Towards City-Scale Neural Rendering

Haithem Turki

CMU-CS-24-132
July 2024

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

*Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy.*

This research was sponsored by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior/Interior Business Center (DOI/IBC) contract number 140D0423C0074 and the National Science Foundation under grant number CNS-2106862. This research was also sponsored by the CMU Argo AI Center for Autonomous Vehicle Research, the Living Edge Lab, the Open Edge Computing Initiative at Carnegie Mellon University, and gifts to faculty from Bosch Research.

Keywords: Computer Vision, Machine Learning, 3D Deep Learning, Neural Rendering, Novel View Synthesis

To Ayden; may your life be full of wonders

Abstract

Advances in neural rendering techniques have led to significant progress towards photo-realistic novel view synthesis. When combined with increases in data processing and compute capability, this promises to unlock numerous VR applications, including virtual telepresence, search and rescue, and autonomous driving. Large-scale virtual reality, long the domain of science fiction [31, 62], feels markedly more tangible.

This thesis explores the frontier of large-scale neural rendering by building upon Neural Radiance Fields (NeRFs) [118], a family of methods attracting attention due to their state-of-the-art rendering quality and conceptual simplicity. Since its inception, at least 3,000 papers have been proposed in less than three years by research groups across the world across numerous use cases [135]. However, many shortcomings remain. The first is scale itself. Only a handful of existing methods capture scenes larger than a single object or room. Those that do only handle static reconstruction, which limits their applicability. Another is speed, as rendering falls below interactive thresholds. Current acceleration methods remain too slow or degrade quality at high resolution. Quality is a third issue, as NeRF assumes ideal viewpoint conditions that are unrealistic in practice and degrades when they are violated.

We first explore scaling within the context of static reconstruction. We design a sparse network structure that specializes parameters to different regions of the scene that can be trained in parallel, allowing us to scale linearly as we increase model capacity (vs quadratically in the original NeRF), and reconstruct urban-scale environments orders of magnitude larger than prior work. We then address dynamic reconstruction of entire cities, and build the largest dynamic NeRF representation to date. To accelerate rendering, we improve sampling efficiency through a hybrid surface-volume representation that encourages the model to represent as much of the world as possible through surfaces (which require few samples per ray) while maintaining the freedom to render transparency and finer details (which pure surface representations struggle to capture). We finally propose a fast anti-aliasing method that greatly improves rendering quality when training with data collected from freeform camera trajectories. Importantly, our method incurs a minimal performance overhead and is compatible with the scale and speed improvements previously mentioned.

Acknowledgments

I'm extremely grateful to my advisor Deva Ramanan for his constant guidance and mentorship throughout my PhD. I couldn't think of a better role model to learn from - our discussions have helped me become a much better researcher and communicator. His passion for knowledge and discovery are truly infectious and have deeply shaped my own interests and worldview. And perhaps most importantly, he working with him has been incredibly fun! I'm also indebted to Mahadev Satyanarayanan (Satya) for his guidance and support while I was first finding my feet at CMU and for convincing me to leave Tokyo for Pittsburgh in the first place.

I would like to thank my committee members - Shubham Tulsiani, Jessica Hodgins, Martial Hebert, and Jon Barron for their invaluable feedback throughout my thesis journey and Angjoo Kanazawa for her guidance during my thesis proposal. I deeply appreciate the time and effort spent on my behalf and their work is the source of great inspiration in my research.

I would also like to convey my gratitude towards my mentors and collaborators during my internships at Meta and Argo AI - Christian Richardt, Michael Zollhöfer, Vasu Agrawal, Samuel Rota Bulò, Lorenzo Porzi, Peter Kontschieder, Li Ma, Francesco Ferroni, and Benjamin Wilson. Their assistance has been immensely helpful in overcoming numerous technical roadblocks, enhancing my presentation skills, and becoming an overall better researcher.

I got the chance to collaborate with and learn from great colleagues during my time at CMU - Jason Zhang, Gengshan Yang, Arun Vasudevan, Kangle Deng, Neehar Peri, Jeff Tan, Tarasha Khurana, Zhiqiu Lin, Nathaniel Chodosh, Gautam Gare, Ravi Mullapudi, Jonathon Luiten, Chonghyuk Song, Shilpa George, Ziqiang Feng, Roger Iyengar, Junjue Wang, Jan Harkes, Tom Eiszler, Padmanabhan Pillai, Jim Blakely, and many others. Interacting with them all was a highlight of my time at CMU and I'm very excited for our paths to cross again in the future.

I was lucky to learn from great teachers during my time at Stanford - Alex Aiken, Daphne Koller, and Pawan Mudigonda. The sense of discovery I felt when exploring research within Alex's group during my Masters became a principal driving factor that led to me returning to academia from industry and pursuing a PhD. I'm also deeply appreciative of my supervisors at Palantir Technologies - Jasjit Grewal, Mark Elliot, and Stephen Cohen, who were supportive of my decision and made the transition as smooth as possible.

I would finally like to thank my family and friends, especially my parents Neila and Lassaad and my wife Helen, whose support has been unwavering through the last five years. Helen has truly been my anchor throughout this journey - amongst other things, she should really have been better credited for her help during the countless iterations it took to gather drone footage for the Mill 19 dataset. Without her, none of this would have been remotely possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Recent technological advancements have brought the promise of virtual reality and creation of large-scale virtual worlds, long the domain of science fiction [31, 62], closer than ever into our everyday lives. Although many challenges remain ahead of world-scale VR, current developments are already making a notable near-term impact across numerous industries. As skilled trade workers depart the workforce in unprecedented numbers, virtual training is becoming an increasingly attractive solution to reskilling the workforce relative to on-the-job training that is often ineffective due to time and resource constraints [3, 7, 10]. The imminent introduction of photo-realistic virtual avatars, such as those developed by Meta [4], promises to revolutionize our social interactions and the future of remote work. As autonomous vehicles expand their commercial footprint [11], closed-loop simulators that mimic complex environmental interactions in high fidelity are increasingly essential to training adequately robust agents [9, 207].

Amongst the many research efforts needed to advance the frontier of virtual reality, including the hardware design of ergonomic headsets, accurate tracking systems, and realistic haptic feedback, the task of efficiently rendering photo-realistic virtual worlds is a long-standing problem. Classical rendering pipelines achieve photo-realism either via painstakingly hand-crafted scene assets, which introduces expensive and tedious manual work from artists as a bottleneck, or through the estimation of various model parameters (camera, geometry, material and light properties) from real-world data, also known as *inverse rendering* [38, 39, 69, 97, 112]. The predefined physical models used in classical inverse rendering often struggle to accurately reproduce the full complexity of real-world physical interactions due to mathematical and computational constraints.

As an alternative to classical rendering, which projects 3D content onto 2D images, image-based rendering methods generate novel views by transforming a reference set of images, typically by warping input pixels into a novel view based on projected geometry and camera poses [34, 158]. Although these approaches can generate high-quality renderings, they often require closely sampled input images, which severely limits their applicability to large-scale environments [133].

More recently, the advent of neural rendering [172], an emerging field that uses neural networks to generate controllable, high-quality scene representations from input images and videos,

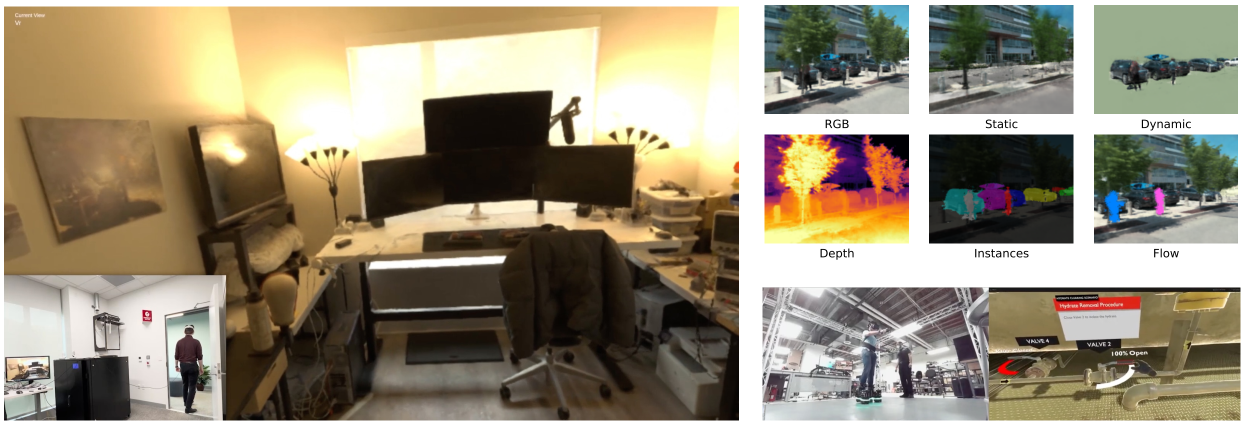


Figure 1.1: **Applications.** Current virtual world applications include walkthrough VR and virtual telepresence [200] **(left)**, closed-loop simulations for autonomous driving across different scene decompositions and modalities [179] **(top right)**, and virtual training of maintenance workflows in the energy industry [5] **(bottom right)**.

has seen considerable research activity. Numerous neural rendering approach address the problem of novel view synthesis [43, 122, 159], of which Neural Radiance Fields (NeRFs) [118] has emerged as the new state-of-the-art method due to their state-of-the-art rendering quality, ease of training, and conceptual simplicity. Since their introduction in 2020, over 3,000 follow-up works has been proposed [135] from research groups around the world targeting various applications including robotics [84, 211, 212], astronomy [93], tomography [147], and content creation [101, 132]. A natural question thus emerges - can NeRFs help facilitate the construction of large-scale virtual worlds?

1.2 Challenges

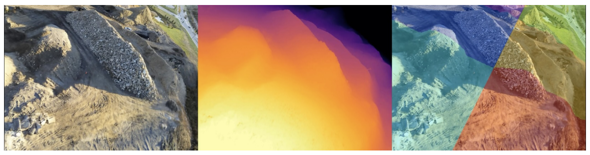
Although NeRFs can generate photo-realistic renderings under ideal conditions, several challenges limit their suitability when rendering virtual worlds, which we group along three principal axes:

Scale. The original NeRF [118] concerns itself with single-object or forward-facing scenes, and the vast majority of follow-up works limit their scope to object-scale or room-scale captures. The few larger-scale variants currently proposed [141, 167, 197] handle only static captures, which limits their applicability to real-world use cases.

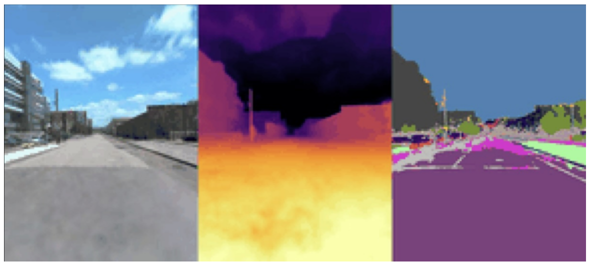
Speed. Conventional NeRF rendering is very slow to render, taking minutes to generate a single 1080p image. This is far too slow for interactivity - conventional virtual-reality headsets such as the Meta Quest 2 require rendering at 2K$\times$2K resolution at over 36 FPS to achieve acceptable user latency. Accelerated NeRF variants such as Instant-NGP [120], TensoRF [26], K-Planes [50], and Nerfacto [168] significantly improve rendering speed, but still reach <5 FPS at 2K$\times$2K resolution. Other methods achieve higher frame-rates by precomputing model outputs [28, 58, 68, 216] or via fewer model queries per pixel [65, 91, 121, 131], but noticeably degrade rendering quality.

Quality. NeRF methods often assume constraints that are unrealistic for real-world captures, such as entirely static scenes (with static lighting), and only observing objects from a roughly constant camera distance [16]. Furthermore, as NeRF methods are usually optimized on a per-scene basis, they are unable to hallucinate accurate geometry in under-observed regions. This is especially problematic within large-scale dynamic worlds where it is impossible to densely sample every location at every time step.

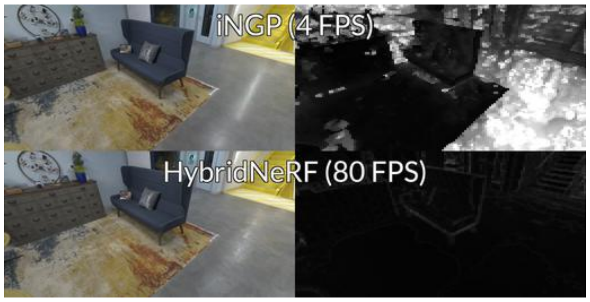
1.3 Contributions



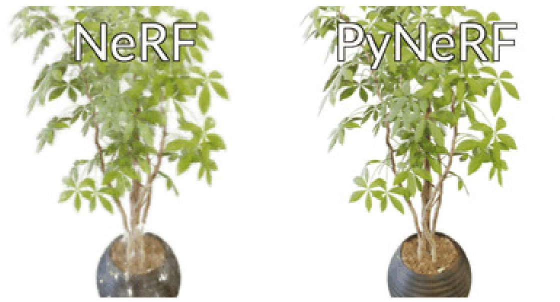
Static Reconstruction of City Blocks
(Chapter 2)



Dynamic City-Scale Reconstruction
(Chapter 3)



Real-Time Rendering at VR Resolution
(Chapter 4)



Fast Anti-Aliasing for Neural Radiance Fields
(Chapter 5)

Figure 1.2: **Thesis contributions.** We build upon NeRF and design methods suitable for city-scale neural rendering. We explore how to scale neural representations, first to build static reconstructions of city blocks (Chapter 2), and then to generate dynamic represntations of entire cities (Chapter 3). We then enable real-time rendering at VR resolution in Chapter 4. We finally improve rendering quality via a fast anti-aliasing method in Chapter 5.

This thesis aims to develop methods that address NeRF's shortcomings along scale, speed, and quality, thus facilitating their suitability towards city-scale neural rendering.

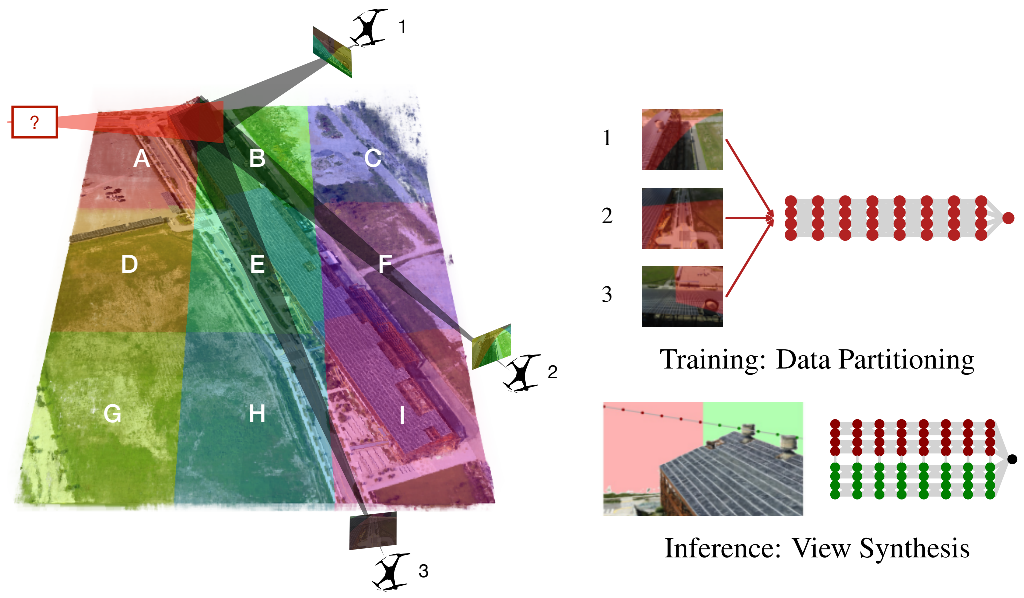


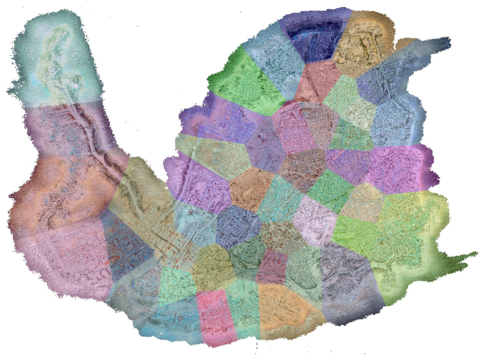
Figure 1.3: **Data parallelism for scalable training.** Mega-NeRF [178] decomposes a scene into a set of spatial cells (**left**), learning a separate NeRF submodule for each in parallel. It trains each submodule via a geometric clustering algorithm that only makes use of pixels whose rays intersect that spatial cell (**top right**). For example, pixels from image 2 are added to the trainset of cells A, B, and F, reducing the size of each training set by 10×. To render new views, Mega-NeRF makes use of standard raycasting and point sampling, but queries the encompassing submodule for each sampled point (**bottom right**). It also makes use of temporal coherence by caching occupancy and color values from nearby previous views to accelerate rendering by 40× relative to the original NeRF.

1.3.1 Static Reconstruction of City Blocks

We begin by extending NeRFs from single-object or room-scale settings to modeling buildings or entire city blocks, which requires increasing model capacity as the scene footprint increases. The original NeRF encodes the scene representation into a monolithic MLP, which causes training and rendering time to increase quadratically as model capacity increases.

Mega-NeRF [178] instead proposes using spatial partitioning to enable *data parallelism for scalable training* (Fig. 1.3). It introduces a sparse network structure that scales *linearly* with model capacity, where parameters are specialized to different regions of the scene, and a simple geometric clustering algorithm that partitions training pixels into different NeRF submodules that can be trained in parallel. Mega-NeRF further exploits spatial locality at render time to implement a just-in-time visualization technique that renders 40× faster than conventional NeRF raycasting, facilitating interactive fly-throughs of captured environments. We evaluate Mega-NeRF on existing datasets [32, 103] along with our publicly released Mill 19 dataset (now used across the research community [83, 98, 102, 116, 202, 226]) and measure a 3× speedup in training time and a 12% PSNR improvement on average.

1.3.2 Dynamic City-Scale Reconstruction



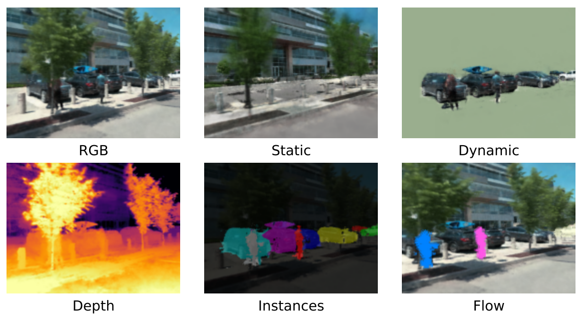


Figure 1.4: **Open-world reconstructions of dynamic cities.** SUDS [179] scales neural reconstructions to city scale by dividing the area into multiple cells and training hash table representations for each. We show the full city-scale reconstruction (**left**) and the derived representations (**right**). Unlike prior methods, SUDS handles dynamism across multiple videos, disentangling dynamic objects from static background and modeling shadow effects. SUDS uses unlabeled inputs to learn scene flow and semantic predictions, enabling category- and object-level scene manipulation.

We then extend neural radiance fields to dynamic large-scale urban scenes, and expand our geospatial footprint from city blocks to entire cities. Prior large-scale NeRF works [116, 141, 167, 197, 202], including Mega-NeRF [178], target purely static captures. Existing dynamic NeRF methods [41, 55, 90, 96, 126, 196] only operate on short video sequences (<1 minute) and have memory requirements that scale on a per-frame and/or per-object basis, which quickly becomes prohibitive. They also often require supervision via 3D bounding boxes and panoptic labels, obtained manually or via category-specific models, which are expensive to reliably obtain at scale in the wild.

SUDS [179] introduces two key innovations as a step towards truly open-world reconstructions of dynamic cities: (a) it factorizes the scene into three separate hash table data structures [120] (which are quicker to train and render than MLPs used by previous NeRF methods) to efficiently encode static, dynamic, and far-field radiance fields, and (b) it makes use of unlabeled target signals consisting of RGB images, sparse LiDAR, off-the-shelf self-supervised 2D descriptors, and 2D optical flow. These unlabeled annotations are far easier to reliably at scale relative to accurate 3D bounding boxes and/or panoptic segmentations, and enable SUDS to decompose dynamic scenes into the static background, individual objects, and their motions via photometric, geometric, and feature-metric reconstruction losses (Fig. 1.4).

