\epsilon = 10^{-8}$). The learning rate is manually tuned for each mapping: 10^{-2} for no mapping, 5×10^{-3} for basic, 5×10^{-4} for positional encoding, and 5×10^{-4} for Gaussian. During training we use batches of 1024 rays.

The original NeRF method [123] uses an input mapping similar to the *Positional encoding* we compare against. The original NeRF mapping is smaller than our mappings (8 vs. 256 frequencies). We include metrics for this mapping in Table 3.5 under *Original pos. enc.* The positional encoding mappings only contain frequencies on the axes, and are therefore biased towards signals with on-axis frequency content. In our experiments we rotate the *Lego* scene, which was manually axis-aligned in the original dataset, for a more equitable comparison. Table 3.5 also reports metrics for positional encodings on the original axis-aligned scene. Results are visualized in Figure 3.10.

Optimizing validation error through the NTK linear dynamics

Using Eqn. 3.3, we can predict what error a trained network will achieve on a set of testing points. Since this equation depends on the composed NTK, we can directly relate predicted test set loss to the Fourier feature mapping parameters a and b for a validation set of signals $\mathbf{y}_{val}$:

$$\mathcal{L}_{\text{opt}} = \left\|\mathbf{u}^{(t)} - \mathbf{y}_{\text{val}}\right\|_2^2 \approx \left\|\mathbf{K}_{\text{val}}\mathbf{K}^{-1}\left(\mathbf{I} - e^{-\eta\mathbf{K}t}\right)\mathbf{y} - \mathbf{y}_{\text{val}}\right\|_2^2, \tag{3.10}$$

	3D NeRF
No mapping	22.41 ± 0.92
Basic	23.16 ± 0.90
Original pos. enc.	24.81 ± 0.88
Positional enc.	25.28 ± 0.83
Gaussian	$\mathbf{25.48 \pm 0.89}$
Original pos. enc. (axis-aligned)	25.60 ± 0.76
Positional enc. (axis-aligned)	26.27 ± 0.91

Table 3.5: 3D NeRF results (mean $\pm$ standard deviation of PSNR). Error is calculated based on held-out images of the scene since the ground truth radiance field is not known.

where $\mathbf{K}_{\text{val}}$ is the composed NTK evaluated between points in a validation dataset $\mathbf{X}_{\text{val}}$ and training dataset $\mathbf{X}$, and η and t are the learning rate and number of iterations that will be used when training the actual network.

In Figure 3.11, we show the results of minimizing Eqn. 3.10 by gradient descent on a_j values (with fixed corresponding "densely sampled" $b_j = j$) for validation sets sampled from three different $1/f^\alpha$ noise families. Note that gradient descent on this theoretical loss approximation produces a_j values which are able to perform as well as the best "power law" a_j values for each respective signal class (compared dashed lines versus $\times$ markers in Figure 3.11b). As mentioned in the main text, we find that this optimization strategy is only viable for small 1D regression problems. In our multidimensional tasks, using densely sampled $\mathbf{b}_j$ values is not tractable due to memory constraints. In addition, the theoretical approximation only holds when training the network using SGD, and in practice we train using the Adam optimizer [86].

Feature sparsity and network depth

In our experiments, we observe that deeper networks need fewer Fourier features than shallow networks. As the depth of the MLP increases, we observe that a sparser set of frequencies can achieve similar performance; Figure 3.12 illustrates this effect in the context of 2D image regression.

Again drawing on NTK theory, we understand this tradeoff as an effect of frequency "spreading," as illustrated in Figure 3.13. A Fourier featurization consists of only discrete frequencies, but when composed with the NTK, the influence of each discrete frequency "spreads" over its local neighborhood in the final spectrum. We find that the "spread" around each frequency feature increases for deeper networks. For an MLP to learn all of the frequency components in the target signal, its corresponding composed NTK must contain adequate power across the frequency support of the target signal. This is accomplished either by including more frequencies in the Fourier features or by spreading those frequencies through sufficient NTK depth.

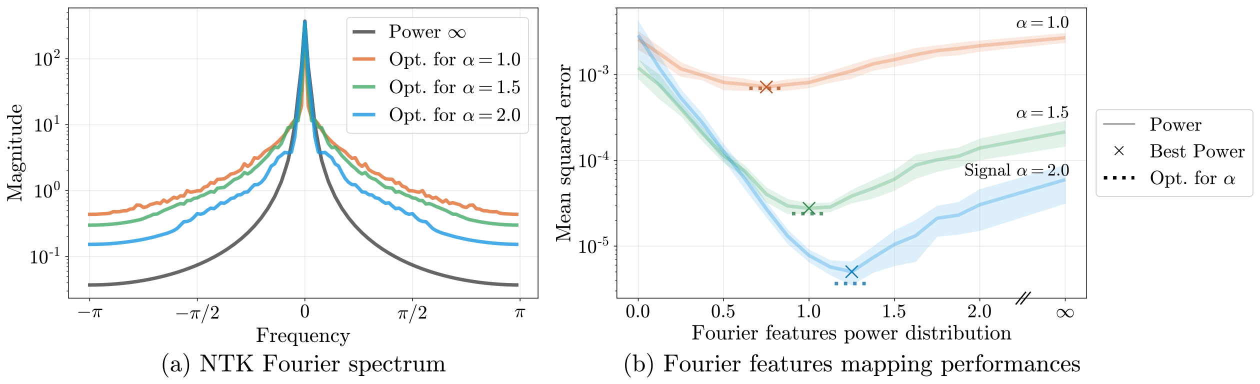


Figure 3.11: The Fourier feature mappings can be optimized for better performance on a class of target signals by using the linearized network approximation. Here we consider target signals sampled from three different power law distributions. In (a) we show the spectrum for composed kernels corresponding to different optimized feature mappings, where the feature mappings are initialized to match the “Power ∞” distribution. In (b) we take an alternative approach where we sweep over "power law" settings for our Fourier features. We find that tuning this simple parameterization is able to perform on par with the optimized feature maps.

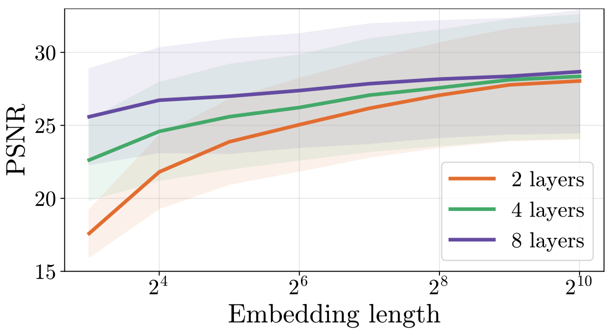
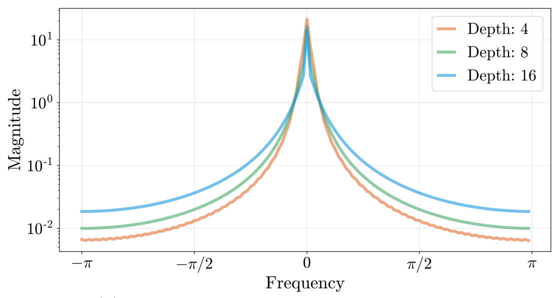


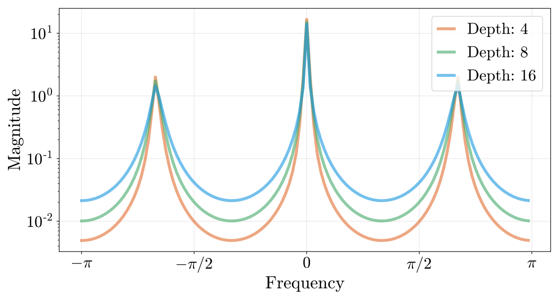
Figure 3.12: In a 2D image regression task (explained in Section 3.6) we find that shallower networks require more Fourier features than deeper networks. This is explained by the frequency spreading effect shown in Figure 3.13. In this experiment we use the *Natural* image dataset and a Gaussian mapping. All of the network layers have 256 channels, and the networks are trained using an Adam [86] optimizer with a learning rate of 10^{-3}.

3.7 Discussion

We leverage NTK theory to show that a Fourier feature mapping can make coordinate-based MLPs better suited for modeling functions in low dimensions, thereby overcoming the spectral bias inherent in coordinate-based MLPs. We experimentally show that tuning the Fourier feature parameters offers control over the frequency falloff of the combined NTK and significantly improves performance across a range of graphics and imaging tasks. These findings shed light on the burgeoning technique of using coordinate-based MLPs to represent 3D shapes in computer vision and graphics pipelines, and provide a simple strategy for practitioners to improve results in these domains. This technique is leveraged utilized in all of the remaining projects in this dissertation.



(a) NTK Fourier spectrum with basic mapping



(b) NTK Fourier spectrum with basic mapping and an additional frequency

Figure 3.13: Each frequency included in a Fourier embedding is "spread" by the NTK, with deeper NTKs causing more frequency spreading. We posit that this frequency spreading is what enables an MLP with a sparse set of Fourier features to faithfully reconstruct a complex signal, which would be poorly reconstructed by either sparse Fourier feature regression or a plain coordinate-based MLP.

Chapter 4

Initializing Coordinate-Based Networks

In the previous chapters we explored the potential of representing complex low-dimensional signals using deep fully-connected neural networks. However, one limitation of these neural representations is that computing network weights θ that reproduce a given signal typically requires solving an optimization problem by running many steps of gradient descent. This can take between seconds (when encoding a small image) and hours (when solving an inverse problem to recover a high resolution radiance field, as in NeRF [123]). Common approaches to address this issue include concatenating a latent vector to the input coordinate and supervising a single neural network to represent an entire class of signals [115, 137], or training a hypernetwork to map from signal observations (or a latent code) to MLP weights [179, 177]. However, each of these strategies is restricted to representing only signals within its learned latent space, potentially limiting its ability to express previously unseen target signals.

Recent work [180] has shown that optimization-based meta-learning can dramatically reduce the number of gradient descent steps required to optimize a neural representation to encode a new signal in the case of signed distance fields of 2D and 3D shapes. In this work, we propose learning the weight initialization for neural representations across a wide variety of underlying signal types, such as images, volumetric data, and 3D scenes. We show that compared to a standard random initialization, using fixed, learned values for the initial network weights acts as a strong prior that enables both faster convergence during optimization and better generalization when only partial observations of the target signal are available. In the context of using neural representations for 3D reconstruction from images, a learned initialization specialized to a particular ShapeNet [21] class allows the network to recover 3D shape from a single image over the course of optimization, whereas a standard randomly initialized network fails unless provided with multiple input views. Given a meta-training set consisting of observations of different signals sampled from a fixed underlying class, our setup applies an optimization-based meta-learning algorithm (MAML [41] or Reptile [129]) in order to produce initial weights better suited for representing that specific signal class (e.g., face images from CelebA [107] or 3D chairs from ShapeNet [21]).

The biggest advantage of our approach is its simplicity. Given an existing framework for test-

This chapter is based on joint work published at CVPR 2021 [194]

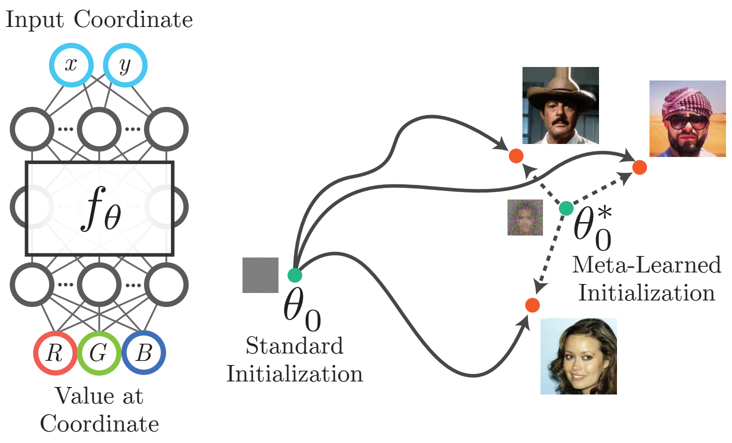


Figure 4.1: A coordinate-based MLP, illustrated on the left, takes a coordinate as input and outputs a value at that location. For example, the network could take in a pixel coordinate (x, y) and emit the (R, G, B) color at that pixel as output, thereby representing a 2D image. The network weights θ are typically optimized via gradient descent to produce the desired image, as depicted on the right. However, finding good parameters can be computationally expensive, and the full optimization process must be repeated for each new target. We propose using meta-learning to find initial network weights θ_0^* that allow for faster convergence and better generalization.

time optimization of a neural representation, implementing an outer loop with MAML or Reptile update steps only requires a few extra lines of code and a dataset of training examples. Once the meta-learning phase is complete, the learned initial weights can be stored and later reloaded in place of a standard network initialization whenever a new signal needs to be encoded. This minor implementation change can significantly alter the behavior of the network during optimization.

4.1 Related Work

Neural Representations Neural representations have recently risen to prominence as compact representations for 3D shapes. These methods represent shapes as implicit surfaces defined as a level set of an MLP network and enable full object reconstruction from incomplete 3D point cloud data or depth scans [26, 34, 53, 55, 76, 115, 118, 137]. Later work combined this idea with various formulations of differentiable rendering to recover neural representations of 3D shape using only 2D image observations [105, 104, 123, 131, 177, 215].

Coordinate-based neural networks have also been used to represent other low-dimensional signals, such as 2D images, where such networks (when trained via genetic algorithms) have been referred to as compositional pattern–producing networks [185]. Recent works have shown that standard ReLU MLPs fail to adequately represent fine details in these complex low-dimensional signals due to a spectral bias [149] and address this issue by either replacing the ReLU activations with sine functions [179] or by lifting the input coordinates into a Fourier feature space [193]. Our work makes use of these observations and presents a technique that enables a coordinate-based MLP to learn from the process of fitting many signals within a category so that it can quickly optimize to fit any new signal using fewer steps and fewer observations.

Meta-learning Meta-learning typically addresses the problem of few-shot learning, where some examples of a given task (including training and test data) are used to learn an algorithm that achieves better performance on new, previously unseen instances of the same task. A prototypical example from computer vision is few-shot image classification, where a network must learn to differentiate between new classes at test time based on only a small number of labeled instances of each class.

Most relevant to this work are optimization-based meta-learning algorithms such as Model-Agnostic Meta Learning (MAML) [41] and Reptile [129], as well as various extensions [6, 40, 42, 98, 153]. Given a network architecture for performing a task, these methods use an outer loop of gradient-based learning to find a weight initialization that allows the network to more efficiently optimize for new instances of the underlying task at test time. These methods assume the use of a standard gradient-based optimization method such as stochastic gradient descent or Adam [86] at test time, making them easy to layer on top of existing implementations, as opposed to more complex methods such as Ravi *et al.* [156], which trains a "meta-learner" LSTM network to perform gradient updates for the underlying task. An exhaustive review of meta-learning algorithms is provided in the survey paper by Hospedales *et al.* [68].

MetaSDF [180] specifically applies this idea of learning a weight initialization to the task of fitting neural representations to represent signed distance fields, and shows that this strategy achieves much more rapid convergence than standard approaches such as DeepSDF [137]. Our work applies meta-learning to neural representations for a wider variety of underlying signal types and further explores the power of using initial weight settings as a prior.

4.2 Overview

We define a finite signal T as a function mapping from a bounded set $C \in \mathbb{R}^d$ to $\mathbb{R}^n$, where we refer to elements $\mathbf{x} \in C$ as d-dimensional coordinates. Examples include images (mapping from 2D pixel coordinates to 3D color values) or volumetric representations for 3D shapes (mapping from 3D locations to 4D tuples of color and density). A coordinate-based neural representation f_θ for T is a fully connected neural network with d input and n output channels whose weights θ are optimized such that f_θ matches T as closely as possible for all coordinates in $\mathbf{x} \in C$.

If direct pointwise observations $\{(\mathbf{x}_i, T(\mathbf{x}_i)\}_i$ of the signal T are available, f_θ can be supervised by gradient descent using a simple L2 loss:

$$L(\theta) = \sum_i \|f_\theta(\mathbf{x}_i) - T(\mathbf{x}_i)\|_2^2 \, . \tag{4.1}$$

Let θ_0 denote the initial network weights before any gradient steps are taken, and let θ_i denote the weights after i steps of optimization. Basic gradient descent applies the rule:

$$\theta_{i+1} = \theta_i - \alpha \nabla_\theta L(\theta)|_{\theta=\theta_i} \,, \tag{4.2}$$

with a learning rate parameter α, whereas more sophisticated optimizers such as Adam [86] keep track of gradient moments over time to redirect the optimization trajectory. Given a fixed budget of m optimization steps, different initial weight values θ_0 will result in different final weights θ_m and signal approximation error $L(\theta_m)$. When emphasizing the functional dependence of θ_m on the initial weights and a particular signal, we will write $\theta_m(\theta_0, T)$.

It is often the case that only indirect observations of T are available, taken through some forward measurement model $M(T, \mathbf{p})$. For example, if T is a 3D object, $M(T, \mathbf{p})$ could be a 2D image captured of the object from camera pose $\mathbf{p}$. In this case, recovering a neural representation for T from observations $\{\mathbf{p}_i, M(T, \mathbf{p}_i)\}_i$ requires solving an inverse problem by taking gradient steps on a loss that incorporates the forward model M:

$$L_M(\theta) = \sum_i \|M(f_\theta, \mathbf{p}_i) - M(T, \mathbf{p}_i)\|_2^2 \,. \tag{4.3}$$

If M discards too much information about T or the set of provided observations is too small, the resulting network f_θ may not match T closely. For example, accurately recovering a 3D object from a single 2D view may not be possible without strong a priori knowledge of the object's shape.

Optimizing initial weights

We assume that we are given a dataset of observations of signals T from a particular distribution $\mathcal{T}$ (e.g., 2D face images or 3D chairs) and our goal is to find initial weights θ_0^* that will result in the lowest possible final loss $L(\theta_m)$ when optimizing a network f_θ to represent a new, previously unseen signal from the same distribution:

$$\theta_0^* = \arg\min_{\theta_0} E_{T\sim\mathcal{T}}[L(\theta_m(\theta_0, T))] \tag{4.4}$$

This problem of trying to learn the initial weights of a network to serve as a good starting point for gradient descent across a distribution of tasks is addressed by a variety of optimization-based meta-learning algorithms, such as MAML [41] and Reptile [129].

MAML [41] Given a task T, calculating the weight values $\theta_m(\theta_0, T)$ requires taking m optimization steps, which are collectively referred to as the *inner loop*. MAML wraps an *outer loop* of meta-learning around this inner loop in order to learn the initial weights θ_0. Each outer loop samples a signal T_j from $\mathcal{T}$ and applies the update rule:

$$\theta_0^{j+1} = \theta_0^j - \beta \nabla_\theta L(\theta_m(\theta, T_j))|_{\theta=\theta_0^j} \tag{4.5}$$

with meta-learning step size β. This update rule applies gradient descent to the loss on the weights $\theta_m(\theta_0^j, T_j)$ resulting from the inner loop optimization.

Reptile [129] Reptile uses the same meta-learning setup as MAML but applies a simpler update rule that does not require calculating second-order gradients:

$$\theta_0^{j+1} = \theta_0^j - \beta(\theta_m(\theta_0^j, T_j) - \theta_0^j)\,. \tag{4.6}$$

This rule moves the previous weight initialization θ_0^j in the direction of the task-optimized weights $\theta_m(\theta_0^j, T_j)$.

Experimental setup

The meta-learning algorithms described previously are conceptually simple, requiring no changes to the architecture or optimization procedure of a coordinate-based neural representation when given a new signal to encode at "test time" (after meta-learning is complete). These algorithms produce only a set of initial network weights θ_0^* that are then used as a starting point for gradient descent. Test-time optimization on new signals is not limited to the same number of steps m as were used in the inner loop during meta-learning; indeed, at test time we often observe benefits from optimizing for significantly more iterations than were used during the inner loop of the meta-learning algorithm.

MAML is typically able to produce a better initialization than Reptile given a fixed number of inner loop steps m, but Reptile can be unrolled for more inner loop steps because it is less memory-intensive than MAML. For some tasks, MAML's limited number of inner loop steps means that it can only observe a small percentage of the observations of a target signal. In these cases, we use Reptile to maximize the number of different observations seen over the course of the inner loop. Experimentally we find it beneficial to unroll more steps for more complex tasks.

Each of our experiments involves two phases:

1. *Meta-learning*, where we use MAML or Reptile in combination with a training dataset of example tasks (observations of different signal instances) to optimize initial network weights for that class of signals, and
2. *Test-time optimization*, where we use standard gradient-based optimization to fit the weights of a network to observations of a previously unseen signal from the same class.

We aim to answer the following question: how do different initial network weight settings influence the ability of a neural representation to fit to a new signal during test-time optimization?

4.3 Implementation details

We found that modifying the weight initialization for these coordinate-based networks drastically changed their convergence behavior during test-time optimization. As a result, we tuned the optimization method and hyperparameters for each part of each experiment (using held-out validation sets) in order to provide the fairest possible comparison and to not bias the results against the non-meta-learned initializations. For example, we often found that SGD outperformed Adam when doing test-time optimization using meta-learned initializations, but that Adam was significantly better than SGD with a standard random initialization.

All experiments are implemented in JAX [15]. Each experiment is trained on either a single NVIDIA V100, 2080 Ti, or 3080 Ti. In all cases where the Adam optimizer [86] is used, we keep the standard parameter choices for $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$.

Image regression

For this task we use a SIREN [179] architecture ($\omega_0 = 200$) with 5 layers of 256 channels each. For the randomly initialized *Standard* baseline, we use the specific initialization procedure as proposed in the SIREN paper.

