Title
Neural Scene Representations for View Synthesis

Author
Mildenhall, Benjamin Joseph

Publication Date
2020

Peer reviewed|Thesis/dissertation

Neural Scene Representations for View Synthesis

by

Benjamin Joseph Mildenhall

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Assistant Professor Ren Ng, Chair
Professor Alexei Efros
Professor Martin Banks

Fall 2020

Neural Scene Representations for View Synthesis

Abstract

Neural Scene Representations for View Synthesis

by

Benjamin Joseph Mildenhall

Doctor of Philosophy in Computer Science

University of California, Berkeley

Assistant Professor Ren Ng, Chair

View synthesis is the problem of using a given set of input images to render a scene from new points of view. Recent approaches have combined deep learning and volume rendering to achieve photorealistic image quality. However, these methods rely on a dense 3D grid representation that only allows for a small amount of local camera movement and scales poorly to higher resolutions.

In this dissertation, we present a new approach to view synthesis based on neural radiance fields, an efficient way to represent a scene as a continuous function parameterized by the weights of a neural network. In contrast to using a feed-forward neural network to predict scene properties from a small number of inputs, a neural radiance field can be directly optimized to globally reconstruct a scene from tens or hundreds of input images and thus achieve high quality novel view synthesis over a large camera baseline.

The key to enabling high fidelity reconstruction of a low-dimensional signal using a neural network is a high frequency mapping of the input coordinates into a higher-dimensional space. We explain the connection between this mapping and the neural tangent kernel, and show how manipulating the frequency spectrum of the mapping provides control over the network's interpolation behavior between supervision points.

Contents

List of Figures

List of Tables

Acknowledgments

I remember meeting Ren for the first time in a cafe in Palo Alto nearly seven years ago and being struck by his vibrant intellectual energy. I biked back to my dorm afterward and immediately started telling one of my friends about this exciting new professor and the group he wanted to start at Berkeley. Looking back as I complete my PhD, I am immeasurably glad I chose to join Ren's research group. He has taught me so much about the research process and the value of distilling and clarifying concepts in papers and in talks.

And I could not have asked for a better group of peers than Pratul Srinivasan, Grace Kuo, Cecilia Zhang, Matt Tancik, Utkarsh Singhal, and Vivien Nguyen for laughing at dumb research ideas, debating real research ideas, playing with ancient VCL treasures and the Vive, going on data collection adventures, and drinking boba way too many times—I will always miss hanging out in the office together. I have especially enjoyed and massively benefited from collaborating closely with Pratul and Matt over the last couple years; all of the work presented here was done jointly with both of them. Pratul and I first began working together in 2017, beginning a chaotic journey that started with us barely being able to calibrate camera poses and continues to this day with a ridiculously named method and results that exceed our wildest original expectations. I am eternally grateful to have a collaborator with such compatible research tastes and unending good humor in the face of setbacks and deadlines.

I am also grateful for the mentorship provided by my intern hosts at Google and Fyusion: thanks to Rob Carroll, Jiawen Chen, Dillon Sharlet, and Andrew Adams for teaching me how cameras work, and to Rodrigo Ortiz-Cayon and Abhishek Kar for showing me what research at a startup is like. Long after Pratul and I finished our first internships at Google, Jon Barron continues to be an amazing collaborator and unending source of helpful research and life advice.

I would also like to acknowledge the Hertz Foundation for providing financial support for the majority of my graduate studies.

Finally, I want to thank my friends and family for their support—countless delicious meals with Grace Kuo and Regina Eckert, relaxing California winter holidays with my parents, and through all the ups and downs, from our first coffee/tea together in Brewed Awakening til now, endless love and support from Coline—we did it!

Chapter 1

Introduction

Modern computer graphics techniques can be used to render images that are virtually indistinguishable from reality, given a sufficiently detailed 3D scene model. People typically imagine computer graphics and visual effects being used to conjure up dramatic explosions or futuristic worlds, but these technologies are more often deployed to imperceptibly add or modify objects, actors, backdrops, or even entire 3D environments in order to simplify the process of creating films, TV shows, advertisements, and so on.

Though the rendering process mapping from a virtual scene to an output image is entirely automated, creating these underlying 3D assets requires an immense amount of human labor; even a highly skilled digital artist would take hundreds of hours using sophisticated 3D modeling software to design, for example, a single realistic scene of a busy city street populated with a variety of buildings, cars, and people.

On the other hand, anyone with a camera can walk out onto a city street and capture a detailed, photorealistic image in the blink of an eye. Thanks to the ubiquity of modern smartphone cameras, humanity now produces an overwhelming deluge of photographs—over 45,000 per second in the year 2020 [43]. How can we exploit the ease of taking high quality 2D images to automate the capture of realistic real-world scenes, bypassing the labor-intensive process of designing detailed 3D models by hand?

1.1 The problem of view synthesis

In this dissertation, we address this question through the lens of *view synthesis*: given a set of photographs of a scene, how can we generate new images of that scene from previously unobserved viewpoints? This would be equivalent to recreating a digital version of the scene and moving the camera while leaving the lighting, objects, materials, etc. unchanged. A perfect view synthesis algorithm could transport a user wearing a virtual reality headset into another place entirely, allowing them to move their head freely within a fixed region of space, as well as enabling many other artistic, scientific, and commercial applications.

This problem setting is quite specific; it does not guarantee accurate recovery of 3D

geometry, disentangle material and lighting properties, or allow for moving objects. However, we posit that view synthesis is a "least common denominator" for these broader problems in inverse graphics: none of them can be tackled without a functioning view synthesis algorithm.

Any system for view synthesis must address the following questions:

- Representation: How will the underlying scene be represented?
- Renderer: How will new views of the scene be rendered from this representation?
- Reconstruction: How will this representation be recovered from the given input images?

Decades of computer graphics research have been spent investigating different representations and renderers for many varied use cases and computational budgets. View synthesis adds the additional constraint that it must be possible to devise an algorithm that recovers the underlying scene representation from images alone.

1.2 Considerations when designing an algorithm

When examining different view synthesis methods, certain design choices arise again and again. The first tradeoff we describe concerns local interpolation between input images versus global reconstruction of a full scene model, which is largely a matter of runtime requirements, available computational resources, and input data quality and may potentially never be fully resolved. The issues regarding surface versus volume rendering and feed-forward versus optimization-based pipelines are most relevant in the context of the recent rise of deep learning methods relying on fully differentiable rendering pipelines, which have made great strides toward photorealistic results on complex real-world scenes over the past five years.

1.2.1 Local interpolation vs. global reconstruction

This question represents a tension between two ideas: rendering an output view by blending patches of pixels taken directly from nearby input images can yield highly realistic results for a single frame, but rendering outputs using a global scene model reconstructed from all input images produces more temporally coherent results when smoothly changing the output viewpoint. Local interpolation relies on having a high sampling rate of input views (tightly packed cameras), while global reconstruction may require fewer views but rely on a representation that implies stronger priors on the scene content. Figure 1.1 illustrates where many existing view synthesis methods fall along this continuum.

Standard light field interpolation [35, 64] is the most basic view synthesis algorithm, making very few assumptions about the captured scene's appearance. This method treats the input views as discrete samples from the four-dimensional light field of all rays viewing the scene, thus allowing for a standard signal processing approach to reconstruction. When the input image positions lie on a regular 2D grid, it is possible to derive a Nyquist sampling

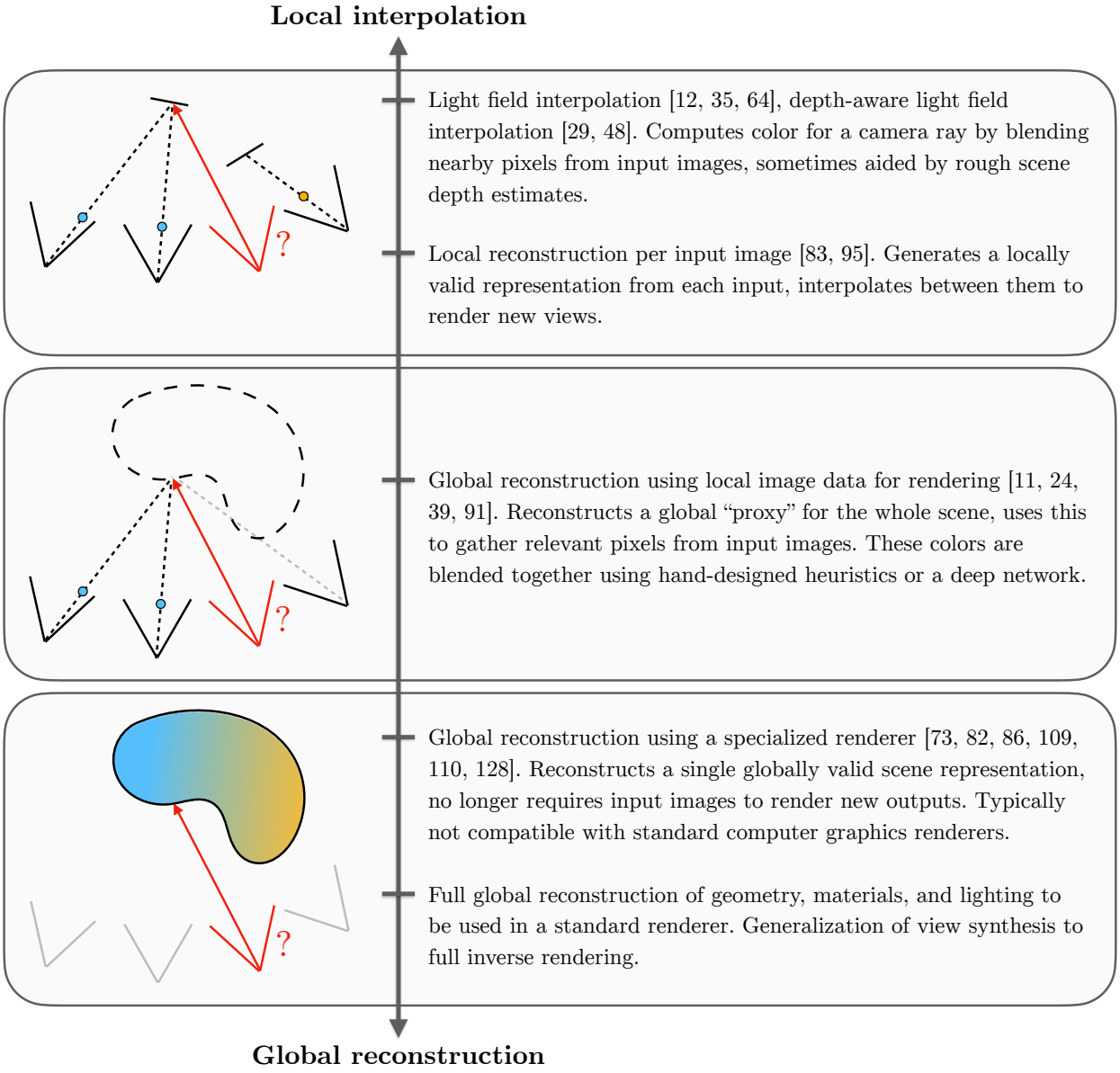


Figure 1.1: Spectrum of view synthesis methods from local interpolation between input images to global reconstruction of a full scene model.

bound that guarantees perfect reconstruction when the input cameras are spaced sufficiently close together [12]. This bound can be relaxed if additional depth information is provided or estimated. Some recent methods use deep learning to predict a depth value for every pixel in the rendered output and interpolate between the corresponding rays from neighboring input views [29, 48]; other methods generate a locally-valid scene representation corresponding to each input image that can be used to render nearby views, then interpolate between different local reconstructions as the novel view pose changes [83, 95].

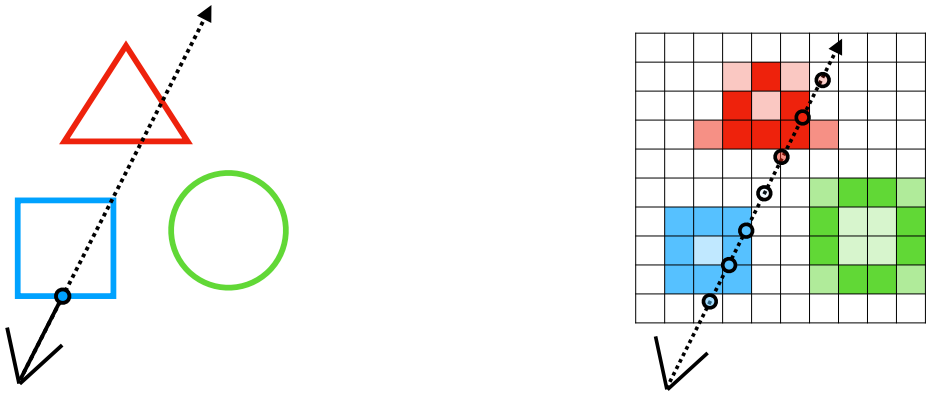
A common compromise between local and global approaches is to reconstruct an intermediate global "proxy" (usually a triangle mesh) that is used to select pixels from the input images, which are then blended to create the final rendered output. The blending step can be performed using hand-designed heuristics for speed and simplicity [11, 24, 91] or by deep networks for better image quality [39]. Most of these methods use a traditional structure-from-motion and dense multivew stereo reconstruction pipeline (such as COLMAP [103]) to generate the global proxy geometry, which can limit their usefulness in challenging scenes with complex geometry or reflective surfaces.

On the other end of the spectrum lie methods that create a single global representation of the scene and do not rely on the input images at all to render new views. In order to achieve photorealism, these methods typically use a custom representation and renderer, rather than generating a reconstruction that works directly with a typical graphics pipeline (such as a triangle mesh with corresponding texture and material maps). One historical example is the *surface light field* method, which computes a compressed representation of the viewpoint-varying color for each vertex in a triangle mesh [128]. This global approach has become much more popular in the era of deep learning, with scene representations designed to be optimized via gradient descent through differentiable rendering pipelines. In this context, deep networks may be used to help reconstruct [73] or render [109] the scene, or even directly serve as the underlying scene representation [82, 86, 110].

1.2.2 Surface vs. volume rendering in differentiable pipelines

Within the last five years, automatic differentiation frameworks such as Tensorflow [76] and PyTorch [94] have enabled an explosion of work on *differentiable* rendering systems in which it is possible to take the gradient of the rendered output image with respect to various scene representation parameters, such as geometry or surface colors. Differentiability is important because it allows these rendering methods to be directly integrated into end-to-end deep learning pipelines that are supervised via gradient descent on a rendered image loss.

In the context of differentiable rendering, we can make a distinction between *surface* and *volume* rendering methods (Figure 1.2). A surface rendering method assumes that 3D space is empty except for an enumerated set of opaque primitives (triangles, points, depth maps, etc.). A ray either does or does not intersect any given primitive; the notion of "visibility" is discrete. In order to produce a differentiable surface renderer, one must appeal to the fact that each camera pixel actually represents an *integral* over a bundle of rays, weighted by a filtering function [59], or approximate the gradients by softening the ray-primitive



(a) **Surface rendering.** The scene is represented by a list of opaque primitive objects (e.g., a triangle mesh). A camera ray traced into the scene only returns the color of the first primitive it intersects.

(b) **Volume rendering.** Scene content can exist at any point in space with some "density" value. A camera ray accumulates color in a continuous and differentiable way as it traverses the scene.

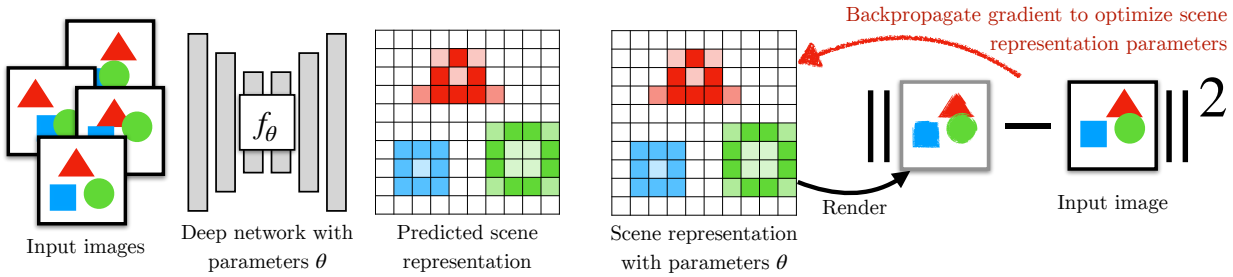
Figure 1.2: Surface vs. volume rendering.

intersection term to be continuous rather than binary [18, 71]. Additionally, optimizing a surface-based representation using gradients requires "physically" moving primitive objects between their initial and final locations—imagine squeezing and stretching a mesh of points connected by springs to contort it into a desired shape. It is also not possible to differentiably modify the number or connective topology of surface primitives in the scene.

In contrast, a volume rendering method assumes that 3D space is densely occupied by a "participating medium" which has some statistical chance of interacting with a traced ray at any point, yielding a continuous, probabilistic notion of visibility. Volume rendering is naturally differentiable and simple to implement in any array-based deep learning framework [73, 121, 140]. Each point in space can modify its own visibility independently of all other points—imagine smoke clouds appearing and disappearing to form object shapes. However, encoding a scene made up of hard surfaces by using a discrete 3D voxel grid is extremely inefficient, since most storage will be devoted to representing empty space. In traditional computer graphics this is not an issue, since an acceleration structure such as an octree can make storing and tracing rays through a mostly-empty volume just as efficient as using a surface representation like a triangle mesh [20]. However, adding such an acceleration structure on top of a 3D volume grid requires making binary decisions about which regions are empty and thus cannot be used in the context of differentiable rendering.

1.2.3 Feed-forward prediction vs. direct optimization

Typical deep learning approaches follow a template of feed-forward prediction. For each new problem instance, some input data is fed into a deep network, which produces an



(a) **Feed-forward prediction.** A small number of input images are fed into a deep network that predicts a corresponding scene representation. The network must be trained over many example input images of different scenes (typically supervised by a loss comparing a rendered output to another held-out image of the same scene).

(b) **Direct optimization.** The scene representation parameters are directly optimized using all input images of a single scene (again, by taking gradient steps on a differentiable rendering loss). No external dataset is needed; however, the optimization process is likely to be much slower than a feed-forward prediction.

Figure 1.3: Feed-forward prediction vs. direct optimization.

output prediction. Given a training dataset consisting of example pairs of input data and corresponding labels, the network is optimized by gradient descent over a loss that encourages its output to match these ground truth labels. In the context of view synthesis, the input data is a set of images of a scene and their camera poses, and the label is usually another held-out image of the same scene. The network outputs parameters for some scene representation, which is used to render an image that is compared to the held-out ground truth by some loss function. If the rendering process is differentiable, the chain rule can be used to propagate gradients back through the whole pipeline, from the loss to the network's parameters.

Feed-forward view synthesis methods derives their power from training over a large dataset of examples, extracting priors that allow them to generalize to arbitrary new scenes. Though the training phase may take multiple days, at inference time the network can usually be applied in a matter of seconds or minutes. However, memory and algorithmic constraints restrict the number of input images these networks can process (typically 2-5 input images [83, 140], or up to 12 in a carefully engineered implementation [28]). Additionally, generating training data requires either creating and rendering images of a large library of high quality synthetic 3D scenes or taking thousands of real-world images by hand.

Directly optimizing scene representation parameters requires no external training data and can result in the highest quality global reconstruction when many input images are available for each scene (tens to hundreds). In this context, we use gradient descent to optimize the scene representation parameters for a single scene such that it best reproduces all provided input images according to a differentiable rendering loss. This dissertation will present *neural radiance fields*, the first scene representation capable of producing photorealistic results at high resolutions using direct optimization.

1.3 Dissertation overview

In this dissertation, we describe two contrasting view synthesis algorithms. The first builds directly on prior learning-based methods, working in a local interpolation framework and using a discrete voxel grid scene representation. The second approaches the problem in a new way, demonstrating for the first time that it is possible to achieve photorealistic results using a memory-efficient neural scene representation and without requiring any external training dataset. Developing this scene representation requires a specific insight into the workings of fully-connected neural networks, which is presented between the two view synthesis methods.

- **Chapter 2** describes Local Light Field Fusion (LLFF), a prototypical feed-forward deep learning method trained over a large dataset.[1] Given a new scene at test time, LLFF builds a locally valid reconstruction for each input image, then interpolates between nearby reconstructions to generate views from new camera poses. We show empirically that this method effectively lowers the required view sampling rate for light field interpolation by a factor of 64^2. The local reconstructions can be generated within minutes for each new scene, and novel views can be rendered at real-time framerates.

  However, each local reconstruction must be predicted and stored as a dense 3D voxel grid, resulting in poor memory scaling as the number of inputs or image resolution increases. Additionally, even a $64^2\times$ reduction in sampling is not sufficient to make very wide baseline captures tractable (e.g., a 360° capture of one object from all directions would still require thousands of views). These limitations motivate us to seek a memory-efficient global scene representation that can be directly optimized via gradient descent to reproduce many input views captured over a wide baseline.