We evaluate SUDS against autonomous driving logs collected across the city of Pittsburgh. Both the scale and the nature of this data is far more challenging than drone-collected datasets [103, 178] that are commonly used to evaluate other large-scale NeRF methods, which are largely static and captured over the course of several hours. In contrast, this street-level footage spans months/years, contains numerous moving vehicles and pedestrians, and is inconsistent in areas covered by overlapping logs due to transient parked cars and differing seasons and weather conditions. We show that SUDS can be scaled to tens of thousands of objects across 1.2 million frames from 1700 videos spanning geospatial footprints of hundreds of kilometers, (to our knowledge) the largest dynamic NeRF built to date.

1.3.3 Real-Time Rendering at VR Resolution

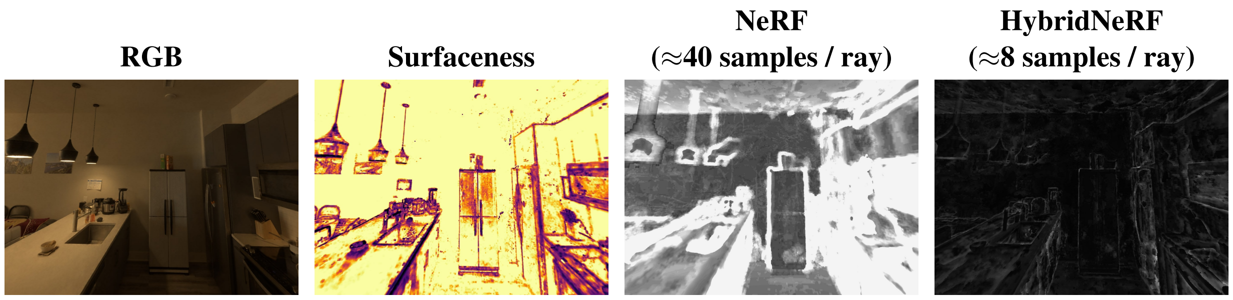


Figure 1.5: **Adaptive volumetric surfaces.** HybridNeRF [181] trains a hybrid surface–volume representation via *surfaceness* parameters that allow it to render most of the scene with few samples. Eikonal loss is tracked during training as surfaceness increases to avoid degrading quality near fine and translucent structures (such as wires). In the two right-most panels, we visualize the number of samples per ray (brighter is higher).

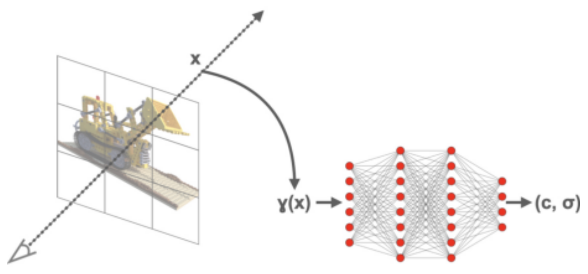
After constructing methods that allow us to efficiently train neural scene representations at scale, we explore how to efficiently render our trained representations. We specifically target real-time rendering at virtual-reality resolutions, which we define to be above 36 FPS at 2K$\times$2K resolution.

Although both Mega-NeRF [178] and SUDS [179] render faster than the original NeRF, they remain below the required threshold for virtual reality, as do other popular accelerated NeRF variants such as Instant-NGP [120], TensoRF [26], K-Planes [50], and Nerfacto [168]. Other methods do achieve acceptable speed by precomputing model outputs into a finite-resolution caching structure such as an octree [28, 58, 68, 216], by improving NeRF's sample efficiency [65, 91, 121, 131], or via rasterization-based techniques [81, 209], but significantly degrade rendering quality.

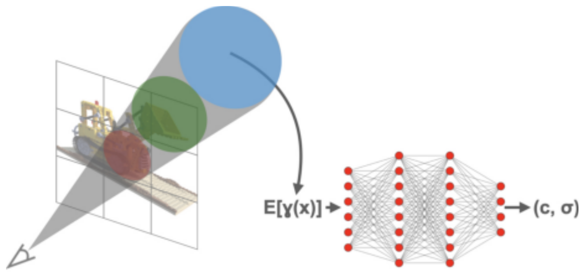
HybridNeRF [181] builds upon NeRF's raycasting paradigm and achieves real-time framerates without degrading quality. We start from the observation that NeRF's volume rendering method, which represents everything as volumes that can require many model queries per ray/pixel, is computationally expensive and often unnecessary since most of the world can instead be efficiently defined via surfaces (requiring as little as one sample per ray at the limit). Signed distance functions (SDFs), which were originally proposed to improve the geometry quality of NeRFs via regularization [125, 185, 208], can therefore *also* be used to dramatically increase efficiency by requiring fewer samples per ray. However, they often struggle to reconstruct scenes with thin structures or view-dependent effects, such as reflections and translucency. HybridNeRF's key innovation is to define a locally varying *surfaceness* parameter that allows it to model most (>95%) of the scene as thin surfaces (needing few samples) while using with more samples near fine and semi-opaque structures (and therefore maintain a high level of visual fidelity) (Fig. 1.5).

We evaluate HybridNeRF against the challenging Eyeful Tower dataset [200], designed for walkthrough virtual reality, along with other commonly used view synthesis datasets. When comparing to state-of-the-art baselines, including recent rasterization-based approaches [81], we improve error rates by 15–30% while rendering over 36 FPS at 2K×2K resolution.

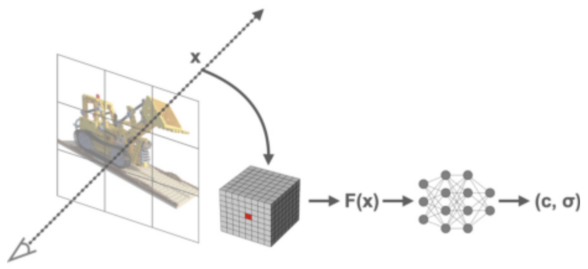
1.3.4 Fast Anti-Aliasing for Neural Radiance Fields



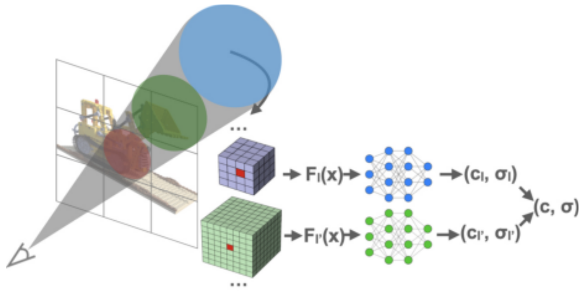
(a) NeRF



(b) Mip-NeRF



(c) Grid Methods (eg: iNGP)



(d) PyNeRF

Figure 1.6: **Comparison of rendering methods.** **(a)** NeRF [118] traces a ray from the camera's center of projection through each pixel and samples points $\mathbf{x}$ along each ray. Sample locations are then encoded with a positional encoding to produce a feature $\gamma(\mathbf{x})$ that is fed into an MLP. **(b)** Mip-NeRF [16] instead reasons about *volumes* by defining a 3D conical frustum per camera pixel. It splits the frustum into sampled volumes, approximates them as multivariate Gaussians, and computes the integral of the positional encodings of the coordinates contained within the Gaussians. Similar to NeRF, these features are then fed into an MLP. **(c)** Accelerated NeRF methods, such as iNGP [120], sample points as in NeRF, but do not use positional encoding and instead featurize each point by interpolating between vertices in a feature grid. These features are then passed into a much smaller MLP, which greatly accelerates training and rendering. **(d)** PyNeRF [180] also uses feature grids, but reasons about volumes by training separate models at different scales (similar to a mipmap). It calculates the area covered by each sample in world coordinates, queries the models at the closest corresponding resolutions, and interpolates their outputs.

We finally investigate how to improve the rendering quality of our representations. As our goal is to efficiently render large-scale virtual worlds, we explore solutions that are compatible with the scaling and speed improvements described in previous chapters.

We adress aliasing that occurs when training and rendering with freeform camera trajectories. Most NeRF methods assume that training and test-time cameras capture scene content from a roughly constant distance. They degrade and render blurry views and aliasing artifacts in less constrained settings. This is because NeRF raycasting is scale-unaware - it samples points along an infinitesimally thin ray and does not consider the area viewed by each pixel.

As a solution, Mip-NeRF [16] proposes to reason about the volume of the conical frustum defined by a camera pixel. It leverages this insight to derive the approximate integral of all NeRF-encoded coordinates contained within the conical frustum. However, this integral is only valid for methods that apply positional encoding [118] to model inputs as in the original NeRF; most accelerated NeRF methods (including SUDS [179] and HybridNeRF [181]) do not. Zip-NeRF [18] also reasons about the volume of the conical frustum, but instead uses a multisampling strategy within the volume. This strategy is compatible with accelerated NeRF methods, but training and rendering speed decreases with the number of additional samples.

PyNeRF [180] takes inspiration from mipmaps [192] used in classical image processing and proposes a simple modification by training a pyramid of NeRFs that divide the scene at different spatial resolutions. Similar to Mip-NeRF, PyNeRF samples volumes within conical frustums - it then simply maps larger volumes to coarser NeRFs and finer volumes to finer NeRFs (Fig. 1.6).

This strategy can be easily applied to most existing accelerated NeRF methods and significantly improves rendering quality (reducing error rates by 20–90% across synthetic and unbounded real-world scenes) while incurring minimal performance overhead (as each individual NeRF in the pyramid is quick to evaluate). Compared to Mip-NeRF, it reduces error rates by 20% while training over 60$\times$ faster.

1.3.5 Excluded Research

Some of the research works that I have contributed to during my PhD are excluded from this thesis, which include other contributions within the neural rendering space and the broader design of scalable, efficient machine learning systems:

1. **SplatGym: Reconstruction of Dynamic Urban Scenes using 3D Gaussians and Simulation of Actors**
   Arun Balajee Vasudevan, Neehar Peri, **Haithem Turki**, Johan Vertens, Shubham Tulsiani, Deva Ramanan
   In submission
2. **SpecNeRF: Gaussian Directional Encoding for Specular Reflections** [111]
   Li Ma, Vasu Agrawal, **Haithem Turki**, Changil Kim, Chen Gao, Pedro V. Sander, Michael Zollhöfer, Christian Richardt
   Conference on Computer Vision and Pattern Recognition (CVPR), 2024
3. **Low-Bandwidth Self-Improving Transmission of Rare Training Data** [61]
   Shilpa George, **Haithem Turki**, Ziqiang Feng, Deva Ramanan, Padmanabhan Pillai, Mahadev Satyanarayanan
   Conference on Mobile Computing and Networking (MobiCom), 2023
4. **Accelerating Silent Witness Storage** [153]
   Mahadev Satyanarayanan, Ziqiang Feng, Shilpa George, Jan Harkes, Roger Iyengar, **Haithem**

Turki, Padmanabhan Pillai
IEEE Micro, 2022

5. **OpenRTiST: End-to-End Benchmarking for Edge Computing** [60]
   Shilpa George, Thomas Eiszler, Roger Iyengar, **Haithem Turki**, Ziqiang Feng, Junjue Wang, Padmanabhan Pillai, Mahadev Satyanarayanan
   IEEE Pervasive Computing, 2020
6. **Edge Computing for Legacy Applications** [152]
   Mahadev Satyanarayanan, Thomas Eiszler, Jan Harkes, **Haithem Turki**, Ziqiang Feng
   IEEE Pervasive Computing, 2020

Chapter 2

Static Reconstruction of City Blocks

The contents of this chapter were published as "Mega-NeRF: Scalable Construction of Large-Scale NeRFs for Virtual Fly-Throughs" in CVPR 2022

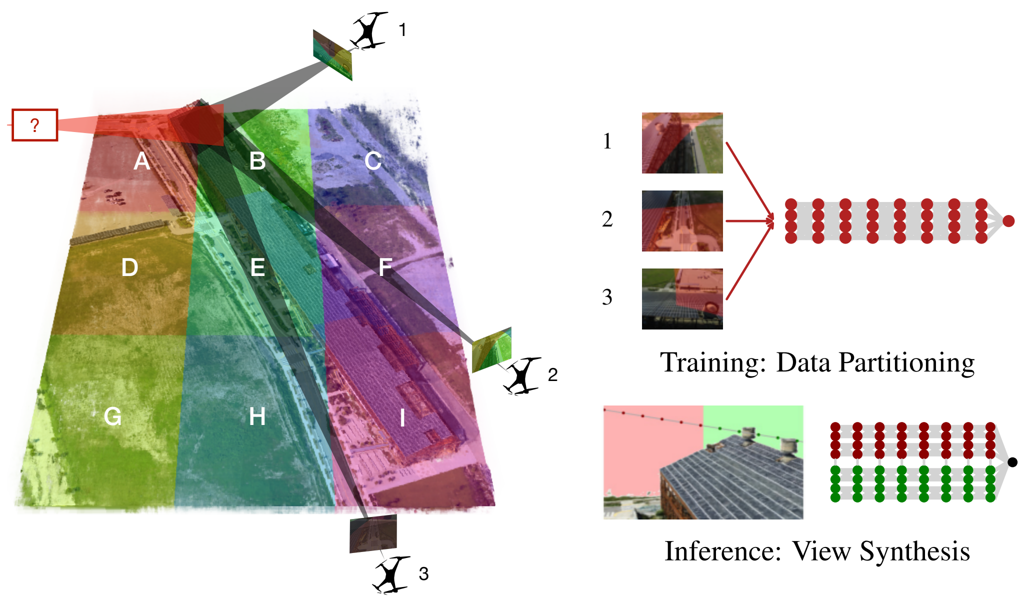


Figure 2.1: We scale neural reconstructions to massive urban scenes 1000× larger than prior work. To do so, Mega-NeRF [178] decomposes a scene into a set of spatial cells (**left**), learning a separate NeRF submodule for each. We train each submodule with geometry-aware pixel-data partitioning, making use of *only* those pixels whose rays intersect that spatial cell (**top right**). For example, pixels from image 2 are added to the trainset of cells A, B, and F, reducing the size of each trainset by 10×. To generate new views for virtual fly-throughs, we make use of standard raycasting and point sampling, but query the encompassing submodule for each sampled point (**bottom right**). To ensure view generation is near-interactive, we make use of temporal coherence by caching occupancy and color values from nearby previous views (Fig. 2.4).

2.1 Introduction

We first explore the scalability of NeRFs in static settings. The vast majority of existing methods explore single-object scenes, often captured indoors or from synthetic data. To our knowledge, Tanks and Temples [88] is the largest dataset used in NeRF evaluation, spanning 463 m^2 on average. In this work, we scale NeRFs to capture and interactively visualize urban-scale environments from drone footage that is orders of magnitude larger than any dataset to date, from 150,000 to over 1,300,000 m^2 per scene.

Search and Rescue. As a motivating use case, consider search-and-rescue, where drones provide an inexpensive means of quickly surveying an area and prioritizing limited first responder resources (e.g., for ground team deployment). Because battery life and bandwidth limits the ability to capture sufficiently detailed footage in real-time [44], collected footage is typically reconstructed into 2D "birds-eye-view" maps that support post-hoc analysis [191]. We imagine a future in which neural rendering lifts this analysis into 3D, enabling response teams to inspect the field as if they were flying a drone in real-time at a level of detail far beyond the achievable with classic Structure-from-Motion (SfM).

Challenges. Within this setting, we encounter multiple challenges. Firstly, applications such as search-and-rescue are time-sensitive. According to the National Search and Rescue Plan [2], "the life expectancy of an injured survivor decreases as much as 80 percent during the first 24 hours, while the chances of survival of uninjured survivors rapidly diminishes after the first 3 days." The ability to train a usable model within a few hours would therefore be highly valuable. Secondly, as our datasets are orders of magnitude larger than previously evaluated datasets (Table 2.1), model capacity must be significantly increased in order to ensure high visual fidelity, further increasing training time. Finally, although interactive rendering is important for fly-through and exploration at the scale we capture, existing real-time NeRF renderers either rely on pretabulating outputs into a finite-resolution structure, which scales poorly and significantly degrades rendering performance, or require excessive preprocessing time.

Mega-NeRF. In order to address these issues, we propose Mega-NeRF, a framework for training large-scale 3D scenes that support interactive human-in-the-loop fly-throughs. We begin by analyzing visibility statistics for large-scale scenes, as shown in Table 2.1. Because only a small fraction of the training images are visible from any particular scene point, we introduce a sparse network structure where parameters are specialized to different regions of the scene. We introduce a simple geometric clustering algorithm that partitions training images (or rather pixels) into different NeRF submodules that can be trained in parallel. We further exploit spatial locality at render time to implement a just-in-time visualization technique that allows for interactive fly-throughs of the captured environment.

Prior art. Our approach of using "multiple" NeRF submodules is closely inspired by the recent work of DeRF [137] and KiloNeRF [139], which use similar insights to accelerate *inference* (or rendering) of an existing, pre-trained NeRF. However, even obtaining a pre-trained NeRF for our scene scales is essentially impossible with current training pipelines. We demonstrate that modularity is vital for *training*, particularly when combined with an intelligent strategy for "sharding" training data into the appropriate modules via geometric clustering.

Contributions. We propose a reformulation of the NeRF architecture that sparsifies layer connections in a spatially-aware manner, facilitating efficiency improvements at training and

	Resolution	# Images	# Pixels/Rays	Scene Captured / Image
Synthetic NeRF - Chair	400 x 400	400	256,000,000	0.271
Synthetic NeRF - Drums	400 x 400	400	256,000,000	0.302
Synthetic NeRF - Ficus	400 x 400	400	256,000,000	0.582
Synthetic NeRF - Hotdog	400 x 400	400	256,000,000	0.375
Synthetic NeRF - Lego	400 x 400	400	256,000,000	0.205
Synthetic NeRF - Materials	400 x 400	400	256,000,000	0.379
Synthetic NeRF - Mic	400 x 400	400	256,000,000	0.518
Synthetic NeRF - Ship	400 x 400	400	256,000,000	0.483
T&T - Barn	1920 x 1080	384	796,262,400	0.135
T&T - Caterpillar	1920 x 1080	368	763,084,800	0.216
T&T - Family	1920 x 1080	152	315,187,200	0.284
T&T - Ignatius	1920 x 1080	263	545,356,800	0.476
T&T - Truck	1920 x 1080	250	518,400,000	0.225
Mill 19 - Building	4608 x 3456	1940	30,894,981,120	0.062
Mill 19 - Rubble	4608 x 3456	1678	26,722,566,144	0.050
Quad 6k	1708 x 1329	5147	11,574,265,679	0.010
UrbanScene3D - Residence	5472 x 3648	2582	51,541,512,192	0.059
UrbanScene3D - Sci-Art	4864 x 3648	3019	53,568,749,568	0.088
UrbanScene3D - Campus	5472 x 3648	5871	117,196,056,576	0.028

Table 2.1: Scene properties from the commonly used Synthetic NeRF and Tanks and Temples datasets (T&T) compared to our target datasets (**below**). Our targets contain an order-of-magnitude more pixels (and hence rays) than prior work. Moreoever, each image captures significantly less of the scene, motivating a modular approach where spatially-localized submodules are trained with a fraction of relevant image data.

	Resolution	# Images	# Pixels/Rays
Synthetic NeRF [118]	400 x 400	400	256,000,000
LLFF [117]	4032 x 3024	41	496,419,840
Light Field [220]	1280 x 720	214	195,910,200
Tanks and Temples [88]	1920 x 1080	283	587,658,240
Phototourism [78]	919 x 794	1708	1,149,113,846
Mill 19	4608 x 3456	1809	28,808,773,632
Quad 6k [32]	1708 x 1329	5147	11,574,265,679
UrbanScene3D [103]	5232 x 3648	3824	74,102,106,112

Table 2.2: Comparison of datasets commonly used in view synthesis (**above**) relative to those evaluated in our work (**below**). We average the resolution, number of images, and total number of pixels across each captured scene. We report statistics for Light Field and Tanks and Temples using the splits in [223] and [216] respectively. For Phototourism we average across the scenes used in [113].

rendering time. We then adapt the training process to exploit spatial locality and train the model subweights in a fully parallelizable manner, leading to a 3$\times$ improvement in training speed while exceeding the reconstruction quality of existing approaches. In conjunction, we evaluate existing fast rendering approaches against our trained Mega-NeRF model and present a novel method that exploits temporal coherence. Our technique requires minimal preprocessing, avoids the finite resolution shortfalls of other renderers, and maintains a high level of visual fidelity. We also present a new large-scale dataset containing thousands of HD images gathered from drone footage over 100,000 m^2 of terrain near an industrial complex.