MAML [41] is trained for 150K iterations. Each iteration has an outer batch size of 3 target images. The inner batch contains all pixels of the target image. The outer loop uses the Adam optimizer with learning rate of 10^{-5}. The inner loop performs two steps of gradient descent with a learning rate of 10^{-2}.

During test-time optimization, we use gradient descent with learning rate of 10^{-2} when starting from the MAML initial weights. For the baseline methods (*Standard*, *Mean*, *Matched*, *Shuffled*) we used Adam with learning rate of 10^{-4}, which performed significantly better than than gradient descent.

CT reconstruction

For this task we use an MLP with 5 layers of 256 channels each. The network uses a ReLU activation after each layer with the exception of the last layer, which has a sigmoid activation. Prior to inputting the coordinates into the network, we encode them using random Fourier features sampled from a normal distribution with $\sigma = 30$, as was done in Tancik *et al.* [193].

Reptile [129] is trained for 100K iterations. Each iteration has an outer batch size of 1. The inner batch contains 20 CT projections, each with 256 measurements, taken from a randomly sampled direction. The outer loop uses the Adam optimizer with learning rate of 5×10^{-5}. The inner loop performs 12 inner loop steps of gradient descent with a learning rate of 10^{1}.

ShapeNet [21] view synthesis

We use a simplified NeRF [123] model for our view synthesis tasks. This model uses a single network rather than two networks (coarse and fine), and we do not provide view directions as input. The network is an MLP with 6 layers, each with 256 channels and ReLU activations. As in NeRF [123], we apply a positional encoding to each input coordinate with the form

$$\bigcup_{i=0}^{N} \left\{ \cos\left(2^{fi/N} x\right), \sin\left(2^{fi/N} x\right) \right\}, \tag{4.7}$$

with $N = 20$ encodings and log-max frequency $f = 8$. We accumulate 128 samples per ray for rendering.

Reptile is trained for 100K iterations with an outer batch size of 1. The inner loop step optimizes over a batch of 128 rays. We perform 32 inner loop steps for every outer loop step. The outer loop uses the Adam optimizer with learning rate 5×10^{-4} for the *Chairs* scenes and 5×10^{-5} for the *Lamps* and *Cars* scenes.

The test-time optimization parameters vary depending on the scene and the number of views available during meta-learning. Each experiment uses an inner batch of 64 rays. The *Shuffled* and *Matched* initializations are computed based on the *MV Meta* weights. For the 25 view chair reconstruction, we use stochastic gradient descent with a learning rate of 10^{-1} for the Reptile initialization; for the standard initialization, we use Adam with a learning rate of 10^{-4}.

Phototourism [77] view synthesis

We use the same architecture as described in §4.3. Reptile is trained for 150K iterations with an outer batch size of 1. The inner loop step optimizes over a batch of 64 rays, with 128 volume rendering samples per ray. The outer loop uses the Adam optimizer with a learning rate of 5^{-4}. We train with 64 inner loop steps using gradient descent with a learning rate of 10. We compare to *Basic NeRF* which has the same setup, but only one inner step. For *Basic NeRF* we train *Trevi* for 60K iterations, *Brandenburg* for 100K iterations, and *Sacre Coeur* for 200K iterations. To transfer the appearance of a new photo during test-time optimization, we take 150 gradient steps with a learning rate of 10.

4.4 Results

We present results on 2D image regression, 2D computed tomography (CT) reconstruction, 3D object reconstruction, and 3D scene reconstruction. For each task, we demonstrate the benefits of using meta-learned initial weights optimized to reconstruct a specific class of signals.

For 2D image regression, a meta-learned weight initialization leads to faster convergence and better performance during test-time optimization. For CT reconstruction, it allows for better reconstruction quality from fewer supervision views during test-time optimization. For 3D shape reconstruction from images, it allows for faster convergence at test time and makes single view reconstruction possible. For Phototourism landmark reconstruction, it can be optimized at test time to transfer the appearance of a single input image onto the whole landmark, which can then be rendered from novel camera views.

Tasks

Image regression A prototypical example of a coordinate-based neural representation is an MLP optimized to represent a 2D image [179, 193] by taking in 2D pixel coordinates and outputting RGB color values. We consider four different distributions $\mathcal{T}$: images of faces (*CelebA* [107]), natural images (*Imagenette* [69]), images of text (*Text*), and 2D signed distance fields of simple curves (*SDF*). Each category contains around ten thousand examples. Given a sampled image $T \sim \mathcal{T}$, we provide all 178×178 pixels as observations for optimizing the network weights θ in the inner loop. Since this task is not memory constrained, we use MAML to meta-learn the weights over 2 unrolled gradient steps (separately for each category $\mathcal{T}$). In each of these inner loop steps, the entire image is reconstructed and used to calculate the loss. For the MLP f_θ, we use 5 layers with 256 channels each and sine function nonlinearities, as in SIREN [177].

CT reconstruction Computed tomography (CT) is a widely used medical imaging technique that captures projective measurements of the volumetric density of a target object. Tancik *et al.* [193] use a coordinate-based neural representation to reconstruct a 2D signal from 1D integral projections; the underlying MLP f_θ takes in a 2D coordinate and outputs a scalar volume density at that location. Here $\mathcal{T}$ is a dataset of 2048 randomly generated 256×256 pixel Shepp-Logan phantoms [175], where we provide 2D integral projections of a bundle of 256 parallel rays from a random angle as the measurement for each sampled signal T during meta-learning. We use Reptile to meta-learn the initial weights over 12 unrolled gradient steps. We found this to outperform MAML, which was limited to 3 unrolled steps due to memory constraints. For the MLP f_θ, we use 5 layers with 256 channels each and ReLU nonlinearities, and we apply random Fourier features to the input coordinates [193].

View synthesis for ShapeNet [21] objects The goal of view synthesis is to generate a novel view of a scene from a set of reference images. We use NeRF described in chapter 2 for this task. The NeRF network is optimized to minimize the residual of re-rendering each of the input reference images from their respective camera poses. In our view synthesis experiments, we use a simplified NeRF model (simple-NeRF) that maintains the same image supervision and volume rendering context. Unlike the original NeRF model, we do not feed in the viewing direction and we use a single model instead of the two “coarse” and “fine” models used by NeRF.

For view synthesis on objects from the ShapeNet [21] dataset, we consider three categories $\mathcal{T}$: *Chairs*, *Cars*, and *Lamps*. We provide 25 128×128 pixel reference images during meta-learning for each 3D object T. The reference viewpoints are randomly distributed on a sphere and are oriented towards the target object, and each object is oriented in the canonical coordinate frame. The scenes are lit by a randomly selected environment map [51] and rendered using ray tracing. We use Reptile to meta-learn the initial weights (for each shape category) over 32 unrolled gradient steps. For the MLP f_θ, we use 6 layers with 256 channels each and ReLU nonlinearities, and apply a positional encoding to the input coordinates [123].

View synthesis for Phototourism [77] scenes This dataset consists of thousands of posed tourist photographs of famous landmarks. Our objective is to use these images to create an underlying representation that can be explored and rendered from novel viewpoints with varying lighting conditions. The primary challenge is the diversity of the capture conditions: the photos are taken with different lighting conditions, camera hardware, camera viewpoint, and varying transient objects like people and cars. Each underlying dataset $\mathcal{T}$ for meta-learning θ_0^* consists of images of a single landmark (*Trevi*, *Sacre Couer*, or *Brandenburg*); the category is the overall 3D structure of the

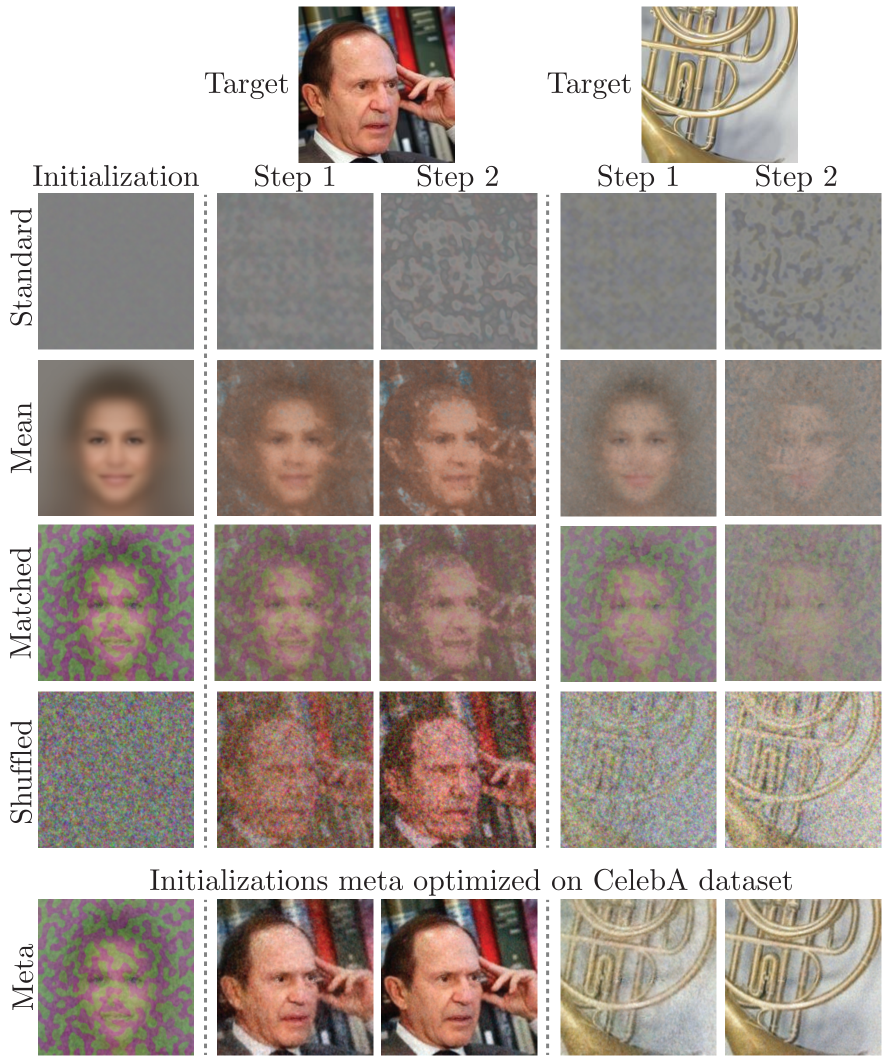


Figure 4.2: **Faster convergence:** Examples of optimizing a network to represent a 2D image from different initial weight settings. The meta-learned initialization (*Meta*) is specialized for the class of human face images but still helps speed up convergence on other natural images (right). Non-meta-initialized networks take 10 to 20 times as many iterations to reach the same quality as the meta-initialized network does after only 2 gradient steps (see Table 4.1).

landmark itself, and the signal is its particular appearance (resulting from the time of day, lighting, weather conditions, etc) within a single photo. If a standard NeRF model is trained directly on this data, it learns a blurry representation of the scene that roughly corresponds to the mean of the environmental conditions. NeRF in the Wild [112] explores these shortcomings and proposes extensive architectural modifications to account for the variations. We find that these shortcomings can be addressed to some degree solely with a better initialization and no architectural changes.

Init. Method	2 Step PSNR ↑	# of iters to match ↓
Standard	10.88	37.92 ± 6.31
Mean	14.48	25.59 ± 4.57
Matched	13.73	26.32 ± 4.17
Shuffled	16.29	25.80 ± 4.02
Meta	**30.37**	-

Table 4.1: Comparison of different initialization methods on an image regression task using the CelebA dataset. We report reconstruction PSNR after two steps of test-time optimization. The meta-learned initialization (*Meta*) significantly outperforms all other initializations. We also report the average number of iterations necessary to match the accuracy of *Meta* after two steps.

		Task			
		CelebA	Imagenette	Text	SDF
Init.	CelebA	**30.37**	26.44	21.53	36.45
	Imagenette	28.51	**27.07**	22.63	34.80
	Text	14.65	15.83	**27.85**	23.14
	SDF	19.80	20.05	17.23	**51.73**

Table 4.2: PSNR comparison of four different learned initializations for image regression. Each row corresponds to an initialization meta-learned over a different underlying image dataset. The columns indicate which dataset images are sampled from during testing. The best initialization for each task (bolded) is the one specifically optimized on training images drawn from the same dataset. We observe that initializations transfer better between more similar datasets (*CelebA* and *Imagenette*, both natural images) and poorly between less similar datasets (the frequency spectrum of *Text* images is unlike that of the other categories).

We apply meta-learning to the same simple-NeRF model from the ShapeNet experiment. The meta-training dataset for each landmark consists of thousands of images with varying resolution and intrinsic/extrinsic camera parameters. We use Reptile to meta-learn the initial weights (for each landmark) over 64 unrolled gradient steps. At test time, we optimize the simple-NeRF (starting from the initial weights θ_0^* for that landmark) to reproduce the appearance of a new image, and then render that simple-NeRF from other viewpoints. For the underlying MLP f_θ, we use 6 layers with 256 channels each and ReLU nonlinearities, and apply positional encoding to the input coordinates [123].

Baselines

As well as a *Standard* randomly initialized network (Glorot *et al.* [57]), we compare to various other initialization schemes in several of our experimental settings:

Init.	PSNR			
Method	1 Views	2 Views	4 Views	8 View
Standard	13.63	14.15	16.31	21.49
Mean	14.72	15.39	17.43	25.19
Matched	14.07	15.51	20.25	24.77
Shuffled	13.64	14.17	16.69	22.09
Meta	**15.09**	**18.70**	**22.00**	**27.34**

Table 4.3: Comparison of initialization methods on a CT reconstruction task. Each "view" consists of 256 parallel rays. The data-dependent prior acquired during meta-learning improves reconstruction quality when fewer views are observed.

- *Mean:* we optimize a network from scratch such that its output matches the mean signal $E_{T\sim\mathcal{T}}[T]$ from the current class $\mathcal{T}$.
- *Matched:* we optimize a network from scratch such that its output matches the output of a network using the meta-learned initialization for the current class $\mathcal{T}$.
- *Shuffled:* we randomly permute the weights (within each network layer) of the meta-learned initialization θ_0^* for the current class $\mathcal{T}$.

Both the *Mean* and *Matched* baselines demonstrate the difference between having a good initialization in *signal* space versus *weight* space—despite *Mean* and *Matched* being initialized so that the loss against a randomly sampled signal will be low, they are a worse starting point for gradient descent than the actual meta-learned initial weights. The *Shuffled* baseline demonstrates that matching the statistical distribution of the meta-learned initial weights is not sufficient for better convergence or generalization. We find that using the Adam [86] optimizer performs best for all of the baseline initializations, but that standard stochastic gradient descent works best for the meta-learned initializations (we choose the best optimizer and hyperparameters for each task and initialization using a held-out validation set).

Faster convergence

Image regression In Figure 4.2, we visualize the network output for a variety of initial weight settings, showing the output images after 0, 1, and 2 gradient steps of test-time optimization. The meta-learned initial weights are optimized to represent face images (CelebA [107]). When using the learned initial weights θ_0^* (*Meta*), the target image is already clearly visible after the very first step. In contrast, the baseline initialization methods take an order of magnitude more iterations to represent the target image to the same accuracy (see Table 4.1). The *Mean*, *Matched*, and *Shuffled* baselines perform better than the completely random *Standard* initialization, but still take over ten times as many iterations to reach the same quality as the meta-initialized network can after 2 steps. In particular, this demonstrates that neither matching the image space output nor the statistical distribution of the meta-learned weights is sufficient for achieving a similar speedup.

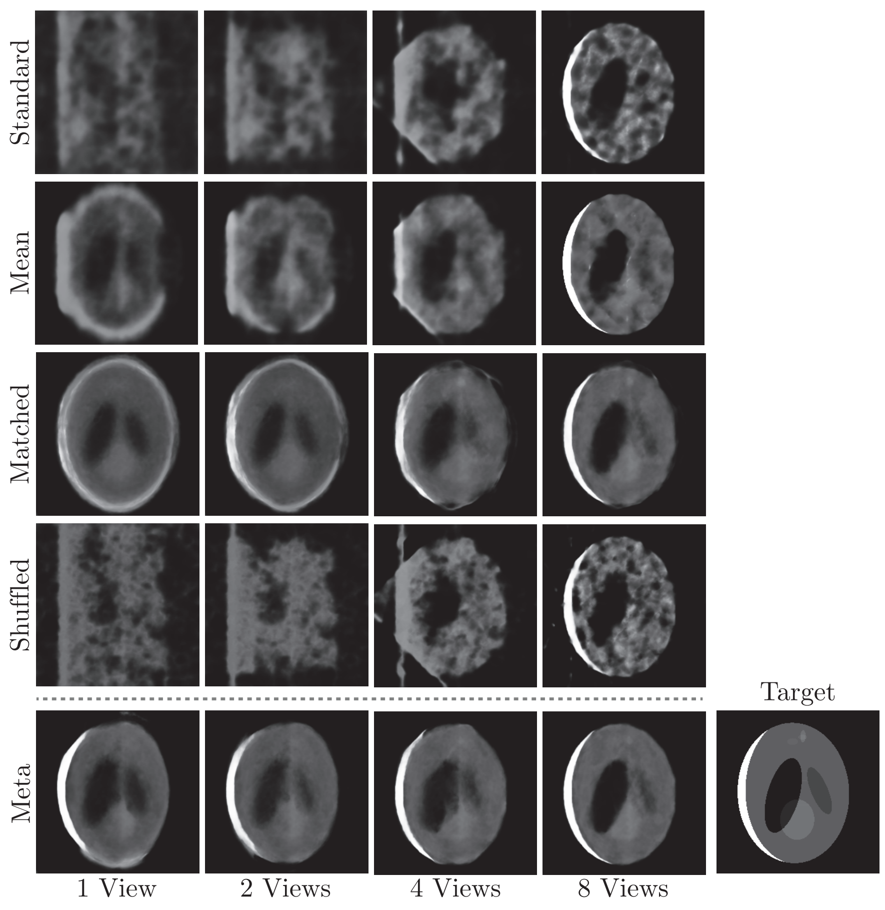


Figure 4.3: **Sparse Recovery:** Examples of CT reconstructions of a Shepp-Logan phantom from a sparse set of views. The meta-learned initial weights encode a data-dependent prior that improves reconstruction in the limited data regime.

View synthesis for ShapeNet [21] objects In Figure 4.5, we plot the image reconstruction accuracy for a held-out test set of objects from the *Chair* category. During test-time optimization, 25 views are observed. We find that starting from the optimized weights θ_0^* allows the network to recover the chair more quickly compared to the *Standard* weight initialization. We note that after many steps, both methods end up at a similar quality.

Generalizing from partial observations

Image regression within a category We perform meta-learning experiments across multiple datasets to determine the extent that the optimized weight initialization acts as a class-specific prior. We compare initializations trained on four different image datasets (*CelebA*, *Imagenette*, *Text*, and *SDF*). Table 4.2 presents a confusion matrix demonstrating that optimizing the network initialization

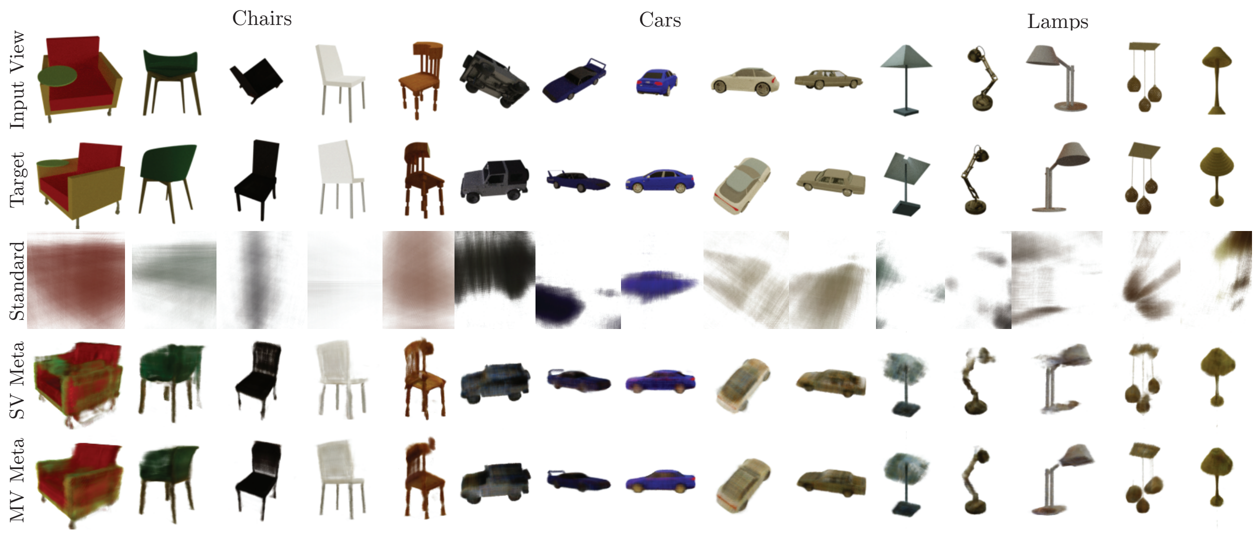


Figure 4.4: **Single view reconstructions of ShapeNet [21] objects.** The simple-NeRF formulation relies on multi-view consistency for supervision and therefore fails if naively applied to the task of single view reconstruction, as seen in the *Standard* column. However, if the model is trained starting from meta-learned initial weights, it is able to recover 3D geometry. The *MV Meta* initialization has access to multiple views per object during meta-learning, whereas the *SV Meta* initialization only has access to a single view per object during meta-learning. All methods only receive a single input view during test-time optimization.

does in fact induce a dataset-dependent prior, with each learned initialization generalizing best to the same dataset distribution it was trained on.