- **Chapter 3** explains how neural networks can act as this memory-efficient representation.[2] We can use a fully-connected neural network as a substitute for a multidimensional array by training the network to map from array coordinate inputs to array value outputs. However, standard networks have an inherent *spectral bias* toward learning low-frequency components of a signal exponentially faster than higher-frequency components, making it infeasible for them to represent complex natural signals such as 2D images or 3D shapes.

  We find that it is possible to overcome this spectral bias passing the input coordinates through many randomly sampled Fourier basis functions before putting them into the network, leveraging recent theoretical work connecting neural networks and kernel regression [45, 5] to show that certain parameters of this *Fourier feature mapping* allow users to directly control the rate at which the network learns different frequency bands of a signal. This insight transforms fully-connected networks into a highly efficient alternative to discrete arrays for representing complicated signals in three dimensions and above.

[1] Based on work originally published in SIGGRAPH 2019 [83].
[2] Based on work originally published in NeurIPS 2020 [119].

- **Chapter 4** presents neural radiance fields (NeRF), a continuous 5D scene representation encoded in a fully-connected neural network that can be directly optimized (using images of a single input scene) to render high-quality novel views.[3] This neural scene representation is trivially differentiable, simple to implement, and over one hundred times more compact than an equivalent dense grid, enabling the first view synthesis algorithm capable of using a globally reconstructed model to produce photorealistic high resolution results over a wide camera baseline.
- **Chapter 5** concludes and discusses potential avenues for future work.

[3]Based on work originally published in ECCV 2020 [82].

Chapter 2

View Synthesis as Local Interpolation: Local Light Field Fusion

This chapter presents a view synthesis algorithm based on light field interpolation in a plenoptic sampling framework. We use a feed-forward neural network to promote each input image into a "local light field" representation capable of rendering nearby views (moving up to a certain pixel disparity away from the image's camera position). This allows us to reduce the required input view sampling rate well below the Nyquist limit for standard light field interpolation. This reudction is critical since Nyquist rate sampling is intractable for scenes with content at nearby distances, as the Nyquist limit increases linearly with the reciprocal of the closest scene depth. For example, for a scene with a subject at a depth of 0.5 meters captured by a mobile phone camera with a typical 64° field of view and rendered at 1 megapixel resolution, the required sampling rate is an intractable 2.5 million images per square meter. Hence, we must move towards view synthesis algorithms that leverage scene priors to effectively predict the missing views.

Our view synthesis approach is grounded within a plenoptic sampling framework and can precisely prescribe how densely a user must capture a given scene for reliable rendering performance. Our method is conceptually simple and consists of two main stages. We first use a deep network to promote each source view to a layered representation of the scene that can render a limited range of views, advancing recent work on the multiplane image (MPI) representation [140]. We then synthesize novel views by blending renderings from adjacent layered representations.

Our theoretical analysis shows that the number of input views required by our method decreases quadratically with the number of planes we predict for each layered scene representation, up to limits set by the camera field of view. We empirically validate our analysis and apply it in practice to render novel views with the same perceptual quality as Nyquist view sampling while using up to $64^2 \approx 4000\times$ fewer images.

It is impossible to break the Nyquist limit with full generality, but we show that it is possible to achieve Nyquist level performance with greatly reduced view sampling by specializing to the subset of natural scenes. This capability is primarily due to our deep

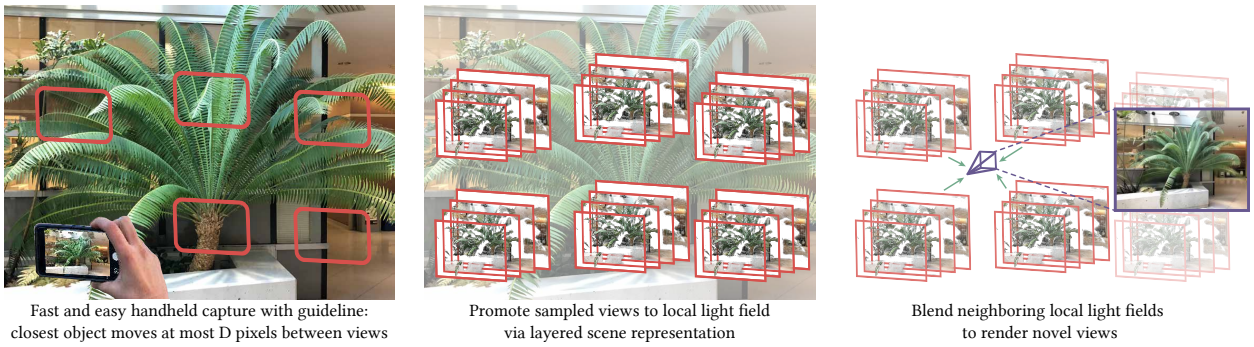


Figure 2.1: We present a simple and reliable method for view synthesis from a set of input images captured by a handheld camera in an irregular grid pattern. We theoretically and empirically demonstrate that our method enjoys a prescriptive sampling rate that requires 4000× fewer input views than Nyquist for high-fidelity view synthesis of natural scenes. Specifically, we show that this rate can be interpreted as a requirement on the pixel-space disparity of the closest object to the camera between captured views (Section 2.2). After capture, we expand all sampled views into layered representations that can render high-quality local light fields. We then blend together renderings from adjacent local light fields to synthesize dense paths of new views (Section 2.3). Our rendering consists of simple and fast computations (homography warping and alpha compositing) that can generate new views in real-time.

learning pipeline, which is trained on renderings of natural scenes to estimate high quality layered scene representations that produce locally consistent light fields.

In summary, our key contributions are:

1. An extension of plenoptic sampling theory that directly specifies how users should sample input images for reliable high quality view synthesis with our method.

2. A practical and robust solution for capturing and rendering complex real world scenes for virtual exploration.

3. A demonstration that carefully crafted deep learning pipelines using local layered scene representations achieve state-of-the-art view synthesis results.

We extensively validate our derived prescriptive view sampling requirements and demonstrate that our algorithm quantitatively outperforms traditional light field reconstruction methods as well as state-of-the-art view interpolation algorithms across a range of sub-Nyquist view sampling rates. We highlight the practicality of our method by developing an augmented reality app that implements our derived sampling guidelines to help users capture input images that produce reliably high-quality renderings with our algorithm. Additionally, we develop mobile and desktop viewer apps that render novel views from our predicted lay-

ered representations in real-time. Finally, we qualitatively demonstrate that our algorithm reliably produces state-of-the-art results across a diverse set of complex real-world scenes.

2.1 Related Work

Image-based rendering (IBR) is the fundamental computer graphics problem of rendering novel views of objects and scenes from sampled views. We find that it is useful to categorize IBR algorithms by the extent to which they use explicit scene geometry, as done by Shum and Kang [107].

2.1.1 Plenoptic Sampling and Reconstruction

Light field rendering [64] eschews any geometric reasoning and simply samples images on a regular grid so that new views can be rendered as slices of the sampled light field. Lumigraph rendering [35] showed that using approximate scene geometry can ameliorate artifacts due to undersampled or irregularly sampled views.

The plenoptic sampling framework [12] analyzes light field rendering using signal processing techniques and shows that the Nyquist view sampling rate for light fields depends on the minimum and maximum scene depths. Furthermore, they discuss how the Nyquist view sampling rate can be lowered with more knowledge of scene geometry. Zhang and Chen [136] extend this analysis to show how non-Lambertian and occlusion effects increase the spectral support of a light field, and also propose more general view sampling lattice patterns.

Rendering algorithms based on plenoptic sampling enjoy the significant benefit of prescriptive sampling; given a new scene, it is easy to compute the required view sampling density to enable high-quality renderings. Many modern light field acquisition systems have been designed based on these principles, including large-scale camera systems [127, 91] and a mobile phone app [22].

We posit that prescriptive sampling is necessary for practical and useful IBR algorithms, and we extend prior theory on plenoptic sampling to show that our deep-learning-based view synthesis strategy can significantly decrease the dense sampling requirements of traditional light field rendering. Our novel view synthesis pipeline can also be used in future light field acquisition hardware systems to reduce the number of required cameras.

2.1.2 Geometry-Based View Synthesis

Many IBR algorithms attempt to leverage explicit scene geometry to synthesize new views from arbitrary unstructured sets of input views. These approaches can be meaningfully categorized as either using global or local geometry.

Techniques that use global geometry generally compute a single global mesh from a set of unstructured input images. Simply texture mapping this global mesh can be effective for

constrained situations such as panoramic viewing with mostly rotational and little translational viewer movement [37, 38], but this strategy can only simulate Lambertian materials. Surface light fields [128] are able to render convincing view-dependent effects, but they require accurate geometry from dense range scans and hundreds of captured images to sample the outgoing radiance at points on an object's surface.

Many free-viewpoint IBR algorithms are based upon a strategy of locally texture mapping a global mesh. The influential view-dependent texture mapping algorithm [23] proposed an approach to render novel views by blending nearby captured views that have been reprojected using a global mesh. Work on Unstructured Lumigraph Rendering [11] focused on computing per-pixel blending weights for reprojected images and proposed a heuristic algorithm that satisfied key properties for high-quality rendering. Unfortunately, it is very difficult to estimate high-quality meshes whose geometric boundaries align well with edges in images, and IBR algorithms based on global geometry typically suffer from significant artifacts. State-of-the-art algorithms [40, 39] attempt to remedy this shortcoming with complicated pipelines that involve both global mesh and local depth map estimation. However, it is difficult to precisely define view sampling requirements for robust mesh estimation, and the mesh estimation procedure typically takes multiple hours, making this strategy impractical for casual content capture scenarios.

IBR algorithms that use local geometry [15, 17, 55, 79, 90] avoid difficult and expensive global mesh estimation. Instead, they typically compute detailed local geometry for each input image and render novel views by reprojecting and blending nearby input images. This strategy has also been extended to simulate non-Lambertian reflectance by using a second depth layer [108]. The state-of-the-art Soft3D algorithm [95] blends between reprojected local layered representations to render novel views, which is conceptually similar to our strategy. However, Soft3D computes each local layered representation by aggregating heuristic measures of depth uncertainty over a large neighborhood of views. We instead train a deep learning pipeline end-to-end to optimize novel view quality by predicting each of our local layered representations from a much smaller neighborhood of views. Furthermore, we directly pose our algorithm within a plenoptic sampling framework, and our analysis directly applies to the Soft3D algorithm as well. We demonstrate that the high quality of our deep learning predicted local scene representations allows us to synthesize superior renderings without requiring the aggregation of geometry estimates over large view neighborhoods, as done in Soft3D. This is especially advantageous for rendering non-Lambertian effects because the apparent depth of specularities generally varies with the observation viewpoint, so smoothing the estimated geometry over large viewpoint neighborhoods prevents accurate rendering of these effects.

Other IBR algorithms [2] have attempted to be more robust to incorrect camera poses or scene motion by interpolating views using more general 2D optical flow instead of 1D depth. Local pixel shifts are also encoded in the phase information, and algorithms have exploited this to extrapolate views from micro-baseline stereo pairs [26, 52, 138] without explicit flow computation. However, these methods require extremely close input views and are not suited for large baseline view interpolation.

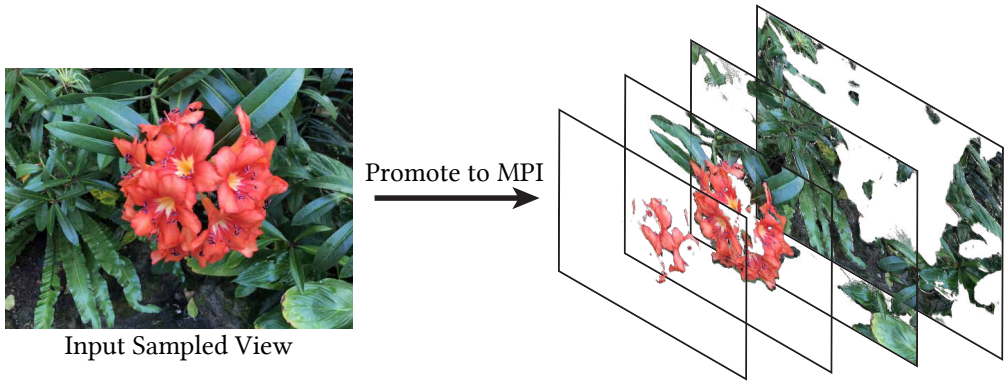


Figure 2.2: We promote each input view sample to an MPI scene representation [140], consisting of D RGBα planes at regularly sampled disparities within the input view's camera frustum. Each MPI can render continuously-valued novel views within a local neighborhood by alpha compositing color along rays into the novel view's camera.

2.1.3 Deep Learning for View Synthesis

Other recent methods have trained deep learning pipelines end-to-end for view synthesis. This includes recent angular superresolution methods [129, 134] that interpolate dense views within a light field camera's aperture but cannot handle sparser input view sampling since they do not model scene geometry. The DeepStereo algorithm [29], deep learning based light field camera view interpolation [48], and single view local light field synthesis [114] each use a deep network to predict depth separately for every novel view. However, predicting local geometry separately for each view results in inconsistent renderings across smoothly-varying viewpoints.

Finally, Zhou *et al.* [140] introduce a deep learning pipeline to predict an MPI from a narrow baseline stereo pair for the task of stereo magnification. As opposed to previous deep learning strategies for view synthesis, this approach enforces consistency by using the same predicted scene representation to render all novel views. We adopt MPIs as our local light field representation (Figure 2.2) and introduce specific technical improvements to enable larger-baseline view interpolation from many input views, in contrast to local view extrapolation from a stereo pair using a single MPI. We predict multiple MPIs, one for each input view, and train our system end-to-end through a blending procedure to optimize the resulting MPIs to be used in concert for rendering output views. We propose a 3D convolutional neural network (CNN) architecture that dynamically adjusts the number of depth planes based on the input view sampling rate, rather than a 2D CNN with a fixed number of output planes. Additionally, we show that state-of-the-art performance requires only an easily-generated synthetic dataset and a small real fine-tuning dataset, rather than a large real dataset. This allows us to generate training data captured on 2D irregular grids similar to handheld view sampling patterns, while the YouTube dataset in Zhou *et al.* [140] is restricted to 1D camera paths.

Table 2.1: Reference for symbols used in Section 2.2.

Symbol	Definition
D	Number of depth planes
W	Camera image width (pixels)
f	Camera focal length (meters)
Δ_x	Pixel size (meters)
Δ_u	Baseline between cameras (meters)
K_x	Highest spatial frequency in sampled light field
B_x	Highest spatial frequency in continuous light field
$z_{\min}$	Closest scene depth (meters)
$z_{\max}$	Farthest scene depth (meters)
$d_{\max}$	Maximum disparity between views (pixels)

2.2 Theoretical Plenoptic Sampling Analysis

The overall strategy of our method is to use a deep learning pipeline to promote each sampled view to a layered scene representation with D depth layers, and render novel views by blending between renderings from neighboring scene representations. In this section, we show that the full set of scene representations predicted by our deep network can be interpreted as a specific form of light field sampling. We extend prior work on plenoptic sampling to show that our strategy can theoretically reduce the number of required sampled views by a factor of D^2 compared to the number required by traditional Nyquist view sampling. Section 2.5.1 empirically shows that we are able to take advantage of this bound to reduce the number of required views by up to $64^2 \approx 4000\times$.

In the following analysis, we consider a "flatland" light field with a single spatial dimension x and view dimension u for notational clarity, but note that all findings apply to general light fields with two spatial and two view dimensions.

2.2.1 Nyquist Rate View Sampling

Initial work on plenoptic sampling [12] derived that the Fourier support of a light field, ignoring occlusion and non-Lambertian effects, lies within a double-wedge shape whose bounds are set by the minimum and maximum scene depths $z_{\min}$ and $z_{\max}$, as visualized in Figure 2.3. Zhang and Chen [136] showed that occlusions expand the light field's Fourier support because an occluder convolves the spectrum of the light field due to farther scene content with a kernel that lies on the line corresponding to the occluder's depth. The light field's Fourier support considering occlusions is limited by the effect of the closest occluder convolving the line corresponding to the furthest scene content, resulting in the parallelogram shape illustrated in Figure 2.4a, which can only be packed half as densely as the double-wedge. The

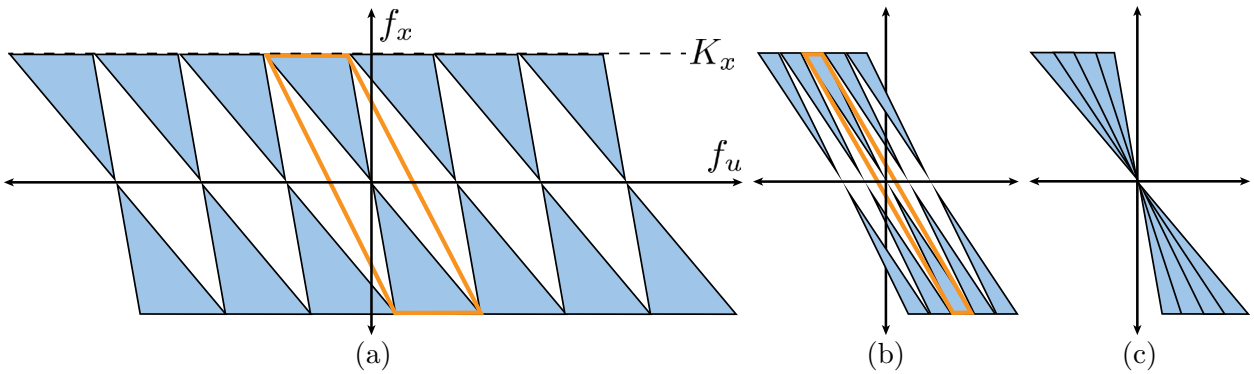


Figure 2.3: Traditional plenoptic sampling without occlusions, as derived in [12]. (a) The Fourier support of a light field without occlusions lies within a double-wedge, shown in blue. Nyquist rate view sampling is set by the double-wedge width, which is determined by the minimum and maximum scene depths $[z_{\min}, z_{\max}]$ and the maximum spatial frequency K_x. The ideal reconstruction filter is shown in orange. (b) Splitting the light field into D non-overlapping layers with equal disparity width decreases the Nyquist rate by a factor of D. (c) Without occlusions, the full light field spectrum is the sum of the spectra from each layer.

required maximum camera sampling interval Δ_u for a light field with occlusions is:

$$\Delta_u \leq \frac{1}{2K_x f \left(1/z_{\min} - 1/z_{\max}\right)}, \tag{2.1}$$

where $K_x = \min\left(B_x, \frac{1}{2\Delta_x}\right)$ is the highest spatial frequency represented in the sampled light field, as determined by the highest spatial frequency in the continuous light field B_x and the camera's spatial resolution Δ_x.

2.2.2 MPI Scene Representation and Rendering

The MPI scene representation [140] consists of a set of fronto-parallel RGBα planes, evenly sampled in disparity within a reference camera's view frustum (see Figure 2.2). We can render novel views from an MPI at continuously-valued camera poses within a local neighborhood by alpha compositing the color along rays into the novel view camera using the "over" operator [96]. This rendering procedure is equivalent to reprojecting each MPI plane onto the sensor plane of the novel view camera and alpha compositing the MPI planes from back to front, as observed in early work on volume rendering [58]. An MPI can be considered as an encoding of a local light field, similar to layered light field displays [125, 126].

2.2.3 View Sampling Rate Reduction

Plenoptic sampling theory [12] additionally shows that decomposing a scene into D depth ranges and separately sampling the light field within each range allows the camera sampling

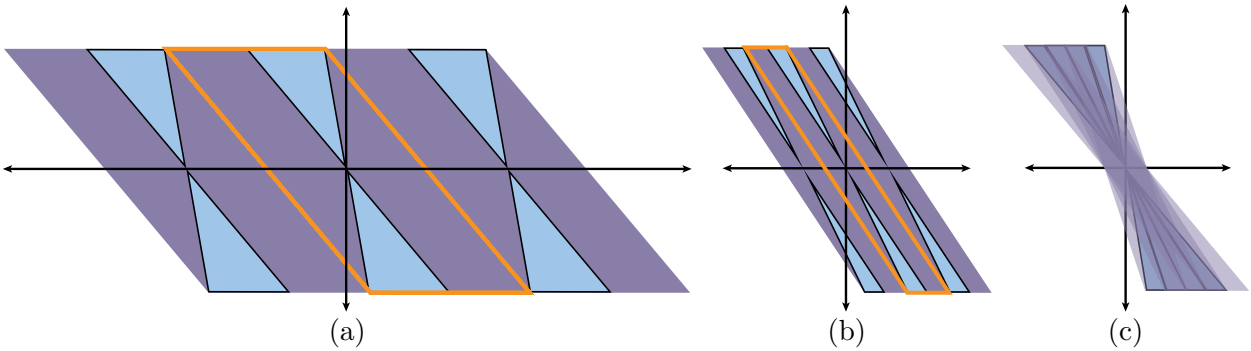


Figure 2.4: We extend traditional plenoptic sampling to consider occlusions when reconstructing a continuous light field from MPIs. (a) Considering occlusions expands the Fourier support to a parallelogram (the Fourier support without occlusions is shown in blue and occlusions expand the Fourier support to additionally include the purple region) and doubles the Nyquist view sampling rate. (b) As in the no-occlusions case, separately reconstructing the light field for D layers decreases the Nyquist rate by a factor of D. (c) With occlusions, the full light field spectrum cannot be reconstructed by summing the individual layer spectra because the union of their supports is smaller than the support of the full light field spectrum (a). Instead, we compute the full light field by alpha compositing the individual light field layers from back to front in the primal domain.

interval to be increased by a factor of D. This is because the spectrum of the light field emitted by scene content within each depth range lies within a tighter double-wedge that can be packed D times more tightly than the full scene's double-wedge spectrum. Therefore, a tighter reconstruction filter with a different shear can be used for each depth range, as illustrated in Figure 2.3b. The reconstructed light field, ignoring occlusion effects, is simply the sum of the reconstructions of all layers, as shown in Figure 2.3c.