2.2 Related Work

Fast rendering. Conventional NeRF rendering falls well below interactive thresholds. Plenoctree [216], SNeRG [68], and FastNeRF [58] speed up the process by storing precomputed non-view dependent model outputs into a separate data structure such as a sparse voxel octree. These renderers then bypass the original model entirely at render time by computing the final view-dependent radiance through a separate smaller multi-layer perceptron (MLP) or through spherical basis computation. Although they achieve interactivity, they suffer from the finite capacity of the caching structure and poorly capture low-level details at scale.

DeRF [137] decomposes the scene into multiple cells via spatial Voronoi partitioning. Each cell is independently rendered using a smaller MLP, accelerating rendering by 3$\times$ over NeRF. KiloNeRF [139] divides the scene into thousands of even smaller networks. Although similar in spirit to Mega-NeRF, these methods use spatial partitioning to speed up *inference* while we use it to enable *data parallelism* for scalable training. Both DeRF and KiloNERF are initialized with a single large network trained on all data which is then distilled into smaller networks for fast inference, increasing processing time by over 2$\times$ for KiloNeRF. Training on all available data is prohibitive at our scale. Instead, our crucial insight is to geometrically partition training pixels

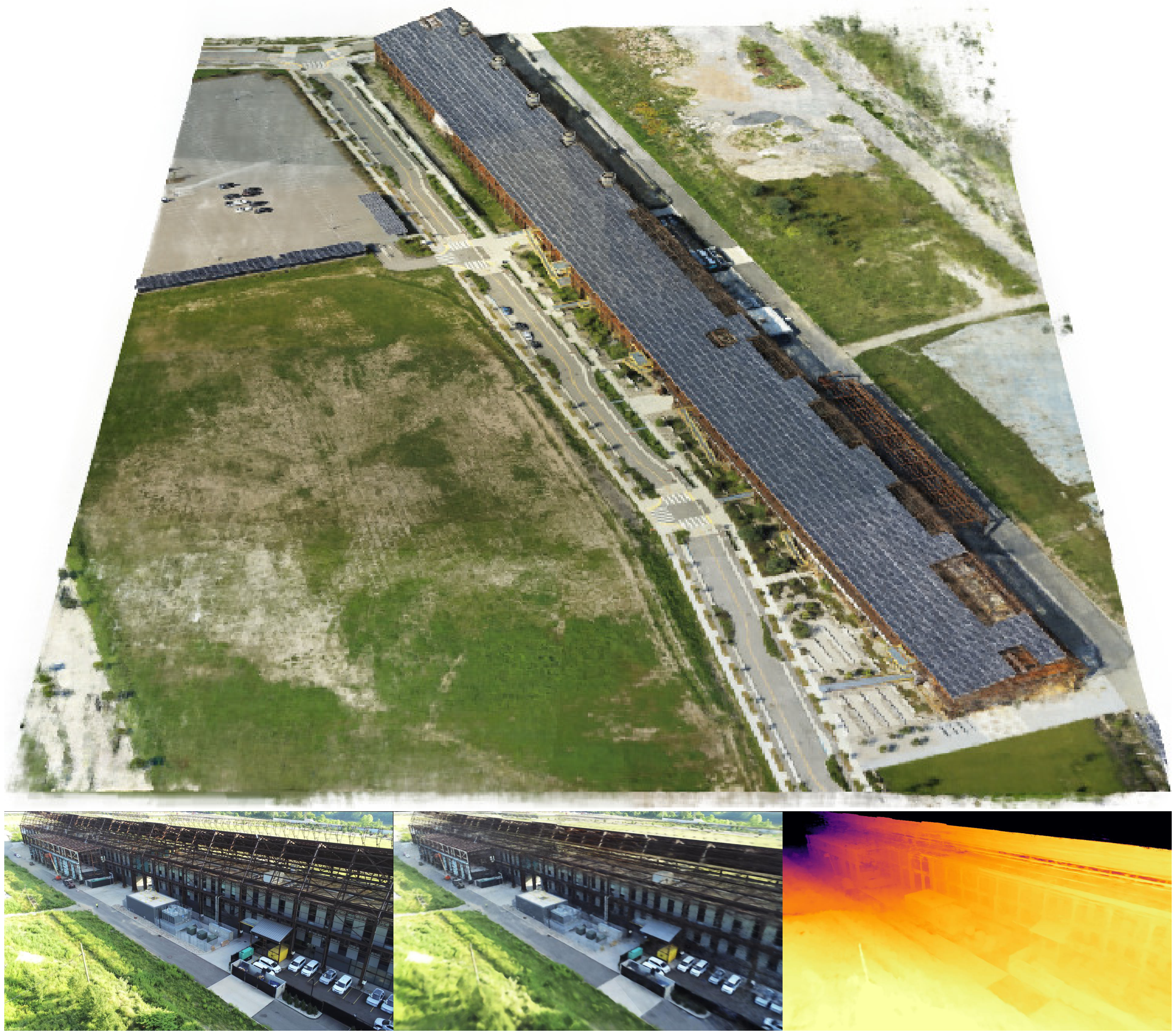


Figure 2.2: Visualization of Mill 19 by Mega-NeRF. The top panel shows a high-level 3D rendering of Mill 19 within our interactive visualizer. The bottom-left panel contains a ground truth image captured by our drone. The following two panels illustrate the model reconstruction along with the associated depth map.

into small data shards relevant for each submodule, which is essential for efficient training and high accuracy.

DONeRF [121] accelerates rendering by significantly reducing the number of samples queried per ray. To maintain quality, these samples are placed more closely around the first surface the ray intersects, similar to our guided sampling approach described in Sec. 2.3.3. In contrast to our method, DONeRF uses a separate depth oracle network trained against ground truth depth data.

Unbounded scenes. Although most NeRF-related work targets indoor areas, NeRF++ [223] handles unbounded environments by partitioning the space into a unit sphere foreground region that encloses all camera poses and a background region that covers the inverted sphere complement. A separate MLP model represents each area and performs ray casting independently before a final composition. Mega-NeRF employs a similar foreground/background partitioning although we further constrain our foreground and sampling bounds as described in Sec. 2.3.1.

NeRF in the Wild [113] augments NeRF's model with an additional transient head and learned per-image embeddings to better explain lighting differences and transient occlusions across images. Although it does not explicitly target unbounded scenes, it achieves impressive results against outdoor sequences in the Phototourism [78] dataset. We adopt similar appearance embeddings for Mega-NeRF and quantify its impact in Sec. 2.4.2.

Concurrent to us, Urban Radiance Fields [141] (URF), BungeeNeRF [197], and BlockNeRF [167] target urban-scale environments. URF makes use of lidar inputs, while CityNeRF makes use of multi-scale data modeling. Both methods can be seen as complementary to our approach, implying combining them with Mega-NeRF is promising. Most related to us is Block-NeRF [167], which decomposes a scene into spatial cells of fixed city blocks. Mega-NeRF makes use of geometry visibility reasoning to decompose the set of training pixels, allowing for pixels captured from far-away cameras to still influence a spatial cell (Fig. 4.1).

Training speed. Several works speed up model training by incorporating priors learned from similar datasets. PixelNeRF [215], IBRNet [186], and GRF [174] condition NeRF on predicted image features while Tancik et al. [166] use meta-learning to find good initial weight parameters that converge quickly. We view these efforts as complementary to ours.

Graphics. We note longstanding efforts within the graphics community covering interactive walkthroughs. Similar to our spatial partioning, Teller and Séquin [171] subdivide a scene into cells to filter out irrelevant geometry and speed up rendering. Funkhouser and Séquin [52] separately describe an adaptive display algorithm that iteratively adjusts image quality to achieve interactive frame rates within complex virtual environments. Our renderer takes inspiration from this gradual refinement approach.

Large-scale SfM. We take inspiration from previous large-scale reconstruction efforts based on classical Structure-from-Motion (SfM), in particular Agarwal et al's seminal "Building Rome in a Day," [13] which describes city-scale 3D reconstruction from internet-gathered data.

2.3 Approach

We first describe our model architecture in Sec. 2.3.1, then our training process in 2.3.2, and finally propose a novel renderer that exploits temporal coherence in 2.3.3.

2.3.1 Model Architecture

Background. We begin with a brief description of Neural Radiance Fields (NeRFs) [118]. NeRFs represent a scene within a continuous volumetric radiance field that captures both geometry and view-dependent appearance. NeRF encodes the scenes within the weights of a multilayer perceptron (MLP). At render time, NeRF projects a camera ray **r** for each image pixel and samples along the ray. For a given point sample p_i, NeRF queries the MLP at position $\mathbf{x}_i = (x, y, z)$ and ray viewing direction $\mathbf{d} = (d_1, d_2, d_3)$ to obtain opacity and color values σ_i and $\mathbf{c}_i = (r, g, b)$. It then composites a color prediction $\hat{C}(\mathbf{r})$ for the ray using numerical quadrature $\sum_{i=0}^{N-1} T_i(1 - \exp(-\sigma_i \delta_i))\,\mathbf{c}_i$, where $T_i = \exp(-\sum_{j=0}^{i-1} \sigma_j \delta_j)$ and δ_i is the distance between samples p_i and p_{i+1}. The training process optimizes the model by sampling batches R of image pixels and minimizing the loss function $\sum_{\mathbf{r}\in\mathcal{R}} \left\|C(\mathbf{r}) - \hat{C}(\mathbf{r})\right\|^2$. NeRF samples camera rays through a two-stage hierarchical sampling process and uses positional encoding to better capture high-frequency details. We refer the reader to the NeRF paper [118] for additional information.

Spatial partitioning. Mega-NeRF decomposes a scene into cells with centroids $\mathbf{n}_{\in\mathcal{N}} = (n_x, n_y, n_z)$ and initializes a corresponding set of model weights $f^{\mathbf{n}}$. Each weight submodule is a sequence of fully connected layers similar to the NeRF architecture. Similar to NeRF in the Wild [113], we associate an additional appearance embedding vector $l^{(a)}$ for each input image a used to compute radiance. This allows Mega-NeRF additional flexibility in explaining lighting differences across images which we found to be significant at the scale of the scenes that we cover. At query time, Mega-NeRF produces an opacity σ and color $\mathbf{c} = (r, g, b)$ for a given position **x**, direction **d**, and appearance embedding $l^{(a)}$ using the model weights $f^{\mathbf{n}}$ closest to the query point:

$$f^{\mathbf{n}}(\mathbf{x}) = \sigma \tag{2.1}$$

$$f^{\mathbf{n}}(\mathbf{x}, \mathbf{d}, l^{(a)}) = \mathbf{c} \tag{2.2}$$

$$\text{where } \mathbf{n} =_{n\in\mathcal{N}} \left\|n - \mathbf{x}\right\|^2 \tag{2.3}$$

Centroid selection. Although we explored several methods, including k-means clustering and uncertainty-based partitioning as in [205], we ultimately found that tessellating the scene into a top-down 2D grid worked well in practice. This method is simple to implement, requires minimal preprocessing, and enables efficient assignment of point queries to centroids at inference time. As the variance in altitude between camera poses in our scenes is small relative to the differences in latitude and longitude, we fix the height of the centroids to the same value.

Foreground and background decomposition. Similar to NeRF++ [223], we further subdivide the scene into a foreground volume enclosing all camera poses and a background covering the complementary area. Both volumes are modeled with separate Mega-NeRFs. We use the same 4D outer volume parameterization and raycasting formulation as NeRF++ but improve upon its unit sphere partitioning by instead using an ellipsoid that more tightly encloses the camera poses and relevant foreground detail. We also take advantage of camera altitude measurements to further refine the sampling bounds of the scene by terminating rays near ground level. Mega-NeRF thus avoids needlessly querying underground regions and samples more efficiently. Fig. 2.3 illustrates the differences between both approaches.

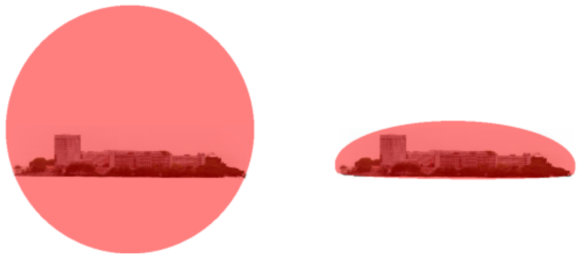


Figure 2.3: **Ray Bounds.** NeRF++ (**left**) samples within a unit sphere centered within and enclosing all camera poses to render its foreground component and uses a different methodology for the outer volume complement to efficiently render the background. Mega-NeRF (**right**) uses a similar background parameterization but models the foreground as an ellipsoid to achieve tighter bounds on the region of interest. It also uses camera altitude measurements to constrain ray sampling and not query underground regions.

2.3.2 Training

Spatial Data Parallelism. As each Mega-NeRF submodule is a self-contained MLP, we can train each in parallel with no inter-module communication. Crucially, as each image captures only a small part of the scene (Table 2.1), we limit the size of each submodule's trainset to only those potentially relevant pixels. Specifically, we sample points along the camera ray corresponding to each pixel for each training image, and add that pixel to the trainset for only those spatial cells it intersects (Fig. 4.1). In our experiments, this visibility partitioning reduces the size of each submodule's trainset by 10× compared to the initial aggregate trainset. This data reduction should be even more extreme for larger-scale scenes; when training a NeRF for North Pittsburgh, one need not add pixels of South Pittsburgh. We include a small overlap factor between cells (15% in our experiments) to further minimize visual artifacts near boundaries.

Spatial Data Pruning. Note that the initial assignment of pixels to spatial cells is based on camera positions, irrespective of scene geometry (because that is not known at initialization). Once NeRF gains a coarse understanding of the scene, one could further prune away irrelevant pixels/rays that don't contribute to a particular NeRF due to an intervening occluder. For example, in Fig. 4.1, early NeRF optimization might infer a wall in cell F, implying that pixels from image 2 can then be pruned from cell A and B. Our initial exploration found that this additional visibility pruning further reduced trainset sizes by 2×. We provide details in Sec. 5.4.6.

2.3.3 Interactive Rendering

We propose a novel interactive rendering method in addition to an empirical evaluation of existing fast renderers on top of Mega-NeRF in Sec. 2.4.3. In order to satisfy our search-and-rescue usecase, we attempt to: (a) preserve visual fidelity, (b) minimize any additional processing time beyond training the base model, and (c) accelerate rendering, which takes over 2 minutes for a 720p frame with normal ray sampling, to something more manageable.

Caching. Most existing fast NeRF renderers make use of cached precomputation to speed up rendering, which may not be effective at our scene scale. For example, Plenoctree [216] precomputes a cache of opacity and spherical harmonic coefficients into a sparse voxel octree.

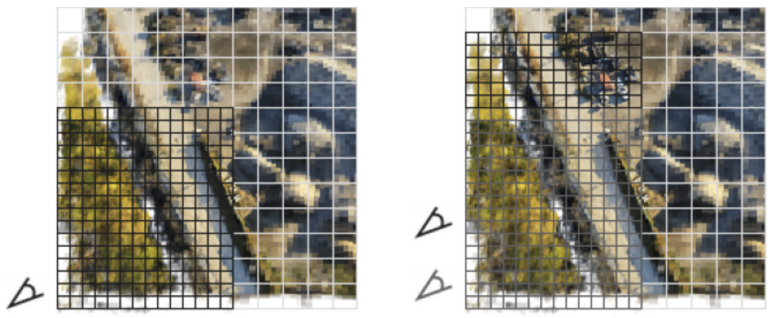

(a) Fixed Octree
(b) Dynamically Expanded Octree
(c) Reused Octree (next frame)

Figure 2.4: **Mega-NeRF-Dynamic.** Current renderers (such as Plenoctree [216]) cache precomputed model outputs into a fixed octree, limiting the resolution of rendered images (**a**). Mega-NeRF-Dynamic *dynamically* expands the octree based on the current position of the fly-through (**b**). Because of the temporal coherence of camera views, the next-frame rendering (**c**) can reuse of much of expanded octree.

Generating the entire 8-level octree for our scenes took an hour of computation and anywhere from 1 to 12 GB of memory depending on the radiance format. Adding a single additional level increased the processing time to 10 hours and the octree size to 55GB, beyond the capacity of all but the largest GPUs.

Temporal coherence. We explore an orthogonal direction that exploits the temporal coherence of interactive fly-throughs; once the information needed to render a given view is computed, we reuse much of it for the *next* view. Similar to Plenoctree, we begin by precomputing a coarse cache of opacity and color. In contrast to Plenoctree, we *dynamically* subdivide the tree throughout the interactive visualization. Fig. 2.4 illustrates our approach. As the camera traverses the scene, our renderer uses the cached outputs to quickly produce an initial view and then performs additional rounds of model sampling to further refine the image, storing these new values into the cache. As each subsequent frame has significant overlap with its predecessor, it benefits from the previous refinement and needs to only perform a small amount of incremental work to maintain quality.

Guided sampling. We perform a final round of guided ray sampling after refining the octree to further improve rendering quality. We render rays in a single pass in contrast to NeRF's traditional two-stage hierachical sampling by using the weights stored in the octree structure. As our refined octree gives us a high-quality estimate of the scene geometry, we need to place only a small number of samples near surfaces of interest. Fig. 2.5 illustrates the difference between both approaches. Similar to other fast renderers, we further accelerate the process by accumulating transmittance along the ray and ending sampling after a certain threshold.

2.4 Experiments

Our evaluation of Mega-NeRF is motivated by the following two questions. First, given a finite training budget, how accurately can Mega-NeRF capture a scene? Furthermore, after training, is it possible to render accurately at scale while minimizing latency?

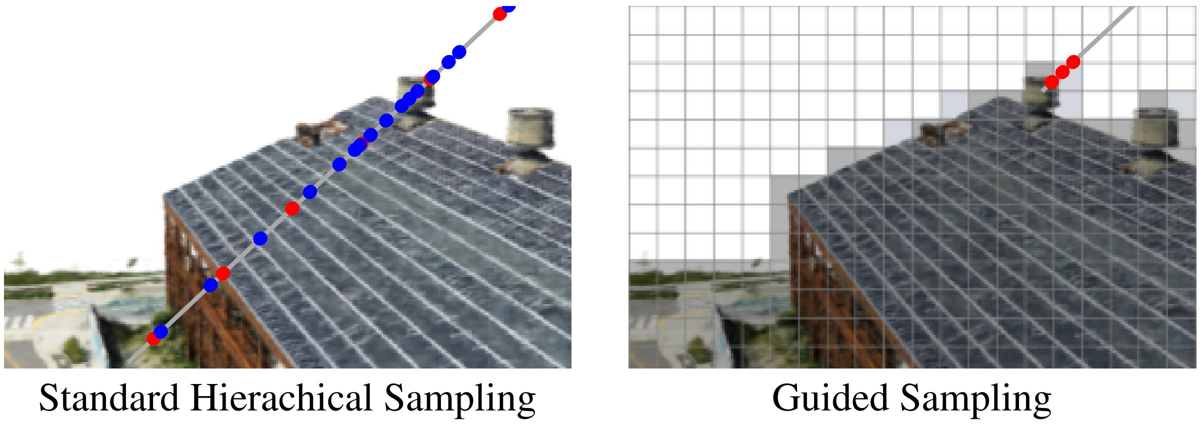


Figure 2.5: **Guided Sampling.** Standard NeRF **(left)** first samples coarsely at uniform intervals along the ray and subsequently performs another round of sampling guided by the coarse weights. Mega-NeRF-Dynamic **(right)** uses its caching structure to skip empty spaces and take a small number of samples near surfaces.

Qualitative results. We present two sets of qualitative results. Fig. 2.6 compares Mega-NeRF's reconstruction quality to existing view synthesis methods. In all cases Mega-NeRF captures a high level of detail while avoiding the numerous artifacts present in the other approaches. Fig. 2.7 then illustrates the quality of existing fast renderers and our method on top of the same base Mega-NeRF model. Our approach generates the highest quality reconstructions in almost all cases, avoiding the pixelization of voxel-based renderers and the blurriness of KiloNeRF.

2.4.1 Evaluation protocols

Datasets. We evaluate Mega-NeRF against multiple varied datasets. Our Mill 19 dataset consists of two scenes we recorded firsthand near a former industrial complex. Mill 19 - Building consists of footage captured in a grid pattern across a large 500×250 m^2 area around an industrial building. Mill 19 - Rubble covers a nearby construction area full of debris in which we placed human mannequins masquerading as survivors. We also measure Mega-NeRF against two publicly available collections - the Quad 6k dataset [32], a large-scale Structure-from-Motion dataset collected within the Cornell Universty Arts Quad, and several scenes from UrbanScene3D [103] which contain high-resolution drone imagery of large-scale urban environments. We refine the initial GPS-derived camera poses in the Mill 19 and UrbanScene3D datasets and the estimates provided in the Quad 6k dataset using PixSFM [104]. We use a pretrained semantic segmentation model [47] to produce masks of common movable objects in the Quad 6k dataset and ignore masked pixels during training.