CT reconstruction from sparse views We report the reconstruction quality over a test set of phantoms given varying numbers of views at test time in Table 4.3 and visualize one test example in Figure 4.3. We observe poor reconstructions from the *Standard* initialization when few views are provided. The meta-learned initializations are consistently able to match the PSNR of *Standard* with half as many views. The *Mean* initialization is generated by training a network to reconstruct the mean of the training phantoms. It is better able to preserve the structure of the phantom compared to *Standard* but still performs worse than the meta-learned initializations.

Single image view synthesis for ShapeNet [21] A simple-NeRF model with a *Standard* random initialization relies on multi-view consistency to reconstruct the appearance of a 3D object. With only a single view, this naïve model is unable to recover any meaningful shape. We find that a learned initialization "bakes in" a class-specific shape prior that enables the recovery of 3D geometry (Figure 4.4, Table 4.4). We can meta-learn an effective weight initialization for single-view reconstruction by optimizing over a dataset with 25 training views of each object (*MV Meta*).

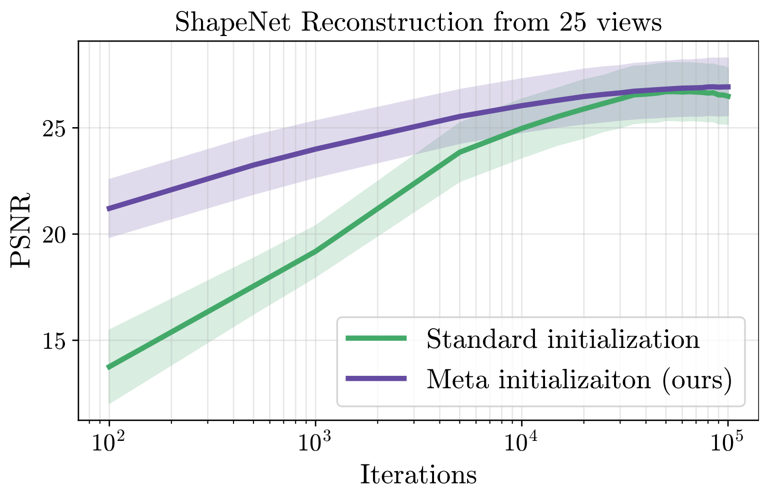


Figure 4.5: Reconstruction quality over the course of training for models optimized to reconstruct ShapeNet chairs from a set of 25 reference images. The model starting from the meta-learned initial weights outperforms the network using a standard random initialization throughout training.

	PSNR		
	Chairs	Cars	Lamps
Standard	12.49	11.45	15.47
MV Matched	16.40	22.39	20.79
MV Shuffled	10.76	11.30	13.88
MV Meta	**18.85**	**22.80**	**22.35**
SV Meta	16.54	22.10	20.95

Table 4.4: Metrics for single image ShapeNet reconstructions using a simple-NeRF model. See Figure 4.4 for image examples and §4.4 for experimental details.

We find that this prior persists even if the meta-training dataset only contains a single reference image per scene (*SV Meta*), meaning that the meta-learning phase has no access to multiview information for any particular object.

View synthesis with appearance transfer for Phototourism [77] As described in §4.4, these images have different camera poses and visual appearance (lighting, sky, etc.) as they are taken by tourists at different times. Our goal at test time is to explore the landmark from varying camera viewpoints but rendered with the same appearance as in a target photograph. In every step of the meta-learning outer loop, we supervise the simple-NeRF model to match the appearance of a random photo of the landmark (with varying pose and appearance). We find that performing

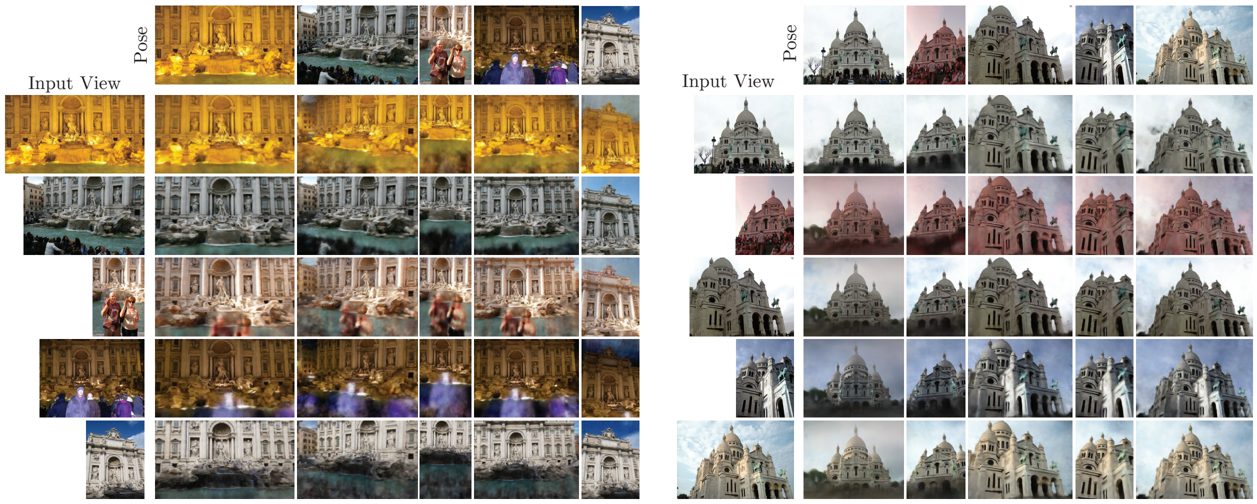


Figure 4.6: Reconstructions of the Trevi Fountain and Sacre Coeur landmarks from the Phototourism dataset [77]. The meta-learning algorithm is run over tourist images taken at different locations and times. During the test-time optimization, the neural representation is trained to recover the input view on the left. The strong prior from the initialization captures the underlying geometry, allowing us to render views from the camera positions of the images in the top row while retaining the appearance of the input view.

	PSNR		
	Trevi	Sacre Coeur	Brandenburg
Basic NeRF	17.14	17.59	17.77
Meta	**19.35**	**19.33**	**19.11**

Table 4.5: Reconstruction results on Phototourism data. Multi-view data with consistent appearance is not available in this dataset, so we optimize on one half of an image and report image metrics on the other half. We compare our Reptile setup (*Meta*) with a standard NeRF network trained on all images of the landmark and then test-time optimized to fit each held-out target image. This is equivalent to training Reptile with one inner loop gradient step.

test-time optimization using a single new photograph allows us to render convincing unobserved viewpoints of the scene with the same environmental conditions.

In Figure 4.6, we show results for two landmarks. We test-time optimize the meta-learned weights for five target images (shown on the left side of the grid), taking 150 gradient steps for each image. We then render each of the resulting simple-NeRF networks from the five different viewpoints (shown in the row above the grid). The result is an image from the camera position of the corresponding top row image and matching the appearance of the left column image.

Quantitative evaluation on the Phototourism dataset is difficult as multiple views with the same environmental conditions do not exist. To overcome this, for Table 4.5 we optimize and evaluate on the same image, by optimizing to match the appearance of the left half of the image and subsequently evaluating metrics on the right half. For comparison, we train a simple-NeRF model with a standard random initialization from scratch on each landmark, then test-time optimize it to match the left half of each new view before evaluating it on the right half. This is algorithmically equivalent to Reptile with one inner optimization step. We find that unrolling Reptile for 64 inner steps performs better, producing significantly clearer renderings of the landmark.

4.5 Discussion

Our results show that simply modifying a coordinate-based neural representation's initial weight values can guide the network along a significantly better optimization trajectory, without changing the underlying architecture or test-time optimization procedure. These meta-learned initial weights can result in faster convergence or act as a strong prior for representing signals from a given distribution. This partially ameliorates a major shortcoming of neural representations (separately optimizing a network for each new signal) without limiting their representational power.

There are many additional directions to explore, such as applying more sophisticated meta-learning algorithms or more precisely characterizing the geometry of weight space for these networks. One limitation of our current approach is that it requires a sizable dataset of example signals from a target distribution in order to derive beneficial initial weights. Another shortcoming is that our method still requires some amount of test-time optimization.

As the number of use cases for neural representations continues to rapidly expand, we believe this work takes an important step toward understanding the importance of their initial weights and optimization behavior. In the following chapters we will explore new use cases for neural representations.

Chapter 5

Scaling Neural Radiance Fields

In this chapter we investigate scaling up NeRFs to arbitrarily large scenes. In the previous chapters, we focused on small-scale and object-centric reconstructions. Though some methods address scenes the size of a single room or building [8], these are generally still limited and do not naïvely scale up to *city-scale* environments. Applying these methods to large environments typically leads to significant artifacts and low visual fidelity due to limited model capacity.

Reconstructing large-scale environments enables several important use-cases in domains such as autonomous driving [136, 96, 214] and aerial surveying [38, 101]. One example is mapping, where a high-fidelity map of the entire operating domain is created to act as a powerful prior for a variety of problems, including robot localization, navigation, and collision avoidance. Furthermore, large-scale scene reconstructions can be used for closed-loop robotic simulations [36]. Autonomous driving systems are commonly evaluated by re-simulating previously encountered scenarios; however, any deviation from the recorded encounter may change the vehicle's trajectory, requiring high-fidelity novel view renderings along the altered path. Beyond basic view synthesis, scene conditioned NeRFs are also capable of changing environmental lighting conditions such as camera exposure, weather, or time of day, which can be used to further augment simulation scenarios.

Reconstructing such large-scale environments introduces additional challenges, including the presence of transient objects (cars and pedestrians), limitations in model capacity, along with memory and compute constraints. Furthermore, training data for such large environments is highly unlikely to be collected in a single capture under consistent conditions. Rather, data for different parts of the environment may need to be sourced from different data collection efforts, introducing variance in both scene geometry (*e.g.* , construction work and parked cars), as well as appearance (*e.g.* , weather conditions and time of day).

We extend NeRF with appearance embeddings and learned pose refinement to address the environmental changes and pose errors in the collected data. We additionally add exposure conditioning to provide the ability to modify the exposure during inference. We refer to this modified model as a Block-NeRF. Scaling up the network capacity of Block-NeRF enables the ability to represent increasingly large scenes. However this approach comes with a number of limitations; rendering

This chapter is based on work completed when the author was an intern at Waymo LLC, published at CVPR 2022 [192] (U.S. Patent Pending App. No. 18/074,371).

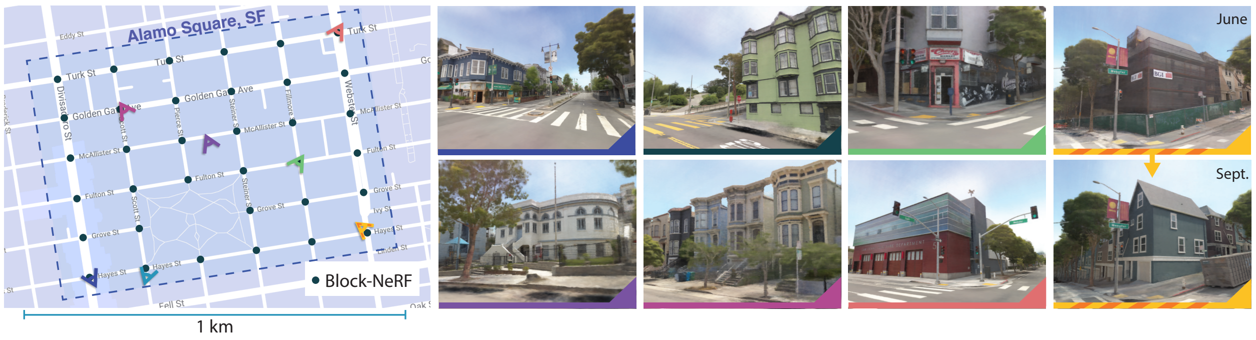


Figure 5.1: **Block-NeRF** is a method that enables large-scale scene reconstruction by representing the environment using multiple compact NeRFs that each fit into memory. At inference time, Block-NeRF seamlessly combines renderings of the relevant NeRFs for the given area. In this example, we reconstruct the Alamo Square neighborhood in San Francisco using data collected over 3 months. Block-NeRF can update individual blocks of the environment without retraining on the entire scene, as demonstrated by the construction on the right. Video results can be found on the project website waymo.com/research/block-nerf.

time scales with the size of the network, networks can no longer fit on a single compute device, and updating or expanding the environment requires retraining the entire network.

To address these challenges, we propose dividing up large environments into individually trained Block-NeRFs, which are then rendered and combined dynamically at inference time. Modeling these Block-NeRFs independently allows for maximum flexibility, scales up to arbitrarily large environments and provides the ability to update or introduce new regions in a piecewise manner without retraining the entire environment as demonstrated in Figure 5.1. To compute a target view, only a subset of the Block-NeRFs are rendered and then composited based on their geographic location compared to the camera. To allow for more seamless compositing, we propose an appearance matching technique which brings different Block-NeRFs into visual alignment by optimizing their appearance embeddings.

5.1 Related Work

Large Scale 3D Reconstruction

Researchers have been developing and refining techniques for 3D reconstruction from large image collections for decades [46, 181, 142, 97, 3, 233], and much current work relies on mature and robust software implementations such as COLMAP to perform this task [170]. Nearly all of these reconstruction methods share a common pipeline: extract 2D image features (such as SIFT [111]), match these features across different images, and jointly optimize a set of 3D points and camera poses to be consistent with these matches (the well-explored problem of bundle adjustment [60, 197]). Extending this pipeline to city-scale data is largely a matter of implementing highly robust and parallelized versions of these algorithms, as explored in work such as Photo Tourism [181] and Building Rome in a Day [3]. Core graphics research has also explored breaking up scenes for fast high quality rendering [110].

These approaches typically output a camera pose for each input image and a sparse 3D point cloud. To get a complete 3D scene model, these outputs must be further processed by a dense multi-view stereo algorithm (*e.g.* , PMVS [47]) to produce a dense point cloud or triangle mesh. This process presents its own scaling difficulties [48]. The resulting 3D models often contain artifacts or holes in areas with limited texture or specular reflections as they are challenging to triangulate across images. As such, they frequently require further postprocessing to create models that can be used to render convincing imagery [174]. However, this task is mainly the domain of novel view synthesis, and 3D reconstruction techniques primarily focus on geometric accuracy.

In contrast, our approach does not rely on large-scale SfM to produce camera poses, instead performing odometry using various sensors on the vehicle as the images are collected [196].

Novel View Synthesis

Given a set of input images of a given scene and their camera poses, novel view synthesis seeks to render observed scene content from previously unobserved viewpoints, allowing a user to navigate through a recreated environment with high visual fidelity.

Geometry-based Image Reprojection. Many approaches to view synthesis start by applying traditional 3D reconstruction techniques to build a point cloud or triangle mesh representing the scene. This geometric "proxy" is then used to reproject pixels from the input images into new camera views, where they are blended by heuristic [18] or learning-based methods [63, 163, 164]. This approach has been scaled to long trajectories of first-person video [88], panoramas collected along a city street [89], and single landmarks from the Photo Tourism dataset [116]. Methods reliant on geometry proxies are limited by the quality of the initial 3D reconstruction, which hurts their performance in scenes with complex geometry or reflectance effects.

Volumetric Scene Representations. Recent view synthesis work has focused on unifying reconstruction and rendering and learning this pipeline end-to-end, typically using a volumetric scene representation. Methods for rendering small baseline view interpolation often use feed-forward networks to learn a mapping directly from input images to an output volume [43, 231], while methods such as Neural Volumes [108] that target larger-baseline view synthesis run a global optimization over all input images to reconstruct every new scene, similar to traditional bundle adjustment.

Neural Radiance Fields (NeRF) [123] combines this single-scene optimization setting with a neural scene representation capable of representing complex scenes much more efficiently than a discrete 3D voxel grid; however, its rendering model scales very poorly to large-scale scenes in terms of compute. Followup work has proposed making NeRF more efficient by partitioning space into smaller regions, each containing its own lightweight NeRF network [157, 158]. Unlike our method, these network ensembles must be trained jointly, limiting their flexibility. Another approach is to provide extra capacity in the form of a coarse 3D grid of latent codes [102]. This approach has also been applied to compress detailed 3D shapes into neural signed distance functions [191] and to represent large scenes using occupancy networks [140].

We build our Block-NeRF implementation on top of mip-NeRF [9], which improves aliasing issues that hurt NeRF's performance in scenes where the input images observe the scene from many different distances. We incorporate techniques from NeRF in the Wild (NeRF-W) [112], which adds a latent code per training image to handle inconsistent scene appearance when applying NeRF to landmarks from the Photo Tourism dataset. NeRF-W creates a separate NeRF for each landmark from thousands of images, whereas our approach combines many NeRFs to reconstruct a coherent large environment from *millions* of images. Our model also incorporates a learned camera pose refinement which has been explored in previous works [216, 186, 100, 207, 219].

Some NeRF-based methods use segmentation data to isolate and reconstruct static [212] or moving objects (such as people or cars) [224, 136] across video sequences. As we focus primarily on reconstructing the environment itself, we choose to simply mask out dynamic objects during training.

Urban Scene Camera Simulation

Camera simulation has become a popular data source for training and validating autonomous driving systems on interactive platforms [5, 85]. Early works [49, 162, 166, 36] synthesized data from scripted scenarios and manually created 3D assets. These methods suffered from domain mismatch and limited scene-level diversity. Several recent works tackle the simulation-to-reality gaps by minimizing the distribution shifts in the simulation and rendering pipeline. Kar *et al.* [80] and Devaranjan *et al.* [35] proposed to minimize the scene-level distribution shift from rendered outputs to real camera sensor data through a learned scenario generation framework. Richter *et al.* [161] leveraged intermediate rendering buffers in the graphics pipeline to improve photorealism of synthetically generated camera images.

Towards the goal of building photo-realistic and scalable camera simulation, prior methods [96, 214, 25] leverage rich multi-sensor driving data collected during a single drive to reconstruct 3D scenes for object injection [25] and novel view synthesis [214] using modern machine learning techniques, including image GANs for 2D neural rendering. Relying on a sophisticated surfel reconstruction pipeline, SurfelGAN [214] is still susceptible to errors in graphical reconstruction and can suffer from the limited range and vertical field-of-view of LiDAR scans. In contrast to existing efforts, our work tackles the 3D rendering problem and is capable of modeling the real camera data captured from multiple drives under varying environmental conditions, such as weather and time of day, which is a prerequisite for reconstructing large-scale areas.

5.2 Background

We build upon NeRF [123] described in Chapter 2 and its extension mip-NeRF [9]. Here, we summarize relevant parts of mip-NeRF. For details, please refer to the original papers.

mip-NeRF Preliminaries

Recall that to enable the NeRF MLPs to represent higher frequency detail [193], the inputs $\mathbf{x}$ and $\mathbf{d}$ are each preprocessed by a componentwise sinusoidal positional encoding γ_{PE}:

$$\gamma_{\text{PE}}(z) = [\sin(2^0 z), \cos(2^0 z), \ldots, \sin(2^{L-1} z), \cos(2^{L-1} z)] \tag{5.1}$$

where L is the number of levels of positional encoding.

NeRF's MLP f_σ takes a single 3D point as input. However, this ignores both the relative footprint of the corresponding image pixel and the length of the interval $[t_{i-1}, t_i]$ along the ray $\mathbf{r}$ containing the point, resulting in aliasing artifacts when rendering novel camera trajectories. Mip-NeRF [9] remedies this issue by using the projected pixel footprint to sample conical frustums along the ray rather than intervals. To feed these frustums into the MLP, mip-NeRF approximates each of them as Gaussian distributions with parameters $\boldsymbol{\mu}_i, \Sigma_i$ and replaces the positional encoding γ_{PE} with its expectation over the input Gaussian

$$\gamma_{\text{IPE}}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \mathbb{E}_{\boldsymbol{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})}[\gamma_{\text{PE}}(\boldsymbol{X})] \,, \tag{5.2}$$

referred to as an *integrated* positional encoding.

5.3 Method

Training a single NeRF does not scale when trying to represent scenes as large as cities. We instead propose splitting the environment into a set of Block-NeRFs that can be independently trained in parallel and composited during inference. This independence enables the ability to expand the environment with additional Block-NeRFs or update blocks without retraining the entire environment (see Figure 5.1). We dynamically select relevant Block-NeRFs for rendering, which are then composited in a smooth manner when traversing the scene. To aid with this compositing, we optimize the appearances codes to match lighting conditions and use interpolation weights computed based on each Block-NeRF's distance to the novel view.

Block Size and Placement

The individual Block-NeRFs should be arranged to collectively ensure full coverage of the target environment. We typically place one Block-NeRF at each intersection, covering the intersection itself and any connected street 75% of the way until it converges into the next intersection (see

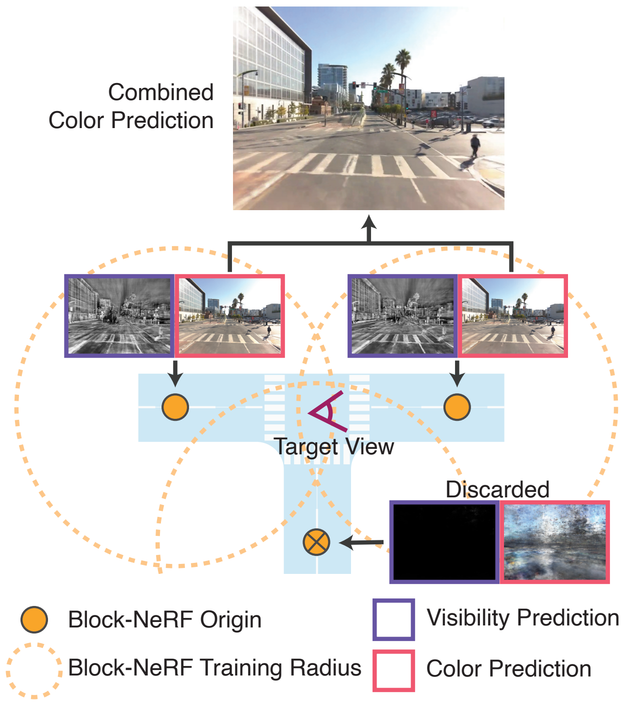


Figure 5.2: The scene is split into multiple Block-NeRFs that are each trained on data within some radius (dotted orange line) of a specific Block-NeRF origin coordinate (orange dot). To render a target view in the scene, the visibility maps are computed for all of the NeRFs within a given radius. Block-NeRFs with low visibility are discarded (bottom Block-NeRF) and the color output is rendered for the remaining blocks. The renderings are then merged based on each block origin's distance to the target view.