However, it is not straightforward to extend this analysis to handle occlusions, because the union of the Fourier spectra for all depth ranges has a smaller support than the original light field with occlusions, as visualized in Figure 2.4c. Instead, we observe that reconstructing a full scene light field from these depth range light fields while respecting occlusions would be much easier given corresponding per-view opacities, or shield fields [60], for each layer. We could then easily alpha composite the depth range light fields from back to front to compute the full scene light field.

Each alpha compositing step increases the Fourier support by convolving the previously-accumulated light field's spectrum with the spectrum of the occluding depth layer. As is well known in signal processing, the convolution of two spectra has a Fourier bandwidth equal to the sum of the original spectra's bandwidths. Figure 2.4b illustrates that the width of the Fourier support parallelogram for each depth range light field, considering occlusions, is:

$$2K_x f \left(1/z_{\min} - 1/z_{\max}\right)/D, \tag{2.2}$$

so the resulting reconstructed light field of the full scene will enjoy the full Fourier support

width.

We apply this analysis to our algorithm by interpreting the predicted MPI layers at each camera sampling location as view samples of scene content within non-overlapping depth ranges, and noting that applying the optimal reconstruction filter [12] for each depth range is equivalent to reprojecting and then blending pre-multiplied RGBα planes from neighboring MPIs. Our MPI layers differ from layered renderings considered in traditional plenoptic sampling because we predict opacities in addition to color for each layer, which allows us to correctly respect occlusions while compositing the depth layer light fields.

In summary, we extend the layered plenoptic sampling framework to correctly handle occlusions by taking advantage of our predicted opacities, and show that this still allows us to increase the required camera sampling interval by a factor of D:

$$\Delta_u \leq \frac{D}{2K_x f\left(1/z_{\min} - 1/z_{\max}\right)}. \tag{2.3}$$

Our framework further differs from classic layered plenoptic sampling in that each MPI is sampled within a reference camera view frustum with a finite field of view, instead of the infinite field of view assumed in prior analyses [12, 136]. In order for the MPI prediction procedure to succeed, every point within the scene's bounding volume should fall within the frustums of at least two neighboring sampled views. The required camera sampling interval Δ_u is then additionally bounded by:

$$\Delta_u \leq \frac{W\Delta_x z_{\min}}{2f} \tag{2.4}$$

where W is the image width in pixels of each sampled view. The overall camera sampling interval must satisfy both constraints:

$$\Delta_u \leq \min\left(\frac{D}{2K_x f\left(1/z_{\min} - 1/z_{\max}\right)}, \frac{W\Delta_x z_{\min}}{2f}\right). \tag{2.5}$$

2.2.4 Image Space Interpretation of View Sampling

It is useful to interpret the required camera sampling rate in terms of the maximum pixel disparity $d_{\max}$ of any scene point between adjacent input views. If we set $z_{\max} = \infty$ to allow scenes with content up to an infinite depth and additionally set $K_x = 1/2\Delta_x$ to allow spatial frequencies up to the maximum representable frequency:

$$\frac{\Delta_u f}{\Delta_x z_{\min}} = d_{\max} \leq \min\left(D, \frac{W}{2}\right). \tag{2.6}$$

Simply put, the maximum disparity of the closest scene point between adjacent views must be less than $\min(D, W/2)$ pixels. When $D = 1$, this inequality reduces to the Nyquist bound: a maximum of 1 pixel of disparity between views.

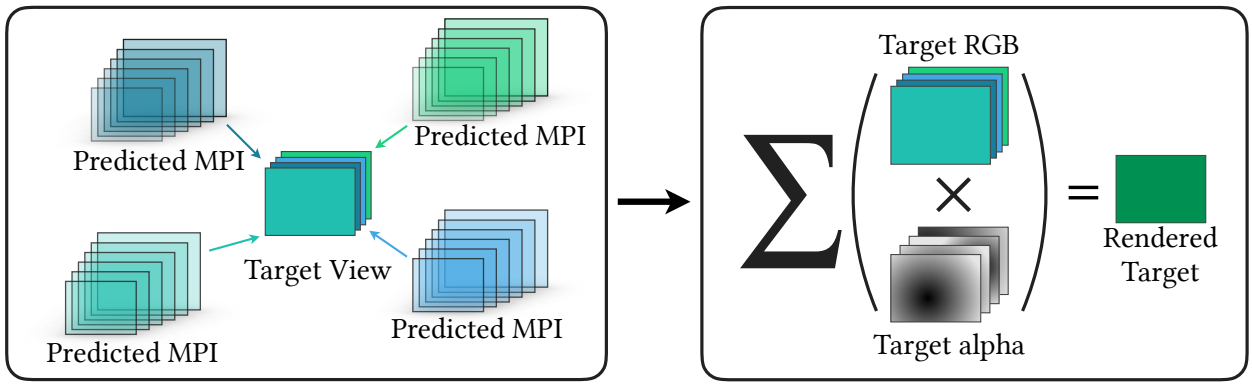


Figure 2.5: We render novel views as a weighted combination of renderings from neighboring MPIs, modulated by the corresponding accumulated alphas.

In summary, promoting each view sample to an MPI scene representation with D depth layers allows us to decrease the required view sampling rate by a factor of D, up to the required field of view overlap for stereo geometry estimation. Light fields for real 3D scenes must be sampled in two viewing directions, so this benefit is compounded into a sampling reduction of D^2. Section 2.5.1 empirically validates that our algorithm's performance matches this theoretical analysis. Section 2.6.1 describes how we apply the above theory along with the empirical performance of our deep learning pipeline to prescribe practical sampling guidelines for users.

2.3 Practical View Synthesis by Blending Local Light Fields

We present a practical and robust method for synthesizing new views from a set of input images and their camera poses. Our method first uses a CNN to promote each captured input image to an MPI, then reconstructs novel views by blending renderings from nearby MPIs. Figures 2.1 and 2.5 visualize this pipeline. We discuss the practical image capture process enabled by our method in Section 2.6.

2.3.1 MPI Prediction for Local Light Field Expansion

The first step in our pipeline is expanding each sampled view to a local light field using an MPI scene representation. Our MPI prediction pipeline takes five views as input: the reference view to be expanded and its four nearest neighbors in 3D space. Each image is reprojected to D depth planes, sampled linearly in disparity within the reference view frustum, to form 5 plane sweep volumes (PSVs) of size $H \times W \times D \times 3$.

Table 2.2: Our network architecture. **k** is the kernel size, **s** the stride, **d** the kernel dilation, **chns** the number of input and output channels for each layer, **in** and **out** are the accumulated stride for the input and output of each layer, **input** denotes the input of each layer with + meaning concatenation, and layers starting with "nnup" perform 2× nearest neighbor upsampling. All layers except the last are followed by a ReLU nonlinearity and layer normalization [62]. The final layer outputs 5 channels. One channel is passed through a sigmoid to generate the output MPI's alpha channel. The other four (along with an all-zero channel) are passed through a softmax to get five blending weights for each voxel which are used to generate the output MPI's color channels, as described in Section 2.3.1. Restricting one of the softmax inputs to always be zero makes the function one-to-one rather than many-to-one.

Layer	**k**	**s**	**d**	**chns**	**in**	**out**	**input**
conv1_1	3	1	1	15/8	1	1	PSVs
conv1_2	3	2	1	8/16	1	2	conv1_1
conv2_1	3	1	1	16/16	2	2	conv1_2
conv2_2	3	2	1	16/32	2	4	conv2_1
conv3_1	3	1	1	32/32	4	4	conv2_2
conv3_2	3	1	1	32/32	4	4	conv3_1
conv3_3	3	2	1	32/64	4	8	conv3_2
conv4_1	3	1	2	64/64	8	8	conv3_3
conv4_2	3	1	2	64/64	8	8	conv4_1
conv4_3	3	1	2	64/64	8	8	conv4_2
nnup5				128/128	8	4	conv4_3 + conv3_3
conv5_1	3	1	1	128/32	4	4	nnup5
conv5_2	3	1	1	32/32	4	4	conv5_1
conv5_3	3	1	1	32/32	4	4	conv5_2
nnup6				64/64	4	2	conv5_3 + conv2_2
conv6_1	3	1	1	64/16	2	2	nnup6
conv6_2	3	1	1	16/16	2	2	conv6_1
nnup7				32/32	2	1	conv6_2 + conv1_2
conv7_1	3	1	1	32/8	1	1	nnup7
conv7_2	3	1	1	8/8	1	1	conv7_1
conv7_3	3	1	1	8/6	1	1	conv7_2

Our 3D CNN takes these 5 PSVs as input, concatenated along the channel dimension. This CNN outputs an opacity α for each MPI coordinate (x, y, d) as well as a set of 5 color selection weights that sum to 1 at each MPI coordinate. These weights parameterize the RGB values in the output MPI as a weighted combination of the input PSVs. Intuitively, each predicted MPI softly "selects" its color values at each MPI coordinate from the pixel colors at that coordinate in each of the input PSVs. We specifically use this RGB parameterization instead of the foreground+background parameterization proposed by Zhou *et al.* [140] because their method does not allow an MPI to directly incorporate content occluded from the reference view but visible in other input views.

Furthermore, we enhance the MPI prediction CNN architecture from the original version to use 3D convolutional layers instead of the original 2D convolutional layers so that our

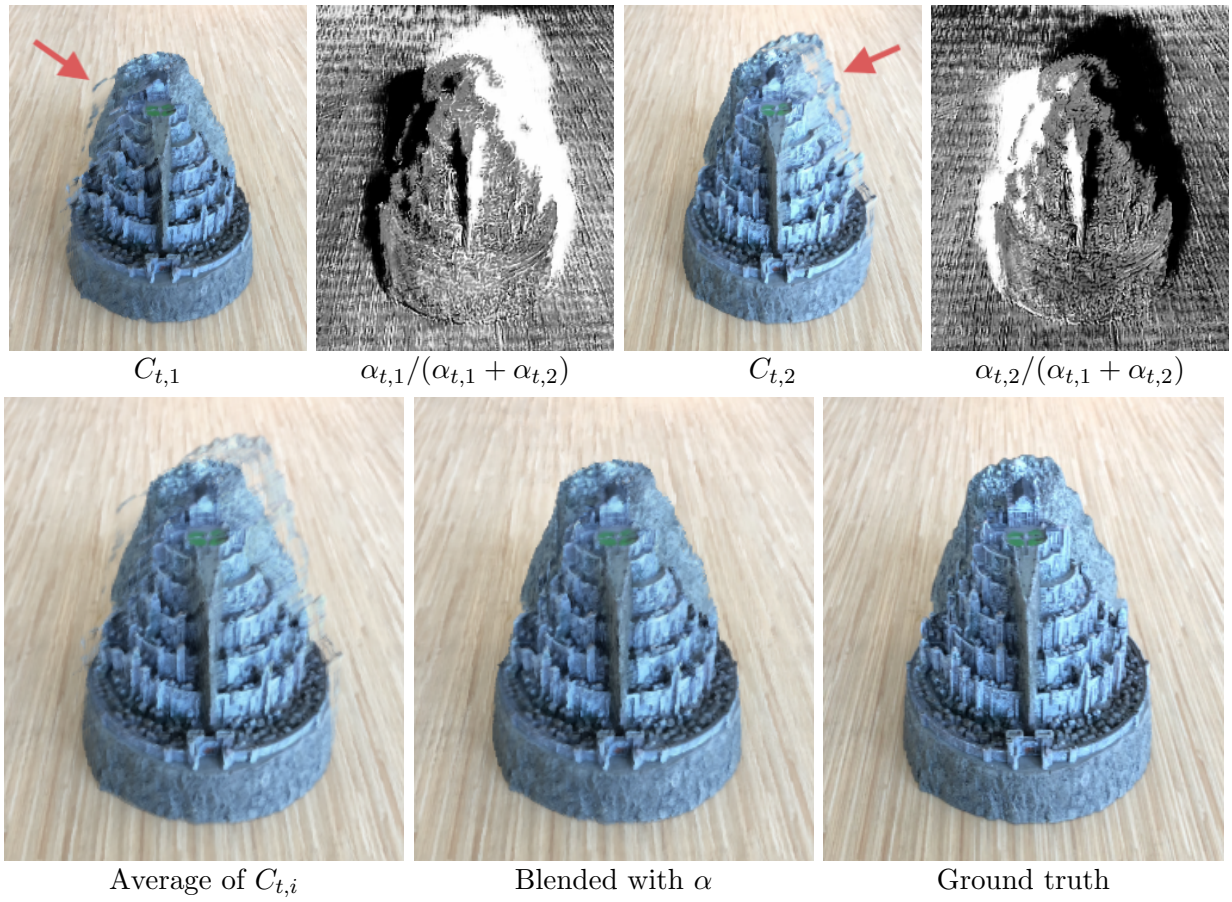


Figure 2.6: An example illustrating the benefits of using accumulated alpha to blend MPI renderings. We render two MPIs at the same new camera pose. In the top row, we display the RGB outputs $C_{t,i}$ from each MPI as well as the accumulated alphas $\alpha_{t,i}$, normalized so that they sum to one at each pixel. In the bottom row, we see that a simple average of the RGB images $C_{t,i}$ retains the stretching artifacts from both MPI renderings, whereas the alpha weighted blending combines only the non-occluded pixels from each input to produce a clean output C_t.

architecture is fully convolutional along the height, width, and depth dimensions. This enables us to predict MPIs with a variable number of planes D so that we can jointly choose the view and disparity sampling densities to satisfy Equation 2.6. Table 2.3 validates the benefit of being able to change the number of MPI planes to correctly match our derived sampling requirements, enabled by our use of 3D convolutions. Our full network architecture can be found in Table 2.2.

2.3.2 Continuous View Reconstruction by Blending

As discussed in Section 2.2, we reconstruct interpolated views as a weighted combination of renderings from multiple nearby MPIs. This effectively combines our local light field approximations into a light field with a near plane spanning the extent of the captured input views and a far plane determined by the field-of-view of the input views. As in standard light field rendering, this allows for a new view path with unconstrained 3D translation and rotation within the range of views made up of rays in the light field.

One important detail in our rendering process is that we consider the accumulated alpha values from each MPI rendering when blending. This allows each MPI rendering to "fill in" content that is occluded from other camera views.

Our MPI prediction network uses a set of RGB images C_k along with their camera poses p_k to produce a set of MPIs M_k (one corresponding to each input image). To render a novel view with pose p_t using the predicted MPI M_k, we homography warp each RGBα MPI plane into the frame of reference of the target pose p_t then alpha composite the warped planes together from back to front. This produces an RGB image and an alpha image, which we denote $C_{t,k}$ and $\alpha_{t,k}$ respectively (subscript t, k indicating that the output is rendered at pose p_t using the MPI at pose p_k).

Since a single MPI alone will not necessarily contain all the content visible from the new camera pose due to occlusions and field of view issues, we generate the final RGB output C_t by blending rendered RGB images $C_{t,k}$ from multiple MPIs, as depicted in Figure 2.5. We use scalar blending weights $w_{t,k}$, each modulated by the corresponding accumulated alpha images $\alpha_{t,k}$ and normalized so that the resulting rendered image is fully opaque ($\alpha = 1$):

$$C_t = \frac{\sum_k w_{t,k}\alpha_{t,k}C_{t,k}}{\sum_k w_{t,k}\alpha_{t,k}}. \tag{2.7}$$

For an example where modulating the blending weights by the accumulated alpha values prevents artifacts in C_t, see Figure 2.6. Table 2.3 demonstrates that blending with alpha gives quantitatively superior results over both using a single MPI and blending multiple MPI renderings without using the accumulated alpha.

The blending weights $w_{t,k}$ can be any sufficiently smooth filter. In the case of data sampled on a regular grid, we use bilinear interpolation from the four nearest MPIs rather than the ideal sinc function interpolation for effiency and due to the limited number of sampled views. For irregularly sampled data, we use the five nearest MPIs and take $w_{t,k} \propto \exp\left(-\gamma\ell(p_t, p_k)\right)$. Here $\ell(p_t, p_k)$ is the L^2 distance between the translation vectors of poses p_t and p_k, and the constant γ is defined as $\frac{f}{Dz_{\min}}$ given focal length f, minimum distance to the scene $z_{\min}$, and number of planes D. (Note that the quantity $\frac{f\ell}{z_{\min}}$ represents ℓ converted into units of pixel disparity.)

Our strategy of blending between neighboring MPIs is particularly effective for rendering non-Lambertian effects. For general curved surfaces, the virtual apparent depth of a specularity changes with the viewpoint [116]. As a result, specularities appear as curves in epipolar slices of the light field, while diffuse points appear as lines. Each of our predicted

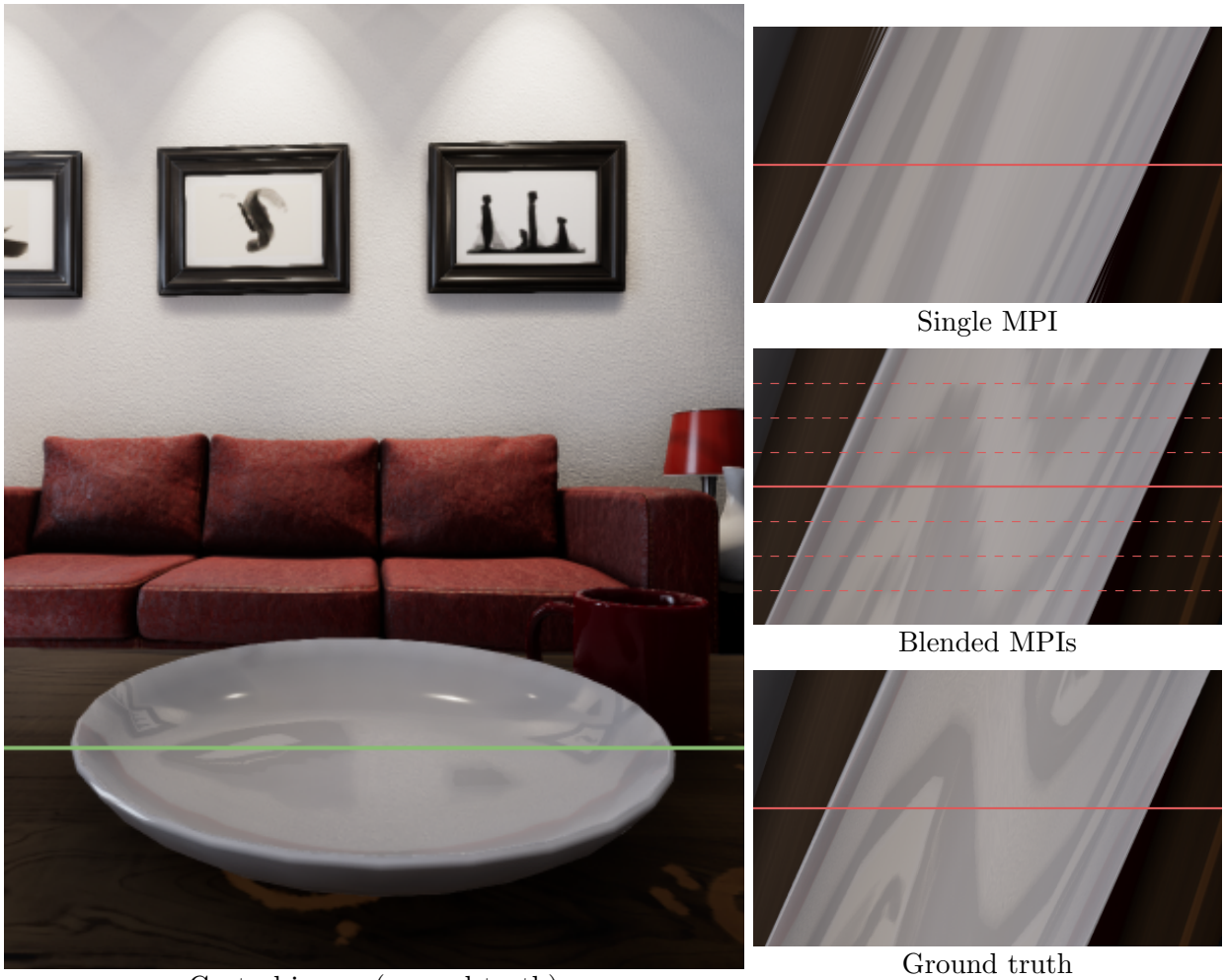


Figure 2.7: We demonstrate that a collection of MPIs can approximate a highly non-Lambertian light field. In this synthetic scene, the curved plate reflects the paintings on the wall, leading to quickly-varying specularities as the camera moves horizontally. This effect can be observed in the ground truth epipolar plot (bottom right). A single MPI (top right) can only place a specular reflection at a single virtual depth, but blending renderings from multiple MPIs (middle right) provides a much better approximation to the true light field. In this example, we blend between MPIs evenly distributed at every 32 pixels of disparity along a horizontal path, indicated by the dashed lines in the epipolar plot.