Training. We evaluate Mega-NeRF with 8 submodules each consisting of 8 layers of 256 hidden units and a final fully connected ReLU layer of 128 channels. We use hierarchical sampling during training with 256 coarse and 512 fine samples per ray in the foreground regions and 128/256 samples per ray in the background. In contrast to NeRF, we use the same MLP to query both coarse and fine samples which reduces our model size and allows us to reuse the coarse network outputs during the second rendering stage, saving 25% model queries per ray. We adopt

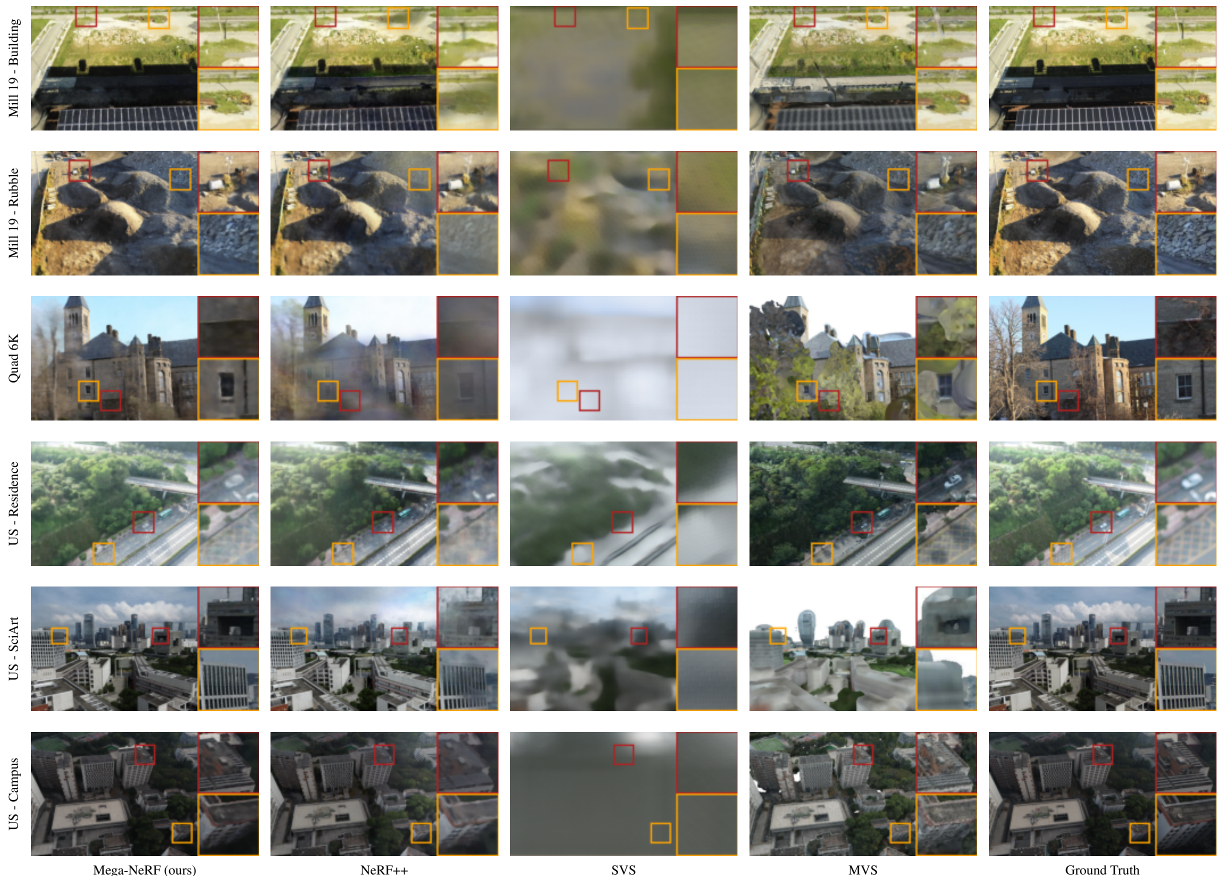


Figure 2.6: **Scalable training.** Mega-NeRF generates the best reconstructions while avoiding the artifacts present in the other approaches.

	Mill 19 - Building				Mill 19 - Rubble				Quad 6k			
	↑PSNR	↑SSIM	↓LPIPS	↓Time (h)	↑PSNR	↑SSIM	↓LPIPS	↓Time(h)	↑PSNR	↑SSIM	↓LPIPS	↓Time(h)
NeRF [118]	19.54	0.525	0.512	59:51	21.14	0.522	0.546	60:21	16.75	0.559	0.616	62:48
NeRF++ [223]	19.48	0.520	0.514	89:02	20.90	0.519	0.548	90:42	16.73	0.560	0.611	90:34
SVS [142]	12.59	0.299	0.778	38:17	13.97	0.323	0.788	37:33	11.45	0.504	0.637	29:48
DeepView [48]	13.28	0.295	0.751	31:20	14.47	0.310	0.734	32:11	11.34	0.471	0.708	19:51
MVS [155]	16.45	0.451	0.545	32:29	18.59	0.478	0.532	31:42	11.81	0.425	**0.594**	**18:55**
Mega-NeRF	**20.93**	**0.547**	**0.504**	**29:49**	**24.06**	**0.553**	**0.516**	**30:48**	**18.13**	**0.568**	0.602	39:43
	UrbanScene3D - Residence				UrbanScene3D - Sci-Art				UrbanScene3D - Campus			
	↑PSNR	↑SSIM	↓LPIPS	↓Time (h)	↑PSNR	↑SSIM	↓LPIPS	↓Time(h)	↑PSNR	↑SSIM	↓LPIPS	↓Time(h)
NeRF [118]	19.01	0.593	0.488	62:40	20.70	0.727	0.418	60:15	21.83	0.521	0.630	61:56
NeRF++ [223]	18.99	0.586	0.493	90:48	20.83	0.755	0.393	95:00	21.81	0.520	0.630	93:50
SVS [142]	16.55	0.388	0.704	77:15	15.05	0.493	0.716	59:58	13.45	0.356	0.773	105:01
DeepView [48]	13.07	0.313	0.767	30:30	12.22	0.454	0.831	31:29	13.77	0.351	0.764	33:08
MVS [155]	17.18	0.532	**0.429**	69:07	14.38	0.499	0.672	73:24	16.51	0.382	**0.581**	96:01
Mega-NeRF	**22.08**	**0.628**	0.489	**27:20**	**25.60**	**0.770**	**0.390**	**27:39**	**23.42**	**0.537**	0.618	**29:03**

Table 2.3: **Scalable training.** We compare Mega-NeRF to NeRF [118], NeRF++ [223], Stable View Synthesis (SVS) [142], DeepView [48], and dense multi-view stereo (MVS) reconstructions from COLMAP [155] after running each method to completion. Mega-NeRF consistently outperforms the baselines even after allowing other approaches to train well beyond 24 hours.

mixed-precision training to further accelerate the process. We sample 1024 rays per batch and use the Adam optimizer [87] with an initial learning rate of 5×10^{-4} decaying exponentially to 5×10^{-5}. We employ the procedure described in [113] to finetune Mega-NeRF's appearance embeddings.

2.4.2 Scalable Training

Baselines. We evaluate Mega-NeRF against the original NeRF [118] architecture and NeRF++ [223]. We also evaluate our approach against Stable View Synthesis [142], an implementation of DeepView [48], and dense reconstructions from COLMAP [155], a traditional Multi-View Stereo approach, as non-neural radiance field-based alternatives.

We use the same Pytorch-based framework and data loading infrastructure across all of NeRF variants to disentangle training speed from implementation specifics. We also use mixed precision training and the same number of samples per ray across all variants. We provide each implementation with the same amount of model capacity as Mega-NeRF by setting the MLP width to 2048 units. We base our DeepView baseline on a publicly available implementation and use the official Stable View Synthesis and COLMAP implementations.

Metrics. We report quantitative results based on PSNR, SSIM [187], and the VGG implementation of LPIPS [225]. We also report training times as measured on a single machine with 8 V100 GPUs.

Results. We run all methods to completion, training all NeRF-based methods for 500,000 iterations. We show results in Table 2.3 along with the time taken to finish training. Mega-NeRF outperforms the baselines even after training the other approaches for longer periods.

2.4.3 Interactive Exploration

Baselines. We evaluate two existing fast renderers, Plenoctree [216] and KiloNeRF [139], in

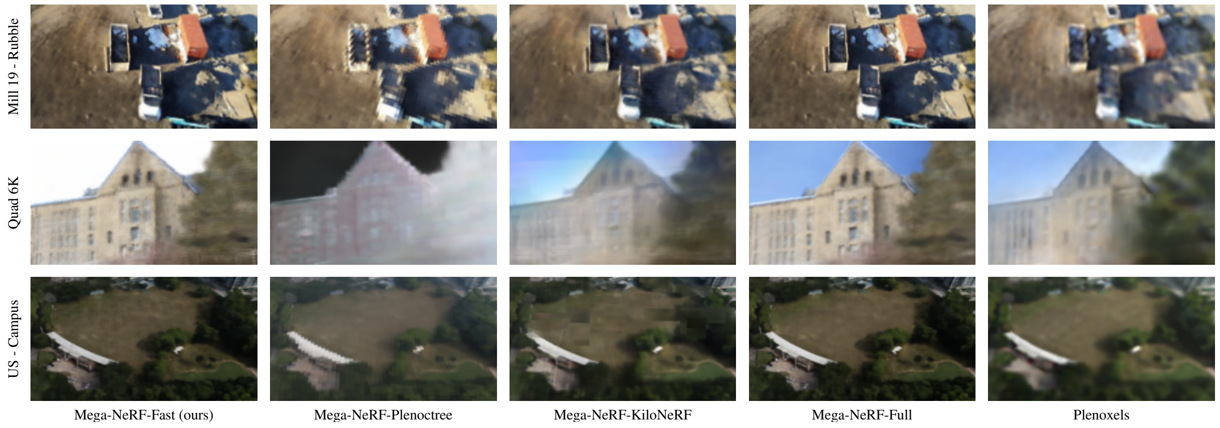


Figure 2.7: **Interactive rendering.** Plenoctree's approach causes significant voxelization and Plenoxel's renderings are blurry. KiloNeRF's results are crisper but capture less detail than Mega-NeRF-Dynamic and contain numerous visual artifacts.

addition to our dynamic renderer. We base all renderers against the same Mega-NeRF model trained in 2.4.2 with the exception of the Plenoctree method which is trained on a variant using spherical harmonics. We accordingly label our rendering variants as Mega-NeRF-Plenoctree, Mega-NeRF-KiloNeRF, and Mega-NeRF-Dynamic respectively. We measure traditional NeRF rendering as an additional baseline, which we refer to as Mega-NeRF-Full, and Plenoxels [149] which generates a sparse voxel structure similar to Plenoctree but with trilinear instead of nearest-neighbor interpolation.

Implementation. We bound the maximum tree size used by Mega-NeRF-Dynamic according available GPU memory and set it to 20M elements in our experiments. We track the number of pixels visible from each node as we traverse the tree when rendering. We then subdivide the top k (16,384) nodes with the most pixels. We observe maximum tree depths of roughly 12 in practice. As we track which nodes contribute to which pixels, we also prune entries that have not recently contributed in order to reclaim space whenever we hit capacity.

Metrics. We report the same perceptual metrics as in 2.4.2 and the time it takes to render a 720p image. We evaluate only foreground regions as Plenoctree and KiloNeRF assume bounded scenes. We also report any additional time needed to generate any additional data structures needed for rendering *beyond* the base model training time in the spirit of enabling fly-throughs within a day. As our renderer presents an initial coarse voxel-based estimate before progressively refining the image, we present an additional set of measurements, labeled as Mega-NeRF-Initial, to quantify the quality and latency of the initial reconstruction.

Results. We list our results in Table 2.4. Although Mega-NeRF-Plenoctree renders most quickly, voxelization has a large visual impact. Plenoxels provides better renderings but still suffers from the same finite resolution shortfalls and is blurry relative to the NeRF-based methods. Mega-NeRF-KiloNeRF comes close to interactivity at 1.1 FPS but still suffers from noticeable visual artifacts. Its knowledge distillation and finetuning processes also require over a day of additional processing. In contrast, Mega-NeRF-Dynamic remains within 0.8 db in PSNR of normal NeRF rendering while providing a 40× speedup. Mega-NeRF-Plenoctree and Mega-

best second-best	Mill 19					Quad 6k					UrbanScene3D				
	↑PSNR	↑SSIM	↓LPIPS	Preprocess Time (h)	Render Time (s)	↑PSNR	↑SSIM	↓LPIPS	Preprocess Time (h)	Render Time (s)	↑PSNR	↑SSIM	↓LPIPS	Preprocess Time (h)	Render Time (s)
Mega-NeRF-Plenoctree	16.27	0.430	0.621	1:26	**0.031**	13.88	0.589	0.427	1:33	**0.010**	16.41	0.498	0.530	**1:07**	**0.025**
Mega-NeRF-KiloNeRF	21.85	0.521	0.512	30:03	0.784	20.61	0.652	0.356	27:33	1.021	21.11	0.542	0.453	34:00	0.824
Mega-NeRF-Full	**22.96**	**0.588**	**0.452**	-	101	**21.52**	**0.676**	0.355	-	174	**24.92**	**0.710**	**0.393**	-	122
Plenoxels [149]	19.32	0.476	0.592	-	0.482	18.61	0.645	0.411	-	0.194	20.06	0.608	0.503	-	0.531
Mega-NeRF-Initial	17.41	0.447	0.570	**1:08**	0.235	14.30	0.585	0.386	**1:31**	0.214	17.22	0.527	0.506	1:10	0.221
Mega-NeRF-Dynamic	22.34	0.573	0.464	**1:08**	3.96	20.84	0.658	**0.342**	**1:31**	2.91	23.99	0.691	0.408	1:10	3.219

Table 2.4: **Interactive rendering.** We evaluate two existing fast renderers on top of our base model, Mega-NeRF-Plenoctree and Mega-NeRF-KiloNeRF, relative to conventional rendering, labeled as Mega-NeRF-Full, Plenoxels, and our novel renderer (**below**). Although PlenOctree achieves a consistently high FPS, its reliance on a finite-resolution voxel structure causes performance to degrade significantly. Our approach remains within 0.8 db in PSNR quality while accelerating rendering by 40× relative to conventional ray sampling.

	1 Submodule					4 Submodules				
	↑PSNR	↑SSIM	↓LPIPS	Train Time (h)	Render Time (s)	↑PSNR	↑SSIM	↓LPIPS	Train Time (h)	Render Time (s)
128 Channels	21.75	0.435	0.670	**18:54**	**2.154**	22.61	0.469	0.631	**18:56**	**2.489**
256 Channels	22.60	0.471	0.622	28:54	3.298	23.63	0.521	0.551	29:09	3.427
512 Channels	**23.40**	**0.512**	**0.559**	52:33	6.195	**24.53**	**0.581**	**0.482**	52:34	6.313
	9 Submodules					16 Submodules				
	↑PSNR	↑SSIM	↓LPIPS	Train Time (h)	Render Time (s)	↑PSNR	↑SSIM	↓LPIPS	Train Time (h)	Render Time (s)
128 Channels	23.08	0.495	0.594	**19:01**	**2.633**	23.34	0.513	0.568	**19:02**	**2.851**
256 Channels	24.17	0.559	0.508	29:13	3.793	24.52	0.584	0.481	29:14	3.991
512 Channels	**25.11**	**0.625**	**0.438**	53:36	6.671	**25.68**	**0.659**	**0.407**	53:45	6.870

Table 2.5: **Model scaling.** We scale up Mega-NeRF with additional submodules (**rows**) and increased channel count per submodule (**columns**). Scaling up both increases reconstruction quality, but increasing channels significantly increases both training and rendering time (as measured for Mega-NeRF-Dynamic).

NeRF-Dynamic both take an hour to build similar octree structures.

2.4.4 Diagnostics

Scaling properties. We explore Mega-NeRF's scaling properties against the Mill 19 - Rubble dataset. We vary the total number of submodules and the number of channels per submodule across 1, 4, 9, and 16 submodules and 128, 256, and 512 channels respectively. We summarize our findings in Table 2.5. Increasing the model capacity along either dimension improves rendering quality, as depicted in Fig. 2.8. However, although increasing the channel count severely penalizes training and rendering speed, the number of submodules has less impact.

Data Pruning. Recall that the initial assignment of pixels to spatial cells is based on camera positions, irrespective of scene geometry (because that is not known at initialization time). However, Sec. 2.3.2 points out that one could repartition our training sets with additional 3D knowledge. Intuitively, one can prune away irrelevant pixel/ray assignments that don't contribute to a particular NeRF submodule due to an intervening occluder (Fig. 2.9).

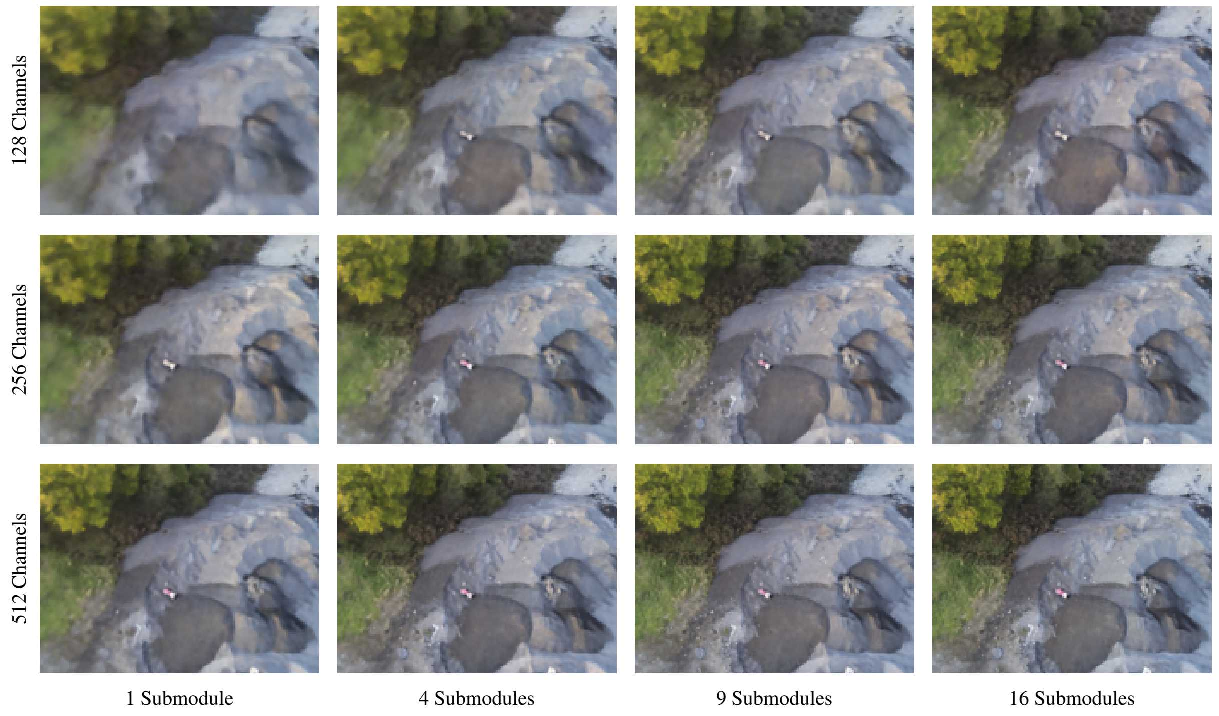


Figure 2.8: **Model scaling.** Example rendering within our Mill 19 - Rubble dataset across different numbers of submodules (**columns**) and channels per submodule (**rows**). Mega-NeRF generates increasingly photo-realistic renderings as capacity increases. Increasing the number of submodules increases the overall model capacity with little impact to training and inference time.