Figure 5.1). This results in a 50% overlap between any two adjacent blocks on the connecting street segment, making appearance alignment easier between them. Following this procedure means that the block size is variable; where necessary, additional blocks may be introduced as connectors between intersections. We ensure that the training data for each block stays exactly within its intended bounds by applying a geographical filter. This procedure can be automated and only relies on basic map data such as OpenStreetMap [58].

Note that other placement heuristics are also possible, as long as the entire environment is covered by at least one Block-NeRF. For example, for some of our experiments, we instead place blocks along a single street segment at uniform distances and define the block size as a sphere around the Block-NeRF Origin (see Figure 5.2).

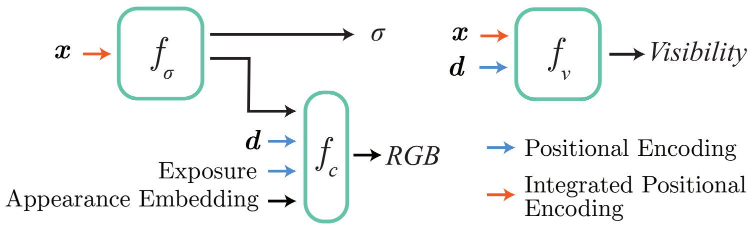


Figure 5.3: Our model is an extension of the model presented in mip-NeRF [9]. The first MLP f_σ predicts the density σ for a position $\mathbf{x}$ in space. The network also outputs a feature vector that is concatenated with viewing direction $\mathbf{d}$, the exposure level, and an appearance embedding. These are fed into a second MLP f_c that outputs the color for the point. We additionally train a visibility network f_v to predict whether a point in space was visible in the training views, which is used for culling Block-NeRFs during inference.

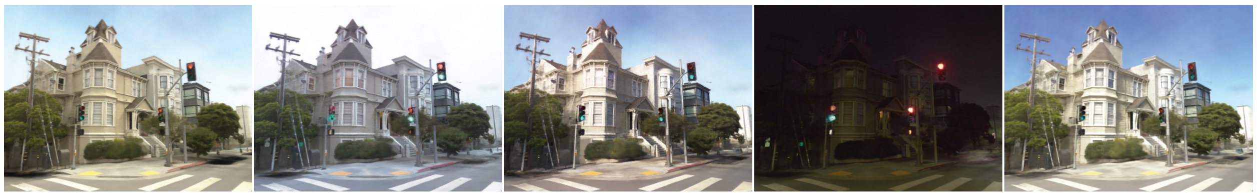


Figure 5.4: The appearance codes allow the model to represent different lighting and weather conditions.

Training Individual Block-NeRFs

Appearance Embeddings

Given that different parts of our data may be captured under different environmental conditions, we follow NeRF-W [112] and use Generative Latent Optimization [13] to optimize per-image appearance embedding vectors, as shown in Figure 5.3. This allows the NeRF to explain away several appearance-changing conditions, such as varying weather and lighting. We can additionally manipulate these appearance embeddings to interpolate between different conditions observed in the training data (such as cloudy versus clear skies, or day and night). Examples of rendering with different appearances can be seen in Figure 5.4. In § 5.3, we use test-time optimization over these embeddings to match the appearance of adjacent Block-NeRFs, which is important when combining multiple renderings.

Learned Pose Refinement

Although we assume that camera poses are provided, we find it advantageous to learn regularized pose offsets for further alignment. Pose refinement has been explored in previous NeRF based models [186, 100, 207, 219]. These offsets are learned per driving segment and include both

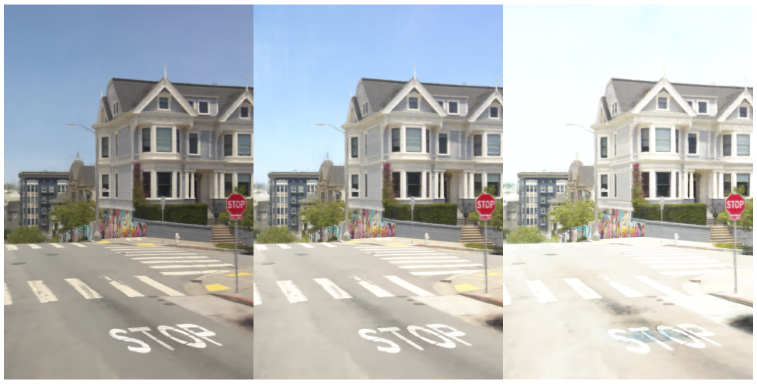


Figure 5.5: Our model is conditioned on exposure, which helps account for exposure changes present in the training data. This allows users to alter the appearance of the output images in a human-interpretable manner during inference.

a translation and a rotation component. We optimize these offsets jointly with the NeRF itself, significantly regularizing the offsets in the early phase of training to allow the network to first learn a rough structure prior to modifying the poses.

Exposure Input

Training images may be captured across a wide range of exposure levels, which can impact NeRF training if left unaccounted for. We find that feeding the camera exposure information to the appearance prediction part of the model allows the NeRF to compensate for the visual differences (see Figure 5.3). Specifically, the exposure information is processed as $\gamma_{\text{PE}}(\text{shutter speed} \times \text{analog gain}/t)$ where γ_{PE} is a sinusoidal positional encoding with 4 levels, and t is a scaling factor (we use 1000 in practice). An example of different learned exposures can be found in Figure 5.5.

Transient Objects

While our method accounts for variation in appearance using the appearance embeddings, we assume that the scene geometry is consistent across the training data. Any movable objects (*e.g.* cars, pedestrians) typically violate this assumption. We therefore use a semantic segmentation model [27] to produce masks of common movable objects, and ignore masked areas during training. While this does not account for changes in otherwise static parts of the environment, *e.g.* construction, it accommodates most common types of geometric inconsistency.

Visibility Prediction

When merging multiple Block-NeRFs, it can be useful to know whether a specific region of space was visible to a given NeRF during training. We extend our model with an additional small MLP f_v that is trained to learn an approximation of the *visibility* of a sampled point (see Figure 5.3).

For each sample along a training ray, f_v takes in the location and view direction and regresses the corresponding transmittance of the point T_i. The model is trained alongside f_σ, which provides supervision. Transmittance represents how visible a point is from a particular input camera: points in free space or on the surface of the first intersected object will have transmittance near 1, and points inside or behind the first visible object will have transmittance near 0. If a point is seen from some viewpoints but not others, the regressed transmittance value will be the average over all training cameras and lie between zero and one, indicating that the point is partially observed. Our visibility prediction is similar to the visibility fields proposed by Srinivasan *et al.* [182]. However, they used an MLP to predict visibility to environment lighting for the purpose of recovering a relightable NeRF model, while we predict visibility to training rays.

The visibility network is small and can be run independently from the color and density networks. This proves useful when merging multiple NeRFs, since it can help to determine whether a specific NeRF is likely to produce meaningful outputs for a given location, as explained in § 5.3. The visibility predictions can also be used to determine locations to perform appearance matching between two NeRFs, as detailed in § 5.3.

Merging Multiple Block-NeRFs

Block-NeRF Selection

The environment can be composed of an arbitrary number of Block-NeRFs. For efficiency, we utilize two filtering mechanisms to only render relevant blocks for the given target viewpoint. We only consider Block-NeRFs that are within a set radius of the target viewpoint. Additionally, for each of these candidates, we compute the associated visibility. If the mean visibility is below a threshold, we discard the Block-NeRF. An example of visibility filtering is provided in Figure 5.2. Visibility can be computed quickly because its network is independent of the color network, and it does not need to be rendered at the target image resolution. After filtering, there are typically one to three Block-NeRFs left to merge.

Block-NeRF Compositing

We render color images from each of the filtered Block-NeRFs and interpolate between them using inverse distance weighting between the camera origin c and the centers x_i of each Block-NeRF. Specifically, we calculate the respective weights as $w_i \propto \mathrm{distance}(c, x_i)^{-p}$, where p influences the rate of blending between Block-NeRF renders. The interpolation is done in 2D image space and produces smooth transitions between Block-NeRFs. We also explore other interpolation methods in § 5.5.

Appearance Matching

The appearance of our learned models can be controlled by an appearance latent code after the Block-NeRF has been trained. These codes are randomly initialized during training and therefore the same code typically leads to different appearances when fed into different Block-NeRFs. This

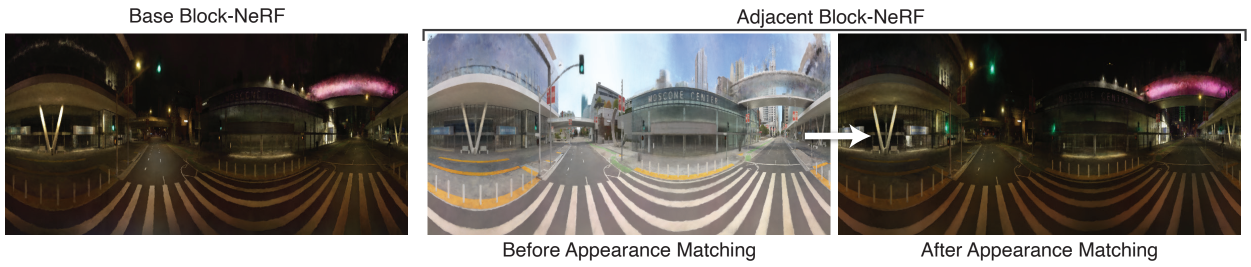


Figure 5.6: When rendering scenes based on multiple Block-NeRFs, we use appearance matching to obtain a consistent appearance across the scene. Given a fixed target appearance for one of the Block-NeRFs (left image), we optimize the appearances of the adjacent Block-NeRFs to match. In this example, appearance matching produces a consistent night appearance across Block-NeRFs.

is undesirable when compositing as it may lead to inconsistencies between views. Given a target appearance in one of the Block-NeRFs, we aim to match its appearance in the remaining blocks. To accomplish this, we first select a 3D *matching location* between pairs of adjacent Block-NeRFs. The visibility prediction at this location should be high for both Block-NeRFs.

Given the matching location, we freeze the Block-NeRF network weights and only optimize the appearance code of the target in order to reduce the ℓ_2 loss between the respective area renders. This optimization is quick, converging within 100 iterations. While not necessarily yielding perfect alignment, this procedure aligns most global and low-frequency attributes of the scene, such as time of day, color balance, and weather, which is a prerequisite for successful compositing. Figure 5.6 shows an example optimization, where appearance matching turns a daytime scene into nighttime to match the adjacent Block-NeRF.

The optimized appearance is iteratively propagated through the scene. Starting from one root Block-NeRF, we optimize the appearance of the neighboring ones and continue the process from there. If multiple blocks surrounding a target Block-NeRF have already been optimized, we consider each of them when computing the loss.

5.4 Results and Experiments

5.5 Model Parameters / Optimization Details

Our network follows the mip-NeRF structure. The network f_σ is composed of 8 layers with width 512 (Mission Bay experiments) or 1024 (all other experiments). f_c has 3 layers with width 128 and f_v has 4 layers with width 128. The appearance embeddings are 32 dimensional. We train each Block-NeRF using the Adam [86] optimizer for $300\,\mathrm{K}$ iterations with a batch size of 16384. Similar to mip-NeRF, the learning rate is an annealed logarithmically from $2 \cdot 10^{-3}$ to $2 \cdot 10^{-5}$, with a warm up phase during the first 1024 iterations. The coarse and fine networks are sampled 256 times during training and 512 times when rendering the videos. The visibility is supervised with

MSE loss and is scaled by 10^{-6}. The learned pose correction consists of a position offset and a 3×3 residual rotation matrix, which is added to the identity matrix and normalized before being applied to ensure it is orthogonal. The pose corrections are initialized to 0 and their element-wise $\ell 2$ norm is regularized during training. This regularization is scaled by 10^5 at the start of training and linearly decays to 10^{-1} after 5000 iterations. This allows the network to learn initial geometry prior to applying pose offsets.

Each Block-NeRF takes between 9 and 24 hours to train (depending on hyperparameters). We train each Block-NeRF on 32 TPU v3 cores available through Google Cloud Compute, which combined offer a total of 1680 TFLOPS and 512 GB memory. Rendering an 1200×900px image for a single Block-NeRF takes approximately 5.9 seconds. Multiple Block-NeRF can be processed in parallel during inference (typically fewer than 3 Block-NeRFs need to be rendered for a single frame).

Datasets

We perform experiments on datasets that we collect specifically for the task of novel view synthesis of large-scale scenes. Our dataset is collected on public roads using data collection vehicles. While several large-scale driving datasets already exist, they are not designed for the task of view synthesis. For example, some datasets lack sufficient camera coverage (*e.g.* , KITTI [52], Cityscapes [30]) or prioritize visual diversity over repeated observations of a target area (*e.g.* , NuScenes [20], Waymo Open Dataset [188], Argoverse [22]). Instead, they are typically designed for tasks such as object detection or tracking, where similar observations across drives can lead to generalization issues.

We capture both long-term sequence data ($100\,\mathrm{s}$ or more), as well as distinct sequences captured repeatedly in a particular target area over a period of several months. We use image data captured from 12 cameras that collectively provide a $360°$ view. 8 of the cameras provide a complete surround view from the roof of the car, with 4 additional cameras located at the vehicle front pointing forward and sideways. Each camera captures images at $10\,\mathrm{Hz}$ and stores a scalar exposure value. The vehicle pose is known and all cameras are calibrated. Using this information, we calculate the corresponding camera ray origins and directions in a common coordinate system, also accounting for the rolling shutter of the cameras. As described in § 5.3, we use a semantic segmentation model [27] to detect movable objects.

San Francisco Alamo Square Dataset. We select San Francisco's Alamo Square neighborhood as the target area for our scalability experiments. The dataset spans an area of approximately $960\,\mathrm{m} \times 570\,\mathrm{m}$, and was recorded in June, July, and August of 2021. We divide this dataset into 35 Block-NeRFs. Example renderings and Block-NeRF placements can be seen in Figure 5.1. Each Block-NeRF was trained on data from 38 to 48 different data collection runs, adding up to a total driving time of 18 to 28 minutes each. After filtering out some redundant image captures (*e.g.* stationary captures), each Block-NeRF is trained on between $64\,575$ to $108\,216$ images. The overall dataset is composed of $13.4\,\mathrm{h}$ of driving time sourced from 1330 different data collection runs, with a total of $2\,818\,745$ training images.

San Francisco Mission Bay Dataset. We choose San Francisco's Mission Bay District as the target area for our baseline, block size, and placement experiments. Mission Bay is an urban environment with challenging geometry and reflective facades. We identified a long stretch on Third Street with far-range visibility, making it an interesting test case. Notably, this dataset was recorded in a single capture in November 2020, with consistent environmental conditions allowing for simple evaluation. This dataset was recorded over $100\,\mathrm{s}$, in which the data collection vehicle traveled $1.08\,\mathrm{km}$ and captured $12\,000$total images from 12cameras. We will release this single-capture dataset to aid reproducibility.

Model Ablations

	NeRFs	PSNR↑	SSIM↑	LPIPS↓
	mip-NeRF	17.86	0.563	0.509
Ours	-Appearance	20.13	0.611	0.458
	-Exposure	23.55	**0.649**	0.418
	-Pose Opt.	23.05	0.625	0.442
	Full	**23.60**	**0.649**	**0.417**

Table 5.1: Ablations of different Block-NeRF components on a single intersection in the Alamo Square dataset. We show the performance of mip-NeRF as a baseline, as well as the effect of removing individual components from our method.

We ablate our model modifications on a single intersection from the Alamo Square dataset. We report PSNR, SSIM, and LPIPS [228] metrics for the test image reconstructions in Table 5.1. The test images are split in half vertically, with the appearance embeddings being optimized on one half and tested on the other. We also provide qualitative examples in Figure 5.7. Mip-NeRF alone fails to properly reconstruct the scene and is prone to adding non-existent geometry and cloudy artifacts to explain the differences in appearance. When our method is not trained with appearance embeddings, these artifacts are still present. If our method is not trained with pose optimization, the resulting scene is blurrier and can contain duplicated objects due to pose misalignment. Finally, the exposure input marginally improves the reconstruction, but more importantly provides us with the ability to change the exposure during inference.

Block-NeRF Size and Placement

We compare performance on our Mission Bay dataset versus the number of Block-NeRFs used. We show details in Table 5.2, where depending on granularity, the Block-NeRF sizes range from as small as $54\,\mathrm{m}$ to as large as $544\,\mathrm{m}$. We ensure that each pair of adjacent blocks overlaps by 50% and compare other overlap percentages. All were evaluated on the same set of held-out test images spanning the entire trajectory. We consider two regimes, one where each Block-NeRF contains

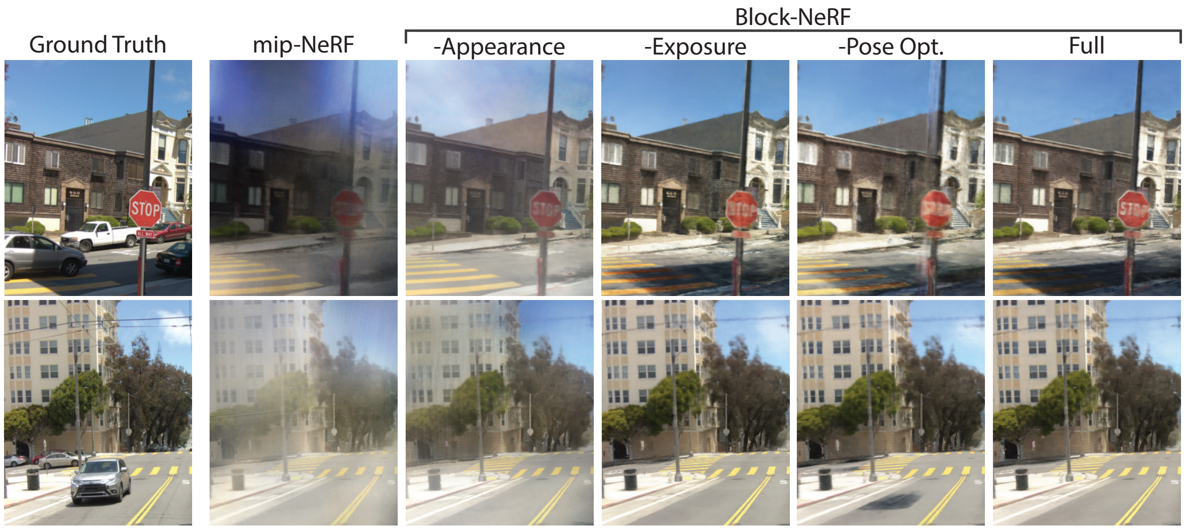


Figure 5.7: Model ablation results on multi segment data. Appearance embeddings help the network avoid adding cloudy geometry to explain away changes in the environment like weather and lighting. Removing exposure slightly decreases the accuracy. The pose optimization helps sharpen the results and removes ghosting from repeated objects, as observed with the telephone pole in the first row.

# Blocks	Weights / Total	Size	Compute	PSNR↑	SSIM↑	LPIPS↓
1	0.25M / 0.25M	544 m	1×	23.83	0.825	0.381
4	0.25M / 1.00M	271 m	2×	25.55	0.868	0.318
8	0.25M / 2.00M	116 m	2×	26.59	0.890	0.278
16	0.25M / 4.00M	54 m	2×	**27.40**	**0.907**	**0.242**
1	1.00M / 1.00M	544 m	1×	24.90	0.852	0.340
4	0.25M / 1.00M	271 m	0.5×	25.55	0.868	0.318
8	0.13M / 1.00M	116 m	0.25×	25.92	0.875	0.306
16	0.07M / 1.00M	54 m	0.125×	**25.98**	**0.877**	**0.305**

Table 5.2: Comparison of different numbers of Block-NeRFs for reconstructing the Mission Bay dataset. Splitting the scene into multiple Block-NeRFs improves the reconstruction accuracy, even when holding the total number of weights constant (bottom section). The number of blocks determines the size of the area each block is trained on and the relative compute expense at inference time.

the same number of weights (top section) and one where the total number of weights across all Block-NeRFs is fixed (bottom section). In both cases, we observe that increasing the number of models improves the reconstruction metrics. In terms of computational expense, parallelization during training is trivial as each model can be optimized independently across devices. At inference, our method only requires rendering Block-NeRFs near the target view. Depending on the scene and NeRF layout, we typically render between one to three NeRFs. We report the relative compute expense in each setting without assuming any parallelization, which however would be possible and lead to an additional speed-up. Our results imply that splitting the scene into multiple lower capacity models can reduce the overall computational cost as not all of the models need to be evaluated (see bottom section of Table 5.2).

Block-NeRF Overlap Comparison

In the main paper, we include experiments on Block-NeRF size and placement (§5.3). For these experiments, we assumed a relative overlap of 50% between each pair of Block-NeRFs, which aids with appearance alignment.