MPIs can represent a specularity for a local range of views by placing the specularity at a single virtual depth. Figure 2.7 illustrates how our rendering procedure effectively models a specularity's curve in the light field by blending locally linear approximations, as opposed to the limited extrapolation provided by a single MPI.

2.4 Training Our View Synthesis Pipeline

2.4.1 Training Dataset

We train our view synthesis pipeline using both renderings and real images of natural scenes. Using synthetic training data crucially enables us to easily generate a large dataset with input view and scene depth distributions similar to those we expect at test time, while using real data helps us generalize to real-world lighting and reflectance effects as well as small errors in pose estimation.

Our synthetic training set consists of images rendered from the SUNCG [111] and UnrealCV [98] datasets. SUNCG contains 45,000 simplistic house and room environments with texture mapped surfaces and low geometric complexity. UnrealCV contains only a few large scale environments, but they are modeled and rendered with extreme detail, providing geometric complexity, texture variety, and non-Lambertian reflectance effects. We generate views for each synthetic training instance by first randomly sampling a target baseline for the inputs (up to 128 pixels of disparity), then randomly perturbing the camera pose in 3D to approximately match this baseline.

Our real training dataset consists of 24 scenes from our handheld cellphone captures, with 20-30 images each. We use the COLMAP structure from motion [103] implementation to compute poses for our real images.

2.4.2 Training Procedure

For each training step, we sample two sets of 5 views each to use as inputs, and a single held-out target view for supervision. We first use the MPI prediction network to predict two MPIs, one from each set of 5 inputs. Next, we render the target novel view from both MPIs and blend these renderings using the accumulated alpha values, as described in Equation 2.7.

The training loss is simply the image reconstruction loss for the rendered novel view. We follow the original work on MPI prediction [140] and use a VGG network activation perceptual loss as implemented by Chen and Koltun [16], which has been consistently shown to outperform standard image reconstruction losses [44, 137]. We are able to supervise only the final blended rendering because both our fixed rendering and blending functions are differentiable. Learning through this blending step trains our MPI prediction network to leave alpha "holes" in uncertain regions for each MPI, in the expectation that this content will be correctly rendered by another neighboring MPI, as illustrated by Figure 2.6.

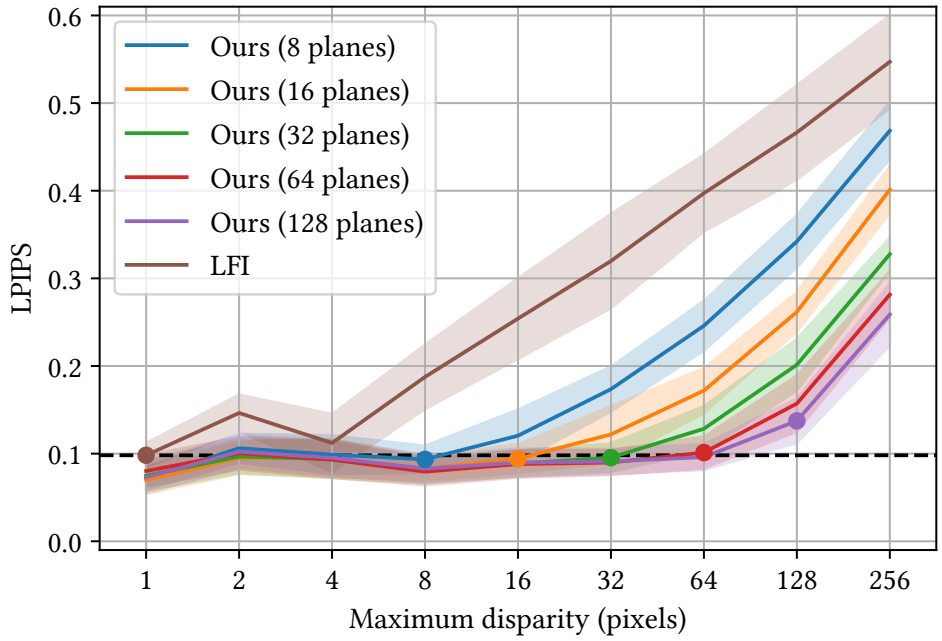


Figure 2.8: We plot the performance of our method (with varying number of planes $D = 8, 16, 32, 64$, and 128) compared to light field interpolation for different input view sampling rates (denoted by maximum scene disparity $d_{\max}$ between adjacent input views). Our method can achieve the same perceptual quality as LFI with Nyquist rate sampling (black dotted line) as long as the number of predicted planes matches or exceeds the undersampling rate, up to an undersampling rate of 128. At $D = 64$, this means we achieve the same quality as LFI with $64^2 \approx 4000\times$ fewer views. We use the LPIPS [137] metric (lower is better) because we primarily value perceptual quality. The colored dots indicate the point on each line where the number of planes equals the maximum scene disparity, where equality is achieved in our sampling bound (Equation 2.6). The shaded region indicates ± 1 standard deviation over all 8 test scenes.

In practice, training through blending is slower than training a single MPI, so we first train the network to render a new view from only one MPI for 500k iterations, then train the full pipeline (blending views from two different MPIs) for 100k iterations. To fine tune the network to process real data, we train on our small real dataset for an additional 10k iterations. We use 320×240 resolution and up to 128 planes for SUNCG training data, and 640×480 resolution and up to 32 planes for UnrealCV training data, due to GPU memory limitations. We implement our full pipeline in Tensorflow [76] and optimize the MPI prediction network parameters using Adam [54] with a learning rate of 2×10^{-4} and a batch size of one. We split the training pipeline across two Nvidia RTX 2080Ti GPUs, using one GPU to generate each MPI.

2.5 Experimental Evaluation

We quantitatively and qualitatively validate our method's prescriptive sampling benefits and ability to render high fidelity novel views of light fields that have been undersampled by up to 4000×, as well as demonstrate that our algorithm outperforms state-of-the-art methods for regular view interpolation. Figure 2.9 showcases these qualitative comparisons on scenes with complex geometry (Fern and T-Rex) and highly non-Lambertian scenes (Air Plants and Pond) that are not handled well by most view synthesis algorithms.

For all quantitative comparisons (Table 2.3), we use a synthetic test set rendered from an UnrealCV [98] environment that was not used to generate any training data. Our test set contains 8 scenes, each rendered at 640×480 resolution and at 8 different view sampling densities such that the maximum disparity between adjacent input views ranges from 1 to 256 pixels (a maximum disparity of 1 pixel between input views corresponds to Nyquist rate view sampling). We restrict our quantitative comparisons to rendered images because a Nyquist rate grid-sampled light field would require at least 384^2 camera views to generate a similar test set, and no such densely-sampled real light field dataset exists to the best of our knowledge. We report quantitative performance using the standard PSNR and SSIM metrics, as well as the state-of-the-art LPIPS [137] perceptual metric, which is based on a weighted combination of neural network activations tuned to match human judgements of image similarity.

Finally, our accompanying video shows results on over 60 additional real-world scenes. These renderings were created completely automatically by a script that takes only the set of captured images and desired output view path as inputs, highlighting the practicality and robustness of our method.

2.5.1 Sampling Theory Validation

Our method is able to render high-quality novel views while significantly decreasing the required input view sampling density compared to standard light field interpolation. Figure 2.8 shows that our method is able to render novel views with Nyquist level perceptual quality with up to $d_{\max} = 64$ pixels of disparity between input view samples, as long as we match the number of planes in each MPI to the maximum pixel disparity between input views. We postulate that our inability to match Nyquist quality from input images with a maximum of 128 pixels of disparity is due to the effect of occlusions. It becomes increasingly likely that any non-foreground scene point will be sampled by fewer input views as the maximum disparity between adjacent views increases. This increases the difficulty of depth estimation and requires the CNN to hallucinate the appearance and depth of occluded points in extreme cases where they are sampled by none of the input views.

Figure 2.8 also shows that once our sampling bound is satisfied, adding additional planes does not increase performance. For example, at 32 pixels of disparity, increasing from 8 to 16 to 32 planes decreases the LPIPS error, but performance stays constant from 32 to 128 planes. This verifies that for scenes up to 64 pixels of disparity, adding additional planes

Table 2.3: We quantitatively show that our method outperforms state-of-the-art baselines and specific ablations of our method, across a wide range of input sampling rates (measured by the maximum pixel disparity $d_{\max}$ between adjacent input views), on a synthetic test set. We display results using the standard PSNR and SSIM metrics (higher is better) as well as the LPIPS perceptual metric [137] (lower is better). The best measurement in each column is bolded. See Sections 2.5.2 and 2.5.3 for details on each comparison.

		Maximum disparity $d_{\max}$ (pixels)											
		16			32			64			128		
	Algorithm	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
Baselines	LFI	26.21	0.7776	0.2541	23.35	0.6982	0.3198	20.60	0.6243	0.3971	18.32	0.5560	0.4665
	ULR	28.17	0.8320	0.1510	26.43	0.7987	0.1820	24.34	0.7679	0.2311	21.24	0.7062	0.3215
	Soft3D	34.48	0.9430	0.1345	32.33	0.9216	0.1795	27.97	0.8588	0.2652	23.11	0.7382	0.3979
	BW Deep	34.18	0.9433	0.1074	34.00	0.9476	0.1128	31.88	0.9192	0.1573	27.59	0.8363	0.2591
Ablations	Single MPI	31.11	0.9482	0.1007	29.38	0.9424	0.1111	26.88	0.9250	0.1363	24.20	0.8734	0.1980
	Avg. MPIs	32.67	0.9560	0.1140	31.34	0.9532	0.1248	29.31	0.9400	0.1423	27.02	0.8999	0.1961
	Ours	**34.57**	**0.9568**	**0.0942**	**34.48**	**0.9569**	**0.0954**	**33.58**	**0.9530**	**0.1012**	**31.96**	**0.9323**	**0.1374**

past the maximum pixel disparity between input views is of limited value, in accordance with our theoretical claim that partitioning a scene with disparity variation of D pixels into D depth ranges is sufficient for continuous reconstruction.

2.5.2 Comparisons to Baseline Methods

We quantitatively (Table 2.3) and qualitatively (Figure 2.9) demonstrate that our algorithm produces superior renderings, particularly for non-Lambertian effects, without the artifacts seen in renderings from competing methods.

We compare our method to state-of-the-art view synthesis techniques as well as view-dependent texture mapping using a global mesh as proxy geometry.

Light Field Interpolation (LFI) [12] This baseline is representative of continuous view reconstruction based on classic signal processing. Following the method of plenoptic sampling [12], we render novel views using a bilinear interpolation reconstruction filter sheared to the mean scene disparity. Figure 2.9 demonstrates that increasing the camera spacing beyond the Nyquist rate results in aliasing and ghosting artifacts when using this method.

Unstructured Lumigraph Rendering (ULR) [11] This baseline is representative of view dependent texture mapping with an estimated global mesh as a geometry proxy. We reconstruct a global mesh from all inputs using the screened Poisson surface reconstruction algorithm [51], and use the heuristic Unstructured Lumigraph blending weights [11] to blend input images after reprojecting them into the novel viewpoint using the global mesh. We use a plane at the mean scene disparity as a proxy geometry to fill in holes in the mesh.

It is particularly difficult to reconstruct a global mesh with geometry edges that are well-aligned with image edges, which causes perceptually jarring artifacts. Furthermore, mesh reconstruction often fails to fill in large portions of the scene, resulting in ghosting artifacts similar to those seen in light field interpolation.

Soft3D [95] Soft3D is a state-of-the-art view synthesis algorithm that is similar to our approach in that it also computes a local layered scene representation for each input view and projects and blends these volumes to render each novel view. However, it uses a hand-crafted pipeline based on classic local stereo and guided filtering to compute each layered representation. Furthermore, since classic stereo methods are unreliable for smooth or repetitive image textures and non-Lambertian materials, Soft3D relies on smoothing their geometry estimation across many (up to 25) input views.

Table 2.3 quantitatively demonstrates that our approach outperforms Soft3D overall. In particular, Soft3D's performance degrades much more rapidly as the input view sampling rate decreases since their aggregation is less effective when fewer input images view the same scene content. Our method is able to predict high-quality geometry in scenarios where Soft3D suffers from noisy and erroneous results of local stereo because we leverage deep learning to learn implicit priors on natural scene geometry. This is in line with recent work that has shown the benefits of deep learning over traditional stereo for depth estimation [53, 44].

Figure 2.9 qualitatively demonstrates that Soft3D generally contains blurred geometry artifacts due to errors in local depth estimation, and that Soft3D's approach fails for rendering non-Lambertian effects because their aggregation procedure blurs the specularity geometry, which changes with the input image viewpoint.

Backwards warping deep network (BW Deep) This baseline subsumes recent deep learning view synthesis techniques [48, 29], which use a CNN to estimate geometry for each novel view and then backwards warp and blend nearby input images to render the target view. We train a network that uses the same 3D CNN architecture as our MPI prediction network but instead outputs a single depth map at the pose of the new target view. We then backwards warp the five input images into the new view using this depth map and use a second 2D CNN to composite these warped input images into a single rendered output view. As shown in Table 2.3, performance for this method degrades quickly as the maximum disparity increases. Although this approach produces comparable images to our method for scenes with small disparities ($d_{\max} = 16, 32$), the renderings suffer from extreme inconsistency when rendering video sequences.

BW Deep methods use a CNN to estimate depth separately for each output viewpoint, so artifacts appear and disappear over only a few frames, resulting in rapid flickers and pops in the rendered camera path. This inconsistency is visible as corruption in the epipolar plots in Figure 2.9 and can be clearly seen in rendered output sequences with a moving camera. Furthermore, backwards warping incentivizes incorrect depth predictions to fill in

disocclusions, so BW Deep methods also produce errors around thin structures and occlusion edges.

2.5.3 Ablation Studies

We validate our overall strategy of blending between multiple MPIs as well as our specific blending procedure using accumulated alphas with the following ablation studies:

Single MPI The fifth row of Table 2.3 shows that using only one MPI to produce new views results in significantly decreased performance due to the limited field of view represented in a single MPI as well as depth discretization artifacts as the target view moves far from the MPI reference viewpoint. Additionally, Figure 2.7 shows an example of complex non-Lambertian reflectance that cannot be represented by a single MPI. This ablation can be considered an upper bound on the performance of Zhou *et al.* [140], since we use one MPI generated by a higher capacity 3D CNN.

Average MPIs The sixth row of Table 2.3 shows that blending multiple MPI outputs for each novel view without using the accumulated alpha channels results in decreased performance. Figure 2.6 visualizes that this simple blending leads to ghosting in regions that are occluded from the poses of any of the MPIs used for rendering, because they will contain incorrect content in disoccluded regions.

2.6 Practical Usage and Scaling Properties

We present guidelines to assist users in sampling views that enable high-quality view interpolation with our algorithm, and showcase our method's practicality with a smartphone camera app that guides users to easily capture such input images. Furthermore, we implement a mobile viewer that renders novel views from our predicted MPIs in real-time. Figure 2.9 showcases examples of rendered results from handheld smartphone captures.

2.6.1 Prescriptive Scene Sampling Guidelines

In a typical capture scenario, a user will have a camera with a field of view θ and a world space plane with side length S that bounds the viewpoints they wish to render. Based on this, we prescribe the design space of image resolution W and number of images to sample N that users can select from to reliably render novel views at Nyquist-level perceptual quality.

Section 2.5.1 shows that the empirical limit on the maximum disparity $d_{\max}$ between adjacent input views for our deep learning pipeline is 64 pixels. Substituting Equation 2.6:

$$\frac{\Delta_u f}{\Delta_x z_{\min}} \leq 64. \tag{2.8}$$

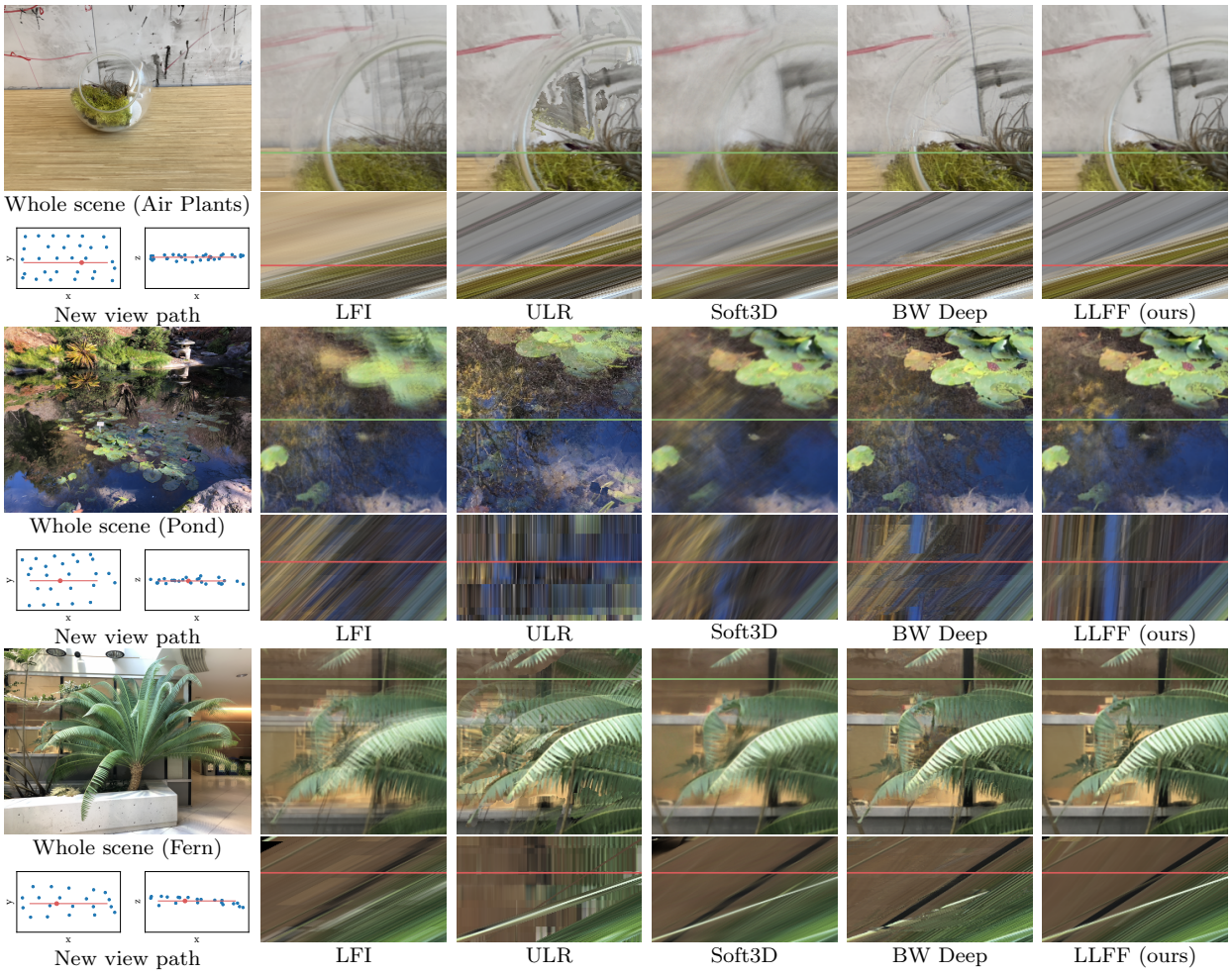


Figure 2.9: Results on real cellphone datasets. We render a sequence of new views and show both a crop from a single rendered output and an epipolar slice of the sequence. We show 2D projections of the input camera poses (blue dots) and new view path (red line) along the z and y axes of the new view camera in the lower left of each row. LFI fails to represent objects at different depths because it only uses a single depth plane for reprojection, leading to ghosting (leaves in Fern, lily pads in Pond) and depth inconsistency visible in all epipolar images. Mesh reconstruction failures cause artifacts visible in both the crops and epipolar images for ULR. Soft3D's depth uncertainty leads to blur, and geometry aggregation across large view neighborhoods results in incorrect specularity geometry (reflections in Pond). BW Deep's use of a CNN to render every novel view causes depth inconsistency, visible as choppiness across the rows of the epipolar images in all examples.