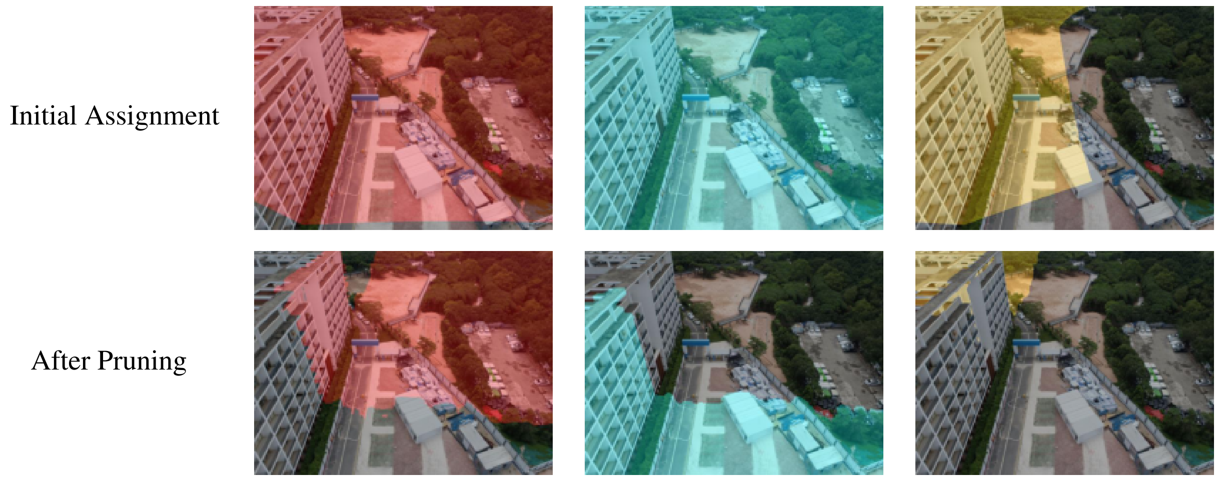


Figure 2.9: **Data pruning.** The initial assignment of pixels to cells is based purely on camera positions. We add each pixel to the training set of **all** cells it traverses, leading to overlap between sets **(top)**. After the model gains a 3D understanding of the scene, we can filter irrelevant pixels by instead assigning pixels based on camera ray intersection with solid surfaces **(bottom)**.

	Mill 19				Quad 6k				UrbanScene3D			
	↑PSNR	↑SSIM	↓LPIPS	↓Pixels	↑PSNR	↑SSIM	↓LPIPS	↓Pixels	↑PSNR	↑SSIM	↓LPIPS	↓Pixels
Original Data	22.50	0.550	0.511	0.211	18.13	0.568	0.602	0.390	23.65	0.644	0.500	0.270
Pruned Data	**22.76**	**0.571**	**0.488**	**0.160**	**18.16**	**0.569**	**0.593**	**0.149**	**23.87**	**0.656**	**0.483**	**0.163**

Table 2.6: **Data pruning.** The initial assignment of pixels to spatial cells is based purely on rays emanating from camera centers, irrespective of scene geometry. However, once a rough Mega-NeRF has been trained, coarse estimates of scene geometry can be used to prune irrelevant pixel assignments. Doing so reduces the amount of training data for each submodule by up to 2× while increasing accuracy for a fixed number of 500,000 iterations.

To explore this optimization, we further prune each data partition early into the training process after the model gains a coarse 3D understanding of the scene (100,000 iterations in our experiments). As directly querying depth information using conventional NeRF rendering is prohibitive at our scale, we instead take inspiration from Plenoctree and tabulate the scene's model opacity values into a fixed resolution structure. We then calculate the intersection of each training pixel's camera ray against surfaces within the structure to generate new assignments. We found that it took around 10 minutes to compute the model density values and 500ms per image to generate the new assignments. We summarize our findings in Table 2.6.

Ablations. We compare Mega-NeRF to several ablations. Mega-NeRF-no-embed removes the appearance embeddings from the model structure. Mega-NeRF-embed-only conversely adds Mega-NeRF's appearance embeddings to the base NeRF architecture. Mega-NeRF-no-bounds uses NeRF++'s unit sphere background/foreground partitioning instead of our formulation described in 2.3.1. Mega-NeRF-dense uses fully connected layers instead of spatially-aware sparse connections. Mega-NeRF-joint uses the same model structure as Mega-NeRF but trains all submodules jointly using the full dataset instead of using submodule-specific data partitions. We

	Mill 19			Quad 6k			UrbanScene3D		
	↑PSNR	↑SSIM	↓LPIPS	↑PSNR	↑SSIM	↓LPIPS	↑PSNR	↑SSIM	↓LPIPS
Mega-NeRF-no-embed	20.42	0.500	0.561	16.16	0.544	0.643	19.45	0.587	0.545
Mega-NeRF-embed-only	21.48	0.494	0.566	17.91	0.559	0.638	22.79	0.611	0.537
Mega-NeRF-no-bounds	22.14	0.534	0.522	18.02	0.565	0.616	23.42	0.636	0.511
Mega-NeRF-dense	21.63	0.504	0.551	17.94	0.562	0.627	22.44	0.605	0.558
Mega-NeRF-joint	21.10	0.490	0.574	17.43	0.560	0.616	21.45	0.595	0.567
Mega-NeRF	**22.34**	**0.540**	**0.518**	**18.08**	**0.566**	**0.602**	**23.60**	**0.641**	**0.504**

Table 2.7: **Ablations.** We compare Mega-NeRF to various ablations after 24 hours of training. Each individual component contributes significantly to overall model performance.

limit training to 24 hours for expediency.

We present our results in Table 2.7. Both the appearance embeddings and the foreground/background decomposition have a significant impact on model performance. Mega-NeRF also outperforms both Mega-NeRF-dense and Mega-NeRF-joint, although Mega-NeRF-dense comes close in several scenes. We however note that model sparsity accelerates rendering by 10$\times$ relative to fully-connected MLPs and is therefore essential for acceptable performance.

2.5 Discussion

We present a modular approach for building NeRFs at previously unexplored scale. We introduce a sparse and spatially aware network structure along with a simple geometric clustering algorithm that partitions training pixels into different NeRF submodules which can be trained in parallel. These modifications speed up training by over 3$\times$ while significantly improving reconstruction quality. Our empirical evaluation of existing fast renderers on top of Mega-NeRF suggests that questions remain as to how to best handle interactive NeRF-based rendering at scale. We advocate leveraging temporal smoothness to minimize redundant computation between views as a valuable first step.

2.5.1 Limitations

Dynamic objects. We did not explicitly address dynamic scenes in this chapter, a relevant factor for many real-world use cases. Several NeRF-related efforts, including NR-NeRF [173], Nerfies [127], NeRFlow [41], DynamicMVS [56], and Neural Scene Graphs [126] focus on dynamism, but are non-trivial to scale to large urban scenes. We discuss this further in Chapter 3.

Rendering speed. While the dynamic renderer avoids the pitfalls of existing fast NeRF approaches, it does not reach the throughput needed for truly interactive applications. We revisit rendering acceleration in Chapter 4.

Training speed. Although our training process is several factors quicker than previous works, training time remains a significant bottleneck towards rapid model deployment. One possible improvement is to use a more efficient representation than MLPs. We propose such a representation in Chapter 3.

Pose accuracy. Pose accuracy is amongst the largest limiting factors to rendering quality. The initial models we trained using raw camera poses collected from standard drone GPS and

IMU sensors were extremely blurry. Although several efforts [30, 76, 100, 114, 189] attempt to jointly optimize camera parameters during NeRF optimization, we found the results lacking relative to using offline structure-from-motion based approaches as a preprocessing step. A hardware-based alternative would be to use higher-accuracy RTK GPS modules when collecting footage.

Chapter 3

Dynamic City-Scale Reconstruction

The contents of this chapter were published as "SUDS: Scalable Urban Dynamic Scenes" in CVPR 2023

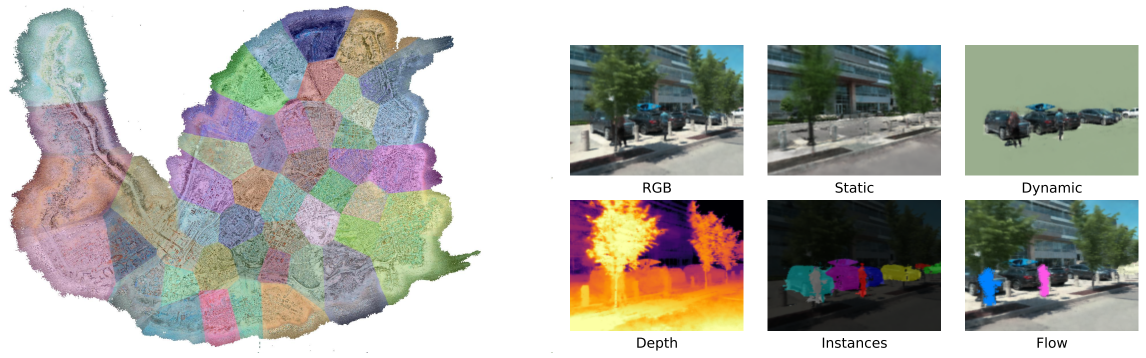


Figure 3.1: **SUDS [179].** We scale neural reconstructions to city scale by dividing the area into multiple cells and training hash table representations for each. We show our full city-scale reconstruction (**left**) and the derived representations (**right**). Unlike prior methods, our approach handles dynamism across multiple videos, disentangling dynamic objects from static background and modeling shadow effects. We use unlabeled inputs to learn scene flow and semantic predictions, enabling category- and object-level scene manipulation.

3.1 Introduction

Chapter 2 explores how to build static neighborhood-representations to enable virtual map reconstruction and photorealistic fly-throughs. However, these maps remain static and frozen in time. This makes capturing bustling human environments—complete with moving vehicles, pedestrians, and objects—impossible, limiting the usefulness of the representation. We now focus on how to best contruct dynamic representations of entire cities.

Challenges. One possible solution is a dynamic NeRF that conditions on time or warps a canonical space with a time-dependent deformation [127]. However, reconstructing dynamic scenes is notoriously challenging because the problem is inherently under-constrained, particularly when input data is constrained to limited viewpoints, as is typical from egocentric video capture [57]. One attractive solution is to scale up reconstructions to *many* videos, perhaps collected at different days (e.g., by an autonomous vehicle fleet). However, this creates additional challenges in jointly modeling fixed geometry that holds for all time (such as buildings), geometry that is locally static but *transient* across the videos (such as a parked car), and geometry that is truly dynamic (such as a moving person).

SUDS. In this paper, we propose SUDS: Scalable Urban Dynamic Scenes, a 4D representation that targets both *scale* and *dynamism*. Our key insight is twofold; (1) SUDS makes use of a rich suite of informative but freely available input signals, such as LiDAR depth measurements and optical flow. Other dynamic scene representations [90, 126] require supervised inputs such as panoptic segmentation labels or bounding boxes, which are difficult to acquire with high accuracy for our in-the-wild captures. (2) SUDS decomposes the world into 3 components: a *static* branch that models stationary topography that is consistent across videos, a *dynamic* branch that handles both transient (e.g., parked cars) and truly dynamic objects (e.g., pedestrians), and an *environment map* that handles far-field objects and sky. We model each branch using a multi-resolution hash table with scene partitioning, allowing SUDS to scale to an entire city spanning over 100 km^2.

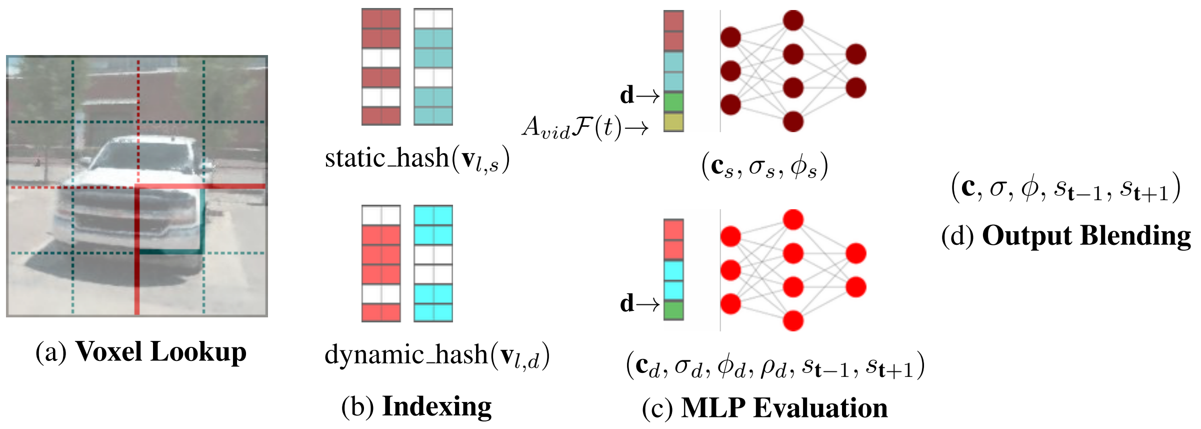
Contributions. We make the following contributions: (1) to our knowledge, we build the first large-scale dynamic NeRF, (2) we introduce a scalable three-branch hash table representation for 4D reconstruction, (3) we present state-of-the-art reconstruction on 3 different datasets. Finally, (4) we showcase a variety of downstream tasks enabled by our representation, including free-viewpoint synthesis, 3D scene flow estimation, and even unsupervised instance segmentation and 3D cuboid detection.

3.2 Related Work

Below, we describe a non-exhaustive list of such approaches along axes relevant to our work.

Scale. The original NeRF operated with bounded scenes. NeRF++ [223] and mip-NeRF 360 [17] use non-linear scene parameterization to model unbounded scenes. However, scaling up the size of the scene with a fixed size MLP leads to blurry details and training instability while the cost of naively increasing the size of the MLP quickly becomes intractable. BungeeNeRF [197] introduced a coarse-to-fine approach that progressively adds more capacity to the network representation. Mega-NeRF (Chapter 2) and Block-NeRF [167] partition the scene spatially and train separate NeRFs for each partition. To model appearance variation, they incorporate per-image embeddings like NeRF-W [113]. Our approach similarly partitions the scene into sub-NeRFs, making use of depth to improve partition efficiency and scaling over an area 200x larger than Block-NeRF's Alamo Square Dataset. Both of these methods work only on static scenes.

Dynamics. Neural 3D Video Synthesis [94] and Space-time Neural Irradiance Fields [196] add time as an input to handle dynamic scenes. Similar to our work, NSFF [96], NeRFlow [41], and DyNeRF [55] incorporate 2D optical flow input and warping-based regularization losses to enforce plausible transitions between observed frames. Multiple methods [127, 128, 134, 173] instead disentangle scenes into a canonical template and per-frame deformation field. BANMo [204]



static_hash($\mathbf{v}_{l,s}$)
dynamic_hash($\mathbf{v}_{l,d}$)
$\mathbf{d}\rightarrow$
$A_{vid}\mathcal{F}(t)\rightarrow$
$(\mathbf{c}_s, \sigma_s, \phi_s)$
$(\mathbf{c}_d, \sigma_d, \phi_d, \rho_d, s_{\mathbf{t}-1}, s_{\mathbf{t}+1})$
$(\mathbf{c}, \sigma, \phi, s_{\mathbf{t}-1}, s_{\mathbf{t}+1})$
(a) **Voxel Lookup**
(b) **Indexing**
(c) **MLP Evaluation**
(d) **Output Blending**

Figure 3.2: **Model Architecture.** (a) For a given input coordinate, we find the surrounding voxels at L resolution levels for both the static and dynamic branches (far-field branch omitted for clarity). (b) We assign indices to their corners by hashing based on position in the static branch and position, time, and video id in the dynamic branch. We look up the feature vectors corresponding to the corners and interpolate according to the relative position of the input coordinate within the voxel. (c) We concatenate the result of each level, along with auxiliary inputs such as viewing direction, and pass the resulting vector into an MLP to obtain per-branch color, density, and feature logits along with scene flow and the shadow ratio. (d) We blend color, opacity, and feature logits as the weighted sum of the branches.

further incorporates deformable shape models and canonical embeddings to train articulated 3D models from multiple videos. These methods focus on single-object scenes, and all but [94] and [204] use single video sequences.

While many of the previous works use segmentation data to factorize dynamic from static objects, D^{2}NeRF [195] does this automatically through regularization and explicitly handling shadows. Neural Groundplans [157] uses synthetic data to do this decomposition from a single image. We borrow some of these ideas and scale beyond synthetic and indoor scenes.

Object-centric approaches. Several approaches [77, 123, 126, 203, 217, 219] represent scenes as the composition of per-object NeRF models and a background model. NSG [126] is most similar to us as it also targets automotive data but cannot handle ego-motion as our approach can. None of these methods target multi-video representations and are fundamentally constrained by the memory required to represent each object, with NSG needing over 1TB of memory to represent a 30 second video in our experience.

Semantics. Follow-up works have explored additional semantic outputs in addition to predicting color. Semantic-NeRF [227] adds an extra head to NeRF that predicts extra semantic category logits for any 3D position. Panoptic-NeRF [51] and Panoptic Neural Fields [90] extend this to produce panoptic segmentations and the latter uses a similar bounding-box based object and background decomposition as NSG. NeSF [182] generalizes the notion of a semantic field to unobserved scenes. As these methods are highly reliant on accurate annotations which are difficult to reliably obtain in the wild at our scale, we instead use a similar approach to recent works [89, 177] that distill the outputs of 2D self-supervised feature descriptors into 3D radiance fields to enable semantic understanding without the use of human labels and extend them to larger dynamic settings.

Fast training. The original NeRF took 1-2 days to train. Plenoxels [150] and DVGO [165] directly optimize a voxel representation instead of an MLP to train in minutes or even seconds. TensoRF [26] stores its representation as the outer product of low-rank tensors, reducing memory usage. Instant-NGP [120] takes this further by encoding features in a multi-resolution hash table, allowing training and rendering to happen in real-time. We use these tables as the base block of our three-branch representation and use our own hashing method to support dynamics across multiple videos.

Depth. Depth provides a valuable supervisory signal for learning high-quality geometry. DS-NeRF [37] and Dense Depth Priors [143] incorporate noisy point clouds obtained by structure from motion (SfM) in the loss function during optimization. Urban Radiance Fields [141] supervises with collected LiDAR data. We also use LiDAR but demonstrate results on dynamic environments.

3.3 Approach

3.3.1 Inputs

Our goal is to learn a global representation that facilitates free-viewpoint rendering, semantic decomposition, and 3D scene flow at arbitrary poses and time steps. Our method takes as input ordered RGB images from N videos (taken at different days with diverse weather and lighting conditions) and their associated camera poses. Crucially, we make use of additional data as “free” sources of supervision given contemporary sensor rigs and feature descriptors. Specifically, we use (1) aligned sparse LiDAR depth measurements, (2) 2D self-supervised pixel (DINO [24]) descriptors to enable semantic manipulation, and (3) 2D optical flow predictions to model scene dynamics. All model inputs are generated without any human labeling or intervention.

3.3.2 Representation

Preliminaries. We build upon NeRF [118], which represents a scene within a continuous volumetric radiance field that captures both geometry and view-dependent appearance. We refer the reader to Chapter 2 and [118] for more details.

Scene composition. To model large-scale dynamic environments, SUDS factorizes the scene into three branches: (a) a static branch containing non-moving topography consistent across videos, (b) a dynamic branch to disentangle video-specific objects [55, 96, 195], moving or otherwise, and (c) a far-field environment map to represent far-away objects and the sky, which we found important to separately model in large-scale urban scenes [141, 178, 223].

However, conventional NeRF training with MLPs is computationally prohibitive at our target scales. Inspired by Instant-NGP [120], we implement each branch using multiresolution hash tables of F-dimensional feature vectors followed by a small MLP, along with our own hash functions to index across videos.