Table 5.3 is a direct extension of Table 5.2 and shows the effect of varying block overlap in the 8 block scenario. Note that varying the overlap changes the spatial block size.

The metrics imply that reducing overlap is beneficial for image quality metrics. However, this can likely be attributed to the resulting reduction in block size. In practice, having an overlap between blocks is important to avoid temporal artifacts when interpolating between Block-NeRFs.

Overlap	Size	PSNR↑	SSIM↑	LPIPS↓
0%	77 m	26.77	0.895	0.262
25%	97 m	26.75	0.894	0.269
50%*	116 m	26.59	0.890	0.278
75%	136 m	26.51	0.887	0.283

Table 5.3: Effect of different NeRF overlaps in the 8 block scenario with 0.25M weights per block (2M weights in total). The default setting used for other experiments is marked *.

Interpolation Methods

We explore different interpolation methods in Table 5.4. We experiment with multiple methods to interpolate between Block-NeRFs and find that simple inverse distance weighting (IDW) in image space produces the most appealing videos due to temporal smoothness. We use an IDW power p of 4 for the Alamo Square renderings and a power of 1 for the Mission Bay renderings. We experiment with 3D inverse distance weighting for each individual pixel by projecting the rendered pixels into 3D space using the expected ray termination depth from the Block-NeRF closest to the target view. The color value of the projected pixel is then determined using inverse distance

Interpolation	Consistent?	PSNR↑	SSIM↑	LPIPS↓
Nearest	–	26.40	0.887	0.280
IDW 2D	✓	26.59	0.890	0.278
IDW 3D	–	26.57	0.890	0.278
Pixelwise Visibility	–	27.39	0.906	0.242
Imagewise Visibility	–	27.41	0.907	0.242

Table 5.4: Comparison of interpolation methods. For our flythrough video results, we opt for 2D inverse distance weighting (IDW) as it produces temporally consistent results.

weighting with the nearest Block-NeRFs. Artifacts occur in the resulting composited renders due to noise in the depth predictions. We also experiment with using the Block-NeRF predicted visibility for interpolation. We consider imagewise visibility where we take the mean visibility of the entire image and pixelwise visibility where were directly utilize the per-pixel visibility predictions. Both of these methods lead to sharper results but come at the cost of temporal inconsistencies. Finally we compare to nearest neighbor interpolation where we only render the Block-NeRF closest to the target view. This results in harsh jumps when transiting between Block-NeRFs.

5.6 Limitations and Future Work

The proposed method handles transient objects by filtering them out during training via masking using a segmentation algorithm. If objects are not properly masked, they can cause artifacts in the resulting renderings. For example, the shadows of cars often remain, even when the car itself is correctly removed. Vegetation also breaks this assumption as foliage changes seasonally and moves in the wind; this results in blurred representations of trees and plants. Similarly, temporal inconsistencies in the training data, such as construction work, are not automatically handled and require the manual retraining of the affected blocks. Further, the inability to render scenes containing dynamic objects currently limits the applicability of Block-NeRF towards closed-loop simulation tasks in robotics. In the future, these issues could be addressed by learning transient objects during the optimization [112], or directly modeling dynamic objects [212, 136]. In particular, the scene could be composed of multiple Block-NeRFs of the environment and individual controllable object NeRFs. Separation can be facilitated by the use of segmentation masks or bounding boxes.

In our model, distant objects in the scene are not sampled with the same density as nearby objects which leads to blurrier reconstructions. This is an issue with sampling unbounded volumetric representations. Techniques proposed in NeRF++ [226] and concurrent Mip-NeRF 360 [8] could potentially be used to produce sharper renderings of distant objects.

In many applications, real-time rendering is key, but NeRFs are computationally expensive to render (up to multiple seconds per image). Several NeRF caching techniques [50, 221, 62] or a sparse voxel grid [102] could be used to enable real-time Block-NeRF rendering. Similarly, multiple concurrent works have demonstrated techniques to speed up training of NeRF style representations by multiple orders of magnitude [126, 187, 45].

5.7 Discussion

In this chapter we proposed Block-NeRF, a method that reconstructs arbitrarily large environments using NeRFs. We demonstrate the method's efficacy by building an entire neighborhood in San Francisco from 2.8M images, forming the largest neural scene representation at the time. We accomplish this scale by splitting our representation into multiple blocks that can be optimized independently. At such a scale, the data collected will necessarily have transient objects and variations in appearance, which we account for by modifying the underlying NeRF architecture. A substantial portion of the project involved re-implementing and combining several advancements from the NeRF literature. This process served as a strong motivation to develop a framework, Nerfstudio [195], that streamlines NeRF development. Further details on Nerfstudio are discussed in Chapter 6.

Chapter 6

Nerfstudio Framework

In the previous chapters we have discussed how NeRFs can be used to create photorealistic 3D scene representations. Ever since the initial NeRF publication, there has been rapid research pushing the field forward.There has been an influx of papers focusing on advancements to the core method including few-image training [220, 206], explicit features for editing [103, 204, 225], surface representations for high-quality 3D mesh exports [134, 217, 205], speed improvements for real-time rendering and training [45, 187, 125], 3D object generation [143], and more [210]. In chapter 5 we combined multiple of these advancements to create the Block-NeRF method.

These research innovations have driven interests in a wide variety of disciplines in both academia and industry. Roboticists have explored using NeRFs for manipulation, motion planning, simulation, and mapping [83, 2, 37, 19, 234, 176]. NeRFs are also explored for tomography applications [168], as well as perceiving people in videos [139]. Visual effects and gaming studios are exploring the technology for production and digital asset creation. News outlets capture NeRF portraits to tell stories in new formats [208]. The potential applications are vast, and even startups [1] are emerging to focus on deploying this technology.

Despite the growing use of NeRFs, support for development is still rudimentary. Due to the influx of papers and lack of code consolidation, tracking progress is difficult. Many papers implement features in their own siloed repository. This complicates the process of transferring features and research contributions across different implementations. Additionally, few tools exist to easily run NeRFs on real-world data collected by users. To address these challenges, we present Nerfstudio (Fig. 6.1), a modular framework that consolidates NeRF research innovations and makes them easier to use in real-world applications.

Furthermore, while NeRFs solve an inherently visual task, there is a lack of comprehensive and extensible tools for visualizing and interacting with NeRFs trained on real-world data. Despite the availability of several NeRF repositories, existing implementations are often focused on achieving state-of-the-art results on metrics such as PSNR, SSIM, and LPIPS. These evaluations are typically based on held-out images along the capture trajectory that are similar to the training images. This often makes them misleading indicators of performance for many real-world applications when

This chapter is based on joint work published at Siggraph 2023 [195]

[1]https://lumalabs.ai/

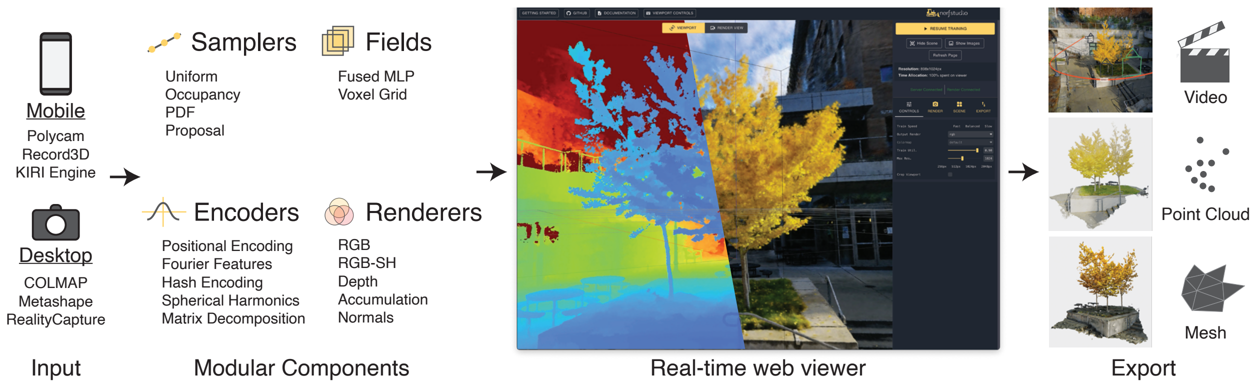


Figure 6.1: **Nerfstudio framework.** Nerfstudio is a Python framework for Neural Radiance Field (NeRF) development. Nerfstudio supports multiple input data pipelines, is built around multiple modular core NeRF components, integrates with a real-time web viewer, and supports multiple export modalities. The goal of the Nerfstudio framework is to simplify the development of custom NeRF methods, processing of real-world data, and interacting with reconstructions.

data is captured in unstructured environments and novel views are rendered with large baselines. Qualitative evaluations have historically been a challenge due to the computational demands of NeRF, which often resulted in rendering times up to multiple seconds per image. Recent developments such as Instant-NGP [125] significantly reduce computational overhead, enabling real-time training and rendering. However, Instant-NGP relies significantly on GPU accelerations with custom CUDA kernels, making development and quick prototyping a challenge. We present a framework that enables interactive visualizations while also being flexible and model-agnostic.

Nerfstudio is an extensible and versatile framework for neural radiance field development. Our design goals are the following:

1. Consolidating various NeRF techniques into reusable, modular components.
2. Enabling real-time visualization of NeRF scenes with a rich suite of controls.
3. Providing an end-to-end, easy-to-use workflow for creating NeRFs from user-captured data.

For modularity, we devise an organization among components across various NeRFs that allows abstracting away method-specific implementations. Our real-time visualizer is designed to work with any model during training or testing. Furthermore, the visualizer is hosted on the web, making it accessible without requiring a local GPU machine. The modular nature of our framework facilitates the integration of various data input formats, thereby simplifying the workflow for incorporating user-captured real-world scenes. We provide support for images and videos with various camera types, as well as other mobile capture applications (Polycam, Record3D, KIRI Engine) and outputs from popular photogrammetry software like RealityCapture and Metashape. In particular, integration with these applications enable users to by-pass structure-from-motion tools

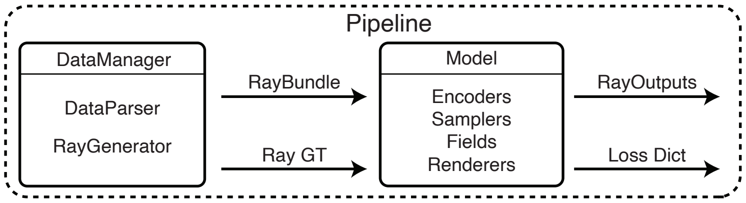


Figure 6.2: **Pipeline components.** Each NeRF method is implemented as a custom Pipeline. DataManagers process input images into bundles of rays (RayBundles) that get rendered by the Model to produce a set of NeRF outputs (RayOutputs). A dictionary of losses supervises the pipeline end-to-end.

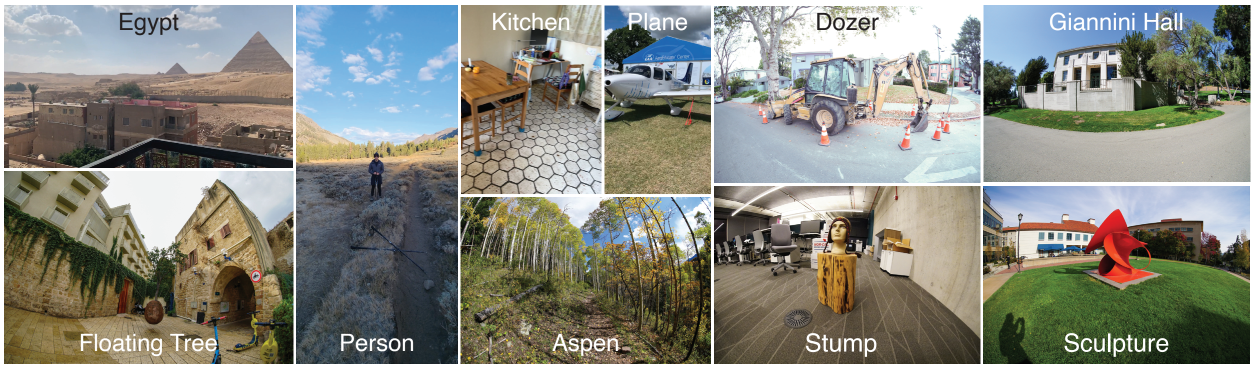


Figure 6.3: **Nerfstudio Dataset.** Our Nerfstudio Dataset contains 10 scenes: 4 phone captures with pinhole lenses and 6 Mirrorless camera captures with a fisheye lens. We focus our efforts on real-world data, and these scenes can help benchmark progress.

like COLMAP [170], which can be time-consuming. Furthermore, we provide support for multiple export formats, including video, depth maps, point clouds, and meshes.

The modularity of Nerfstudio enables developing Nerfacto, our method that combines components from recent papers to achieve a balance between speed and quality. We show that this method is comparable to the other state-of-the-art methods such as MipNeRF-360 [8] while achieving an order of magnitude speedup. We also conduct an ablation study that demonstrates its flexibility on a new in-the-wild dataset consisting of 10 in-the-wild scenes. Our findings highlight the limitations of commonly used NeRF metrics and the importance of a real-time viewer for qualitative assessments. The potential of our framework as a consolidated codebase for NeRF research is reflected in the traction thus far with extensions such as SDFStudio [222]. Furthermore, Nerfstudio is an open-source project with active improvements from both academic and industry contributors.

6.1 Related Works

Frameworks and tools

Software frameworks have played a crucial role in consolidating and driving the advancement of various fields. In deep learning, Caffe [75], TensorFlow [1], and PyTorch [138] provide readily usable machine learning functionalities. Similarly, frameworks such as PyTorch3D [155] and Kornia [160] provide reusable components for 3D computer vision tasks. Other examples of frameworks include Mitsuba3 [72], Halide [147], Taichi [70], and Reyes [29] for graphics, Phototourism [181] and COLMAP [170, 172, 171] for photogrammetry and visualization, and AverageExplorer [232] for data collection. Despite the diversity of topics covered, each of these frameworks originated from the need to provide reusability and reproducibility to a rapidly expanding field. In light of the fast-paced growth of NeRFs in both academia and industry, Nerfstudio aims to streamline advancements in neural rendering by offering a flexible and comprehensive framework for development.

NeRF codebases

In recent years, several codebases for NeRFs have gained popularity among the research community, including the original NeRF codebase[123], nerf-pytorch [218, 127], Nerf_pl [146], Instant NGP [125], torch-ngp, Ngp_pl, and MultiNeRF [121]. Due to the lack of consolidation, there exists a significant number of NeRF repositories that focus on improving specific components of specific algorithms. For example, Mip-NeRF [9] aims to address the anti-aliasing problem of NeRF [123] and Mip-NeRF 360 [8] addresses the limitations of Mip-NeRF. Additionally, Plenoxels [45], TensoRF [23] and InstantNGP [125] propose different approaches to address the problem of computational efficiency. Furthermore, RawNeRF [122], Ref-NeRF [200], and NeRF-W [112] each address distinct challenges related to NeRF, resulting in parallel, non-interacting implementations. Nerfstudio aims to address the lack of consolidated development in the field of NeRFs by consolidating critical techniques introduced in the existing literature. This allows for more efficient and effective experimentation with combining components from multiple solutions into a single, comprehensive method, and facilitates the ability of the community to build upon existing prior approaches.

Neural rendering frameworks

Concurrent efforts such as NeRF-Factory [73], NerfAcc [94], MultiNeRF [121], and Kaolin-Wisp [190] all make significant efforts in advancing the usability of NeRFs. While NeRF-Factory consolidates multiple prior works into a single repository, it places less emphasis on reusable modules shared across these prior works and focuses more on benchmarking. NerfAcc prioritizes pythonic modularity, but focuses primarily on the lower-level components rather than the entire pipeline. Kaolin-Wisp and Multi-NeRF each consolidate multiple paper implementations into a single repository. None of these repositories are as comprehensive as Nerfstudio in delivering our three design goals: modularity, real-time visualization, and end-to-end usability for user-captured data. Furthermore, Nerfstudio is released under an Apache2 license, which allows for its use by both researchers and companies.

6.2 Framework Design

The goals of Nerfstudio are to provide (1) modularity, (2) real-time visualization for development, and (3) ease of use with real data. In designing the framework, we consider trade-offs against designs that optimize for faster rendering or higher quality results on synthetic scenes. For instance, we prefer an implementation that allows for a modularized pythonic non-CUDA method over one that supports a faster, non-modularized CUDA method. Additionally, our design choices lead to simpler interfacing with an extensive visualization ecosystem which supports real-time rendering during test and train with custom camera paths. Finally, we focus on delivering results for real-world data rather than synthetic scenes to address audiences outside research including those in industry and non-technical users.

With these three goals, the design of Nerfstudio promotes collaborations by providing a consolidated platform on which people can request for or contribute to new features. The long-term goal is for Nerfstudio to continue improving through community-driven contributions.

Modularity

We propose an organization of components that is both intuitive and abstract, enabling the implementation of existing and novel NeRFs by swapping reusable components. Fig. 6.1 shows a subset of the components types and implementations we currently have available in Nerfstudio.

Visualization for development

The Nerfstudio real-time viewer offers an interactive and intuitive way to visualize Neural Radiance Fields (NeRFs) during both training and testing phases. To ensure ease of use, the visualizer is simple to install, works seamlessly across both local and remote GPU compute environments, supports different models, and offers a user interface for creating and rendering custom camera paths, shown in Fig. 6.5 (a).

Our real-time visualization interface is particularly useful for qualitatively evaluating a model, allowing for more informed decisions during method development. While metrics such as PSNR can provide some insight, they do not offer a comprehensive understanding of performance–especially for views that are far away from the capture trajectory. Qualitative evaluation with an interactive viewer addresses these limitations and allows developers to gain a more holistic understanding of the model performance.

Easy workflow for user-captured data

While we offer support for synthetic datasets (Blender [123], D-NeRF [145]), in Nerfstudio we focus primarily on "real world data" — images or videos from a physical phone or camera. To this end, we present a new Nerfstudio Dataset (shown in Fig. 6.3) composed of real-world scenes casually captured with mobile phones and a mirrorless camera. Our motivation is to provide a framework compatible with a diverse array of applications which requires supporting real data. For instance, a few use cases for Nerfstudio outside of research include VFX, gaming, and non-technical film-makers who create 3D and video art. To support this wide range of expertise in NeRFs, we ensure our codebase is easily installable and deployable.

6.3 Core components

The proposed framework of Nerfstudio, illustrated in Fig. 6.2, is based on the conceptual grouping of NeRF methods into a series of basic building blocks. Nerfstudio takes a set of posed images and optimizes for a 3D representation of the scene, which is defined by radiance (color), density (structure), and possibly other quantities (semantics, normals, features, etc.). We ingest these inputs into the framework which comprises of a DataManager and a Model, where the DataManager is responsible for (1) parsing image formats via a DataParser and (2) generating rays as RayBundles. These rays are then passed into a Model, which will query Fields and render quantities. Finally, the whole Pipeline is supervised end-to-end with a loss.

DataManagers and DataParsers

The first step of the Pipeline is the DataManager which is responsible for turning posed images into RayBundles, which are slices of 3D space that start at a camera origin. Within the DataManager, the DataParser first loads the input images and camera data. The DataParser is designed to be compatible with arbitrary data formats such as COLMAP. Previous research codebases primarily utilize COLMAP with helper scripts [125], however, COLMAP can be challenging to install and use for non-technical users. To make the framework more accessible to a wider range of users, including scientists, artists, photographers, hobbyists and journalists, we have implemented DataParsers for mobile apps (Record3D, Polycam, KIRI Engine) and 3D tools such as Metashape and Reality Capture. Once the images are properly loaded and formatted, the DataManager iterates through the data, generating RayBundles and ground truth supervision. It can also optimize camera poses during training.

RayBundles, RaySamples, and Frustums

NeRFs operate on regions of 3D space, which can be parametrized in many different ways. We have adopted a more generic representation of 3D space through the use of Frustum for both point-based and volume-based samples. The RayBundles, which are primitives that represent a slice through 3D space, are parameterized with an origin, direction, and other meta-information such as camera indices and time. By specifying the interval bin spacing, the RayBundles generate RaySamples, which represent sampled chunks of 3D space along each ray. These chunks, represented as Frustums, can be encoded either as point samples [123] or as Gaussians with mean and covariance [9], which have been shown to help with anti-aliasing. This abstraction allows for flexibility in representation, as the user can decide which representation to use with a simple function call. A visualization of this abstraction can be found in Fig. 6.4.

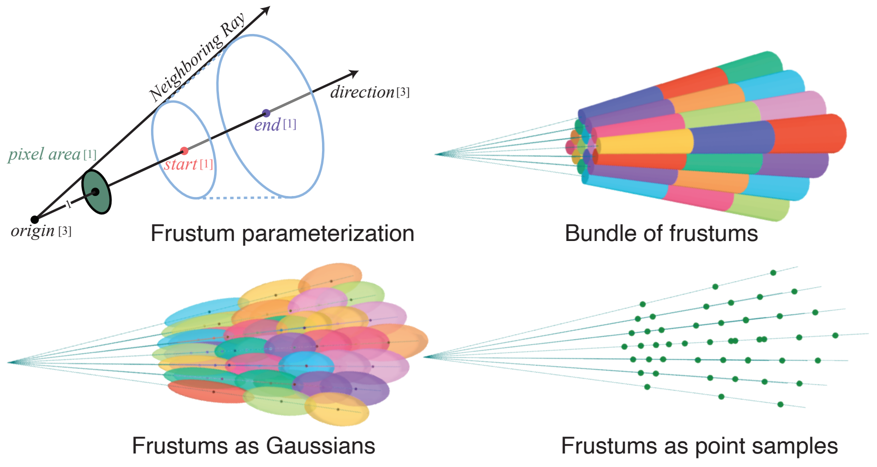


Figure 6.4: **Sample representations.** (Top) We define a frustum as a cone with a start and end. This region of space can be converted into Gaussians (bottom left) or point samples (bottom right) depending on the field input format.