We translate this into user-friendly quantities by noting that $\Delta_u = S/\sqrt{N}$ and that the ratio of sensor width to focal length $W\Delta_x/f = 2\tan\theta/2$:

$$\frac{W}{\sqrt{N}} \leq \frac{128 z_{\min} \tan(\theta/2)}{S}. \tag{2.9}$$

Using a smartphone camera with a 64° field of view, this is simply:

$$\frac{W}{\sqrt{N}} \leq \frac{80 z_{\min}}{S}. \tag{2.10}$$

Intuitively, once a user has determined the extent of viewpoints they wish to render and the depth of the closest scene point, they can choose any target rendering resolution W and number of images to capture N such that the ratio $W/\sqrt{N}$ satisfies the above expression.

2.6.2 Asymptotic Rendering Time and Space Complexity

Within the possible choices of rendering resolution W and number of sampled views N that satisfy the above guideline, different users may value capture time, rendering time, and storage costs differently. We derive the asymptotic complexities of these quantities to further assist users in choosing correct parameters for their application.

First, the capture time is simply $O(N)$. The render time of each MPI generated is proportional to the number of planes times the pixels per plane:

$$W^2 D = \frac{W^3 S}{2\sqrt{N} z_{\min} \tan(\theta/2)} = O(W^3 N^{-1/2}). \tag{2.11}$$

Note that the rendering time for each MPI decreases as the number of sampled images N increases, because this allows us to use fewer planes per MPI. The total MPI storage cost is proportional to:

$$W^2 D \cdot N = \frac{W^3 S \sqrt{N}}{2 z_{\min} \tan(\theta/2)} = O(W^3 N^{1/2}). \tag{2.12}$$

Practically, this means that users should determine their specific rendering time and storage constraints, and then maximize the image resolution and number of sampled views that satisfy their constraints as well as the guideline in Equation 2.9. Figure 2.10 visualizes these constraints for an example user.

2.6.3 Smartphone Capture App

We develop an app for iOS smartphones, based on the ARKit framework, that guides users to capture input views for our view synthesis algorithm. The user first taps the screen to mark the closest object, and the app uses the corresponding scene depth computed by ARKit as $z_{\min}$. Next, the user selects the size of the view plane S within which our algorithm will

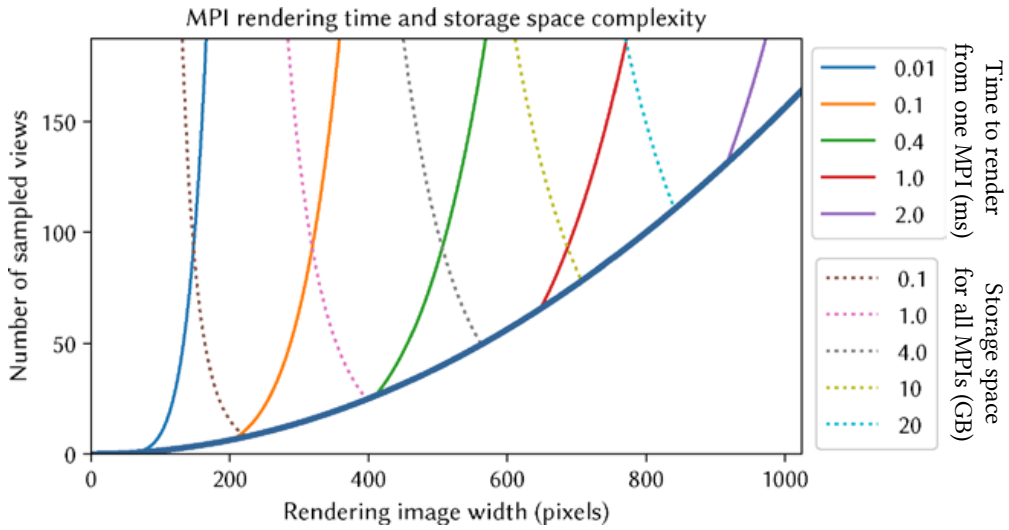


Figure 2.10: Time and storage cost tradeoff within the space of rendering resolution and number of sampled views that result in Nyquist level perceptual quality (space above the thick blue curve signifying $D = d_{\max} \leq 64$, as in Equation 2.10). We plot isocontours of rendering time and storage space for an example scene with close depth $z_{\min} = 1.0m$ and target view plane with side length $0.5m$, captured with a camera with a 64° field of view. We use the average rendering speed from our desktop viewer and the storage requirement from uncompressed 8-bit MPIs. Users can select the point where their desired rendering speed and storage space isocontours intersect to determine the minimum required number of views and maximum affordable rendering resolution.

render novel views. We fix the rendering resolution for the smartphone app to $W = 500$ which therefore fixes the prescribed number and spacing of required images based on Equation 2.10 and the definition $\Delta_u = S/\sqrt{N}$. Our app then guides the user to capture these views using the intuitive augmented reality overlay shown in Figure 2.11. When the phone detects that the camera has been moved to a new sample location, it automatically records an image and highlights the next sampling point.

2.6.4 Preprocessing

After capturing the required input images, the only preprocessing required before being able to render novel views is estimating the input camera poses and using our trained network to predict an MPI for each input view. Unfortunately, camera poses from ARKit are currently not accurate enough for acceptable results, so we use the open source COLMAP software package [103], which takes about 2-6 minutes for sets of 20-30 input images.

We use the deep learning pipeline described in Section 2.3.1 to predict an MPI for each input sampled view. On an Nvidia GTX 1080Ti GPU, This takes approximately 0.5 seconds

(a) Grid of guides shows user where to move phone

(b) Image automatically captured when phone aligns with guide

Figure 2.11: Equation 2.6 prescribes a simple sampling bound related only to the maximum scene disparity. We take advantage of the augmented reality toolkits available in modern smartphones to create an app that helps the user sample a real scene for rendering with our method. (a) We use built-in software to track the phone's position and orientation, providing sampling guides that allow the user to space photos evenly at the target disparity. (b) Once the user has centered the phone so that the RGB axes align with one of the guides, the app automatically captures a photo.

for a small MPI ($500 \times 350 \times 32 \approx 6$ megavoxels) or 12 seconds for a larger MPI that must be output in overlapping patches ($1000 \times 700 \times 64 \approx 45$ megavoxels). In total, our method only requires about 10 minutes of preprocessing to estimate poses and predict MPIs before being able to render novel views at a 1 megapixel image resolution.

With the increasing investment in smartphone AR and on-device deep learning accelerators, we expect that smartphone pose estimation will soon be accurate enough and on-device network inference will be powerful enough for users to go from capturing images to rendering novel views within a few seconds.

2.6.5 Real-Time Viewers

We implement novel view rendering from a single MPI by rasterizing each plane from back to front using texture mapped rectangles in 3D space, invoking a standard shader API to correctly handle the alpha compositing, perspective projection, and texture resampling. For each new view, we determine the MPIs to be blended, as discussed in Section 2.3.2, and render them into separate framebuffers. We then use a simple fragment shader to perform the alpha-weighted blending described in Section 2.3.2. We implement this rendering pipeline as desktop viewer using OpenGL which renders views with 1000×700 resolution at 60 frames per second, as well as an iOS mobile viewer using the Metal API which renders views with 500×350 resolution at 30 frames per second.

2.7 Discussion

This chapter presents a simple and practical method for view synthesis that works reliably for complex real-world scenes, including objects with non-Lambertian materials. This algorithm uses a feed-forward neural network to generate a local light field representation from each input image, then renders novel views in real time by interpolating between outputs computed from nearby representations. We provide an empirically determined plenoptic sampling guideline for the maximum allowable pixel disparity between input views.

A main limitation is that the MPI network sometimes assigns high opacity to incorrect layers in regions of ambiguous or repetitive texture and regions where scene content moves between input images. This can cause floating or blurred patches in the rendered output sequence, which is a common failure mode in methods that rely on texture matching cues to infer depth. In addition, these ambiguity errors may result in different geometry in MPIs corresponding to different input images, leading to temporal inconsistency as a rendered camera path moves from one input's neighborhood to another. It is difficult to completely avoid these ambiguity and inconsistency errors when using a feed-forward method, since the number of views that can be processed by the network is limited.

Another issue is the difficulty of scaling to higher image resolutions. As evident in Equations 2.11 and 2.12, discrete 3D grid-based approaches such as our method are limited by complexities that scale cubically with the image width or height in pixels. Furthermore, increasing the image resolution requires a convolutional network with a larger receptive field.

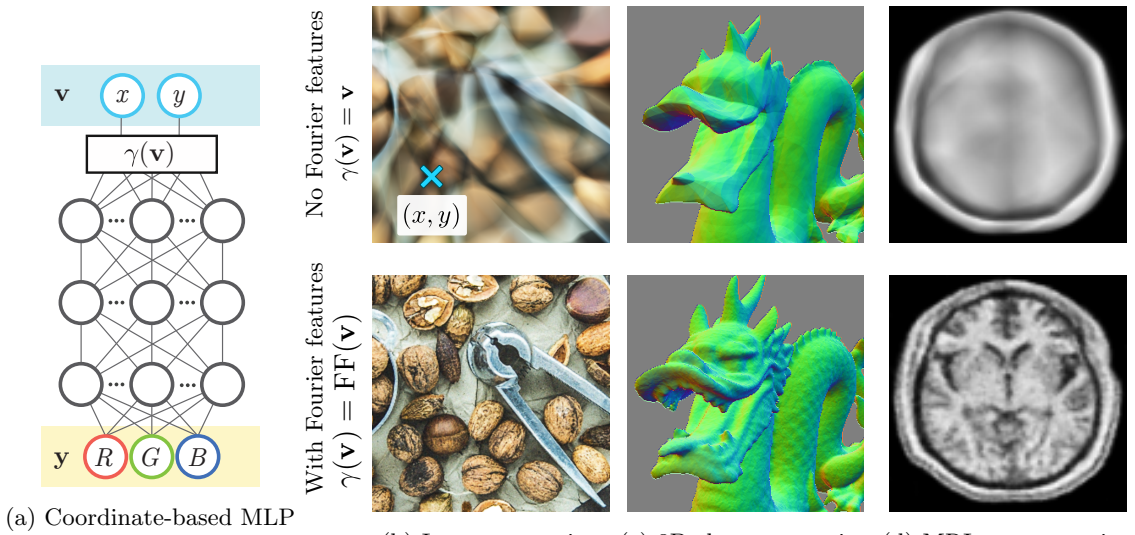
These problems motivate our search for a *global* scene representation that we can directly optimize to reproduce *all* input images of a scene, rather than only five nearby inputs at a time. A discrete 3D voxel grid is too memory-inefficient to serve as such a global scene representation that can be rendered from any viewpoint, since this would require isotropic sampling in space (as opposed to the MPI's heavily warped sampling pattern), leading to intractable scaling properties. In Chapter 3, we explore how neural networks can serve as a highly efficient representation for low-dimensional signals, and in Chapter 4, we present a view synthesis algorithm based on this concept that requires hundreds to thousands of times less storage, generates more temporally stable renderings, and can be directly optimized over all input images, in contrast to LLFF.

Chapter 3

An Efficient Global Representation: Coordinate-Based Networks with Fourier Feature Mappings

The previous chapter describes a view synthesis method based on generating many locally-valid scene representations, each stored as a large discretized 3D array. Dense voxel grids are frequently used to represent scenes in other view synthesis works [28, 73, 140] and for 3D shape reconstruction tasks [49, 121], largely because most deep learning research (across all areas of computer vision) has focused on solving problems using convolutional neural networks, which must be applied to data on a regularly-sampled grid. However, in three dimensions and higher, these dense arrays become highly impractical due to their $O(N^d)$ scaling in dimension d and linear resolution N. Typical 3D objects and scenes are particularly ill-suited for this representation, given that their surfaces only occupy a roughly two-dimensional subset of space, with the rest consisting of free space or object interiors. Given a known 3D shape, acceleration structures such as octrees [20] can be used to create an efficient sparse grid representation. However, this strategy does not work when our problem is to *recover* the underlying 3D shape from a set of 2D images or other inputs. When using machine learning methods to solve such an inverse problem, we need a signal representation that is memory-efficient but still fully differentiable.

A recent line of research in computer vision and graphics proposes replacing traditional discrete representations of objects and scenes using continuous functions parameterized by deep fully-connected networks (also called multilayer perceptrons or MLPs). These MLPs, which we will call "coordinate-based" MLPs, take 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 [84, 115] (referred to as "compositional pattern producing networks"), occupancy [80], and signed distance [92], and have achieved state-



(a) Coordinate-based MLP

(b) Image regression $(x,y) \rightarrow$ RGB

(c) 3D shape regression $(x,y,z) \rightarrow$ occupancy

(d) MRI reconstruction $(x,y,z) \rightarrow$ density

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) 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.

of-the-art results across a variety of tasks such as shape representation [19, 25, 32, 34, 46, 81, 92], texture synthesis [41, 89], shape inference from images [70, 72], and novel view synthesis [86, 102, 110].

Recent progress in modeling the behavior of deep networks using kernel regression with a neural tangent kernel (NTK) [45] can be leveraged 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" [4, 99]. 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.

In this chapter, we demonstrate how a *Fourier feature mapping* [100] can fix this problem, making coordinate-based networks a tractable alternative to discrete grids for representing high-frequency natural signals. A Fourier feature mapping lifts input coordinates $\mathbf{v}$ into a higher-dimensional space before passing them into the MLP:

$$\gamma(\mathbf{v}) = \left[a_1 \cos(2\pi \mathbf{b}_1^{\mathrm{T}} \mathbf{v}), a_1 \sin(2\pi \mathbf{b}_1^{\mathrm{T}} \mathbf{v}), \ldots, a_m \cos(2\pi \mathbf{b}_m^{\mathrm{T}} \mathbf{v}), a_m \sin(2\pi \mathbf{b}_m^{\mathrm{T}} \mathbf{v})\right]^{\mathrm{T}} \tag{3.1}$$

We show that this mapping transforms the NTK into a stationary (shift-invariant) 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 (Sections 3.3, 3.4). 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 (Figure 3.4). Figure 3.1 highlights how this mapping dramatically improves the performance of coordinate-based MLPs across a range of tasks relevant to the computer vision and graphics communities.

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 [115] and 3D scenes [80, 92]. In particular, our analysis explains experimental results demonstrating that an input mapping of coordinates (often referred to as a "positional encoding") using sinusoids with logarithmically-spaced axis-aligned frequencies improves the performance of coordinate-based MLPs [139]. We show that positional encoding is a special case of a Fourier feature mapping, which modifies the MLP's NTK to allow it to learn higher frequency functions more quickly.

Prior works in natural language processing and time series analysis [50, 122, 131] have used a similar positional encoding to represent time or 1D position. In particular, Xu *et al.* [131] use random Fourier features (RFF) [100] 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 [3, 7, 27, 45, 61]. In particular, we use the analyses from Lee *et al.* [61] and Arora *et al.* [3], 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 [3, 4, 7, 61, 132]. 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 [4, 5, 99].

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 Kernel Regression and the Spectral Bias of Deep Networks

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 [124]. 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:

$$\hat{f}(\mathbf{x}) = \sum_{i=1}^{n} \left(\mathbf{K}^{-1}\mathbf{y}\right)_i k(\mathbf{x}_i, \mathbf{x}) \,, \tag{3.2}$$

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.* [45] and extended by others [3, 4, 61] 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:

$$k_{\text{NTK}}(\mathbf{x}_i, \mathbf{x}_j) = \mathbb{E}_{\theta \sim \mathcal{N}} \left\langle \frac{\partial f(\mathbf{x}_i; \theta)}{\partial \theta}, \frac{\partial f(\mathbf{x}_j; \theta)}{\partial \theta} \right\rangle . \tag{3.3}$$

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 $h_{\text{NTK}}(\mathbf{x}_i^{\mathrm{T}} \mathbf{x}_j)$ for a scalar function $h_{\text{NTK}} : \mathbb{R} \to \mathbb{R}$).

Prior work [3, 4, 45, 61] 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.* [61], the network's output for any data $\mathbf{X}_{\text{test}}$ after t training iterations can be approximated as:

$$\hat{\mathbf{y}}^{(t)} \approx \mathbf{K}_{\text{test}} \mathbf{K}^{-1} \left(\mathbf{I} - e^{-\eta \mathbf{K} t}\right) \mathbf{y} \,, \tag{3.4}$$

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 $\hat{\mathbf{y}}^{(t)}_{\text{train}} - \mathbf{y}$, where $\hat{\mathbf{y}}^{(t)}_{\text{train}}$ 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 $e^{-\eta \mathbf{K} t} = \mathbf{Q} e^{-\eta \mathbf{\Lambda} t} \mathbf{Q}^{\mathrm{T}}$:

$$\mathbf{Q}^{\mathrm{T}}(\hat{\mathbf{y}}^{(t)}_{\text{train}} - \mathbf{y}) \approx \mathbf{Q}^{\mathrm{T}} \left(\left(\mathbf{I} - e^{-\eta \mathbf{K} t} \right) \mathbf{y} - \mathbf{y} \right) = -e^{-\eta \mathbf{\Lambda} t} \mathbf{Q}^{\mathrm{T}} \mathbf{y} \,. \tag{3.5}$$

This means that if we consider training convergence in the eigenbasis of the NTK, the i^{th} component of the absolute error $|\mathbf{Q}^{\mathrm{T}}(\hat{\mathbf{y}}^{(t)}_{\text{train}} - \mathbf{y})|_i$ 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 [5, 7, 36]. 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 create 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.

2. We would like to control the bandwidth of the NTK to improve training speed and generalization. As we see from Eqn. 3.5, 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 3.4 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 [100], 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:

$$\gamma(\mathbf{v}) = \left[a_1 \cos(2\pi \mathbf{b}_1^{\mathrm{T}} \mathbf{v}), a_1 \sin(2\pi \mathbf{b}_1^{\mathrm{T}} \mathbf{v}), \ldots, a_m \cos(2\pi \mathbf{b}_m^{\mathrm{T}} \mathbf{v}), a_m \sin(2\pi \mathbf{b}_m^{\mathrm{T}} \mathbf{v})\right]^{\mathrm{T}} . \tag{3.6}$$

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

$$k_\gamma(\mathbf{v}_1, \mathbf{v}_2) = \gamma(\mathbf{v}_1)^{\mathrm{T}} \gamma(\mathbf{v}_2) = \sum_{j=1}^{m} a_j^2 \cos\left(2\pi \mathbf{b}_j^{\mathrm{T}} (\mathbf{v}_1 - \mathbf{v}_2)\right) = h_\gamma(\mathbf{v}_1 - \mathbf{v}_2) , \tag{3.7}$$

$$\text{where } h_\gamma(\mathbf{v}_\Delta) \triangleq \sum_{j=1}^{m} a_j^2 \cos(2\pi \mathbf{b}_j^{\mathrm{T}} \mathbf{v}_\Delta) . \tag{3.8}$$

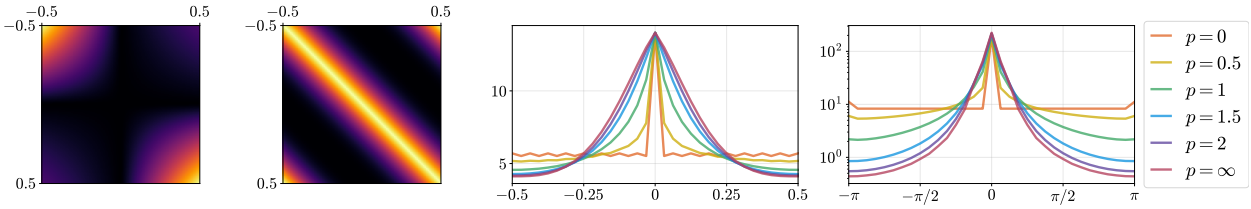
Note that this kernel is stationary (a function of only the difference between points). We can think of the mapping as a Fourier approximation of a kernel function: $\mathbf{b}_j$ are the Fourier basis frequencies used to approximate the kernel, and a_j^2 are the corresponding Fourier series coefficients.