Hash tables (Fig. 5.1). For a given input coordinate $(\mathbf{x}, \mathbf{d}, \mathbf{t}, \mathbf{vid})$ denoting the position $\mathbf{x} \in \mathbb{R}^3$, viewing direction $\mathbf{d} \in \mathbb{R}^3$, frame index $F \in \{1, ..., T\}$, and video id $\mathbf{vid} \in \{1, ..., N\}$, we find the surrounding voxels in each table at $l \in L$ resolution levels, doubling the resolution between levels, which we denote as $\mathbf{v}_{l,s}$, $\mathbf{v}_{l,d}$, $\mathbf{v}_{l,e}$ for the static, dynamic, and far-field. The static branch makes use of 3D spatial voxels $\mathbf{v}_{l,s}$, while the dynamic branch makes use of 4D spacetime voxels $\mathbf{v}_{l,d}$. Finally, the far-field branch makes use of 3D voxels $\mathbf{v}_{l,e}$ (implemented via normalized 3D direction vectors) that index an environment map. Similar to Instant-NGP [120], rather than storing features at voxel corners, we compute hash indices $\mathbf{i}_{l,s}$ (or $\mathbf{i}_{l,d}$ or $\mathbf{i}_{l,e}$) for each corner with the following hash functions:

$$\mathbf{i}_{l,s} = \text{static\_hash}(space(\mathbf{v}_{l,s})) \tag{3.1}$$

$$\mathbf{i}_{l,d} = \text{dynamic\_hash}(space(\mathbf{v}_{l,d}), time(\mathbf{v}_{l,d}), \mathbf{vid}) \tag{3.2}$$

$$\mathbf{i}_{l,e} = \text{env\_hash}(dir(\mathbf{v}_{l,e}), \mathbf{vid}) \tag{3.3}$$

We linearly interpolate features up to the nearest voxel vertices (but now relying on *quad*linear interpolation for the dynamic 4D branch) and rely on gradient averaging to handle hash collisions. Finally, to model the fact that different videos likely contain distinct moving objects and illumination conditions, we add **vid** as an auxiliary input to the hash, but do *not* use it for interpolation (since averaging across distinct movers is unnatural). From this perspective, we leverage hashing to effectively index separate interpolating functions for each video, *without* a linear growth in memory with the number of videos. We concatenate the result of each level into a feature vector $f \in \mathbb{R}^{LF}$, along with auxiliary inputs such as viewing direction, and pass the resulting vector into an MLP to obtain per-branch outputs.

Static branch. We generate RGB images by combining the outputs of our three branches. The static branch maps the feature vector obtained from the hash table into a view-dependent color $\mathbf{c}_s$ and a view-independent density σ_s. To model lighting variations which could be dramatic across videos but smooth *within* a video, we condition on a latent embedding computed as a product of a video-specific matrix A_{vid} and a fourier-encoded time index $\mathcal{F}(t)$ (as in [204]):

$$\sigma_s(\mathbf{x}) \in \mathbb{R} \tag{3.4}$$

$$\mathbf{c}_s(\mathbf{x}, \mathbf{d}, A_{vid}\mathcal{F}(t)) \in \mathbb{R}^3. \tag{3.5}$$

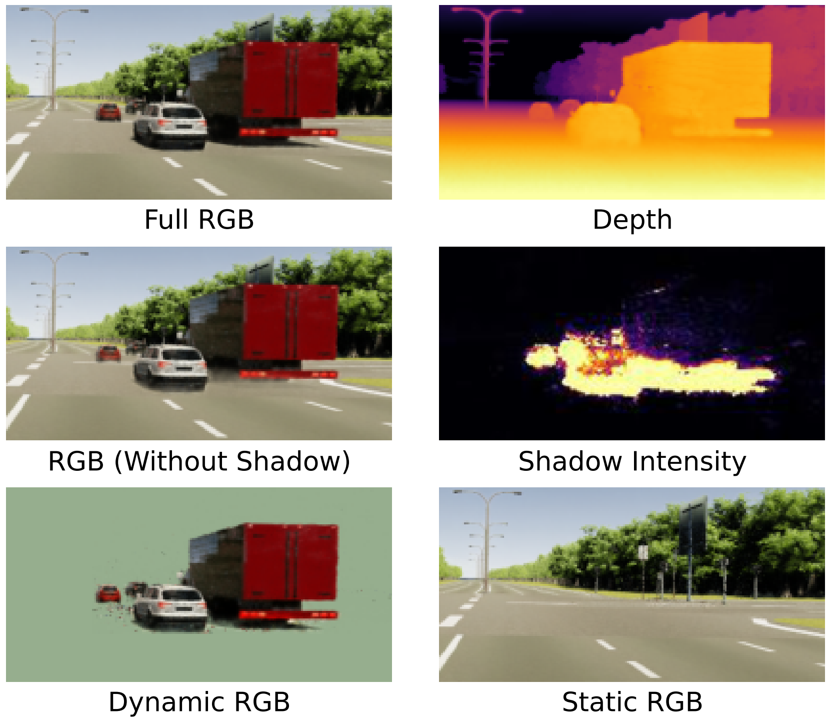
Dynamic branch. While the static branch assumes the density σ_s is static, the dynamic branch allows both the density σ_d and color $\mathbf{c}_d$ to depend on time (and video). We therefore omit the latent code when computing the dynamic radiance. Because we find shadows to play a crucial role in the appearance of urban scenes (Fig. 3.3), we explicitly model a *shadow field* of scalar values $\rho_d \in [0, 1]$, used to scale down the static color $\mathbf{c}_s$ (as done in [195]):

$$\sigma_d(\mathbf{x}, \mathbf{t}, \mathbf{vid}) \in \mathbb{R} \tag{3.6}$$

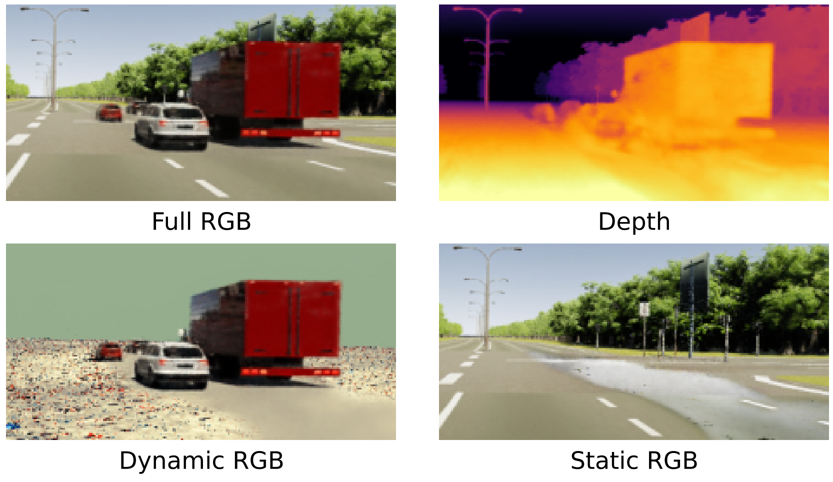
$$\rho_d(\mathbf{x}, \mathbf{t}, \mathbf{vid}) \in [0, 1] \tag{3.7}$$

$$\mathbf{c}_d(\mathbf{x}, \mathbf{t}, \mathbf{vid}, \mathbf{d}) \in \mathbb{R}^3 \tag{3.8}$$

Far-field branch. Because the sky requires reasoning about far-field radiance and because it can change dramatically across videos, we model far-field radiance with an environment map $\mathbf{c}_e(\mathbf{d}, \mathbf{vid}) \in \mathbb{R}^3$ that depends on viewing direction $\mathbf{d}$ [66, 141] and a video id $\mathbf{vid}$.



(a) Shadow Field



(b) No Shadow Field

Figure 3.3: **Shadows.** We learn an explicit shadow field (**a**) as a pointwise reduction on static color, enabling better depth reconstruction and static/dynamic factorization than without (**b**).

Rendering. We derive a single density and radiance value for any position by computing the weighted sum of the static and dynamic components, combined with the pointwise shadow reduction:

$$\sigma(\mathbf{x}, \mathbf{t}, \mathbf{vid}) = \sigma_s(\mathbf{x}) + \sigma_d(\mathbf{x}, \mathbf{t}, \mathbf{vid}) \tag{3.9}$$

$$\begin{aligned} \mathbf{c}(\mathbf{x}, \mathbf{t}, \mathbf{vid}, \mathbf{d}) = & \frac{\sigma_s}{\sigma}(1-\rho_d)\mathbf{c}_s(\mathbf{x}, \mathbf{d}, A_{vid}\mathcal{F}(t)) \\ & + \frac{\sigma_d}{\sigma}\mathbf{c}_d(\mathbf{x}, \mathbf{t}, \mathbf{vid}, \mathbf{d}) \end{aligned} \tag{3.10}$$

We then calculate the color $\hat{C}$ for a camera ray $\mathbf{r}$ with direction $\mathbf{d}$ at a given frame $\mathbf{t}$ and video $\mathbf{vid}$ by accumulating the transmittance along sampled points $\mathbf{r}(t)$ along the ray, forcing the ray to intersect the far-field environment map if it does not hit geometry within the foreground:

$$\begin{aligned} \hat{C}(\mathbf{r}, \mathbf{t}, \mathbf{vid}) = & \int_0^{+\infty} T(t)\sigma(\mathbf{r}(t), \mathbf{t}, \mathbf{vid})\mathbf{c}(\mathbf{r}(t), \mathbf{t}, \mathbf{vid}, \mathbf{d})dt \\ & + T(+\infty)\mathbf{c}_e(\mathbf{d}, \mathbf{vid}), \end{aligned} \tag{3.11}$$

$$\text{where } T(t) = \exp\left(-\int_0^t \sigma(\mathbf{r}(s), \mathbf{t}, \mathbf{vid})ds\right). \tag{3.12}$$

Feature distillation. We build semantic awareness into SUDS to enable the open-world tasks described in Sec. 3.4.2. Similar to recent work [89, 177], we distill the outputs of a self-supervised 2D feature extractor, namely DINO [24], as a teacher model into our network. For a feature extractor that transforms an image into a dense $\mathbb{R}^{H \times W \times C}$ feature grid, we add a C-dimensional output head to each of our branches:

$$\Phi_s(\mathbf{x}) \in \mathbb{R}^C \tag{3.13}$$

$$\Phi_d(\mathbf{x}, \mathbf{t}, \mathbf{vid}) \in \mathbb{R}^C \tag{3.14}$$

$$\Phi_e(\mathbf{d}, \mathbf{vid}) \in \mathbb{R}^C, \tag{3.15}$$

which are combined into a single value Φ at any 3D location and rendered into $\hat{F}(\mathbf{r})$ per camera ray, following the equations for color (3.10, 3.11).

Scene flow. We train our model to predict 3D scene flow and model scene dynamics. Inspired by previous work [41, 55, 96], we augment our dynamic branch to predict forward and backward 3D scene flow vectors $s_{t' \in [-1,1]}(\mathbf{x}, \mathbf{t}, \mathbf{vid}) \in \mathbb{R}^3$. We make use of these vectors to enforce consistency between observed time steps through multiple loss terms (Sec. 3.3.3), which we find crucial to generating plausible renderings at novel time steps (Table 5.7).

Spatial partitioning. We scale our representation to arbitrarily large environments by decomposing the scene into individually trained models [167, 178], each with its own static, dynamic, and far-field branch. Intuitively, the reconstruction for neighborhood X can be done largely independently of the reconstruction in neighborhood Y, provided one can assign the relevant input data to each reconstruction. To do so, we follow the approach of Mega-NeRF (Chapter 2) and split the scene into K spatial cells with centroids $k \in \mathbb{R}^3$. Crucially, we generate separate training datasets for each spatial cell by making use of *visibility* reasoning [53]. Mega-NeRF includes only those datapoints whose associated camera rays intersect the spatial cell.

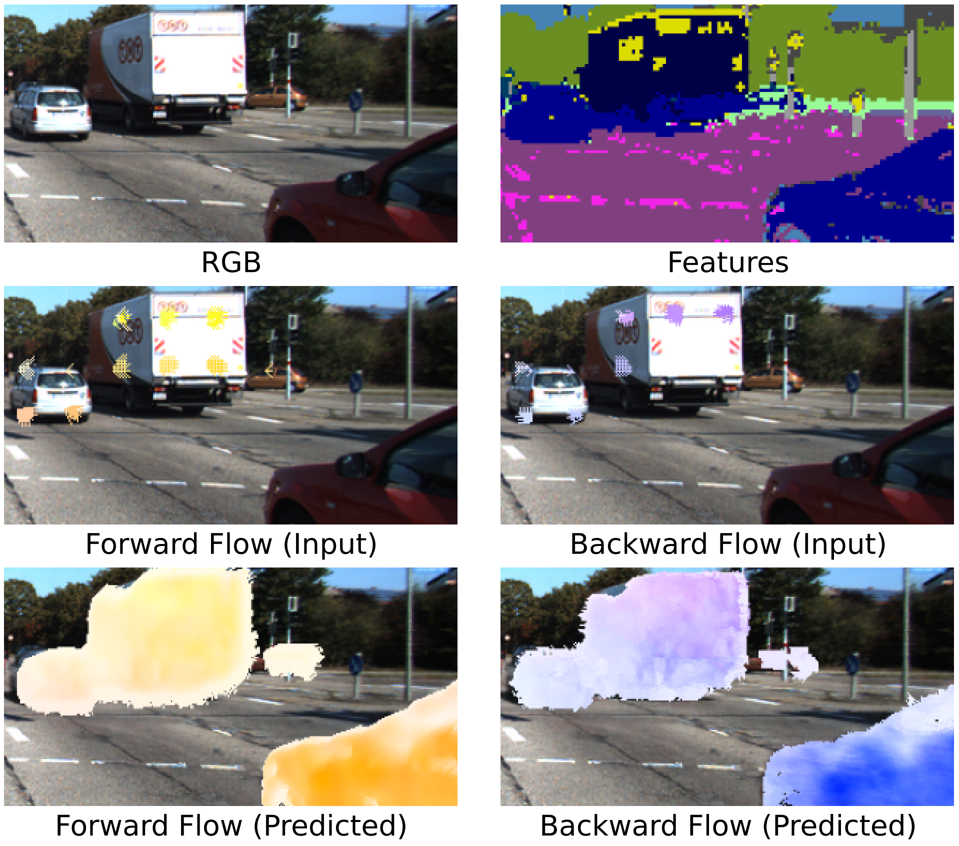


Figure 3.4: **Scene Flow.** We minimize the photometric and feature-metric loss of warped renderings relative to ground truth inputs **(top)**. We use 2D optical flow from off-the-shelf estimators or sparse correspondences computed directly from 2D DINO features [15] **(middle)** to supervise our flow predictions **(bottom)**.

However, this may still include datapoints that are not visible due to an intervening occluder (e.g., a particular camera in neighborhood X can be *pointed* at neighborhood Y, but may not see anything there due to occluding buildings). To remedy this, we make use of depth measurements to prune irrelevant pixel rays that do not terminate within the spatial cell of interest (making use of nearest-neighbor interpolation to impute depth for pixels without a LiDAR depth measurement). This further reduces the size of each trainset by 2x relative to Mega-NeRF. Finally, given such separate reconstructions, one can still produce a globally consistent rendering by querying the appropriate spatial cell when sampling points along new-view rays (as in Chapter 2).

3.3.3 Optimization

We jointly optimize all three of our model branches along with the per-video weight matrices A_{vid} by sampling random batches of rays across our N input videos and minimizing the following loss:

$$\mathcal{L} = \underbrace{\Big(\mathcal{L}_c + \lambda_f \mathcal{L}_f + \lambda_d \mathcal{L}_d + \lambda_o \mathcal{L}_o\Big)}_{\text{reconstruction losses}} + \underbrace{\Big(\mathcal{L}_c^w + \lambda_f \mathcal{L}_f^w\Big)}_{\text{warping losses}}$$
$$\lambda_{flo}\underbrace{\Big(\mathcal{L}_{cyc} + \mathcal{L}_{sm} + \mathcal{L}_{slo}\Big)}_{\text{flow losses}} + \underbrace{\Big(\lambda_e \mathcal{L}_e + \lambda_d \mathcal{L}_d\Big)}_{\text{static-dynamic factorization}} + \lambda_\rho \mathcal{L}_\rho. \quad (3.16)$$

Reconstruction losses. We minimize the L2 photometric loss $\mathcal{L}_c(\mathbf{r}) = \left\|C(\mathbf{r}) - \hat{C}(\mathbf{r})\right\|^2$ as in the original NeRF equation [118]. We similarly minimize the L1 difference $\mathcal{L}_f(\mathbf{r}) = \left\|F(\mathbf{r}) - \hat{F}(\mathbf{r})\right\|_1$ between the feature outputs of the teacher model and that of our network.

To make use of our depth measurements, we project the LiDAR sweeps onto the camera plane and compare the expected depth $\hat{D}(r)$ with the measurement $D(\mathbf{r})$ [37, 141]:

$$\mathcal{L}_d(\mathbf{r}) = \left\|D(\mathbf{r}) - \hat{D}(\mathbf{r})\right\|^2 \quad (3.17)$$
$$\text{where } \hat{D}(\mathbf{r}) = \int_0^{+\infty} T(s)\sigma(\mathbf{r}(s))ds \quad (3.18)$$

Flow. We supervise our 3D scene flow predictions based on 2D optical flow (Sec. 3.4.1). We generate a 2D displacement vector for each camera ray by first predicting its position in 3D space as the weighted sum of the scene flow neighbors along the ray:

$$\hat{X}_{t'}(\mathbf{r}) = \int_0^{+\infty} T(t)\sigma(r(t))(r(t) + s_{t'}(\mathbf{r}(t)))dt \quad (3.19)$$

which we then "render" into 2D using the camera matrix of the neighboring frame index. We minimize its distance from the observed optical flow via $\mathcal{L}_o(\mathbf{r}) = \sum_{t' \in [-1,1]} \left\|X(\mathbf{o}) - \hat{X}_{t'}(\mathbf{r})\right\|_1$. We anneal λ_o over time as these estimates are noisy.

3D warping. The above loss ensures that rendered 3D flow will be consistent with the observed 2D flow. We also found it useful to enforce 3D color (and feature) constancy; i.e., colors remain constant even when moving. To do so, we use the predicted forward and backward 3D flow $s_{\mathbf{t}+1}$ and $s_{\mathbf{t}-1}$ to *advect* each sample along the ray into the next/previous frame:

$$\sigma_{t'}^w(\mathbf{x} + s_{t'}, \mathbf{t} + t', \mathbf{vid}) \in \mathbb{R} \quad (3.20)$$
$$\mathbf{c}_{t'}^w(\mathbf{x} + s_{t'}, \mathbf{t} + t', \mathbf{vid}, \mathbf{d}) \in \mathbb{R}^3 \quad (3.21)$$
$$\Phi_{t'}^w(\mathbf{x} + s_{t'}, \mathbf{t} + t', \mathbf{vid}) \in \mathbb{R}^C \quad (3.22)$$

The warped radiance $\mathbf{c}^w$ and density σ^w are rendered into warped color $\hat{C}^w(\mathbf{r})$ and feature $\hat{F}^w(\mathbf{r})$ (3.10, 3.11). We add a loss to ensure that the warped color (and feature) match the ground-truth input for the current frame, similar to [55, 96]. As in NSFF [96], we found it important to downweight this loss in ambiguous regions that may contain occlusions. However, instead of learning explicit occlusion weights, we take inspiration from Kwea's method [8] and use the difference between the dynamic geometry and the warped dynamic geometry to downweight the loss:

$$w_{t'}(\mathbf{x}, \mathbf{t}, \mathbf{vid}) = \left| \frac{\sigma_d}{\sigma} - \frac{\sigma^w_{t'}}{\sigma} \right| \tag{3.23}$$

$$\hat{W}_{t'}(\mathbf{r}) = \int_0^{+\infty} T(t)\sigma(r(t))w_{t'}(r(t))dt \tag{3.24}$$

resulting in the following warping loss terms:

$$\mathcal{L}^w_c(\mathbf{r}) = \sum_{t' \in [-1,1]} (1 - W_{t'})(\mathbf{r})) \left\| C(\mathbf{r}) - \hat{C}^w_{t'}(\mathbf{r}) \right\|^2 \tag{3.25}$$

$$\mathcal{L}^w_f(\mathbf{r}) = \sum_{t' \in [-1,1]} (1 - W_{t'})(\mathbf{r}) \left\| F(\mathbf{r}) - \hat{F}^w_{t'}(\mathbf{r}) \right\|_1 \tag{3.26}$$

Flow regularization. As in prior work [55, 96] we use a 3D scene flow cycle term to encourage consistency between forward and backward scene flow predictions, down-weighing the loss in areas ambiguous due to occlusions:

$$\mathcal{L}_{cyc}(\mathbf{r}) = \sum_{t' \in [-1,1]} \sum_{\mathbf{x}} w_{t'}(\mathbf{x}, \mathbf{t}) \left\| s_{t'}(\mathbf{x}, \mathbf{t}) + s_{\mathbf{t}}(\mathbf{x} + s_{t'}, \mathbf{t} - t') \right\|_1, \tag{3.27}$$

with **vid** omitted for brevity. We also encourage spatial and temporal smoothness through the same priors as NSFF [96]:

$$\begin{aligned} \mathcal{L}_{sm}(\mathbf{r}) = &\sum_{\mathbf{x}} \sum_{t' \in [-1,1]} e^{-2\left\|\mathbf{x}-\mathbf{x}'\right\|_2} \left\| s_{t'}(\mathbf{x}, \mathbf{t}) - s_{t'}(\mathbf{x}', \mathbf{t}) \right\|_1 \\ &+ \sum_{\mathbf{x}} \left\| s_{\mathbf{t}-1}(\mathbf{x}, \mathbf{t}) + s_{\mathbf{t}+1}(\mathbf{x}, \mathbf{t}) \right\|_1, \end{aligned} \tag{3.28}$$

where $\mathbf{x}$ and $\mathbf{x}'$ indicate neighboring points along the camera ray $\mathbf{r}$.