Models and Fields

The RayBundles are sent to Models as input, which samples them into RaySamples. The RaySamples are consumed by Fields to turn regions of space (i.e., Frustums) into quantities such as color or density. The Nerfstudio framework contains various implementations of models and fields. We've implemented various feature encoding schemes including fourier features, hash encodings [125], spherical harmonics, and matrix decompositions [23]. Field components include fused MLPs, voxel grids, and surface normal MLPs [200], activation functions, spatial distortions [8], and temporal distortions [145].

Real-time web viewer

We draw inspiration from the real-time viewer presented in Instant NGP [125], which facilitates real-time rendering during training. However, the viewer in Instant NGP is designed to work on local compute, which can be cumbersome to setup in remote settings. To address this issue, we have developed a ReactJS-based web viewer packaged as a publicly hosted website at https://viewer.nerf.studio.

The viewer is designed to be accessible to a wide range of users, including those utilizing both local and remote GPUs. The process of utilizing remote compute is streamlined, requiring only the forwarding of a port locally via SSH. Once training begins, the web interface renders the NeRF in real-time as training progresses (See Fig. 6.9). Users can pan, zoom and rotate around the scene as the optimization runs or while evaluating a trained model.

Implementation

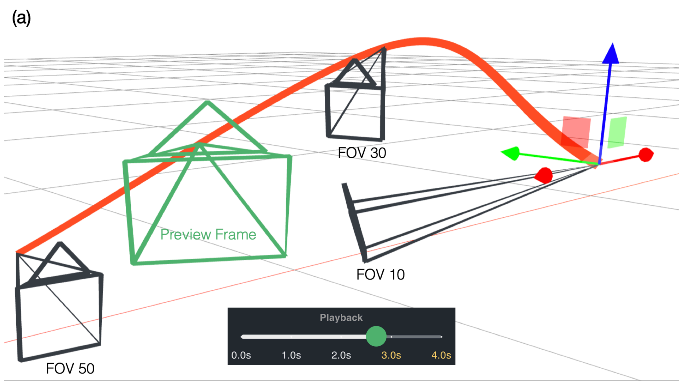
Real-time training visualization utilizes WebSockets to establish a connection between the NeRF training session and the web client. This approach eliminates the need to install local screens and other GUI software. Upon opening the web viewer, a WebSocket connection is established with the training session, which subsequently populates the scene with training images. The web viewer continuously streams the viewport camera pose to the training session during the training process. The training session utilizes this camera pose to render images and transmits them via the websocket. Additionally, the viewer camera controls and UI are implemented using ThreeJS, allowing us to overlay 3D assets such as images, splines, and cropping boxes in front of the NeRF renderings. For instance, the viewer displays training images at their capture locations, letting users intuitively compare performance at seen and novel viewpoints.

Viewer features

Our viewer is compatible with different models of varying rendering speeds. We accomplish this by balancing the computation of training and viewer rendering on a single GPU. Similar to Instant-NGP [125], we adjust the rendering resolution based on the speed of the camera movement. When the camera moves quickly, the rendering resolution will be smaller to maintain a frame rate and prevent lag in the user experience. We can also reduce the time spent on training and allocate more resources for rendering in the viewer. Some of the features of our viewer include:

- Switching between various model outputs (e.g., rgb, depth, normals, semantics).
- Creating custom camera paths composed of keyframes with position and focal length interpolation (Fig. 6.5).
- Visualizing the captured training images in 3D.
- Crop and export options for point clouds and meshes.
- Mouse and keyboard controls to easily navigate in the scene.

The viewer played an instrumental role in providing qualitative assessments that informed design choices in our default method Nerfacto. Other codebases have integrated our viewer into their own codebases, including ArcNerf [223] and SDFStudio [222].



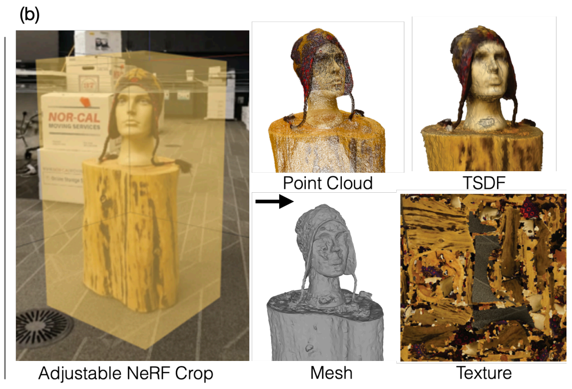


Figure 6.5: **Exporting videos and geometry.** We make exporting videos (a) and geometry (b) easy with real-data captures. The left side shows the interactive camera trajectory editor, which allows animatable poses, FOVs, and speed, to eventually render videos of NeRF's outputs. On the right we show the cropping interface in the viewer and resulting export formats including point clouds, TSDFs, and textured meshes.

Geometry export

Many creators and artists have workflows that require exporting to point clouds or meshes for further processing and incorporation in downstream tools such as game engines. Hence, our framework accommodates various export methods and facilitates the easy addition of new export methods. Fig. 6.5b illustrates our export interface, as well as some of the supported formats, including point clouds, a truncated signed distance function (TSDF) to mesh, and Poisson surface reconstruction [82]. We apply texture to the mesh by densely sampling the texture image, utilizing barycentric interpolation to determine corresponding 3D point locations, and rendering short rays near the surface along the normals to obtain RGB values.

6.4 Nerfacto Method

We leverage our modular design to integrate ideas from multiple research papers into our default and recommended method, Nerfacto. This method is heavily influenced by the structure of MipNeRF-360 [8], but certain parts of the original design are replaced to improve performance. We reference papers such as NeRF-- [207], Instant-NGP [125], NeRF-W [112], and Ref-NeRF [200] in Nerfacto. Fig. 6.6 illustrates how these papers are used.

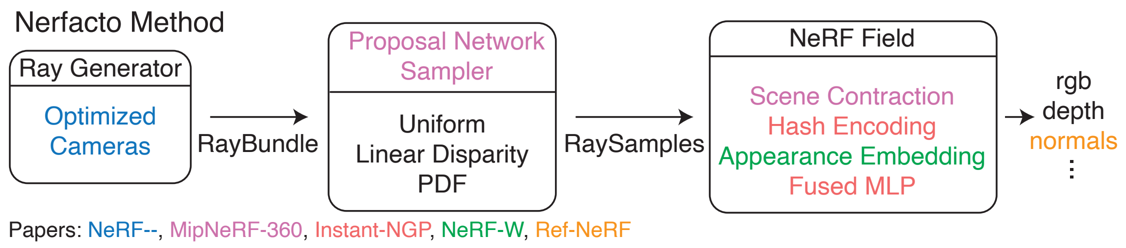


Figure 6.6: **Nerfacto method.** Diagram of the Nerfacto method. It combines features from many papers (bottom left). The method will evolve over time as new papers and features are added to the Nerfstudio codebase.

Ray generation and sampling

The Nerfacto method first optimizes camera views using an optimized SE(3) transformation [207, 100, 192]. These camera views are then used to generate RayBundles. To improve the efficiency and effectiveness of the sampling process, we employ a piece-wise sampler. This sampler samples uniformly up to a fixed distance from the camera, followed by samples that are distributed such that the step size increases with each sample. This allows efficient sampling of distant objects while still maintaining a dense set of samples for nearby objects. These samples are then fed into a proposal network sampler, proposed in the MipNeRF-360 method [8]. The proposal sampler consolidates the sample locations into regions of the scene that contribute most to the final render, typically the first surface intersection. This importance sampling greatly improves reconstruction quality. Furthermore, we use a small fused MLP with a hash encoding [125] for the scene's density function as it has been found to have sufficient accuracy and is computationally efficient. To further reduce the number of samples along rays, the proposal network sampler can contain multiple density fields. These density fields iteratively reduce the number of samples. Empirically, using two density fields works well. In our base Nerfacto configuration, we generate 256 samples from the piece-wise sampler, which gets resampled into 96 samples in the first iteration of the proposal sampler followed by 48 samples in the second.

Scene contraction and NeRF field

Many real-world scenes are unbounded, meaning they could extend indefinitely. This poses a challenge for processing as input samples could have position values that vary across many scales of magnitude. To overcome this issue, we utilize *scene contraction*, which compresses the infinite space into a fixed-size bounding box. Our method of contraction is based on the one proposed in MipNeRF-360 [8], but we use L^∞ norm contraction instead of L^2 norm, which contracts to a cube rather than a sphere. The cube better aligns with voxel-based hash encodings. Fig. 6.7 illustrates how L^∞ contraction maps samples into the range with minimum values of -2,-2,-2 and maximum values of 2,2,2. These samples can then be used with the hash encoding introduced by Instant-NGP and is available via the tiny-cuda-nn [124] Python bindings.

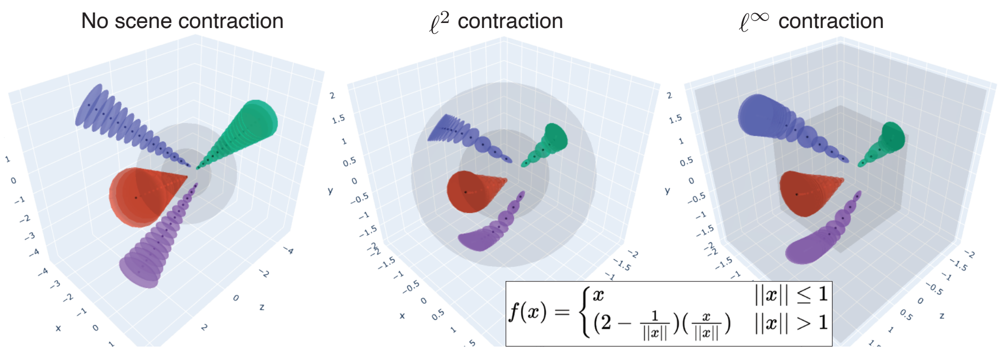

No scene contraction
ℓ^2 contraction
ℓ^∞ contraction
$f(x) = \begin{cases} x & ||x|| \leq 1 \\ (2 - \frac{1}{||x||})(\frac{x}{||x||}) & ||x|| > 1 \end{cases}$

Figure 6.7: **Scene contraction.** Here we show cameras contained in an inner sphere with Gaussian samples along rays. Scene contraction warps the unbounded samples into bounded space before querying a NeRF field. We use L^∞ contraction rather than MipNeRF-360's L^2 contraction to better accommodate the geometry/capacity of the hash grid.

Nerfacto's field incorporates per-image appearance embeddings to account for differences in exposure among training cameras [112]. Additionally, we use techniques from Ref-NeRF [200] to compute and predict normals. Nerfacto is implemented using PyTorch, which allows for easy customization and eliminates the need for complex and custom CUDA code. We will incorporate new papers into Nerfacto as the field progresses.

6.5 Nerfstudio Dataset

Our "Nerfstudio Dataset" includes 10 in-the-wild captures obtained using either a mobile phone or a mirror-less camera with a fisheye lens. We processed the data using either COLMAP or the Polycam app to obtain camera poses and intrinsic parameters. Our goal is to provide researchers with more 360 real-world captures that are not limited to forward-facing scenes [120]. Our dataset is similar to MipNeRF-360 [8] but does not focus on a central object and includes captures with varying degrees quality. We have used this dataset to select the default settings for our proposed NeRF-based method, Nerfacto, and we encourage other researchers to similarly employ real-world data in the development and evaluation of NeRF methods.

6.6 Experiments

We benchmark Nerfacto against a state-of-the-art method MipNeRF-360 and emphasize the modularity of our repository by conducting ablation studies. Furthermore, we highlight the limitations of commonly used evaluation metrics such as PSNR, SSIM, and LPIPS when applied to subsampled evaluation images.

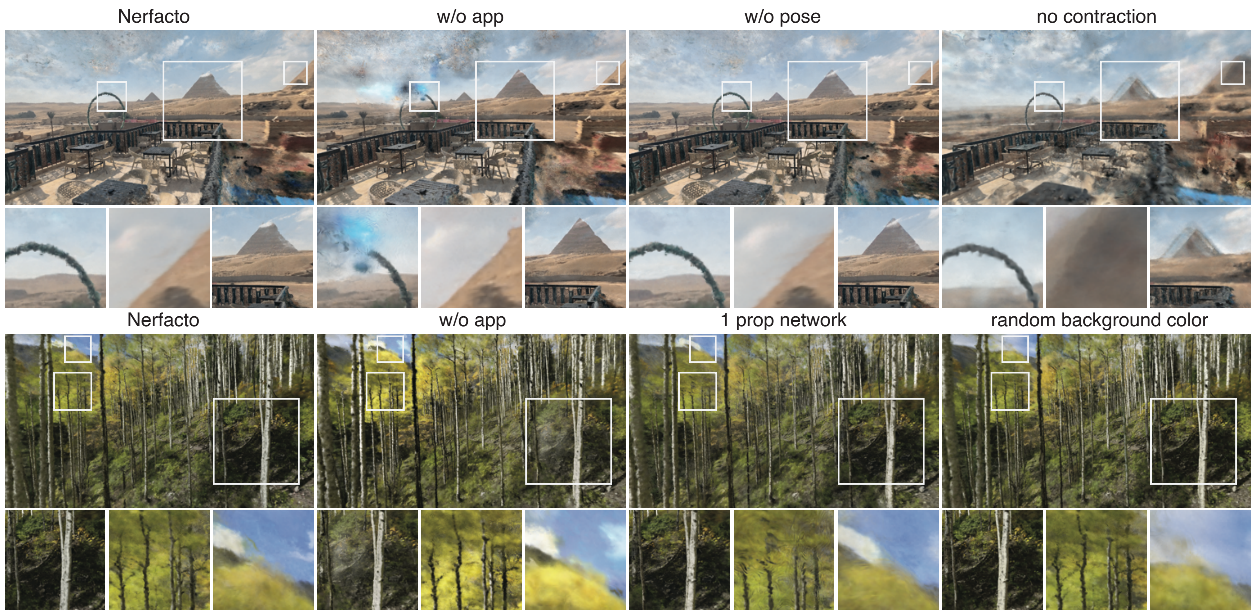


Figure 6.8: **Nerfstudio ablation qualitative examples.** Here we show renderings from different Nerfacto ablation variants. (Top) is the "Egypt" capture and (bottom) is the "aspen" capture from the Nerfstudio Dataset. These novel views are far from the training images to get a sense of how well these methods perform qualitatively. We zoom in on crops to highlight differences in the rendered images.

Mip-NeRF 360 dataset comparison

Here we compare Nerfacto with numbers reported in the MipNeRF-360 [8] paper. We evaluate on their 7 publicly available scenes. We train our method for up to 30K iterations (30 minutes) on an NVIDIA RTX A5000, but we also report results at 5K iterations (5 minutes).

Evaluation protocol. The evaluation protocol followed is similar to that of MipNeRF360, but we process their data using our COLMAP pipeline to recover poses. The original images were downsampled by a factor of 4x. We used 7/8 of the images for training and the remaining 1/8 images were evenly spaced and used for evaluation. Note that this protocol does not include camera pose optimization as it is not an option implemented in MipNeRF360.

Findings. Table 6.1 presents the averages of the results across the 7 captures in the MipNeRF-360 dataset. The complete table can be found in the appendix. In as little as 5K iterations ($\sim$5 minutes), our Nerfacto method achieves reasonable quality in contrast to MipNeRF-360 which takes several hours on a TPU with 32 cores. Training for up to 30K ($\sim$30 minutes) iterations further improves quality. While Nerfacto falls short of metric results obtained by MipNeRF-360, we prioritize efficiency and general usability over optimizing quantitative metrics on this particular benchmark. Fig. 6.9 shows qualitative results on the "garden" scene in our viewer after only a few minutes.

Method	**PSNR** ↑	**SSIM** ↑	**LPIPS** ↓
MipNeRF-360	29.23	0.844	0.207
Nerfacto (ours)	26.75 / 25.38	0.748 / 0.688	0.307 / 0.390

Table 6.1: **Average metrics on the MipNeRF360 dataset.** Our methods are evaluated without pose optimization or per-image appearance embeddings. MipNeRF-360 takes several hours to train. Our metrics reported as { after 30K iterations (∼30min) / after 5k iterations (∼5min) }.

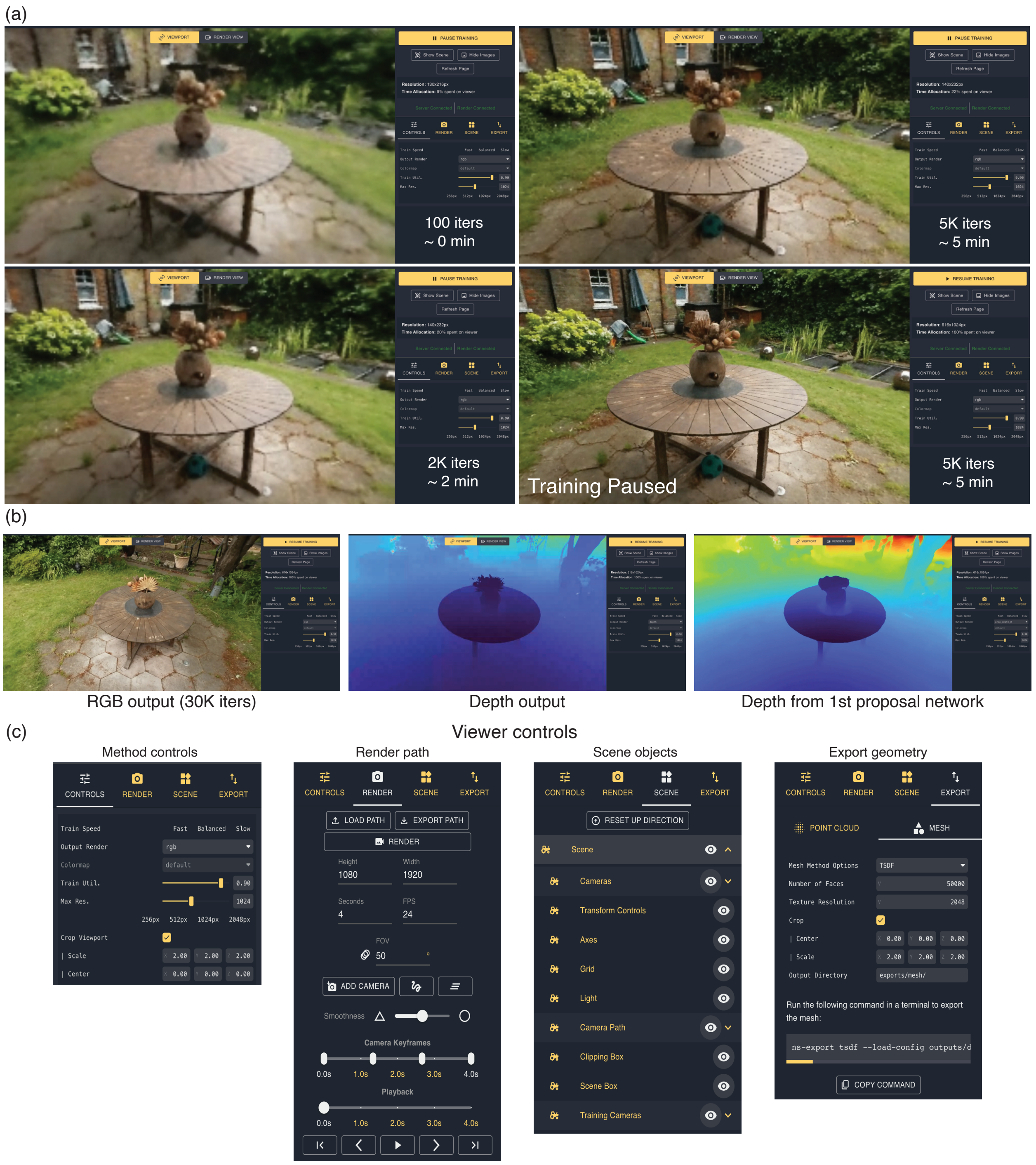
It is worth emphasizing that our Nerfacto method is optimized for qualitative novel-view quality by using the web viewer, rather than solely relying on common metrics. For further illustration, we refer the reader to the appendix where we provide rendered videos from our Nerfacto method.

Nerfacto component ablations

Given the modularity of our codebase, we can easily conduct ablation studies on our method Nerfacto, a unified approach that combines important components from various papers to achieve a fast, high-quality method. We experiment with disabling the pose optimization, appearance embeddings, scene contraction, and variations of the proposal networks, and more. The modularity of our codebase allows for easy implementation of these modifications through the use of different flags with the command line interface.

Evaluation protocol. In our ablation study, we utilize the Nerfstudio Dataset for evaluation. Due to the complexity of the appearance embeddings and pose optimization modules, we adopt a test-time optimization procedure for the evaluation. Specifically, we employ Adam optimizers to optimize the evaluation camera poses. Once the camera poses are fixed, we randomly select either the left or right side of the evaluation image and optimize the appearance code as done in Martin et al. [112]. Finally, with the optimized camera pose and appearance embedding, we compute PSNR, SSIM, and LPIPS. For these experiments, we hold out 1 in every 10 frames of our data as the evaluation set to evaluate on a representative distribution of our data.. We will release this evaluation protocol so future work can run similar experiments.