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 $h_{\mathrm{NTK}}(\mathbf{x}_i^{\mathrm{T}} \mathbf{x}_j)$. In our case, $\mathbf{x}_i = \gamma(\mathbf{v}_i)$ so the composed kernel becomes:

$$h_{\mathrm{NTK}}(\mathbf{x}_i^{\mathrm{T}} \mathbf{x}_j) = h_{\mathrm{NTK}}\left(\gamma(\mathbf{v}_i)^{\mathrm{T}} \gamma(\mathbf{v}_j)\right) = h_{\mathrm{NTK}}\left(h_\gamma(\mathbf{v}_i - \mathbf{v}_j)\right) . \tag{3.9}$$

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

$$\hat{f} = (h_{\mathrm{NTK}} \circ h_\gamma) * \sum_{i=1}^{n} w_i \delta_{\mathbf{v}_i} \tag{3.10}$$



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

Figure 3.2: 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. 3.3) 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 $\gamma(v) = [\cos 2\pi v, \sin 2\pi v]^{\text{T}}$ makes the composed NTK $k_{\text{NTK}}(\gamma(v_i), \gamma(v_j))$ shift-invariant (stationary). (c) A Fourier feature input mapping (Eqn. 3.6) 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 3.3).

where $\mathbf{w} = \mathbf{K}^{-1}\mathbf{y}$ (from Eqn. 3.2). 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 Manipulating Network Behavior using a 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. 3.6, 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 simply taking the Fourier transform of a single row. All experiments are implemented in JAX [9] and the NTK functions are calculated automatically using the Neural Tangents library [88].

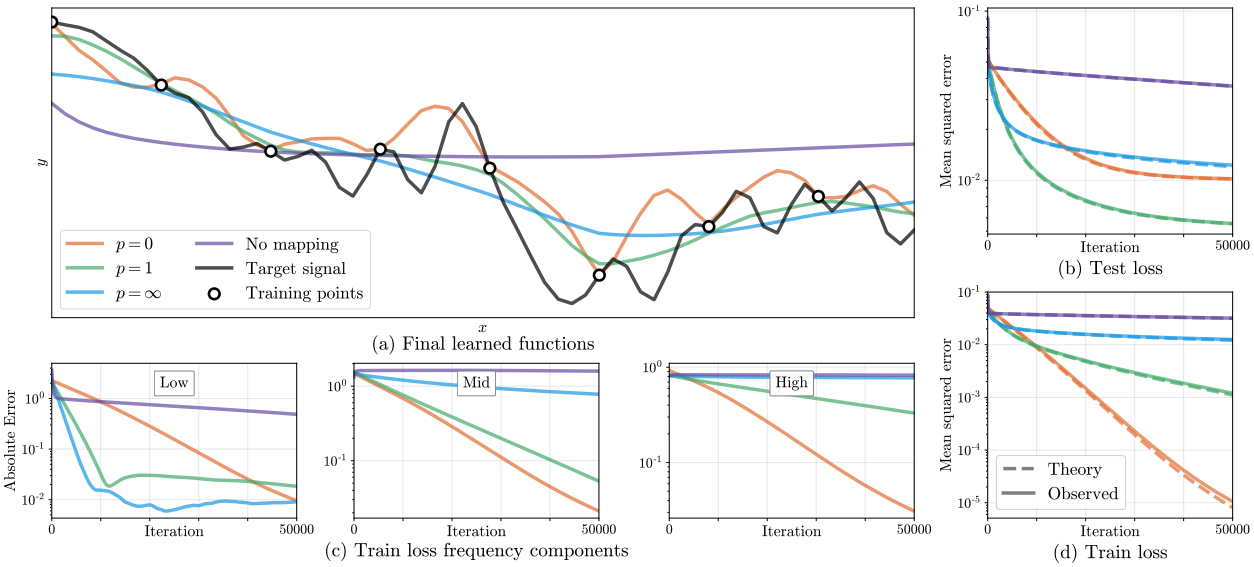
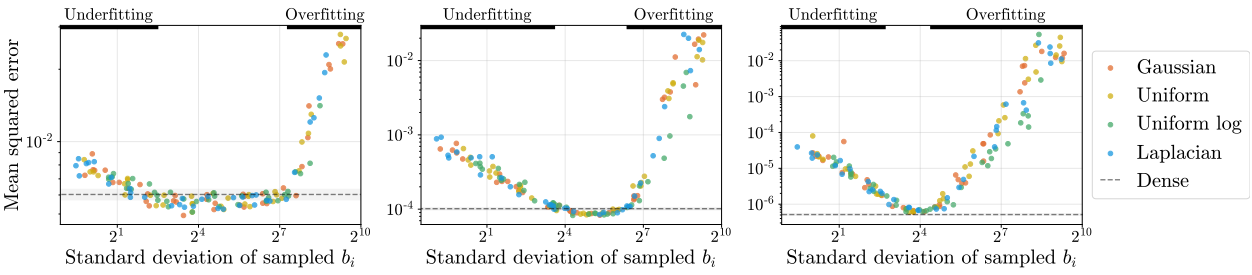


Figure 3.3: Combining a network with a Fourier feature mapping has dramatic effects on convergence and generalization. Here we train a network on 32 sampled points from a 1D function (a) using mappings shown in Fig. 3.2. A mapping with a smaller p value yields a composed NTK with more power in higher 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. 3.5, 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.

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 3.2 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 3.3, 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



(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 3.4: 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.

values. Figures 3.3b and 3.3d 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 3.3c 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. 3.4 allows us to estimate a trained network's theoretical 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 does achieve the best performance when training networks. However, 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 [100] from a parametric distribution. We find that the exact sampling distribution family is much less important than the distribution's scale (standard deviation).

Figure 3.4 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 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 show that this holds for higher-dimensional tasks in Section 3.5.3. 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 3.5 and treat the scale as a hyperparameter to tune on a validation dataset.

3.5 Experiments on 2D and 3D Tasks

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.

3.5.1 Compared mappings

In Table 3.1, 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 [122, 139]. 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

	Direct supervision			Indirect supervision		
	2D image		3D shape [80]	2D CT		3D MRI
	Natural	Text		Shepp	ATLAS	ATLAS
No mapping	19.32	18.40	0.864	16.75	15.44	26.14
Basic	21.71	20.48	0.892	23.31	16.95	28.58
Positional enc.	24.95	27.57	0.960	26.89	19.55	32.23
Gaussian	**25.57**	**30.47**	**0.973**	**28.33**	**19.88**	**34.51**

Table 3.1: We compare four different input mappings on a variety of low-dimensional regression tasks. All results are reported in PSNR except *3D shape*, which uses IoU (higher is better for all). *No mapping* represents using a standard MLP with no feature mapping. *Basic*, *Positional encoding*, and *Gaussian* are different variants of Fourier feature maps. For the *Direct supervision* tasks, the network is supervised using ground truth labels for each input coordinate. For the *Indirect supervision* tasks, the network outputs are passed through a forward model before the loss is applied (integral projection for CT and the Fourier transform for MRI). Fourier feature mappings improve results across all tasks, with random Gaussian features performing best.

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.

3.5.2 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 regression. In this task, we train an MLP to regress from a 2D input pixel coordinate to the corresponding RGB value of an image. For each test image, we train an MLP on a regularly-spaced grid containing $1/4$ of the pixels and report test error on the

remaining pixels. We compare input mappings over a dataset of natural images and a dataset of text images (Figure 3.5).

3D shape regression. Occupancy Networks [80] implicitly represent a 3D shape as the "decision boundary" of an MLP, which is trained to output 0 for points outside the shape and 1 for points inside the shape. Each batch of training data is generated by sampling points uniformly at random from the bounding box of the shape and calculating their labels using the ground truth mesh. Test error is calculated using intersection-over-union versus ground truth on a set of points randomly sampled near the mesh surface to better highlight the different mappings' abilities to resolve fine details. Figure 3.6 shows comparisons for four complex shapes.

2D computed tomography (CT). In CT, we observe integral projections of a density field instead of direct measurements. In our 2D CT experiments, we train an MLP that takes in a 2D pixel coordinate and predicts the corresponding volume density at that location. The network is indirectly supervised by the loss between a sparse set of ground-truth integral projections and integral projections computed from the network's output. We conduct experiments using two datasets: procedurally-generated Shepp-Logan phantoms [106] and 2D brain images from the ATLAS dataset [67] (Figure 3.7).

3D magnetic resonance imaging (MRI). In MRI, we observe Fourier transform coefficients of atomic response to radio waves under a magnetic field. In our 3D MRI experiments, we train an MLP that takes in a 3D voxel coordinate and predicts the corresponding response at that location. The network is indirectly supervised by the loss between a sparse set of ground-truth Fourier transform coefficients and Fourier transform coefficients computed from discretely querying the MLP on a voxel grid. We conduct experiments using the ATLAS dataset [67] (Figure 3.8).

3.5.3 Visualizing underfitting and overfitting in 2D

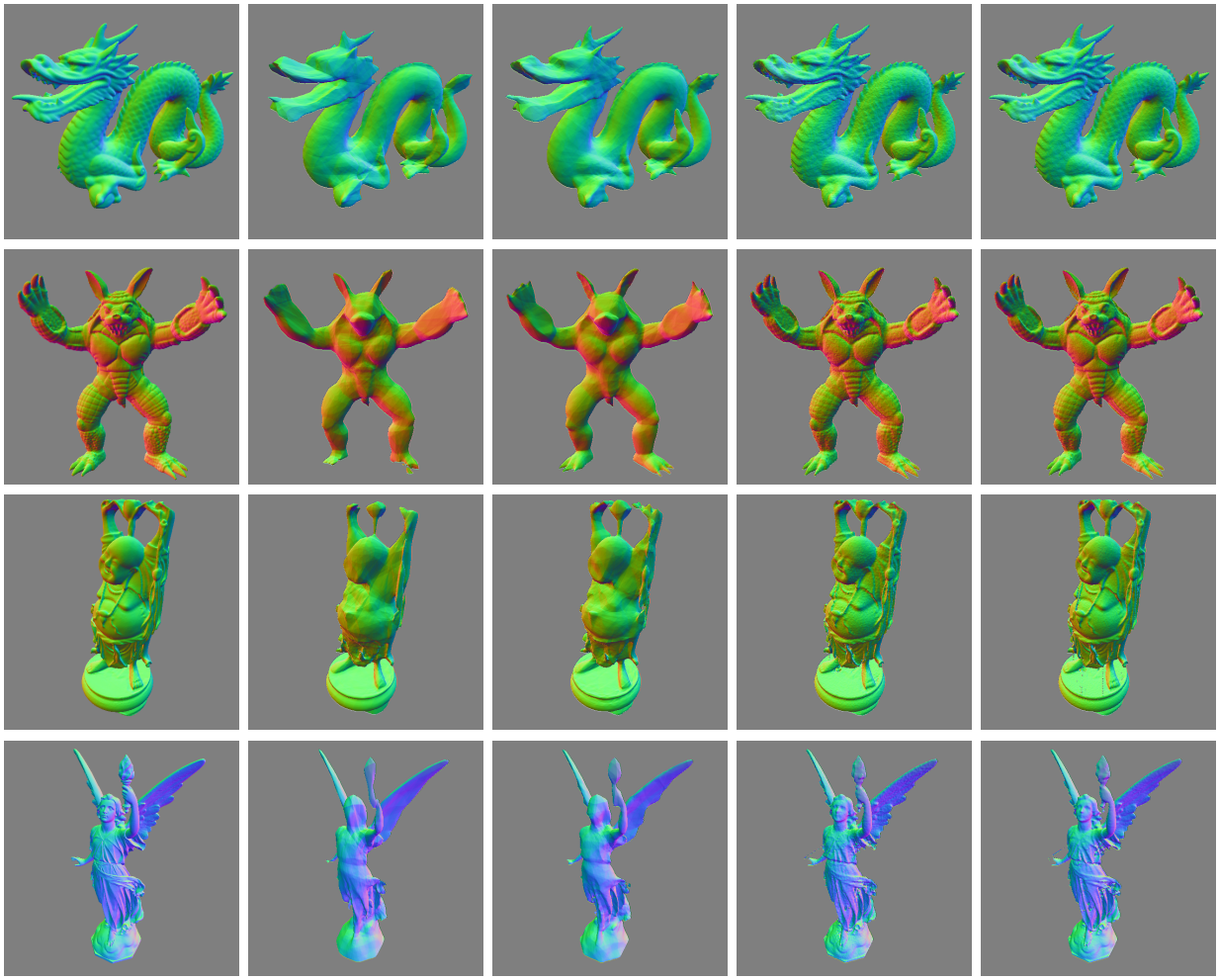
Figure 3.4 shows (in a 1D setting) that as the scale of the Fourier feature sampling distribution increases, the trained network's error traces out a curve that starts in an underfitting regime (only low frequencies are learned) and ends in an overfitting regime (the learned function includes high-frequency detail not present in the training data). In Figure 3.9, we show analogous behavior for 2D image regression, demonstrating that the same phenomenon holds in a multidimensional problem. In Figure 3.10, we show how changing the scale for Gaussian Fourier features qualitatively affects the final result in the 2D image regression task.

3.6 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 their inherent spectral bias. 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. In the next chapter, we will demonstrate how these coordinate-based networks can be used to develop a view synthesis algorithm that is able to render high-resolution outputs over a wide camera baseline.

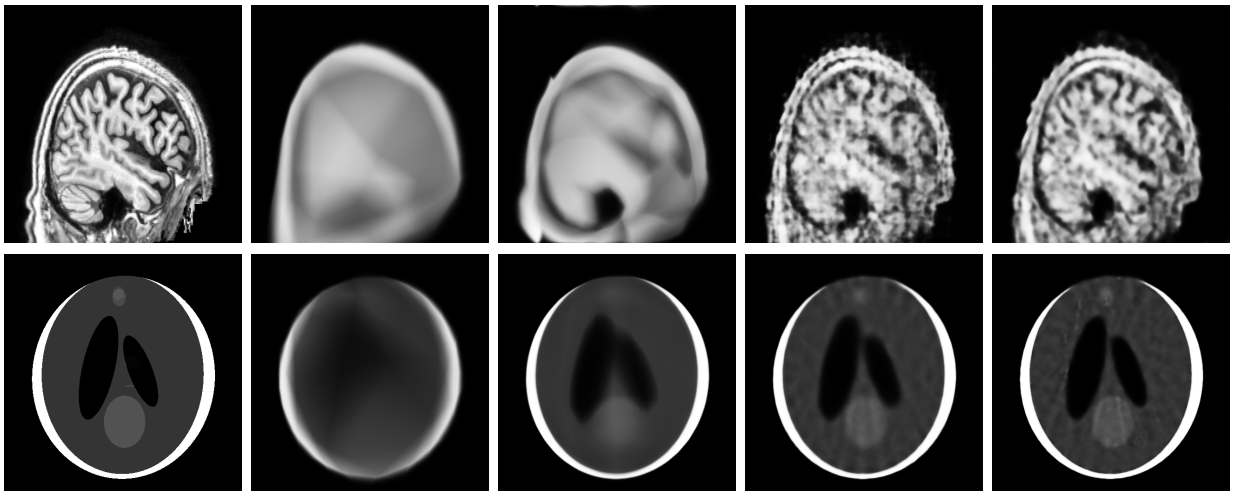
(a) Ground Truth (b) No mapping (c) Basic (d) Positional enc. (e) Gaussian

Figure 3.5: Results for the 2D image regression task, for three images from our *Natural* dataset (top) and two images from our *Text* dataset (bottom). Standard ReLU MLP networks (b) learn low-frequency components of the image exponentially faster than high-frequency details. Even a basic mapping (c) of each input pixel coordinate onto the 2D unit circle dramatically improves the network's representational power. A higher-dimensional Fourier feature mapping (d, e) essentially allows the network to perfectly fit the training points (every other pixel) while retaining reasonable interpolation behavior.



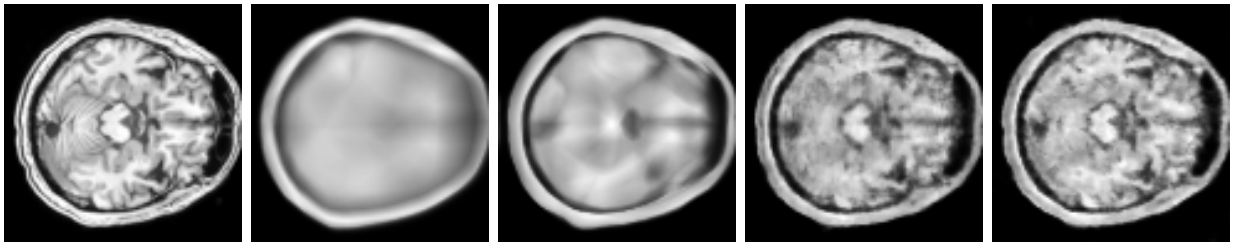
(a) Ground Truth (b) No mapping (c) Basic (d) Positional enc. (e) Gaussian

Figure 3.6: Results for the 3D shape occupancy task [80]. A standard network (b) can represent the rough outline of the shape, but a high-frequency Fourier feature mapping (d, e) is required to reproduce fine surface details.



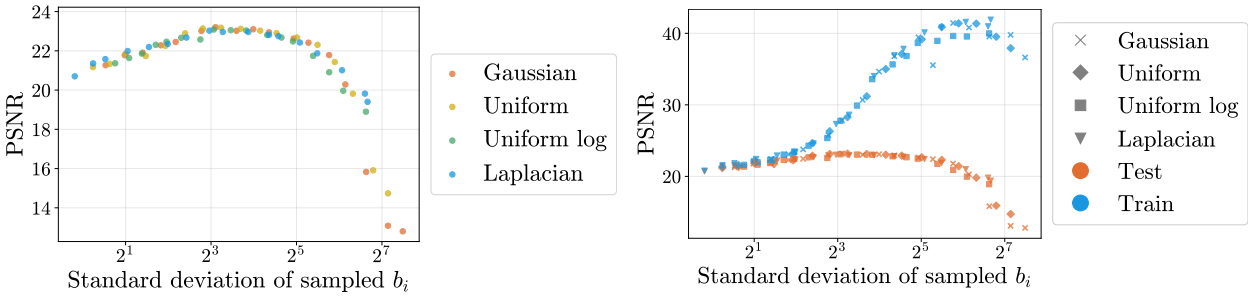
(a) Ground Truth (b) No mapping (c) Basic (d) Positional enc. (e) Gaussian

Figure 3.7: Results for the 2D CT task.



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

Figure 3.8: Results for the 3D MRI task.



(a) Test error for 2D image task

(b) Train and test error for 2D image task

Figure 3.9: An alternate version of Figure 3.4 from the main text where the underlying signal is a 2D image (see 2D image task details in Section 3.5.2) instead of 1D signal. This multi-dimensional case exhibits the same behavior as was seen in the 1D case: we see the same underfitting/overfitting pattern for four different isotropic Fourier feature distributions, and the distribution shape matters less than the scale of sampled b_i values.

$\sigma = 1$ $\sigma = 2$ $\sigma = 10$ $\sigma = 32$ $\sigma = 64$

Figure 3.10: A visualization of the 2D image regression task with different Gaussian scales (corresponding to points along the curve shown in Figure 3.9). Low values of σ underfit, resulting in oversmoothed interpolation, and large values of σ overfit, resulting in noisy interpolation. We find that $\sigma = 10$ performs best for our *Natural* image dataset.

Chapter 4

View Synthesis as Global Reconstruction: Neural Radiance Fields

In this chapter, we address the problem of view synthesis in a new way, using the coordinate-based network representation explored in the previous chapter to reconstruct a scene as a *neural radiance field* in a direct optimization framework. In contrast to the local interpolation method presented in Chapter 2, this approach does not require an external training dataset beyond images of the scene to be reconstructed, and its neural representation is over one hundred times more compact than a densely sampled array with equivalent resolution.

We represent a 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 coordinate-based neural network 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 directly optimize this model to represent a complex scene 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 4.2 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 cam-

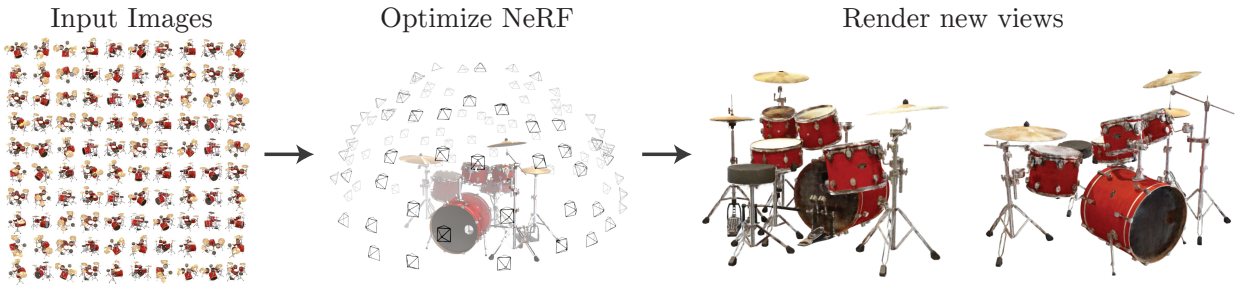


Figure 4.1: We present a method that optimizes a continuous 5D neural radiance field representation (volume density and view-dependent color at any continuous location) of a scene from a set of input images. We use techniques from volume rendering to accumulate samples of this scene representation along 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.

era ray. We address these issues by transforming input 5D coordinates with a positional encoding that enables the MLP to represent higher frequency functions (as described in the previous chapter), 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: it can represent complex real-world geometry and appearance and is well suited for gradient-based optimization using projected images. Crucially, our method is designed to overcome the prohibitive storage costs of *discretized* voxel grids when modeling complex scenes at high-resolutions.