We finally regularize the magnitude of predicted scene flow vectors to encourage the scene to be static through $\mathcal{L}_{slo}(\mathbf{r}) = \sum_{t' \in [\mathbf{t}-1, \mathbf{t}+1]} \sum_{\mathbf{x}} \left\| s_{t'}(\mathbf{x}, \mathbf{t}) \right\|_1$.

Static-dynamic factorization. As physically plausible solutions should have any point in space occupied by *either* a static or dynamic object, we encourage the spatial ratio of static vs dynamic density to either be 0 or 1 through a skewed binary entropy loss that favors static explanations of the scene [195]:

$$\mathcal{L}_e(\mathbf{r}) = \int_0^{+\infty} H\left(\frac{\sigma_d(\mathbf{r}(t))}{\sigma_s(\mathbf{r}(t)) + \sigma_d(\mathbf{r}(t))}^k \right) dt \tag{3.29}$$
$$\text{where } H(x) = -(x \cdot log(x) + (1 - x) \cdot log(1 - x)),$$

and with k set to 1.75, and further penalize the maximum dynamic ratio $\mathcal{L}_d(\mathbf{r}) = \max(\frac{\sigma_d(\mathbf{r}(t))}{\sigma_s + \sigma_d})$ along each ray.

Shadow loss. We penalize the squared magnitude of the shadow ratio $\mathcal{L}_\rho(\mathbf{r}) = \int_0^{+\infty} \rho_d(\mathbf{r}(t))^2 dt$ along each ray to prevent it from over-explaining dark regions [195].

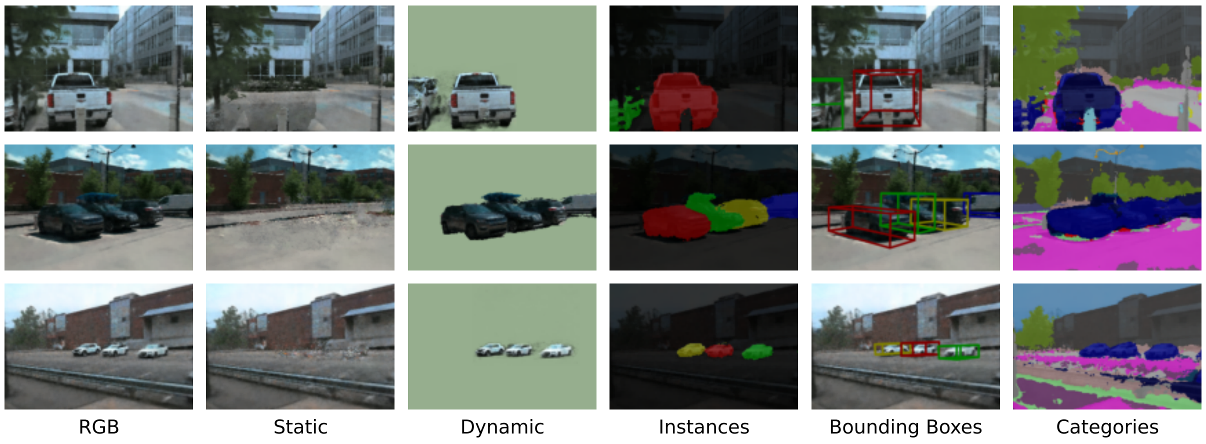


Figure 3.5: **City-1M.** We demonstrate SUDS's capabilities on multiple downstream tasks, including instance segmentation and 3D bounding box estimation without any labeled data (by just making use of geometric clustering). In the last column, we show category-level semantic classification by matching 3D (DINO) descriptors to a held-out video annotated with semantic labels. Please see text for more details.

3.4 Experiments

We demonstrate SUDS's city-scale reconstruction capabilities by presenting quantitative results against baseline methods (Table 3.1). We also show initial qualitative results for a variety of downstream tasks (Sec. 3.4.2). Even though we focus on reconstructing dynamic scenes at city scale, to faciliate comparisons with prior work, we also show results on small-scale but highly-benchmarked datasets such as KITTI and Virtual KITTI 2 (Sec. 3.4.3). We evaluate the various components of our method in Sec. 5.4.6.

3.4.1 Experimental Setup

2D feature extraction. We use Amir et al's feature extractor implementation [15] based on the dino_vits8 model. We downsample our images to fit into GPU memory and then upsample with nearest neighbor interpolation. We L2-normalize the features at the 11th layer of the model and reduce the dimensionality to 64 through incremental PCA [6].

Flow supervision. We explored using an estimator trained on synthetic data [170] in addition to directly computing 2D correspondences from DINO itself [15]. Although the correspondences are sparse (less than 5% of pixels) and expensive to compute, we found its estimates more robust and use it for our experiments unless otherwise stated.

Training. We train SUDS for 250,000 iterations with 4098 rays per batch and use a proposal sampling strategy similar to Mip-NeRF 360 [17]. We use Adam [87] with a learning rate of 5×10^{-3} decaying to 5×10^{-4}.

Metrics. We report quantitative results based on PSNR, SSIM [187], and the AlexNet implementation of LPIPS [225].

	Mega-NeRF (Chapter 2)	Mega-NeRF-T	Mega-NeRF-A	SUDS
PSNR $\uparrow$	16.42	16.46	16.70	**21.67**
SSIM $\uparrow$	0.493	0.493	0.493	**0.562**
LPIPS $\downarrow$	0.879	0.877	0.850	**0.554**

Table 3.1: **City-scale view synthesis on City-1M.** SUDS outperforms all baselines by a wide margin.

	$\leq$ 15k	15-30k	30-45k	$\geq$ 45k
$\uparrow$PSNR	22.86	21.99	21.35	20.75
$\uparrow$SSIM	0.583	0.569	0.557	0.538
$\downarrow$LPIPS	0.516	0.545	0.564	0.578

Images

	$\leq$ 60	60-90	90-120	$\geq$ 120
$\uparrow$PSNR	22.47	21.72	21.68	21.11
$\uparrow$SSIM	0.587	0.556	0.559	0.555
$\downarrow$LPIPS	0.526	0.557	0.557	0.565

Videos

	$\leq 2\ km^2$	2-3 km^2	3-4 km^2	$\geq 4\ km^2$
$\uparrow$PSNR	22.73	21.47	21.53	22.18
$\uparrow$SSIM	0.609	0.556	0.561	0.557
$\downarrow$LPIPS	0.512	0.564	0.555	0.536

Area

Table 3.2: **City-1M scaling.** We evaluate the effect of geographic coverage and the number of images and videos on cell quality. Although performance degrades sublinearly across all metrics, image and video counts have the largest impact.

3.4.2 City-Scale Reconstruction

City-1M dataset. We evaluate SUDS's large-scale reconstruction abilities on our collection of 1.28 million images across 1700 videos gathered across a 105 km^2 urban area using a vehicle-mounted platform with seven ring cameras and two LiDAR sensors. Due to the scale, we supervise optical flow with an off-the-shelf estimator trained on synthetic data [170] instead of DINO for efficiency. We divide City-1M into 48 cells using camera-based k-means clustering. Each cell covers 2.9 km^2 and 32k frames across 98 videos on average.

Baselines. We compare SUDS to the official Mega-NeRF (Chapter 2) implementation alongside two variants: Mega-NeRF-T which directly adds time as an input parameter to compute density and radiance, and Mega-NeRF-A which instead uses the latent embedding $A_{vid}\mathcal{F}(t)$ used by SUDS.

Results. We train both SUDS and the baselines using 48 cells and summarize our results in Table 3.1. SUDS outperforms all Mega-NeRF variants by a large margin. We provide qualitative results on view synthesis, static/dynamic factorization, unsupervised 3D instance segmentation and unsupervised 3D cuboid detection in Fig. 3.5 and tracking results in Fig. 3.6. We evaluate the effect of geographic coverage and number of frames/videos on cell quality in Table 3.2.

Instance segmentation. We derive the instance count as in prior work [157] by sampling dynamic density values σ_d, projecting those above a given threshold onto a discretized ground

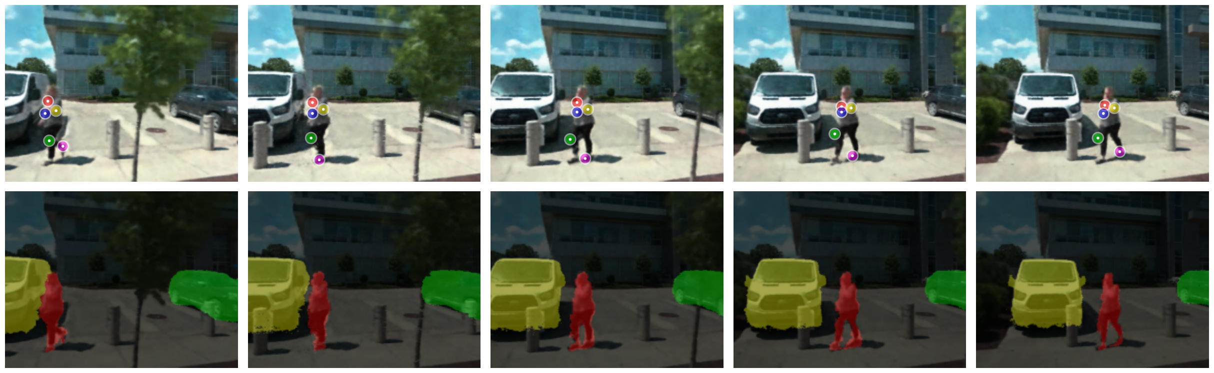


Figure 3.6: **Tracking.** We track keypoints (**above**) and instance masks (**below**) across several frames. As a 3D representation, SUDS can track correspondences through 2D occluders.

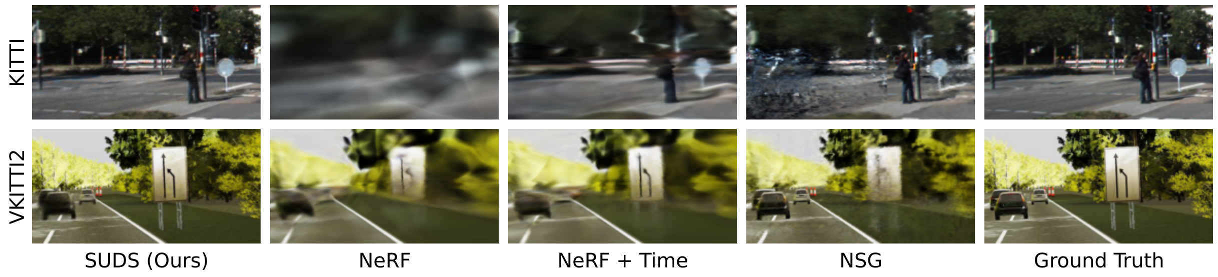


Figure 3.7: **KITTI and VKITTI2 view synthesis**. Prior work fails to represent the scene and NSG [126] renders ghosting artifacts near areas of movement. Our method forecasts plausible trajectories and generates higher-quality renderings.

plane before applying connected component labeling. We apply k-means to obtain 3D centroids and volume render instance predictions as for semantic segmentation.

3D cuboid detection. After computing point-wise instance assignments in 3D, we derive oriented bounding boxes based on the PCA of the convex hull of points belonging to each instance [1].

Tracking. We can compute mask and keypoint-level correspondences across frames after detecting instances (Sec. 3.4.2) by using Best-Buddies similarity [36] on features Φ within or between instances. As a 3D representation, SUDS can track correspondences through 2D occluders. We show an example in Fig. 3.6.

Semantic segmentation. Note the above tasks of instance segmentation and 3D cuboid detection do not require any additional labels as they make use of geometric clustering. We now show that the representation learned by SUDS can also enable downstream semantic tasks, by making use of a small number of 2D segmentation labels provided on a held-out video sequence. We compute the average 2D DINO descriptor for each semantic class from the held out frames and derive 3D semantic labels for all reconstructions by matching each 3D descriptor to the closest class centroid. This allows to produce 3D semantic label fields that can then be rendered in 2D as shown in Fig. 3.5.

	KITTI - 75%			KITTI - 50%			KITTI - 25%		
	↑PSNR	↑SSIM	↓LPIPS	↑PSNR	↑SSIM	↓LPIPS	↑PSNR	↑SSIM	↓LPIPS
NeRF [118]	18.56	0.557	0.554	19.12	0.587	0.497	18.61	0.570	0.510
NeRF + Time	21.01	0.612	0.492	21.34	0.635	0.448	19.55	0.586	0.505
NSG [126]	21.53	0.673	0.254	21.26	0.659	0.266	20.00	0.632	0.281
SUDS	**22.77**	**0.797**	**0.171**	**23.12**	**0.821**	**0.135**	**20.76**	**0.747**	**0.198**
	VKITTI2 - 75%			VKITTI2 - 50%			VKITTI2 - 25%		
	↑PSNR	↑SSIM	↓LPIPS	↑PSNR	↑SSIM	↓LPIPS	↑PSNR	↑SSIM	↓LPIPS
NeRF [118]	18.67	0.548	0.634	18.58	0.544	0.635	18.17	0.537	0.644
NeRF + Time	19.03	0.574	0.587	18.90	0.565	0.610	18.04	0.545	0.626
NSG [126]	23.41	0.689	0.317	23.23	0.679	0.325	21.29	0.666	0.317
SUDS	**23.87**	**0.846**	**0.150**	**23.78**	**0.851**	**0.142**	**22.18**	**0.829**	**0.160**

Table 3.3: **Novel View Synthesis.** As the fraction of training views decreases, accuracy drops for all methods. However, SUDS consistently outperforms prior work, presumably due to more accurate representations learned by our diverse input signals (such as depth and flow).

	SRN [159]	NeRF [118]	NeRF + Time	NSG [126]	PNF [90]	Ours
PSNR ↑	18.83	23.34	24.18	26.66	27.48	**28.31**
SSIM ↑	0.590	0.662	0.677	0.806	0.870	**0.876**

Table 3.4: **KITTI image reconstruction.** We outperform past work on image reconstruction accuracy, following their experimental protocol and self-reported accuracies [90, 126].

3.4.3 KITTI Benchmarks

Baselines. We compare SUDS to SRN [159], the original NeRF implementation [118], a variant of NeRF taking time as an additional input, NSG [126], and PNF [90]. Both NSG and PNF are trained and evaluated using ground truth object bounding box and category-level annotations.

Image reconstruction. We compare SUDS's reconstruction capabilities using the same KITTI [59] subsequences and experimental setup as prior work [90, 126]. We present results in Table 3.4. As PNF's implementation is not publicly available, we rely on their reported numbers. SUDS surpasses the state-of-the-art in PSNR and SSIM.

Novel view synthesis. We demonstrate SUDS's capabilities to generate plausible renderings at time steps unseen during training. As NSG does not handle scenes with ego-motion, we use subsequences of KITTI and Virtual KITTI 2 [54] with little camera movement. We evaluate the methods using different train/test splits, holding out every 4th time step, every other time step, and finally training with only one in every four time steps. We summarize our findings in Table 3.3 along with qualitative results in Fig. 3.7. SUDS achieves the best results across all splits and metrics. Both NeRF variants fail to properly represent the scene, especially in dynamic areas. Although we provide NSG with the ground truth object poses at render time, it fails to learn a clean decomposition between objects and the background, especially as the number of training view decreases, and generates ghosting artifacts near areas of movement.

	↑PSNR	↑SSIM	↓LPIPS
w/o Depth loss	22.74	0.715	0.292
w/o Optical flow loss	22.18	0.708	0.302
w/o Warping loss	17.53	0.622	0.478
w/o Appearance embedding	22.54	0.704	0.296
w/o Occlusion weights	22.56	0.711	0.297
w/o Separate branches	19.73	0.570	0.475
Full Method	**22.95**	**0.718**	**0.289**

Table 3.5: **Diagnostics.** Flow-based warping is the single-most important input, while depth is the least crucial input.

3.4.4 Diagnostics

We ablate the importance of major SUDS components by removing their respective loss terms along with occlusion weights, the latent embedding $A_{vid}\mathcal{F}(t)$ used to compute static color $\mathbf{c}_s$, and separate model branches. We run all approaches for 125,000 iterations across our datasets and summarize the results in Table 5.7. Although all components help performance, flow-based warping is by far the single most important input. Interestingly, depth is the least crucial input, suggesting that SUDS can generalize to settings where depth measurements are not available.

3.5 Discussion

We present a modular approach towards building dynamic neural representations at previously unexplored scale. Our multi-branch hash table structure enables us to disentangle and efficiently encode static geometry and transient objects across thousands of videos. SUDS makes use of unlabeled inputs to learn semantic awareness and scene flow, allowing it to perform several downstream tasks while surpassing state-of-the-art methods that rely on human labeling.

3.5.1 Limitations

Although SUDS scales neural rendering to (in our knowledge) the largest dynamic NeRF representation to date, many open challenges remain. Its rendering quality, especially with regards to dynamic objects, does not reach photorealistic levels. Its rendering speed, while faster than Mega-NeRF (Chapter 2), does not reach real-time requirements at HD resolution. We address this shortcoming in Chapter 4. We list additional limitations below.

Video boundaries. Although our global representation of static geometry is consistent across all videos used for reconstruction, all dynamic objects are video-specific. Put otherwise, our method does not allow us to extrapolate the movement of objects outside of the boundaries of videos from which they were captured, nor does it provide a straightforward way of rendering dynamic visuals at boundaries where camera rays intersect regions with training data originating from disjoint video sequences.

Flow quality. Although our method tolerates some degree of noisiness in the supervisory optical flow input, high-quality flow still has a measurable impact on model performance (and completely incorrect supervision degrades quality). We also assume that flow is linear between observed timestamps to simplify our scene flow representation.

Resources. Modeling city scale requires a large amount of dataset pre-processing, including, but not limited to: extracting DINO features, computing optical flow, deriving normalized coordinate bounds, and storing randomized batches of training data to disk. Collectively, our intermediate representation required more than 20TB of storage even after compression.

Shadows. SUDS attempts to disentangle shadows underneath transient objects. However, if a shadow is present in all observations for a given location (eg: a parking spot that is always occupied, even by different cars), SUDS may attribute the darkness to the static topology, as evidenced in several of our videos, even if the origin of the shadow is correctly assigned to the dynamic branch.

Instance-level tasks. Although we provide initial qualitative results on instance-level tasks as a first step towards true 3D segmentation backed by neural radiance field, SUDS is not competitive with conventional approaches.

Chapter 4

Real-Time Rendering at VR Resolution

The contents of this chapter were published as "HybridNeRF: Efficient Neural Rendering via Adaptive Volumetric Surfaces" in CVPR 2024

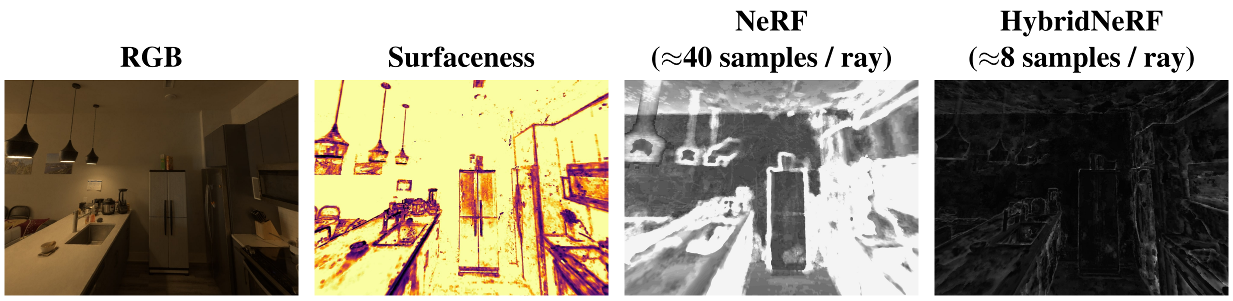


Figure 4.1: **HybridNeRF [181].** We train a hybrid surface–volume representation via *surfaceness* parameters that allow us to render most of the scene with few samples. We track Eikonal loss as we increase surfaceness to avoid degrading quality near fine and translucent structures (such as wires). In the two right-most panels, we visualize the number of samples per ray (brighter is higher). Our model renders in high fidelity at 2K$\times$2K resolution at real-time frame rates.