Findings. Table 6.2 presents the average results of our ablation studies. The complete table for all 10 scenes can be found in the appendix. This study highlights the challenge in extracting meaningful insights from quantitative metrics alone (Table 6.2), due to the fact that held-out evaluation images are close to the training images. For instance, disabling the appearance embeddings ("w/o app") leads to an improvement in PSNR and SSIM. However, Fig. 6.8 illustrates that the "w/o app" method results in the production of blurry "floater" artifacts. These artifacts correspond with the training camera locations because the model overfits to small discrepancies in lighting conditions in the training data by placing these artifacts directly in front of the training cameras. (bottom row, bottom left crop). Furthermore, ablations such as "1 prop network" result in subtle changes in the metrics but are more evident in visualizations of the novel views. The use of "1 prop network" as opposed to "Nerfacto (default)" with 2 prop networks leads to aliasing artifacts as can



Train Speed Fast Balanced Slow
Output Render rgb
Colormap default
Train Util. 0.90
Max Res. 1024
256px 512px 1024px 2048px
Crop Viewport ✓
| Scale 2.00 2.00 2.00
| Center 0.00 0.00 0.00

Height 1080
Width 1920
Seconds 4
FPS 24
FOV 50 °
0.0s 1.0s 2.0s 3.0s 4.0s

Mesh Method Options TSDF
Number of Faces 50000
Texture Resolution 2048
Crop ✓
| Center 0.00 0.00 0.00
| Scale 2.00 2.00 2.00
Output Directory exports/mesh/

Run the following command in a terminal to export the mesh:

```
ns-export tsdf --load-config outputs/d
```

Figure 6.9: **Real-time viewer use.** (a) Training Nerfacto on the MipNeRF-360 garden scene. Good quality can be achieved after a few minutes. Pausing the training increases the rendered resolution. (b) Visualizing different model outputs with the viewer. (c) Viewer controls and settings available in the viewer.

Nerfacto method	**PSNR** ↑	**SSIM** ↑	**LPIPS** ↓
Nerfacto (default)	20.99	0.663	0.389
w/o pose	20.93	0.659	0.393
w/o app	22.65	0.672	0.406
w/o pose & app	22.53	0.671	0.411
1 prop network	21.07	0.669	0.396
l2 contraction	20.98	0.664	0.388
shared prop network	20.95	0.661	0.391
random backg. color	21.00	0.663	0.392
no contraction	18.59	0.534	0.506
synthetic on real	20.09	0.542	0.509

Table 6.2: **Average metrics for ablations on the Nerfstudio Dataset.** We remove and change various components of the Nerfacto method and report { PSNR, SSIM, LPIPS } on the Nerfstudio Dataset. Further details on the experiments can be found in the Appendix.

be seen around the small tree branches (bottom row, middle crop). While these artifacts are visible to the eye especially in the interactive viewer, such temporal discontinuity caused by aliasing is not captured by the quantitative metrics. Furthermore, scene contraction is necessary to correctly recover far objects (top row, right crop).

Overall, the real-time viewer proves to be useful for viewing out-of-distribution renders. The crops in Fig. 6.8 aid in illustrating where certain methods excel over others, regardless of the metrics on the evaluation images. Developing more appropriate evaluation metrics is an important avenue for future research.

6.7 Open-source Contributions

One of the key strengths of our proposed framework is its versatility and ease of use, as demonstrated by our open-source contributions. Our GitHub repository has grown to include over 60 contributors and over 3K stars, reflecting a strong and active community. Additionally, two new libraries, SDFStudio [222] and ArcNerf [223], have been built on top of our framework. Since the release of Nerfstudio in October 2022, our contributors have enhanced and expanded Nerfstudio by addressing various GitHub issues and feature requests including improved camera paths, colab support, additional camera models, reconstruction of dynamic objects.

6.8 Discussion

We draw upon existing techniques and propose a framework that supports a more modularized approach to NeRF development, allows for real-time visualization, and is readily usable with real-world data. We emphasize the importance of utilizing the interactive real-time viewer during training to compensate for imperfect quantitative metrics in model design decisions. We hope the consolidation brought about by this new framework will facilitate the development of NeRF-based methods, thereby accelerating advances in the neural rendering community. Future research directions include the development of more appropriate evaluation metrics and integration of the framework with other fields such as computer vision, computer graphics, and machine learning.

Chapter 7

Conclusion

In this dissertation, we introduce NeRFs (Chapter 2) as a novel approach to achieve high-quality novel view synthesis, showcasing photorealistic results with non-Lambertian lighting effects. We investigate the efficacy of positional encodings for capturing fine details (chapter 3), extend the generalization capabilities of NeRF in chapter 4, and extend its scalability to handle arbitrarily large scenes in chapter 5. Finally, we present the Nerfstudio framework (chapter 6), offering a valuable tool to facilitate future NeRF development and exploration.

7.1 Why NeRFs and the Role of Representation

The popularity of NeRFs prompts an intriguing question: why did they gain widespread attention? While it is true that NeRFs produce high-quality results, so did previous methods [120] implying that there may be other factors at play. One potential reason is the simplicity of the representation employed by NeRFs, which sets them apart from other approaches prevalent at the time, such as mesh or signed distance function (SDF) based methods. NeRFs are more similar in setup to other deep learning methods, making them accessible to researchers familiar with the principles of deep learning and modern machine learning frameworks like PyTorch, TensorFlow, and JAX. With just a few hundred lines of code, one can quickly set up a basic version of NeRF, enabling researchers from the computer vision community to contribute and participate without requiring extensive expertise in the field.

As the field has progressed, it has become evident that the core aspect for successful NeRF-based reconstruction lies in the volume rendering technique, rather than the specific neural network representation itself. In fact, studies have shown that comparable reconstruction quality can be achieved using voxel-based methods alone [45]. This realization highlights that the underlying representation plays a more crucial role in optimization and inference speed, rather than solely determining the output quality. As a result, many recent approaches have adopted a hybrid approach, combining the strengths of neural networks with other representations [126, 187, 23]. It is still an open question why global optimization of the volume, supervised by volume rendering, is so effective at explaining sparse views.

It is intriguing to note that while neural networks played a pivotal role in popularizing NeRFs, they are also the component undergoing significant changes to make these representations more practical. As we delve into developing lower-level methods, it becomes crucial to provide abstractions that ensure accessibility. This need for accessibility was one of the driving factors behind the development of Nerfstudio. By providing a user-friendly framework, Nerfstudio enables researchers from domains outside of 3D computer vision to more easily utilize and contribute to NeRFs. Furthermore, for those engaged in NeRF-related research, we have developed nerfacc [94], a NeRF acceleration toolbox implemented in CUDA with a Python API. These tools and abstractions aim to facilitate the adoption and development of NeRFs, allowing researchers to focus on advancing the field without being hindered by implementation complexities.

7.2 Limitations

While NeRFs have demonstrated impressive capabilities in scene reconstruction, they are not without their limitations. The original formulation described in Chapter 2 made several assumptions that constrained its applicability. It assumed static scenes, where objects did not move between images, and lighting and exposure remained constant. It also assumed images were captured without motion blur and that camera poses were accurate. Furthermore, the method was limited to bounded or forward-facing scenes and lacked generalization due to the absence of learned priors. Additionally, NeRF required thorough scene coverage and suffered from slow reconstruction times, often taking up to a day for a single scene.

In Chapters 4 and 5, we addressed a subset of these limitations by exploring topics such as generalization, appearance changes, pose errors, and unbounded scenes. It is worth noting that the academic community has shown significant interest in NeRF, resulting in over a thousand follow-up papers since its original publication. In response to this growing body of work, Chapter 6 presents Nerfstudio, a framework that consolidates and builds upon these advancements.

Despite the progress made, several challenges remain to be addressed. These include achieving high-quality dynamic reconstruction of arbitrary objects beyond humans, developing compact and efficient methods that can be trained and deployed on devices with limited computational resources (e.g., mobile devices), and enabling scene editing and synthesis using generative approaches. There is also a question regarding the choice of inputs. Currently, many methods assume prior knowledge of camera poses before training, but it is reasonable to expect that NeRF optimization will be able to learn and optimize the poses from scratch in the future. It is peculiar that we currently employ complete structure-from-motion pipelines like COLMAP, only to discard everything except the poses. There are already some promising developments in this direction [100, 114, 207, 74].

Another interesting direction to explore is the application of NeRFs beyond view synthesis. This includes leveraging NeRFs for tasks such as navigation, scene understanding, and robotic simulation. While there have been some works in these areas [234, 167, 176, 87, 84, 83], they are relatively nascent and hold significant potential for further exploration and development. By extending the capabilities of NeRFs to address these diverse applications, we can unlock new possibilities in fields such as robotics and virtual reality.

7.3 The Future of 3D

This dissertation has been dedicated to the replication of our world, a task historically challenging and limited to technical experts. However, the desire to not only replicate but also modify and create entirely new worlds is becoming increasingly prevalent. Similar to replication, creation currently requires specialized expertise, presenting a significant area for future development.

We have recently witnessed the transformative power of generative 2D tools [154, 165, 16, 227], which have revolutionized image creation and editing. This shift has moved the field beyond being purely technical, allowing for more conceptual and creative exploration. These methods enable the synthesis of high-quality images from text or other simple operations, empowering users to unleash their artistic vision like never before. Similar transformations are expected to unfold in the realm of 3D. Already, we are witnessing the emergence of text-to-3D models, exemplified by tools like Dreamfusion [143], as well as techniques for scene modification using high-level instructions, as demonstrated by Instruct-NeRF2NeRF [59].

The era of relying on complex tools like Blender to create production-quality 3D assets is gradually fading. Instead, we should anticipate a future where untrained individuals can generate assets of comparable quality using nothing more than their smartphones. For example, users could capture a few photos or a video of a scene and have it converted to 3D, or start from a text prompt. They can then iteratively modify the colors, textures, and shapes of objects through text prompts and simple interactions such as clicking, unlocking the ability to create their desired scene. This paradigm shift empowers users to effortlessly create and customize 3D content, democratizing the process and unlocking endless creative possibilities. While this vision holds tremendous promise, it necessitates further research and advancement.

We find ourselves at an exciting juncture where the potential for 3D creation and manipulation is poised to reach new heights. The journey toward realizing this future entails continued exploration and innovation in the field, as we strive to unlock the full potential of accessible and user-friendly 3D tools. With sustained research efforts, we can propel the field of 3D graphics into an era of unprecedented creativity and inclusivity.

Bibliography

[1] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015.

[2] Michal Adamkiewicz et al. "Vision-Only Robot Navigation in a Neural Radiance World". In: *CoRR* abs/2110.00168 (2021). arXiv: `2110.00168`. URL: `https://arxiv.org/abs/2110.00168`.

[3] Sameer Agarwal et al. "Building rome in a day". In: *Communications of the ACM* (2011).

[4] Eirikur Agustsson and Radu Timofte. "NTIRE 2017 Challenge on Single Image Super-Resolution: Dataset and Study". In: *CVPR Workshops* (2017).

[5] Alexander Amini et al. "Learning robust control policies for end-to-end autonomous driving from data-driven simulation". In: *IEEE Robotics and Automation Letters* (2020).

[6] Antreas Antoniou, Harrison Edwards, and Amos Storkey. "How to train your MAML". In: *ICLR* (2018).

[7] Sanjeev Arora et al. "Fine-Grained Analysis of Optimization and Generalization for Over-parameterized Two-Layer Neural Networks". In: *ICML* (2019).

[8] 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.

[9] 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.

[10] Ronen Basri et al. "Frequency Bias in Neural Networks for Input of Non-Uniform Density". In: *arXiv preprint arXiv:2003.04560* (2020).

[11] Ronen Basri et al. "The Convergence Rate of Neural Networks for Learned Functions of Different Frequencies". In: *NeurIPS* (2019).

[12] Alberto Bietti and Julien Mairal. "On the Inductive Bias of Neural Tangent Kernels". In: *NeurIPS* (2019).

[13] Piotr Bojanowski et al. "Optimizing the latent space of generative networks". In: *arXiv:1707.05776* (2017).

[14] Blake Bordelon, Abdulkadir Canatar, and Cengiz Pehlevan. "Spectrum Dependent Learning Curves in Kernel Regression and Wide Neural Networks". In: *arXiv preprint arXiv:2002.02561* (2020).

[15] James Bradbury et al. *JAX: composable transformations of Python+NumPy programs*. Version 0.1.68. `http://github.com/google/jax`. 2018. URL: `http://github.com/google/jax`.

[16] Tim Brooks, Aleksander Holynski, and Alexei A Efros. "Instructpix2pix: Learning to follow image editing instructions". In: *arXiv preprint arXiv:2211.09800* (2022).

[17] Chris Buehler et al. "Unstructured Lumigraph Rendering". In: *SIGGRAPH*. 2001.

[18] Chris Buehler et al. "Unstructured lumigraph rendering". In: *Computer graphics and interactive techniques* (2001).

[19] Arunkumar Byravan et al. *NeRF2Real: Sim2real Transfer of Vision-guided Bipedal Motion Skills using Neural Radiance Fields*. 2022. DOI: `10.48550/ARXIV.2210.04932`. URL: `https://arxiv.org/abs/2210.04932`.

[20] Holger Caesar et al. "nuscenes: A multimodal dataset for autonomous driving". In: *CVPR* (2020).

[21] Angel X Chang et al. "Shapenet: An information-rich 3d model repository". In: *arXiv:1512.03012* (2015).

[22] Ming-Fang Chang et al. "Argoverse: 3d tracking and forecasting with rich maps". In: *CVPR* (2019).

[23] Anpei Chen et al. "TensoRF: Tensorial Radiance Fields". In: *European Conference on Computer Vision (ECCV)*. 2022.

[24] Wenzheng Chen et al. "Learning to Predict 3D Objects with an Interpolation-based Differentiable Renderer". In: *NeurIPS*. 2019.

[25] Yun Chen et al. "GeoSim: Realistic Video Simulation via Geometry-Aware Composition for Self-Driving". In: *CVPR* (2021).

[26] Zhiqin Chen and Hao Zhang. "Learning Implicit Fields for Generative Shape Modeling". In: *CVPR*. 2019.

[27] Bowen Cheng et al. "Panoptic-deeplab: A simple, strong, and fast baseline for bottom-up panoptic segmentation". In: *CVPR* (2020).

[28] Michael Cohen et al. "The Lumigraph". In: *SIGGRAPH*. 1996.

[29] Robert L Cook, Loren Carpenter, and Edwin Catmull. "The Reyes image rendering architecture". In: *ACM SIGGRAPH Computer Graphics* 21.4 (1987), pp. 95–102.

[30] Marius Cordts et al. "The cityscapes dataset for semantic urban scene understanding". In: *CVPR* (2016).

[31] Brian Curless and Marc Levoy. "A volumetric method for building complex models from range images". In: *SIGGRAPH*. 1996.

[32] Abe Davis, Marc Levoy, and Fredo Durand. "Unstructured Light Fields". In: *Eurographics*. 2012.

[33] Paul Debevec, Camillo J. Taylor, and Jitendra Malik. "Modeling and Rendering Architecture from Photographs: A Hybrid Geometry-and Image-Based Approach". In: *SIGGRAPH*. 1996.

[34] Boyang Deng et al. "Neural Articulated Shape Approximation". In: *arXiv preprint arXiv:1912.03207* (2019).

[35] Jeevan Devaranjan, Amlan Kar, and Sanja Fidler. "Meta-sim2: Unsupervised learning of scene structure for synthetic data generation". In: *ECCV* (2020).

[36] Alexey Dosovitskiy et al. "CARLA: An open urban driving simulator". In: *Conference on robot learning* (2017).

[37] Danny Driess et al. "Learning Multi-Object Dynamics with Compositional Neural Radiance Fields". In: *arXiv preprint arXiv:2202.11855* (2022).

[38] Dawei Du et al. "The unmanned aerial vehicle benchmark: Object detection and tracking". In: *ECCV* (2018).

[39] Simon S. Du et al. "Gradient Descent Provably Optimizes Over-parameterized Neural Networks". In: *ICLR* (2019).

[40] Alireza Fallah, Aryan Mokhtari, and Asuman Ozdaglar. "On the convergence theory of gradient-based model-agnostic meta-learning algorithms". In: *AISTATS*. 2020.

[41] Chelsea Finn, Pieter Abbeel, and Sergey Levine. "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks". In: *ICML* (2017).

[42] Sebastian Flennerhag et al. "Meta-learning with warped gradient descent". In: *ICLR* (2020).

[43] John Flynn et al. "Deepstereo: Learning to predict new views from the world's imagery". In: *CVPR* (2016).

[44] John Flynn et al. "DeepView: view synthesis with learned gradient descent". In: *CVPR*. 2019.

[45] Sara Fridovich-Keil et al. "Plenoxels: Radiance Fields Without Neural Networks". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022, pp. 5501–5510.

[46] Christian Früh and Avideh Zakhor. "An automated method for large-scale, ground-based city model acquisition". In: *IJCV* (2004).

[47] Yasutaka Furukawa and Jean Ponce. "Accurate, Dense, and Robust Multi-View Stereopsis". In: *IEEE TPAMI* (2010).

[48] Yasutaka Furukawa et al. "Towards internet-scale multi-view stereo". In: *CVPR* (2010).

[49] Adrien Gaidon et al. "Virtual worlds as proxy for multi-object tracking analysis". In: *CVPR* (2016).

[50] Stephan J Garbin et al. “Fastnerf: High-fidelity neural rendering at 200fps”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 14346–14355.

[51] Marc-André Gardner et al. “Learning to predict indoor illumination from a single image”. In: *arXiv preprint arXiv:1704.00090* (2017).

[52] Andreas Geiger, Philip Lenz, and Raquel Urtasun. “Are we ready for autonomous driving? the kitti vision benchmark suite”. In: *CVPR* (2012).

[53] Kyle Genova et al. “Learning Shape Templates with Structured Implicit Functions”. In: *ICCV* (2019).

[54] Kyle Genova et al. “Local Deep Implicit Functions for 3D Shape”. In: *CVPR*. 2020.

[55] Kyle Genova et al. “Local Deep Implicit Functions for 3D Shape”. In: *CVPR* (2020).

[56] Kyle Genova et al. “Unsupervised Training for 3D Morphable Model Regression”. In: *CVPR*. 2018.

[57] Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *AISTATS* (2010).

[58] Mordechai Haklay and Patrick Weber. “Openstreetmap: User-generated street maps”. In: *IEEE Pervasive computing* (2008).

[59] Ayaan Haque et al. “Instruct-NeRF2NeRF: Editing 3D Scenes with Instructions”. In: *arXiv preprint arXiv:2303.12789* (2023).

[60] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Second. Cambridge University Press, 2004.

[61] Reinhard Heckel and Mahdi Soltanolkotabi. “Compressive sensing with un-trained neural networks: Gradient descent finds the smoothest approximation”. In: *arXiv preprint arXiv:2005.03991* (2020).

[62] Peter Hedman et al. “Baking neural radiance fields for real-time view synthesis”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 5875–5884.

[63] Peter Hedman et al. “Deep blending for free-viewpoint image-based rendering”. In: *ACM Transactions on Graphics (TOG)* (2018).

[64] Philipp Henzler, Niloy J Mitra, and Tobias Ritschel. “Learning a Neural 3D Texture Space from 2D Exemplars”. In: *CVPR* (2020).

[65] Philipp Henzler, Niloy J. Mitra, and Tobias Ritschel. “Learning a Neural 3D Texture Space from 2D Exemplars”. In: *CVPR*. 2020.

[66] Philipp Henzler et al. “Single-Image Tomography: 3D Volumes from 2D Cranial X-Rays”. In: *Eurographics*. 2018.

[67] K. Hornik, M. Stinchcombe, and H. White. “Multilayer Feedforward Networks Are Universal Approximators”. In: *Neural Networks* (1989).

[68] Timothy Hospedales et al. “Meta-learning in neural networks: A survey”. In: *arXiv preprint arXiv:2004.05439* (2020).

[69] Jeremy Howard. *imagenette*. URL: `https://github.com/fastai/imagenette/`.

[70] Yuanming Hu et al. “Taichi: a language for high-performance computation on spatially sparse data structures”. In: *ACM Transactions on Graphics (TOG)* 38.6 (2019), pp. 1–16.

[71] Arthur Jacot, Franck Gabriel, and Clément Hongler. “Neural Tangent Kernel: Convergence and generalization in neural networks”. In: *NeurIPS* (2018).

[72] Wenzel Jakob et al. *Mitsuba 3 renderer*. Version 3.1.1. https://mitsuba-renderer.org. 2022.

[73] Yoonwoo Jeong, Seungjoo Shin, and Kibaek Park. *NeRF-Factory: An awesome PyTorch NeRF collection*. 2022. URL: `https://github.com/kakaobrain/NeRF-Factory/`.

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

[75] Yangqing Jia et al. “Caffe: Convolutional architecture for fast feature embedding”. In: *Proceedings of the 22nd ACM international conference on Multimedia*. 2014, pp. 675–678.

[76] Chiyu Jiang et al. “Local Implicit Grid Representations for 3D Scenes”. In: *CVPR*. 2020.

[77] Yuhe Jin et al. “Image matching across wide baselines: From paper to practice”. In: *International Journal of Computer Vision* (2020), pp. 1–31.

[78] James T. Kajiya and Brian P. Von Herzen. “Ray Tracing Volume Densities”. In: *Computer Graphics (SIGGRAPH)* (1984).

[79] Abhishek Kar, Christian Häne, and Jitendra Malik. “Learning a Multi-View Stereo Machine”. In: *NeurIPS*. 2017.

[80] Amlan Kar et al. “Meta-sim: Learning to generate synthetic datasets”. In: *ICCV* (2019).

[81] Seyed Mehran Kazemi et al. “Time2Vec: Learning a Vector Representation of Time”. In: *arXiv preprint arXiv:1907.05321* (2019).