In summary, our key 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, NeRF is 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.

4.1 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 [21] 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.

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 a signed distance function [92] or to an occupancy field [80]. 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 [14]. 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.* [86] 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 then provided as the input to a neural 3D texture field that predicts a diffuse color for that point. Sitzmann *et al.* [110] 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 arbitrarily complicated and high-resolution scene geometries, they have so far been limited to simple shapes with low geometric complexity resulting and produce oversmoothed rendered views. 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 The computer graphics community has made significant progress in photorealistic novel view synthesis by predicting traditional geometry and appearance representations from observed images. One popular class of approaches uses mesh-based representations of scenes with either diffuse [123] or view-dependent [11, 23, 128] appearance. Differentiable rasterizers [18, 33, 71, 74] or pathtracers [65, 87] can directly optimize mesh representations to reproduce a set of input images using gradient descent. However, gradient-based mesh optimization based on 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

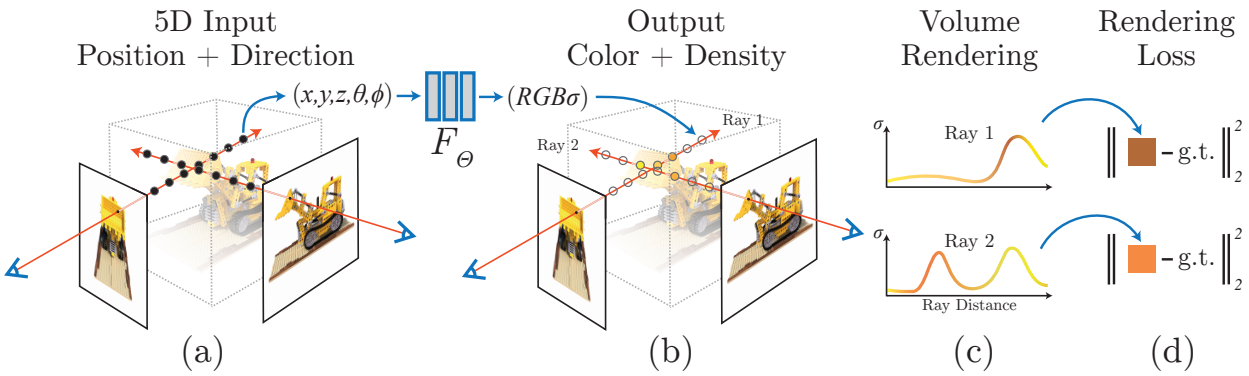


Figure 4.2: An overview of our neural radiance field scene representation and differentiable rendering procedure. We synthesize images by sampling 5D coordinates (location and viewing direction) along camera rays (a), feeding those locations into an MLP to produce a color and volume density (b), and 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).

an initialization before optimization [65], which is typically unavailable for unconstrained real-world scenes.

Another class of methods use volumetric representations to specifically 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 [57, 105, 117]. More recently, several methods [28, 83, 95, 113, 140] 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 alpha-compositing [96] 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 [109] or allow the predicted voxel grids to vary based on input time or animation controls [73]. 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.

4.2 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 $F_\Theta : (\mathbf{x}, \mathbf{d}) \rightarrow (\mathbf{c}, \sigma)$ and optimize its weights Θ to map 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 4 additional fully-connected layers (using ReLU activations and 128 channels per layer) that output the view-dependent RGB color.

See Fig. 4.3 for an example of how our method uses the input viewing direction to represent non-Lambertian effects. As shown in Fig. 4.4, a model trained without view dependence (only $\mathbf{x}$ as input) has difficulty representing specularities.

4.3 Volume Rendering with Radiance Fields

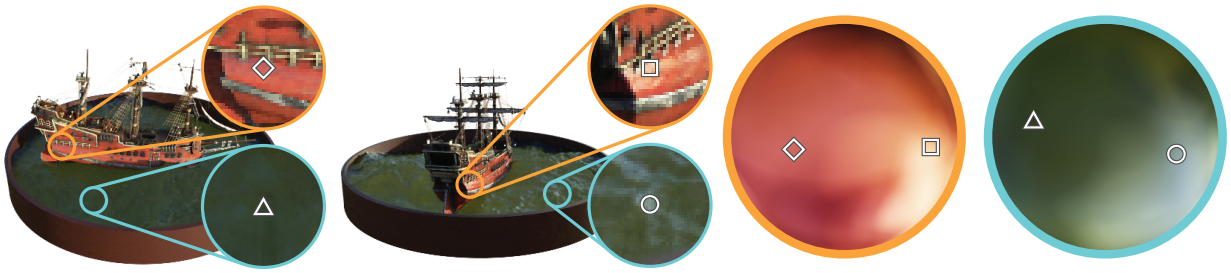
Our 5D neural radiance field represents a scene as the volume density and directional emitted radiance at any point in space. We render the color of any ray passing through the scene using principles from classical volume rendering [47]. The volume density $\sigma(\mathbf{x})$ can be interpreted as the 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:

$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), \mathbf{d})dt\,, \text{ where } T(t) = \exp\left(-\int_{t_n}^{t} \sigma(\mathbf{r}(s))ds\right). \tag{4.1}$$

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:

$$t_i \sim \mathcal{U}\left[t_n + \frac{i-1}{N}(t_f - t_n),\ t_n + \frac{i}{N}(t_f - t_n)\right]. \tag{4.2}$$



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

Figure 4.3: 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.

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 [78]:

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i(1 - \exp(-\sigma_i\delta_i))c_i\,, \text{ where } T_i = \exp\left(-\sum_{j=1}^{i-1}\sigma_j\delta_j\right), \tag{4.3}$$

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 (c_i, σ_i) values is trivially differentiable and reduces to traditional alpha compositing with alpha values $\alpha_i = 1 - \exp(-\sigma_i\delta_i)$.

4.4 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 4.5.3). We introduce two improvements to enable representing high-resolution complex 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.

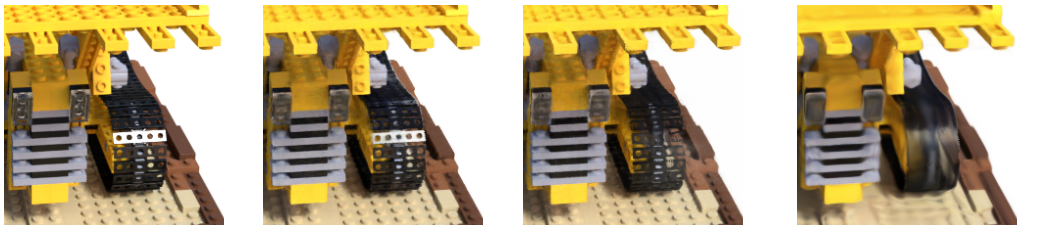


Figure 4.4: Here we visualize how our full model benefits from representing view-dependent emitted radiance and from passing our input coordinates through a high-frequency positional encoding. Removing view dependence prevents the model from recreating the specular reflection on the bulldozer tread. Removing the positional encoding drastically decreases the model's ability to represent high frequency geometry and texture, resulting in an over-smoothed appearance.

4.4.1 Positional encoding

Despite the fact that neural networks are universal function approximators [42], 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, as described in the previous chapter.

We reformulate F_Θ as a composition of two functions $F_\Theta = F'_\Theta \circ \gamma$, a standard MLP and a Fourier feature mapping, in order to significantly improve performance (see Fig. 4.4 and Table 4.2). Here γ is specifically the "positional encoding" variant of Fourier feature mapping, lifting the input component-wise from $\mathbb{R}$ into a higher dimensional space $\mathbb{R}^{2L}$. Formally, the encoding function we use is:

$$\gamma(p) = \big(\ \sin(2^0\pi p),\ \ \cos(2^0\pi p),\ \ \cdots,\ \ \sin\big(2^{L-1}\pi p\big),\ \ \cos\big(2^{L-1}\pi p\big)\ \big)\,. \tag{4.4}$$

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 $\gamma(\mathbf{d})$.

4.4.2 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 [63] 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. 4.2 and 4.3. 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. 4.3 as a weighted sum of all sampled colors c_i along the ray:

$$\hat{C}_c(\mathbf{r}) = \sum_{i=1}^{N_c} w_i c_i \,, \qquad w_i = T_i(1 - \exp(-\sigma_i \delta_i)) \,. \tag{4.5}$$

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. 4.3 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.

4.4.3 Implementation details

We optimize a separate neural continuous volume representation network for each scene (see Figure 4.5 for architecture details). 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 [103] 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. 4.4.2 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. 4.3 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:

$$\mathcal{L} = \sum_{\mathbf{r} \in \mathcal{R}} \left[\left\| \hat{C}_c(\mathbf{r}) - C(\mathbf{r}) \right\|_2^2 + \left\| \hat{C}_f(\mathbf{r}) - C(\mathbf{r}) \right\|_2^2 \right] \tag{4.6}$$

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.

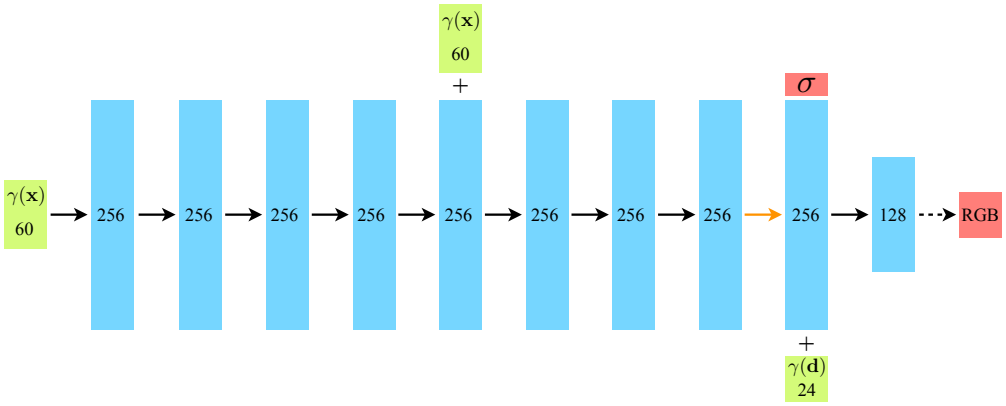


Figure 4.5: A visualization of our fully-connected network architecture. Input vectors are shown in green, intermediate hidden layers are shown in blue, output vectors are shown in red, and the number inside each block signifies the vector's dimension. All layers are standard fully-connected layers, black arrows indicate ReLU activation, orange arrows indicate no activation, dashed black arrows indicate sigmoid activation, and "+" denotes concatenation. The positional encoding of the input location ($\gamma(\mathbf{x})$) is passed through 8 fully-connected ReLU layers, each with 256 channels. We follow the DeepSDF [92] architecture and include a skip connection that concatenates this input to the fifth layer's activation. The penultimate layer outputs the volume density σ (passed through a ReLU to ensure it is nonnegative) and a 256-dimensional feature vector. This feature vector is concatenated with the positional encoding of the input viewing direction ($\gamma(\mathbf{d})$), which is processed by a final layer with 128 channels. A sigmoid activation produces the emitted RGB radiance at position $\mathbf{x}$, as viewed by a ray with direction $\mathbf{d}$.

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 [54] 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).

To render new views at test time, we sample 64 points per ray through the coarse network and $64+128 = 192$ points per ray through the fine network, for a total of 256 network queries per ray. Our realistic synthetic dataset requires 640k rays per image, and our real scenes require 762k rays per image, resulting in between 150 and 200 million network queries per rendered image. On an NVIDIA V100, this takes approximately 30 seconds per frame.

Method	Diffuse Synthetic 360° [109]			Realistic Synthetic 360°			Real Forward-Facing [83]		
	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
SRN [110]	33.20	0.963	0.073	22.26	0.846	0.170	22.84	0.668	0.378
NV [73]	29.62	0.929	0.099	26.05	0.893	0.160	-	-	-
LLFF [83]	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 4.1: Our method quantitatively outperforms prior work on datasets of both synthetic and real images. We report PSNR/SSIM (higher is better) and LPIPS [137] (lower is better). The DeepVoxels [109] 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.

4.5 Results

We quantitatively (Table 4.1) and qualitatively (Figs. 4.6 and 4.7) show that our method outperforms prior work, and provide extensive ablation studies to validate our design choices (Table 4.2). Our video results more clearly demonstrate our method's significant improvement over baseline methods when rendering smooth paths of novel views.

4.5.1 Datasets

Synthetic renderings of objects We first show experimental results on two datasets of synthetic renderings of objects (Table 4.1, "Diffuse Synthetic 360°" and "Realistic Synthetic 360°"). The DeepVoxels [109] dataset contains four Lambertian objects with relatively 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 4.1, "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.

4.5.2 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 [83], 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) [73] 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) [110] 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 [109] by the same authors, which is why we do not include comparisons to DeepVoxels.

Local Light Field Fusion (LLFF) [83] LLFF (described in Chapter 2) 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 [140]) for each input view, then renders novel views by alpha compositing and blending nearby MPIs into the novel viewpoint.

4.5.3 Ablation studies

We validate our algorithm's design choices and parameters with an extensive ablation study in Table 4.2. We present results on one of our synthetic scenes with complex geometry and non-Lambertian materials (*Lego*). 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 provides the largest quantitative benefit of these three contributions (row 2), followed by view-dependence (row 3), and then 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

	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)	**31.01**	**0.947**	**0.081**

Table 4.2: An ablation study of our model. Metrics are averaged over the 8 scenes from our realistic synthetic dataset. See Sec. 4.5.3 for detailed descriptions.

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).

4.6 Discussion

NeRF outperforms both compared methods that use direct optimization to also train one network per scene (NV [73] and SRN [110]). It also produces superior renderings to the feed-forward prediction method LLFF (Chapter 2 and [83]) without requiring a large external training dataset.

The SRN method also represents each scene using an MLP but can only generate blurry, low-frequency reconstructions, likely due to the fact that it does not use a Fourier feature mapping to encode the input coordinates. Its representational power for view synthesis is also limited by its surface rendering model that only calculates color at a single point on each camera ray. The NV baseline is able to capture reasonably detailed volumetric geometry and appearance, but its use of an underlying discrete 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 (see Section 2.5.1), 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 interpolates between different local scene representations for rendering different views, resulting in perceptually distracting inconsistency when rendering smooth camera paths.

The biggest practical tradeoffs between these methods are time versus space. All compared direct optimization methods that optimize a global representation per-scene take at least 12 hours to train. In contrast, as a feed-forward method, LLFF can process a small

input dataset in under 10 minutes. However, the large 3D voxel grid generated for every input image by LLFF result in enormous storage requirements (over 15GB for one "Realistic Synthetic" scene). NeRF requires only 5 MB for the network weights (a relative compression of 3000× compared to LLFF), which is even less memory than the *input images alone* for a single scene from any of our datasets.

In summary, NeRF directly addresses deficiencies of prior work that uses MLPs as a global representation for objects and scenes as continuous functions by incorporating a Fourier feature mapping and adopting a volume rendering forward model. 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.

Figure 4.6: 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 4.7: 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.

Chapter 5

Conclusion

Even in the short time since the work in this dissertation was originally published, various groups have made large strides toward productionizing practical and robust view synthesis systems based on feed-forward machine learning. Broxton *et al.* [10] at Google have developed a system (building on their earlier work in Overbeck *et al.* [91] and Flynn *et al.* [28]) that can capture, reconstruct, compress, and render high quality light field video using input data captured by a spherical array of 46 cameras, releasing reconstructed scenes that can be viewed in a virtual reality headset or even directly in a web browser. Kopf *et al.* [56] recently incorporated a "3D photography" feature directly into the Facebook app that is optimized to run in a few seconds on a mobile device, using monocular depth estimation to add a 3D viewing effect to any user-uploaded photo.

The simple global scene representation, volumetric rendering method, and optimization algorithm described in NeRF have served as a basis for a number of followup papers that improve and build upon the original in various ways, such as

- Extending the representation to include material properties, allowing the recovered scene to be rendered under new lighting conditions [6, 8, 112],
- Speeding up the rendering process via spatial decomposition [69, 101] or directly learning volume rendering integrals [68],
- Allowing recovery and rendering of deformable [93, 97] or dynamic [30, 66, 130] scenes,
- Recovering a radiance field from one or few images by training over many example scenes [31, 118, 120, 135],
- Creating purely generative models for 3D shapes from a particular class (such as human faces, chairs, or cars) [13, 85, 104],
- Generalizing to scenes where the input images are taken under many different lighting and appearance conditions [77, 118].

Seeing such rapid progress less than a year after NeRF's release has been incredibly exciting, and I am certain that the future has many more developments in store. One of the most intriguing directions is investigating the inner workings of these coordinate-based neural representations. As a community, we have largely borrowed the same conventions used when training large feed-forward convolutional neural networks despite applying them to a radically different use case. I am optimistic that further work will uncover different optimizers, architectures, and training schemes that can further improve the performance of these networks, making them even more appealing as differentiable compressed representation to use in-the-loop for inverse rendering tasks and other applications.

The questions raised in the introduction are far from settled. Local interpolation methods based on warping and blending pixels from the input images can achieve convincing results in many scenarios and will likely continue to be used where a limited compute budget makes generating a global reconstruction infeasible. Volume rendering is straightforward to implement in array-based deep learning frameworks, but recent work on using implicit differentiation to calculate gradients for surface-based neural representations has shown promising results on recovering 3D shape from images [86], particularly when combined with positional encoding [133]. Even in cases where volume rendering is used during optimization, converting the reconstructed scene into a surface-based representation would likely make rendering new views much more efficient. Permutation-invariant [1] and attention-based network layers [122] may provide a solution to the problem of passing arbitrary numbers of input views through a feed-forward network, allowing them to feasibly be used for global reconstruction and thus sidestepping the time-consuming process of directly optimizing a new network for every scene; this idea has already begun to be explored by followup work to NeRF [120, 135]. Hybrid approaches may combine the best of both worlds, such as passing gradient information into a feed-forward network to allow it to refine its output iteratively [28], or fine-tuning a feed-forward network's initial output using direct optimization on a specific scene to achieve higher quality [75]. As the purview of deep learning continues to expand into a more general realm of differentiable computing, I expect to see many further innovations in how we represent, render, and reconstruct scenes for view synthesis.

The most important takeaway from NeRF is that a simple approach *works*: given enough input images, directly optimizing a sufficiently expressive scene representation to reproduce those inputs using basic gradient descent produces a high-quality reconstruction. By no means is NeRF the end of the line for view synthesis—rather, its simplicity makes it an optimal starting point for a wide variety of exciting extensions and new applications (as we have already seen!). I eagerly anticipate further progress that continues to make it easier for anyone to generate, modify, and share 3D content freely, and I believe that making 3D data as ubiquitous as 2D imagery is today will spur people to develop creative tools and applications that greatly exceed the current limits of our imagination.

Bibliography

[1] Miika Aittala and Fredo Durand. "Burst Image Deblurring Using Permutation Invariant Convolutional Neural Networks". In: *ECCV*. 2018.

[2] Robert Anderson, David Gallup, Jonathan T. Barron, Janne Kontkanen, Noah Snavely, Carlos Hernández, Sameer Agarwal, and Steven M Seitz. "Jump: Virtual Reality Video". In: *SIGGRAPH Asia*. 2016.

[3] Sanjeev Arora, Simon Du, Wei Hu, Zhiyuan Li, and Ruosong Wang. "Fine-Grained Analysis of Optimization and Generalization for Overparameterized Two-Layer Neural Networks". In: *ICML* (2019).

[4] Ronen Basri, Meirav Galun, Amnon Geifman, David Jacobs, Yoni Kasten, and Shira Kritchman. "Frequency Bias in Neural Networks for Input of Non-Uniform Density". In: *arXiv preprint arXiv:2003.04560* (2020).

[5] Ronen Basri, David Jacobs, Yoni Kasten, and Shira Kritchman. "The Convergence Rate of Neural Networks for Learned Functions of Different Frequencies". In: *NeurIPS* (2019).

[6] Sai Bi, Zexiang Xu, Pratul P. Srinivasan, Ben Mildenhall, Kalyan Sunkavalli, Milos Hasan, Yannick Hold-Geoffroy, David Kriegman, and Ravi Ramamoorthi. "Neural Reflectance Fields for Appearance Acquisition". In: *arXiv* (2020).

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

[8] Mark Boss, Raphael Braun, Varun Jampani, Jonathan T. Barron, Ce Liu, and Hendrik P.A. Lensch. "NeRD: Neural Reflectance Decomposition from Image Collections". In: *arXiv* (2020).

[9] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, and Skye Wanderman-Milne. *JAX: composable transformations of Python+NumPy programs*. Version 0.1.68. `http://github.com/google/jax`. 2018.

[10] Michael Broxton, John Flynn, Ryan Overbeck, Daniel Erickson, Peter Hedman, Matthew DuVall, Jason Dourgarian, Jay Busch, Matt Whalen, et al. "Immersive Light Field Video with a Layered Mesh Representation". In: *ACM Transactions on Graphics (Proc. SIGGRAPH)*. 2020.

[11] Chris Buehler, Michael Bosse, Leonard McMillan, Steven Gortler, and Michael Cohen. "Unstructured Lumigraph Rendering". In: *SIGGRAPH*. 2001.

[12] Jin-Xiang Chai, Xin Tong, Sing-Chow Chan, and Heung-Yeung Shum. "Plenoptic Sampling". In: *SIGGRAPH*. 2000.

[13] Eric Chan, Marco Monteiro, Petr Kellnhofer, Jiajun Wu, and Gordon Wetzstein. "pi-GAN: Periodic Implicit Generative Adversarial Networks for 3D-Aware Image Synthesis". In: *arXiv*. 2020.

[14] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, et al. "Shapenet: An information-rich 3d model repository". In: *arXiv:1512.03012* (2015).

[15] Gaurav Chaurasia, Sylvain Duchêne, Olga Sorkine-Hornung, and George Drettakis. "Depth Synthesis and Local Warps for Plausible Image-based Navigation". In: *SIGGRAPH*. 2013.

[16] Qifeng Chen and Vladlen Koltun. "Photographic Image Synthesis With Cascaded Refinement Networks". In: *ICCV*. 2017.

[17] Shenchang Eric Chen and Lance Williams. "View Interpolation for Image Synthesis". In: *SIGGRAPH*. 1993.

[18] Wenzheng Chen, Jun Gao, Huan Ling, Edward J. Smith, Jaakko Lehtinen, Alec Jacobson, and Sanja Fidler. "Learning to Predict 3D Objects with an Interpolation-based Differentiable Renderer". In: *NeurIPS*. 2019.

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

[20] Cyril Crassin, Fabrice Neyret, Sylvain Lefebvre, and Elmar Eisemann. "Gigavoxels: Ray-guided streaming for efficient and detailed voxel rendering". In: *Proceedings of the 2009 symposium on Interactive 3D graphics and games*. 2009.

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

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

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

[24] Paul Debevec, Yizhou Yu, and George Borshukov. "Efficient view-dependent image-based rendering with projective texture-mapping". In: *Eurographics Workshop on Rendering Techniques*. 1998.

[25] Boyang Deng, JP Lewis, Timothy Jeruzalski, Gerard Pons-Moll, Geoffrey Hinton, Mohammad Norouzi, and Andrea Tagliasacchi. "Neural Articulated Shape Approximation". In: *arXiv preprint arXiv:1912.03207* (2019).

[26] Piotr Didyk, Pitchaya Sitthi-Amorn, William T. Freeman, Fredo Durand, and Wojciech Matusik. "3DTV at Home: Eulerian-Lagrangian Stereo-to-Multiview Conversion". In: *SIGGRAPH Asia*. 2013.

[27] Simon S. Du, Xiyu Zhai, Barnabas Poczos, and Aarti Singh. "Gradient Descent Provably Optimizes Over-parameterized Neural Networks". In: *ICLR* (2019).

[28] John Flynn, Michael Broxton, Paul Debevec, Matthew DuVall, Graham Fyffe, Ryan Overbeck, Noah Snavely, and Richard Tucker. "DeepView: view synthesis with learned gradient descent". In: *CVPR*. 2019.

[29] John Flynn, Ivan Neulander, James Philbin, and Noah Snavely. "DeepStereo: Learning to Predict New Views From the World's Imagery". In: *CVPR*. 2016.

[30] Guy Gafni, Justus Thies, Michael Zollhöfer, and Matthias Nießner. "Dynamic Neural Radiance Fields for Monocular 4D Facial Avatar Reconstruction". In: *arXiv* (2020).

[31] Chen Gao, Yichang Shih, Wei-Sheng Lai, Chia-Kai Liang, and Jia-Bin Huang. "Portrait Neural Radiance Fields from a Single Image". In: *arXiv preprint arXiv:2012.05903* (2020).

[32] Kyle Genova, Forrester Cole, Aaron Sarna Daniel Vlasic, William T. Freeman, and Thomas Funkhouser. "Learning Shape Templates with Structured Implicit Functions". In: *ICCV* (2019).

[33] Kyle Genova, Forrester Cole, Aaron Maschinot, Aaron Sarna, Daniel Vlasic, and William T Freeman. "Unsupervised training for 3d morphable model regression". In: *CVPR*. 2018.

[34] Kyle Genova, Forrester Cole, Avneesh Sud, Aaron Sarna, and Thomas Funkhouser. "Local Deep Implicit Functions for 3D Shape". In: *CVPR*. 2020.

[35] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. "The Lumigraph". In: *SIGGRAPH*. 1996.

[36] 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).

[37] Peter Hedman, Suhib Alsisan, Richard Szeliski, and Johannes Kopf. "Casual 3D Photography". In: *SIGGRAPH Asia*. 2017.

[38] Peter Hedman and Johannes Kopf. "Instant 3D Photography". In: *SIGGRAPH*. 2018.

[39] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. "Deep Blending for Free-Viewpoint Image-Based Rendering". In: *SIGGRAPH Asia*. 2018.

[40] Peter Hedman, Tobias Ritschel, George Drettakis, and Gabriel Brostow. "Scalable Inside-Out Image-Based Rendering". In: *SIGGRAPH Asia*. 2016.

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

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

[43] *How many photos will be taken in 2020?* `https://focus.mylio.com/tech-today/how-many-photos-will-be-taken-in-2020`. Accessed: 2020-12-07.

[44] Po-Han Huang, Kevin Matzen, Johannes Kopf, Narendra Ahuja, and Jia-Bin Huang. "DeepMVS: Learning Multi-View Stereopsis". In: *CVPR*. 2018.

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

[46] Chiyu Jiang, Avneesh Sud, Ameesh Makadia, Jingwei Huang, Matthias Nießner, and Thomas Funkhouser. "Local Implicit Grid Representations for 3D Scenes". In: *CVPR*. 2020.

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

[48] Nima Khademi Kalantari, Ting-Chun Wang, and Ravi Ramamoorthi. "Learning-Based View Synthesis for Light Field Cameras". In: *SIGGRAPH Asia*. 2016.

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

[50] Seyed Mehran Kazemi, Rishab Goel, Sepehr Eghbali, Janahan Ramanan, Jaspreet Sahota, Sanjay Thakur, Stella Wu, Cathal Smyth, Pascal Poupart, et al. "Time2Vec: Learning a Vector Representation of Time". In: *arXiv preprint arXiv:1907.05321* (2019).

[51] Michael Kazhdan and Hugues Hoppe. "Screened Poisson Surface Reconstruction". In: *SIGGRAPH*. 2013.

[52] Petr Kellnhofer, Piotr Didyk, Szu-Po Wang, Pitchaya Sitthi-Amorn, William Freeman, Fredo Durand, and Wojciech Matusik. "3DTV at Home: Eulerian-Lagrangian Stereo-to-Multiview Conversion". In: *SIGGRAPH*. 2017.

[53] Alex Kendall, Hayk Martirosyan, Saumitro Dasgupta, Peter Henry, Ryan Kennedy, Abraham Bachrach, and Adam Bry. "End-to-End Learning of Geometry and Context for Deep Stereo Regression". In: *ICCV*. 2017.

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

[55] Johannes Kopf, Fabian Langguth, Daniel Scharstein, Richard Szeliski, and Michael Goesele. "Image-Based Rendering in the Gradient Domain". In: *SIGGRAPH Asia*. 2013.

[56] Johannes Kopf, Kevin Matzen, Suhib Alsisan, Ocean Quigley, Francis Ge, Yangming Chong, Josh Patterson, Jan-Michael Frahm, Shu Wu, et al. "One Shot 3D Photography". In: *Transactions on Graphics (Proceedings of SIGGRAPH)*. 2020.

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

[58] Philippe Lacroute and Marc Levoy. "Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation". In: *SIGGRAPH*. 1994.

[59] Samuli Laine, Janne Hellsten, Tero Karras, Yeongho Seol, Jaakko Lehtinen, and Timo Aila. "Modular Primitives for High-Performance Differentiable Rendering". In: *ACM Transactions on Graphics* 39.6 (2020).

[60] Douglas Lanman, Ramesh Raskar, Amit Agrawal, and Gabriel Taubin. "Shield Fields: Modeling and Capturing 3D Occluders". In: *SIGGRAPH Asia*. 2008.

[61] Jaehoon Lee, Lechao Xiao, Samuel Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. "Wide neural networks of any depth evolve as linear models under gradient descent". In: *NeurIPS* (2019).

[62] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. "Layer Normalization". In: *arXiv:1607.06450*. 2016.

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

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

[65] Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. "Differentiable Monte Carlo Ray Tracing through Edge Sampling". In: *ACM Transactions on Graphics (SIGGRAPH Asia)* (2018).

[66] Zhengqi Li, Simon Niklaus, Noah Snavely, and Oliver Wang. "Neural Scene Flow Fields for Space-Time View Synthesis of Dynamic Scenes". In: *arXiv* (2020).

[67] Sook-Lei Liew, Julia M. Anglin, Nick W. Banks, Matt Sondag, Kaori L. Ito, Kim, et al. "A large, open source dataset of stroke anatomical brain images and manual lesion segmentations". In: *Scientific Data* (2018).

[68] David B. Lindell, Julien N. P. Martel, and Gordon Wetzstein. "AutoInt: Automatic Integration for Fast Neural Volume Rendering". In: *arXiv* (2020).

[69] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. "Neural Sparse Voxel Fields". In: *NeurIPS* (2020).

[70] Shaohui Liu, Yinda Zhang, Songyou Peng, Boxin Shi, Marc Pollefeys, and Zhaopeng Cui. "DIST: Rendering Deep Implicit Signed Distance Function with Differentiable Sphere Tracing". In: *CVPR*. 2019.

[71] Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. "Soft Rasterizer: A Differentiable Renderer for Image-based 3D Reasoning". In: *ICCV*. 2019.

[72] Shichen Liu, Shunsuke Saito, Weikai Chen, and Hao Li. "Learning to Infer Implicit Surfaces without 3D Supervision". In: *NeurIPS*. 2019.

[73] Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. "Neural volumes: Learning dynamic renderable volumes from images". In: *ACM Transactions on Graphics (SIGGRAPH)* (2019).

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

[75] Xuan Luo, Jia-Bin Huang, Richard Szeliski, Kevin Matzen, and Johannes Kopf. "Consistent Video Depth Estimation". In: 39.4 (2020).

[76] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. URL: `https://www.tensorflow.org/`.

[77] Ricardo Martin-Brualla, Noha Radwan, Mehdi S. M. Sajjadi, Jonathan T. Barron, Alexey Dosovitskiy, and Daniel Duckworth. "NeRF in the Wild: Neural Radiance Fields for Unconstrained Photo Collections". In: *arXiv*. 2020.

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

[79] Leonard McMillan and Gary Bishop. "Plenoptic Modeling: An Image-Based Rendering System". In: *SIGGRAPH*. 1995.

[80] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. "Occupancy Networks: Learning 3D Reconstruction in Function Space". In: *CVPR*. 2019.

[81] Mateusz Michalkiewicz, Jhony K Pontes, Dominic Jack, Mahsa Baktashmotlagh, and Anders Eriksson. "Implicit surface representations as layers in neural networks". In: *ICCV*. 2019.

[82] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis". In: *arXiv preprint arXiv:2003.08934* (2020).

[83] Ben Mildenhall, Pratul P. Srinivasan, Rodrigo Ortiz-Cayon, Nima K. Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. "Local Light Field Fusion: Practical View Synthesis with Prescriptive Sampling Guidelines". In: *ACM Transactions on Graphics (SIGGRAPH)* (2019).

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

[85] Michael Niemeyer and Andreas Geiger. "GIRAFFE: Representing Scenes as Compositional Generative Neural Feature Fields". In: *arXiv* (2020).

[86] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. "Differentiable Volumetric Rendering: Learning Implicit 3D Representations without 3D Supervision". In: *CVPR*. 2019.

[87] Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. "Mitsuba 2: A Retargetable Forward and Inverse Renderer". In: *ACM Transactions on Graphics (SIGGRAPH Asia)* (2019).

[88] Roman Novak, Lechao Xiao, Jiri Hron, Jaehoon Lee, Alexander A. Alemi, Jascha Sohl-Dickstein, and Samuel S. Schoenholz. "Neural Tangents: Fast and Easy Infinite Neural Networks in Python". In: *ICLR* (2020).

[89] Michael Oechsle, Lars Mescheder, Michael Niemeyer, Thilo Strauss, and Andreas Geiger. "Texture fields: Learning texture representations in function space". In: *ICCV*. 2019.

[90] Rodrigo Ortiz-Cayon, Abdelaziz Djelouah, and George Drettakis. "A Bayesian Approach for Selective Image-Based Rendering using Superpixels". In: *International Conference on 3D Vision (3DV)*. 2015.

[91] Ryan S. Overbeck, Daniel Erickson, Daniel Evangelakos, Matt Pharr, and Paul Debevec. "A System for Acquiring, Processing, and Rendering Panoramic Light Field Stills for Virtual Reality". In: *SIGGRAPH Asia*. 2018.

[92] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. "DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation". In: *CVPR*. 2019.

[93] Keunhong Park, Utkarsh Sinha, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Steven M. Seitz, and Ricardo Martin-Brualla. "Deformable Neural Radiance Fields". In: (2020).

[94] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *NeurIPS*. 2019.

[95] Eric Penner and Li Zhang. "Soft 3D Reconstruction for View Synthesis". In: *SIGGRAPH Asia*. 2017.

[96] Thomas Porter and Tom Duff. "Compositing Digital Images". In: *SIGGRAPH*. 1984.

[97] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. "D-NeRF: Neural Radiance Fields for Dynamic Scenes". In: *arXiv preprint arXiv:2011.13961* (2020).

[98] Weichao Qiu, Fangwei Zhong, Yi Zhang, Siyuan Qiao, Zihao Xiao, Tae Soo Kim, Yizhou Wang, and Alan Yuille. "UnrealCV: Virtual Worlds for Computer Vision". In: *ACM Multimedia Open Source Software Competition*. 2017.

[99] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Dräxler, Min Lin, Fred A. Hamprecht, Yoshua Bengio, and Aaron C. Courville. "On the Spectral Bias of Neural Networks". In: *ICML*. 2018.

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

[101] Daniel Rebain, Wei Jiang, Soroosh Yazdani, Ke Li, Kwang Moo Yi, and Andrea Tagliasacchi. "DeRF: Decomposed Radiance Fields". In: *arXiv preprint arXiv:2011.12490* (2020).

[102] Shunsuke Saito, Zeng Huang, Ryota Natsume, Shigeo Morishima, Angjoo Kanazawa, and Hao Li. "PIFu: Pixel-Aligned Implicit Function for High-Resolution Clothed Human Digitization". In: *ICCV*. 2019.

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

[104] Katja Schwarz, Yiyi Liao, Michael Niemeyer, and Andreas Geiger. "GRAF: Generative Radiance Fields for 3D-Aware Image Synthesis". In: *NeurIPS*. 2020.

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

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

[107] Heung-Yeung Shum and Sing Bing Kang. "A Review of Image-Based Rendering Techniques". In: *Proceedings of Visual Communications and Image Processing*. 2000.

[108] Sudipta Sinha, Johannes Kopf, Michael Goesele, Daniel Scharstein, and Richard Szeliski. "Image-Based Rendering for Scenes with Reflections". In: *SIGGRAPH*. 2012.

[109] Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Nießner, Gordon Wetzstein, and Michael Zollhöfer. "DeepVoxels: Learning Persistent 3D Feature Embeddings". In: *CVPR*. 2019.

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

[111] Shuran Song, Fisher Yu, Andy Zeng, Angel X Chang, Manolis Savva, and Thomas Funkhouser. "Semantic Scene Completion from a Single Depth Image". In: *CVPR*. 2017.

[112] Pratul P. Srinivasan, Boyang Deng, Xiuming Zhang, Matthew Tancik, Ben Mildenhall, and Jonathan T. Barron. "NeRV: Neural Reflectance and Visibility Fields for Relighting and View Synthesis". In: *arXiv* (2020).

[113] Pratul P. Srinivasan, Richard Tucker, Jonathan T. Barron, Ravi Ramamoorthi, Ren Ng, and Noah Snavely. "Pushing the Boundaries of View Extrapolation with Multiplane Images". In: *CVPR*. 2019.

[114] Pratul P. Srinivasan, Tongzhou Wang, Ashwin Sreelal, Ravi Ramamoorthi, and Ren Ng. "Learning to Synthesize a 4D RGBD Light Field from a Single Image". In: *ICCV*. 2017.

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

[116] Rahul Swaminathan, Sing Bing Kang, Richard Szeliski, Antonio Criminisi, and Shree K. Nayar. "On the Motion and Appearance of Specularities in Image Sequences". In: *ECCV*. 2002.

[117] Richard Szeliski and Polina Golland. "Stereo matching with transparency and matting". In: *ICCV*. 1998.

[118] Matthew Tancik, Ben Mildenhall, Terrance Wang, Divi Schmidt, Pratul P. Srinivasan, Jonathan T. Barron, and Ren Ng. "Learned Initializations for Optimizing Coordinate-Based Neural Representations". In: *arXiv* (2020).

[119] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. "Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains". In: *NeurIPS*. 2020.

[120] Alex Trevithick and Bo Yang. "GRF: Learning a General Radiance Field for 3D Scene Representation and Rendering". In: *arXiv preprint arXiv:2010.04595* (2020).

[121] Shubham Tulsiani, Tinghui Zhou, Alexei A. Efros, and Jitendra Malik. "Multi-view Supervision for Single-view Reconstruction via Differentiable Ray Consistency". In: *CVPR*. 2017.

[122] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need". In: *NeurIPS*. 2017.

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

[124] 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.

[125] Gordon Wetzstein, Douglas Lanman, Wolfgang Heidrich, and Ramesh Raskar. "Layered 3D: Tomographic Image Synthesis for Attenuation-based Light Field and High Dynamic Range Displays". In: *SIGGRAPH*. 2011.

[126] Gordon Wetzstein, Douglas Lanman, Matthew Hirsch, and Ramesh Raskar. "Tensor Displays: Compressive Light Field Synthesis using Multilayer Displays with Directional Backlighting". In: *SIGGRAPH*. 2012.

[127] Bennett Wilburn, Neel Joshi, Vaibhav Vaish, Eino-Ville Talvala, Emilio Antunez, Adam Barth, Andrew Adams, Marc Levoy, and Mark Horowitz. "High Performance Imaging Using Large Camera Arrays". In: *SIGGRAPH*. 2005.

[128] Daniel N. Wood, Daniel I. Azuma, Ken Aldinger, Brian Curless, Tom Duchamp, David H. Salesin, and Werner Stuetzle. "Surface Light Fields for 3D Photography". In: *SIGGRAPH*. 2000.

[129] Gaochang Wu, Mandan Zhao, Liangyong Wang, Qionghai Dai, Tianyou Chai, and Yebin Liu. "Light Field Reconstruction Using Deep Convolutional Network on EPI". In: *CVPR*. 2017.

[130] Wenqi Xian, Jia-Bin Huang, Johannes Kopf, and Changil Kim. "Space-time Neural Irradiance Fields for Free-Viewpoint Video". In: *arXiv preprint arXiv:2011.12950* (2020).

[131] Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. "Self-attention with Functional Time Representation Learning". In: *NeurIPS* (2019).

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

[133] Lior Yariv, Yoni Kasten, Dror Moran, Meirav Galun, Matan Atzmon, Basri Ronen, and Yaron Lipman. "Multiview Neural Surface Reconstruction by Disentangling Geometry and Appearance". In: *NeurIPS* (2020).

[134] Henry Wing Fung Yeung, Junhui Hou, Jie Chen, Yuk Ying Chung, and Xiaoming Chen. "Fast Light Field Reconstruction with Deep Coarse-to-Fine Modeling of Spatial-Angular Clues". In: *ECCV*. 2018.

[135] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. "pixelNeRF: Neural Radiance Fields from One or Few Images". In: (2020).

[136] Cha Zhang and Tsuhan Chen. "Spectral Analysis for Sampling Image-Based Rendering Data". In: *IEEE Transactions on Circuits and Systems for Video Technology*. 2003.

[137] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. "The Unreasonable Effectiveness of Deep Features as a Perceptual Metric". In: *CVPR*. 2018.

[138] Zhoutong Zhang, Yebin Liu, and Qionghai Dai. "Light Field from Micro-Baseline Image Pair". In: *CVPR*. 2015.

[139] Ellen D. Zhong, Tristan Bepler, Joseph H. Davis, and Bonnie Berger. "Reconstructing continuous distributions of 3D protein structure from cryo-EM images". In: *ICLR*. 2020.

[140] Tinghui Zhou, Richard Tucker, John Flynn, Graham Fyffe, and Noah Snavely. "Stereo Magnification: Learning View Synthesis using Multiplane Images". In: *SIGGRAPH*. 2018.