[82] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. “Poisson surface reconstruction”. In: *Proceedings of the fourth Eurographics symposium on Geometry processing*. Vol. 7. 2006.

[83] Justin Kerr et al. “Evo-NeRF: Evolving NeRF for Sequential Robot Grasping of Transparent Objects”. In: *6th Annual Conference on Robot Learning*. 2022.

[84] Justin Kerr et al. “LERF: Language Embedded Radiance Fields”. In: *arXiv preprint arXiv:2303.09553* (2023).

[85] Seung Wook Kim et al. “DriveGAN: Towards a Controllable High-Quality Neural Simulation”. In: *CVPR* (2021).

[86] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *ICLR*. 2015.

[87] Sosuke Kobayashi, Eiichi Matsumoto, and Vincent Sitzmann. "Decomposing NeRF for Editing via Feature Field Distillation". In: *Advances in Neural Information Processing Systems*. Vol. 35. 2022. URL: `https://arxiv.org/pdf/2205.15585.pdf`.

[88] Johannes Kopf, Michael Cohen, and Rick Szeliski. "First-person Hyperlapse Videos". In: *SIGGRAPH* (2014).

[89] Johannes Kopf et al. "Street slide: browsing street level imagery". In: *ACM Transactions on Graphics (TOG)* (2010).

[90] Kiriakos N. Kutulakos and Steven M. Seitz. "A theory of shape by space carving". In: *International Journal of Computer Vision* (2000).

[91] Jaehoon Lee et al. "Wide neural networks of any depth evolve as linear models under gradient descent". In: *NeurIPS* (2019).

[92] Marc Levoy. "Efficient Ray Tracing of Volume Data". In: *ACM Transactions on Graphics* (1990).

[93] Marc Levoy and Pat Hanrahan. "Light Field Rendering". In: *SIGGRAPH*. 1996.

[94] Ruilong Li, Matthew Tancik, and Angjoo Kanazawa. "NerfAcc: A General NeRF Accleration Toolbox." In: *arXiv preprint arXiv:2210.04847* (2022).

[95] Tzu-Mao Li et al. "Differentiable Monte Carlo Ray Tracing through Edge Sampling". In: *ACM Transactions on Graphics (SIGGRAPH Asia)* (2018).

[96] Wei Li et al. "AADS: Augmented autonomous driving simulation using data-driven algorithms". In: *Science robotics* (2019).

[97] Xiaowei Li et al. "Modeling and recognition of landmark image collections using iconic scene graphs". In: *ECCV* (2008).

[98] Zhenguo Li et al. "Meta-sgd: Learning to learn quickly for few-shot learning". In: *arXiv preprint arXiv:1707.09835* (2017).

[99] Sook-Lei Liew et al. "A large, open source dataset of stroke anatomical brain images and manual lesion segmentations". In: *Scientific Data* (2018).

[100] Chen-Hsuan Lin et al. "Barf: Bundle-adjusting neural radiance fields". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 5741–5751.

[101] Andrew Liu et al. "Infinite nature: Perpetual view generation of natural scenes from a single image". In: *ICCV* (2021).

[102] Lingjie Liu et al. "Neural Sparse Voxel Fields". In: *NeurIPS* (2020).

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

[104] Shaohui Liu et al. "DIST: Rendering Deep Implicit Signed Distance Function with Differentiable Sphere Tracing". In: *CVPR*. 2019.

[105] Shichen Liu et al. “Learning to Infer Implicit Surfaces without 3D Supervision”. In: *NeurIPS*. 2019.

[106] Shichen Liu et al. “Soft Rasterizer: A Differentiable Renderer for Image-based 3D Reasoning”. In: *ICCV*. 2019.

[107] Ziwei Liu et al. “Deep Learning Face Attributes in the Wild”. In: *ICCV* (2015).

[108] Stephen Lombardi et al. “Neural volumes: Learning dynamic renderable volumes from images”. In: *ACM Transactions on Graphics (SIGGRAPH)* (2019).

[109] Matthew M. Loper and Michael J. Black. “OpenDR: An Approximate Differentiable Renderer”. In: *ECCV*. 2014.

[110] Frank Losasso and Hugues Hoppe. “Geometry clipmaps: terrain rendering using nested regular grids”. In: *Siggraph* (2004).

[111] David G Lowe. “Distinctive image features from scale-invariant keypoints”. In: *IJCV* (2004).

[112] Ricardo Martin-Brualla et al. “NeRF in the Wild: Neural Radiance Fields for Unconstrained Photo Collections”. In: *CVPR* (2021), pp. 7210–7219.

[113] Nelson Max. “Optical models for direct volume rendering”. In: *IEEE Transactions on Visualization and Computer Graphics* (1995).

[114] Quan Meng et al. “GNeRF: GAN-based Neural Radiance Field without Posed Camera”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2021.

[115] Lars Mescheder et al. “Occupancy Networks: Learning 3D Reconstruction in Function Space”. In: *CVPR*. 2019.

[116] Moustafa Meshry et al. “Neural Rerendering in the Wild”. In: *CVPR* (2019).

[117] Michael Dawson-Haggerty et al. *trimesh*. Version 3.2.0. `https://trimsh.org/`. 2019.

[118] Mateusz Michalkiewicz et al. “Implicit surface representations as layers in neural networks”. In: *ICCV*. 2019.

[119] Ben Mildenhall et al. “Local Light Field Fusion: Practical View Synthesis with Prescriptive Sampling Guidelines”. In: *ACM Transactions on Graphics (SIGGRAPH)* (2019).

[120] 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.

[121] Ben Mildenhall et al. *MultiNeRF: A Code Release for Mip-NeRF 360, Ref-NeRF, and RawNeRF*. 2022. URL: `https://github.com/google-research/multinerf`.

[122] Ben Mildenhall et al. “Nerf in the dark: High dynamic range view synthesis from noisy raw images”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 16190–16199.

[123] Ben Mildenhall et al. “NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis”. In: *ECCV*. 2020.

[124] Thomas Müller. *tiny-cuda-nn*. Version 1.6. Apr. 2021. URL: `https://github.com/NVlabs/tiny-cuda-nn`.

[125] Thomas Müller et al. "Instant neural graphics primitives with a multiresolution hash encoding". In: *arXiv preprint arXiv:2201.05989* (2022).

[126] Thomas Müller et al. "Instant neural graphics primitives with a multiresolution hash encoding". In: *ACM Transactions on Graphics (ToG)* 41.4 (2022), pp. 1–15.

[127] Krishna Murthy. *nerf-pytorch: A PyTorch re-implementation*. 2020. URL: `https://github.com/krrish94/nerf-pytorch`.

[128] Anh Nguyen, Jason Yosinski, and Jeff Clune. "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images". In: *CVPR* (2015).

[129] Alex Nichol, Joshua Achiam, and John Schulman. "On first-order meta-learning algorithms". In: *arXiv preprint arXiv:1803.02999* (2018).

[130] Michael Niemeyer et al. "Differentiable Volumetric Rendering: Learning Implicit 3D Representations without 3D Supervision". In: *CVPR*. 2019.

[131] Michael Niemeyer et al. "Differentiable Volumetric Rendering: Learning Implicit 3D Representations without 3D Supervision". In: *CVPR* (2020).

[132] Merlin Nimier-David et al. "Mitsuba 2: A Retargetable Forward and Inverse Renderer". In: *ACM Transactions on Graphics (SIGGRAPH Asia)* (2019).

[133] Roman Novak et al. "Neural Tangents: Fast and Easy Infinite Neural Networks in Python". In: *ICLR* (2020).

[134] Michael Oechsle, Songyou Peng, and Andreas Geiger. "Unisurf: Unifying neural implicit surfaces and radiance fields for multi-view reconstruction". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 5589–5599.

[135] Michael Oechsle et al. "Texture fields: Learning texture representations in function space". In: *ICCV*. 2019.

[136] Julian Ost et al. "Neural scene graphs for dynamic scenes". In: *CVPR* (2021).

[137] Jeong Joon Park et al. "DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation". In: *CVPR*. 2019.

[138] Adam Paszke et al. "Pytorch: An imperative style, high-performance deep learning library". In: *Advances in neural information processing systems* 32 (2019).

[139] Georgios Pavlakos et al. "The One Where They Reconstructed 3D Humans and Environments in TV Shows". In: *European Conference on Computer Vision*. Springer. 2022, pp. 732–749.

[140] Songyou Peng et al. "Convolutional occupancy networks". In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*. Springer. 2020, pp. 523–540.

[141] Eric Penner and Li Zhang. "Soft 3D Reconstruction for View Synthesis". In: *ACM Transactions on Graphics (SIGGRAPH Asia)* (2017).

[142] Marc Pollefeys et al. "Detailed real-time urban 3d reconstruction from video". In: *IJCV* (2008).

[143] Ben Poole et al. "DreamFusion: Text-to-3D using 2D Diffusion". In: *The Eleventh International Conference on Learning Representations*. 2023. URL: `https://openreview.net/forum?id=FjNys5c7VyY`.

[144] Thomas Porter and Tom Duff. "Compositing Digital Images". In: *Computer Graphics (SIGGRAPH)* (1984).

[145] Albert Pumarola et al. "D-nerf: Neural radiance fields for dynamic scenes". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 10318–10327.

[146] Chen Quei-An. *Nerf_pl: a pytorchlightning implementation of NeRF*. 2020. URL: `https://github.com/kwea123/nerf%5C_pl/`.

[147] Jonathan Ragan-Kelley et al. "Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines". In: *Acm Sigplan Notices* 48.6 (2013), pp. 519–530.

[148] Nasim Rahaman et al. "On the Spectral Bias of Neural Networks". In: *ICML*. 2018.

[149] Nasim Rahaman et al. "On the spectral bias of neural networks". In: *ICML* (2019).

[150] Ali Rahimi and Benjamin Recht. "Random features for large-scale kernel machines". In: *NeurIPS* (2007).

[151] Gilles Rainer et al. "Neural BTF Compression and Interpolation". In: *Computer Graphics Forum (Eurographics)* (2019).

[152] Gilles Rainer et al. "Unified Neural Encoding of BTFs". In: *Computer Graphics Forum (Eurographics)* (2020).

[153] Aravind Rajeswaran et al. "Meta-learning with implicit gradients". In: *NeurIPS* (2019).

[154] Aditya Ramesh et al. "Zero-shot text-to-image generation". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 8821–8831.

[155] Nikhila Ravi et al. "Accelerating 3d deep learning with pytorch3d". In: *arXiv preprint arXiv:2007.08501* (2020).

[156] S. Ravi and H. Larochelle. "Optimization as a Model for Few-Shot Learning". In: *ICLR* (2017).

[157] Daniel Rebain et al. "Derf: Decomposed radiance fields". In: *CVPR* (2021).

[158] Christian Reiser et al. "KiloNeRF: Speeding Up Neural Radiance Fields With Thousands of Tiny MLPs". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2021, pp. 14335–14345.

[159] Peiran Ren et al. “Global Illumination with Radiance Regression Functions”. In: *ACM Transactions on Graphics* (2013).

[160] Edgar Riba et al. “Kornia: an open source differentiable computer vision library for pytorch”. In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2020, pp. 3674–3683.

[161] Stephan R Richter, Hassan Abu AlHaija, and Vladlen Koltun. “Enhancing photorealism enhancement”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45.2 (2022), pp. 1700–1715.

[162] Stephan R Richter et al. “Playing for data: Ground truth from computer games”. In: *ECCV* (2016).

[163] Gernot Riegler and Vladlen Koltun. “Free View Synthesis”. In: *ECCV* (2020).

[164] Gernot Riegler and Vladlen Koltun. “Stable View Synthesis”. In: *CVPR* (2021).

[165] Robin Rombach et al. “High-resolution image synthesis with latent diffusion models”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 10684–10695.

[166] German Ros et al. “The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes”. In: *CVPR* (2016).

[167] Antoni Rosinol, John J Leonard, and Luca Carlone. “NeRF-SLAM: Real-Time Dense Monocular SLAM with Neural Radiance Fields”. In: *arXiv preprint arXiv:2210.13641* (2022).

[168] Darius Rückert et al. “NeAT: Neural Adaptive Tomography”. In: *ACM Trans. Graph.* 41.4 (2022). ISSN: 0730-0301. DOI: `10.1145/3528223.3530121`. URL: `https://doi.org/10.1145/3528223.3530121`.

[169] Shunsuke Saito et al. “PIFu: Pixel-Aligned Implicit Function for High-Resolution Clothed Human Digitization”. In: *ICCV* (2019).

[170] Johannes Lutz Schönberger and Jan-Michael Frahm. “Structure-from-Motion Revisited”. In: *CVPR*. 2016.

[171] Johannes Lutz Schönberger et al. “A Vote-and-Verify Strategy for Fast Spatial Verification in Image Retrieval”. In: *Asian Conference on Computer Vision (ACCV)*. 2016.

[172] Johannes Lutz Schönberger et al. “Pixelwise View Selection for Unstructured Multi-View Stereo”. In: *European Conference on Computer Vision (ECCV)*. 2016.

[173] Steven M. Seitz and Charles R. Dyer. “Photorealistic scene reconstruction by voxel coloring”. In: *International Journal of Computer Vision* (1999).

[174] Qi Shan et al. “The Visual Turing Test for Scene Reconstruction”. In: *3DV* (2013).

[175] Lawrence A. Shepp and Benjamin F. Logan. “The Fourier reconstruction of a head section”. In: *IEEE Transactions on nuclear science* (1974).

[176] Anthony Simeonov et al. “Neural Descriptor Fields: SE(3)-Equivariant Object Representations for Manipulation”. In: *ICRA*. 2022, pp. 6394–6400. URL: `https://doi.org/10.1109/ICRA46639.2022.9812146`.

[177] Vincent Sitzmann, Michael Zollhoefer, and Gordon Wetzstein. “Scene Representation Networks: Continuous 3D-Structure-Aware Neural Scene Representations”. In: *NeurIPS*. 2019.

[178] Vincent Sitzmann et al. “DeepVoxels: Learning Persistent 3D Feature Embeddings”. In: *CVPR*. 2019.

[179] Vincent Sitzmann et al. “Implicit Neural Representations with Periodic Activation Functions”. In: *NeurIPS* (2020).

[180] Vincent Sitzmann et al. “MetaSDF: Meta-Learning Signed Distance Functions”. In: *NeurIPS* (2020).

[181] Noah Snavely, Steven M Seitz, and Richard Szeliski. “Photo tourism: exploring photo collections in 3D”. In: *ACM siggraph 2006 papers*. 2006, pp. 835–846.

[182] Pratul P. Srinivasan et al. “NeRV: Neural reflectance and visibility fields for relighting and view synthesis”. In: *CVPR* (2021).

[183] Pratul P. Srinivasan et al. “Pushing the Boundaries of View Extrapolation with Multiplane Images”. In: *CVPR*. 2019.

[184] Kenneth O Stanley. “Compositional pattern producing networks: A novel abstraction of development”. In: *Genetic programming and evolvable machines* (2007).

[185] Kenneth O Stanley. “Compositional pattern producing networks: A novel abstraction of development”. In: *Genetic programming and evolvable machines* 8.2 (2007), pp. 131–162.

[186] Shih-Yang Su et al. “A-nerf: Articulated neural radiance fields for learning human shape, appearance, and pose”. In: *Advances in Neural Information Processing Systems* 34 (2021).

[187] Cheng Sun, Min Sun, and Hwann-Tzong Chen. “Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 5459–5469.

[188] Pei Sun et al. “Scalability in perception for autonomous driving: Waymo open dataset”. In: *CVPR* (2020).

[189] Richard Szeliski and Polina Golland. “Stereo matching with transparency and matting”. In: *ICCV*. 1998.

[190] Towaki Takikawa et al. *Kaolin Wisp: A PyTorch Library and Engine for Neural Fields Research*. `https://github.com/NVIDIAGameWorks/kaolin-wisp`. 2022.

[191] Towaki Takikawa et al. “Neural Geometric Level of Detail: Real-time Rendering with Implicit 3D Shapes”. In: *CVPR* (2021).

[192] 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.

[193] Matthew Tancik et al. “Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains”. In: *NeurIPS* (2020).

[194] Matthew Tancik et al. “Learned Initializations for Optimizing Coordinate-Based Neural Representations”. In: *CVPR*. 2021.

[195] Matthew Tancik et al. “Nerfstudio: A modular framework for neural radiance field development”. In: *arXiv preprint arXiv:2302.04264* (2023).

[196] Sebastian Thrun. “Probabilistic robotics”. In: *Communications of the ACM* (2002).

[197] Bill Triggs et al. “Bundle adjustment—a modern synthesis”. In: *International workshop on vision algorithms* (1999).

[198] Shubham Tulsiani et al. “Multi-view Supervision for Single-view Reconstruction via Differentiable Ray Consistency”. In: *CVPR*. 2017.

[199] Ashish Vaswani et al. “Attention is all you need”. In: *NeurIPS*. 2017.

[200] Dor Verbin et al. “Ref-nerf: Structured view-dependent appearance for neural radiance fields”. In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2022, pp. 5481–5490.

[201] Michael Waechter, Nils Moehrle, and Michael Goesele. “Let There Be Color! Large-Scale Texturing of 3D Reconstructions”. In: *ECCV*. 2014.

[202] Martin J. Wainwright. “Reproducing Kernel Hilbert Spaces”. In: *High-Dimensional Statistics: A Non-Asymptotic Viewpoint*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 2019, pp. 383–415. DOI: 10.1017/9781108627771.012.

[203] Ingo Wald et al. “Embree: a kernel framework for efficient CPU ray tracing”. In: *ACM Transactions on Graphics (TOG)* (2014).

[204] Can Wang et al. “Clip-nerf: Text-and-image driven manipulation of neural radiance fields”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 3835–3844.

[205] Peng Wang et al. “NeuS: Learning Neural Implicit Surfaces by Volume Rendering for Multi-view Reconstruction”. In: *NeurIPS* (2021).

[206] Qianqian Wang et al. “Ibrnet: Learning multi-view image-based rendering”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 4690–4699.

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

[208] Katherine Watson et al. *Creating workflows for NeRF Portraiture*. 2022. URL: `https://rd.nytimes.com/projects/creating-workflows-for-nerf-portraiture`.

[209] Daniel N. Wood et al. "Surface Light Fields for 3D Photography". In: *SIGGRAPH*. 2000.

[210] Yiheng Xie et al. "Neural fields in visual computing and beyond". In: *Computer Graphics Forum*. Vol. 41. 2. Wiley Online Library. 2022, pp. 641–676.

[211] Da Xu et al. "Self-attention with Functional Time Representation Learning". In: *NeurIPS* (2019).

[212] Bangbang Yang et al. "Learning Object-Compositional Neural Radiance Field for Editable Scene Rendering". In: *ICCV* (2021).

[213] Greg Yang and Hadi Salman. "A fine-grained spectral perspective on neural networks". In: *arXiv preprint arXiv:1907.10599* (2019).

[214] Zhenpei Yang et al. "SurfelGAN: Synthesizing realistic sensor data for autonomous driving". In: *CVPR* (2020).

[215] Lior Yariv et al. "Multiview Neural Surface Reconstruction by Disentangling Geometry and Appearance". In: *NeurIPS* (2020).

[216] Lior Yariv et al. "Multiview Neural Surface Reconstruction by Disentangling Geometry and Appearance". In: *Advances in Neural Information Processing Systems* 33 (2020).

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

[218] Lin Yen-Chen. *NeRF-pytorch*. `https://github.com/yenchenlin/nerf-pytorch/`. 2020.

[219] Lin Yen-Chen et al. "iNeRF: Inverting Neural Radiance Fields for Pose Estimation". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2021.

[220] Alex Yu et al. "pixelnerf: Neural radiance fields from one or few images". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 4578–4587.

[221] Alex Yu et al. "Plenoctrees for real-time rendering of neural radiance fields". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 5752–5761.

[222] Zehao Yu et al. *SDFStudio: A Unified Framework for Surface Reconstruction*. 2022. URL: `https://github.com/autonomousvision/sdfstudio`.

[223] Yan-Pei Cao Yue Luo. *ArcNerf: Nerf-based object/scene rendering and extraction framework*. 2022. URL: `https://github.com/TencentARC/arcnerf/`.

[224] Jiakai Zhang et al. "Editable free-viewpoint video using a layered neural representation". In: *ACM Transactions on Graphics (TOG)* (2021).

[225] Kai Zhang et al. "Arf: Artistic radiance fields". In: *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXI*. Springer. 2022, pp. 717–733.

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

[227] Lvmin Zhang and Maneesh Agrawala. *Adding Conditional Control to Text-to-Image Diffusion Models*. 2023. arXiv: `2302.05543 [cs.CV]`.

[228] Richard Zhang et al. “The Unreasonable Effectiveness of Deep Features as a Perceptual Metric”. In: *CVPR*. 2018.

[229] Ellen D. Zhong et al. “Reconstructing continuous distributions of 3D protein structure from cryo-EM images”. In: *ICLR*. 2020.

[230] Tinghui Zhou et al. “Stereo Magnification: Learning View Synthesis using Multiplane Images”. In: *ACM Transactions on Graphics (SIGGRAPH)* (2018).

[231] Tinghui Zhou et al. “Stereo magnification: Learning view synthesis using multiplane images”. In: *arXiv:1805.09817* (2018).

[232] Jun-Yan Zhu, Yong Jae Lee, and Alexei A Efros. “Averageexplorer: Interactive exploration and alignment of visual data collections”. In: *ACM Transactions on Graphics (TOG)* 33.4 (2014), pp. 1–11.

[233] Siyu Zhu et al. “Very large-scale global SFM by distributed motion averaging”. In: *CVPR* (2018).

[234] Zihan Zhu et al. “NICE-SLAM: Neural Implicit Scalable Encoding for SLAM”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022.