4.1 Introduction

The previous chapters focused on how to efficiently neural scene representations at scale. This chapter explores how to efficiently render such trained representations.

Efficiency We seek to construct a representation that enables high-quality efficient rendering, which is necessary for immersive applications, such as augmented reality or virtual teleconferencing.

While recent rasterization-based techniques, such as mesh baking [209] or Gaussian splatting [81], are very efficient, they still struggle to capture transparent or fine structures, and view-dependent effects (like reflections or specularities), respectively. Instead, we focus on NeRF's

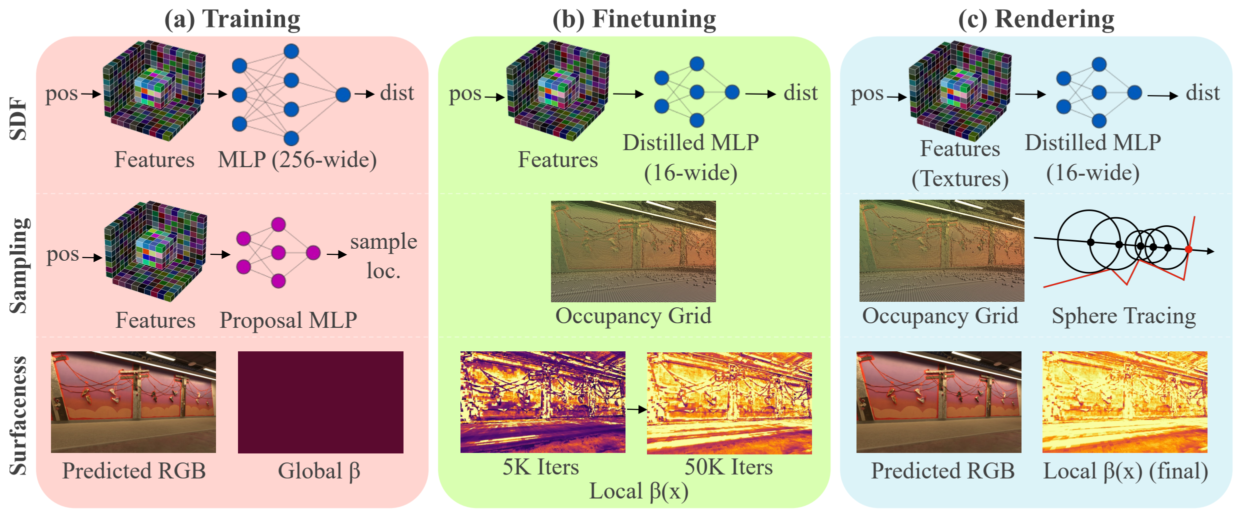


Figure 4.2: **Approach.** In the first phase of our pipeline (**a**), we train a VolSDF-like [208] model with distance-adjusted Eikonal loss to model backgrounds without a separate NeRF (Section 4.3.3). We then crucially transition from a uniform surfaceness parameter β to position-dependent $\beta(\boldsymbol{x})$ values to model most of the scene as thin surfaces (needing few samples) without degrading quality near fine and semi-opaque structures (**b**). Since our model behaves as a valid SDF in >95% of the scene, we use sphere tracing at render time (**c**) along with lower-level optimizations (hardware texture interpolation) to query each sample as efficiently as possible.

standard ray casting paradigm, and propose techniques that enable a better speed–quality trade-off.

Rendering We start with the observation that neural implicit surface representations, such as signed distance functions (SDFs), which were originally proposed to improve the geometry quality of NeRFs via regularization [185, 208], can *also* be used to dramatically increase efficiency by requiring fewer samples per ray. In the limit, only a single sample on the surface is required. In practice, renderers still need to identify the location of the target sample(s), which can be done by generating samples via an initial proposal network [17] or other techniques, such as sphere tracing [108].

Surfaceness While surface-based neural fields are convenient for rendering, they often struggle to reconstruct scenes with thin structures or view-dependent effects, such as reflections and translucency. This is one reason that surfaces are often transformed into volumetric models for rendering [208]. A crucial transformation parameter is a scalar temperature β that is used to convert a β-scaled signed distance value into a density. Higher temperatures tend to produce an ideal binary occupancy field that can improve rendering speed but can struggle for challenging regions as explained above. Lower temperatures allow the final occupancy field to remain flexible, whereby the β-scaled SDF essentially acts as a reparameterization of the underlying occupancy field. As such, we refer to β as the *surfaceness* of the underlying scene (see Fig. 4.1). Prior work treats β as a global parameter that is explicitly scheduled or learned via gradient descent [208].

We learn it in a spatially adaptive manner.

Contributions Our primary contribution is a hybrid surface–volume representation that combines the best of both worlds. Our key insight is to replace the global parameter β with spatially-varying parameters $\beta(\boldsymbol{x})$ corresponding to the surfaceness of regions in the 3D scene. At convergence, we find that most of the scene ($> 95\%$) can be efficiently modeled as a surface. This allows us to render with far fewer samples than fully volumetric methods, while achieving higher fidelity than pure surface-based approaches. Additionally,

1. We propose a weighted Eikonal regularization that allows our method to render high-quality complex backgrounds without a separate background model.
2. We implement specific rendering optimizations, such as hardware texture interpolation and sphere tracing, to significantly accelerate rendering at high resolutions.
3. We present state-of-the-art reconstruction results on three different datasets, including the challenging Eyeful Tower dataset [201], while rendering almost $10\times$ faster.

4.2 Related Work

Many works try to accelerate the rendering speed of neural radiance fields (NeRF). We discuss a representative selection of such approaches below.

Voxel baking Some of the earliest NeRF acceleration methods store precomputed non-view dependent model outputs, such as spherical harmonics coefficients, into finite-resolution structures [28, 42, 58, 68, 216]. These outputs are combined with viewing direction to compute the final radiance at render time, bypassing the original model entirely. Although these methods render extremely quickly (some >200 FPS [58]), they are limited by the finite capacity of the caching structure and cannot capture fine details at room scale.

Feature grids Recent methods use a hybrid approach that combines a learned feature grid with a much smaller MLP than the original NeRF [26, 50, 120]. Instant-NGP [120] (iNGP), arguably the most popular of these methods, encodes features into a multi-resolution hash table. Although these representations speed up rendering, they cannot reach the level needed for real-time HD rendering alone, as even iNGP reaches less than 10 FPS on real-world datasets at high resolution. MERF [140] comes closest through a baking pipeline that uses various sampling and memory layout optimizations that we also make use of in our implementation.

Surface–volume representations Several methods [125, 185, 208] derive density values from the outputs of a signed distance function, which are then rendered volumetrically as in NeRF. These hybrid representations retain NeRF's ease of optimization while improving surface geometry. Follow-ups [64, 209] bake the resulting surface geometry into a mesh that is further optimized and simplified. Similar to early voxel-baking approaches, these methods render quickly (>70 FPS) but are limited by the capacity of the mesh and texture, and thus struggle to model thin structures, transparency, and view-dependent effects. We train a similar SDF representation in our method but continue using the base neural model at render time. Concurrent to our work, Adaptive shells [188] augments NeuS [185] with a spatially-varying kernel similar to our adaptive surfaceness described in Section 4.3.2.

Sample efficiency Several approaches accelerate rendering by intelligently placing far fewer samples along each ray than the original hierarchical strategy proposed by NeRF [17, 65, 91, 121, 131]. These methods all train auxiliary networks that are cheaper to evaluate than the base model. However, as they are based on purely volumetric representations, they are limited in practice as to how few samples they can use per ray without degrading quality, and therefore exhibit a different quality–performance tradeoff curve than ours.

Gaussians Recent methods take inspiration from NeRF's volume rendering formula but discard the neural network entirely and instead parameterize the scene through a set of 3D Gaussians [81, 85, 86, 183]. Of these, 3D Gaussian splatting [81] has emerged as the new state of the art, rendering at $>$100 FPS with higher fidelity than previous non-neural approaches. Although encouraging, it is sensitive to initialization (especially in far-field areas) and limited in its ability to reason about inconsistencies within the training dataset (such as transient shadows) and view dependent effects.

4.3 Method

Given a collection of RGB images and camera poses, our goal is to learn a 3D representation that generates novel views at VR resolution (at least 2K$\times$2K pixels) in real-time (at least 36 FPS), while achieving a high degree of visual fidelity. As we target captures taken under real-world conditions, our representation must be able to account for inconsistencies across training images due to lighting changes and shadows (even in "static" scenes). We build upon NeRF's raycasting paradigm, which can generate highly photorealistic renderings, and improve upon its efficiency. As the world mostly consists of surfaces, we train a representation that can render surfaces with few samples and without degrading the rest of the scene. We outline our method in Fig. 4.2 and present our model architecture and the first training stage in Section 4.3.1, which is followed by finetuning of our model to accelerate rendering without compromising quality in Section 4.3.2. We discuss how to model unbounded scenes in Section 4.3.3 and present final render-time optimizations in Section 4.3.4.

4.3.1 Representation

Preliminaries NeRF [118] represents a scene as a continuous volumetric radiance field that encodes the scene's geometry and view-dependent appearance within the weights of an MLP. NeRF renders pixels by sampling positions $\boldsymbol{x}_i$ along the corresponding camera ray, querying the MLP to obtain density and color values, $\sigma_i \coloneqq \sigma(\boldsymbol{x}_i)$ and $\mathbf{c}_i \coloneqq \boldsymbol{c}(\boldsymbol{x}_i, \boldsymbol{d}_r)$, respectively (with $\boldsymbol{d}_r$ as the ray direction). The density values σ_i are converted into opacity values $\alpha_i \coloneqq 1 - \exp(-\sigma_i \delta_i)$, where δ_i is the distance between samples. The final ray color $\hat{\boldsymbol{c}}_r \coloneqq \sum_{i=0}^{N-1} \boldsymbol{c}_i w_i$

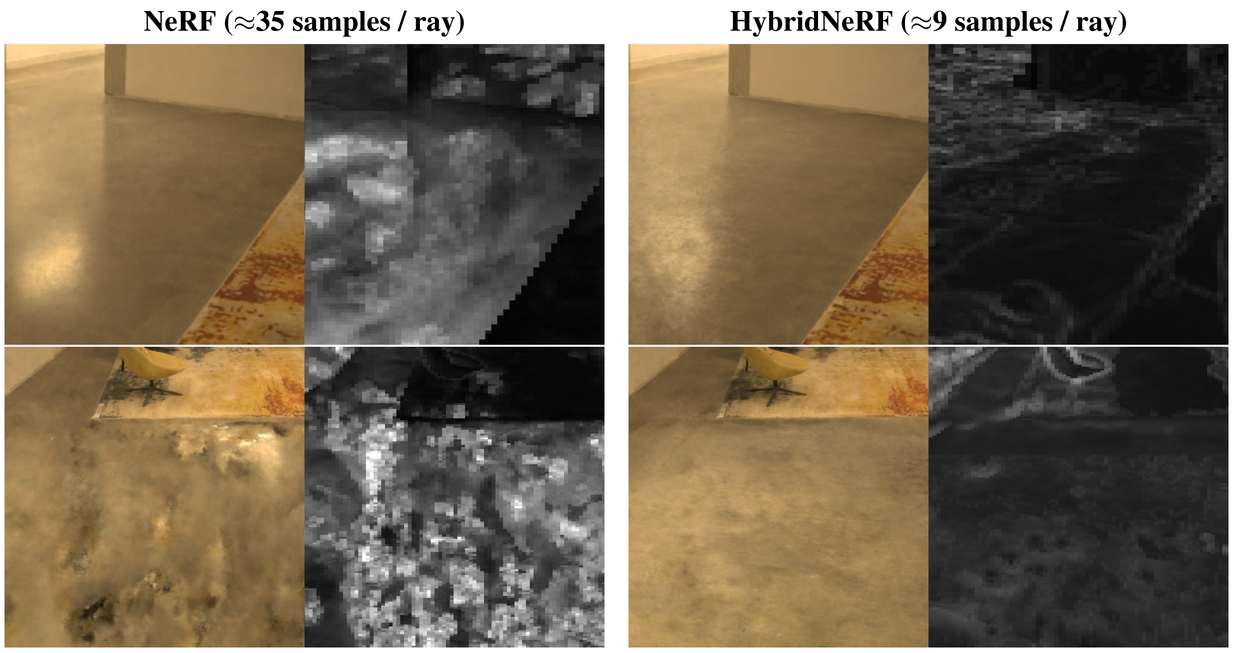


Figure 4.3: **Surfaces.** Since NeRF directly predicts density, it often 'cheats' by modeling specular surfaces, such as floors, as semi-transparent volumes that require many samples per ray (heatmaps shown on the **right**, with brighter values corresponding to more samples). Methods that derive density from signed distances, such as ours, improve surface geometry and appearance while using fewer samples per ray.

is obtained as the combination of the color samples $\boldsymbol{c}_i$ with weights $w_i := \exp(-\sum_{j=0}^{i-1}\sigma_j\delta_j)\alpha_i$. The training process optimizes the model by sampling batches of image pixels and minimizing the L2 reconstruction loss. We refer to Mildenhall et al. [118] for details.

Modeling density The original NeRF representation has the flexibility of representing semi-transparent surfaces, for the density field is not forced to saturate. However, the model often abuses this property by generating semi-transparent volumes to mimic reflections and other view-dependent effects (Fig. 4.3). This hampers our goal of minimizing the samples per ray needed for rendering.

To address this problem, surface–volume representations [125, 185, 208] learn well-defined surfaces by interpreting MLP outputs $f(\boldsymbol{x})$ as a signed distance field (SDF) to represent scene surfaces as the zero-level set of the function f.

As the norm of the gradient of an SDF should typically be 1, the MLP is regularized via the Eikonal loss:

$$\mathcal{L}_{\text{Eik}}(\mathbf{r}) := \sum_{i=0}^{N-1} \eta_i(\|\nabla f(\boldsymbol{x}_i)\| - 1)^2, \tag{4.1}$$

where η_i is a per-sample loss weight typically set to 1. The signed distances are converted into densities σ_{SDF} that are paired with color predictions, and rendered as in NeRF. Specifically, we

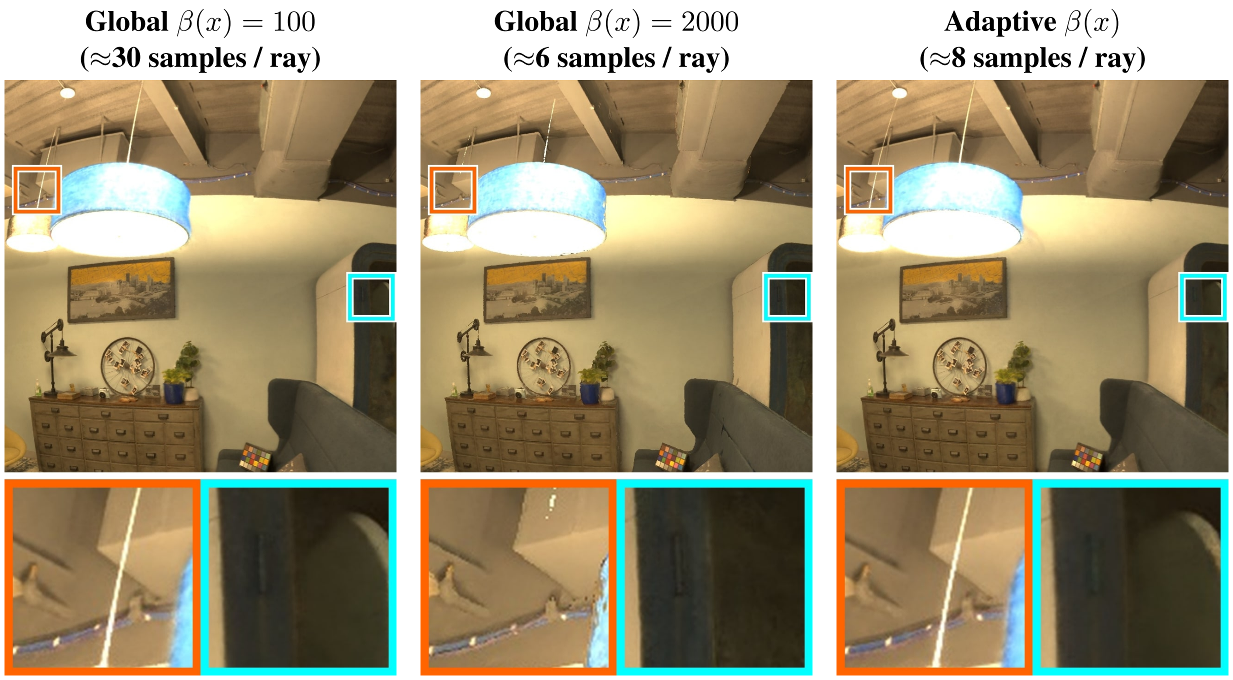


Figure 4.4: **Choice of β.** Increasing β reduces the number of samples needed to render per ray, but negatively impacts quality near fine objects (lamp wires) and transparent structures (glass door).

follow VolSDF's approach [208] and define:

$$\sigma_{\text{SDF}}(\boldsymbol{x}) \coloneqq \beta(\boldsymbol{x})\Psi(f(\boldsymbol{x})\beta(\boldsymbol{x})), \tag{4.2}$$

where $\beta(\boldsymbol{x}) > 0$ determines the *surfaceness* of point $\boldsymbol{x}$, *i.e.* how concentrated the density should be around the zero-level set of f, and Ψ is the CDF of a standard Laplace distribution:

$$\Psi(s) = \begin{cases} \frac{1}{2}\exp(-s) & \text{if } s > 0 \\ 1 - \frac{1}{2}\exp(s) & \text{if } s \leq 0. \end{cases} \tag{4.3}$$

In prior works, the surfaceness $\beta(\boldsymbol{x})$ is independent of position $\boldsymbol{x}$. We instead consider a surfaceness *field* implemented as a 512^3 grid of values queried via nearest-neighbor interpolation. We first constrain the surfaceness parameters to be globally uniform, and allow them to diverge spatially during the finetuning stage (Section 4.3.2).

Model architecture We render color and distance as follows:

$$\boldsymbol{c}(\boldsymbol{x}, \boldsymbol{d}) = \text{MLP}_{\text{col}}(\Gamma_{\text{col}}(\boldsymbol{x}), \text{SH}(\boldsymbol{d})) \tag{4.4}$$
$$f(\boldsymbol{x}) = \text{MLP}_{\text{dist}}(\Gamma_{\text{dist}}(\boldsymbol{x})), \tag{4.5}$$

where Γ_{col} and Γ_{dist} are separate spatial feature encodings.

For the encodings, we use dense multi-resolution 3D feature grids in combination with multi-resolution triplanes [23, 50] to featurize 3D sample locations. We predict color $\boldsymbol{c}$ and signed distance f with separate grids, each followed by an MLP, and use a small proposal network similar to that used by Nerfacto [169] to improve sampling efficiency. For a given 3D point, we fetch $K = 4$ features per level from (1) the 3D feature grids at 3 resolution levels (128^3, 256^3 and 512^3) via trilinear interpolation, and (2) from triplanes at 7 levels (from 128^2 to $8{,}192^2$) via bilinear interpolation. We sum the features across levels (instead of concatenation [50, 120]), and concatenate the summed features from the 3D grid to those from the 3 triplanes to obtain a $4K = 16$-dimensional MLP input. We encode viewing direction through spherical harmonics (up to the 4th degree) as an auxiliary input to the color MLP. As our feature grid is multi-resolution, we handle aliasing as in VR-NeRF [201] by dampening high-resolution grid features based on pixel footprint. For a given sample $\boldsymbol{x}$, we derive a pixel radius $p(\boldsymbol{x})$ in the contracted space, and calculate the optimal feature level $L(\boldsymbol{x})$ based on the Nyquist–Shannon sampling theorem:

$$L(\boldsymbol{x}) \coloneqq -\log_2(2s \cdot p(\boldsymbol{x})), \tag{4.6}$$

where s is our base grid resolution (128). We then multiply grid features at resolution level L with per-level weights w_L:

$$w_L = \begin{cases} 1 & \text{if } L < \lfloor L(\boldsymbol{x}) \rfloor \\ L(\boldsymbol{x}) - \lfloor L(\boldsymbol{x}) \rfloor & \text{if } \lfloor L(\boldsymbol{x}) \rfloor < L \leq \lceil L(\boldsymbol{x}) \rceil \\ 0 & \text{if } \lceil L(\boldsymbol{x}) \rceil < L. \end{cases} \tag{4.7}$$

Optimization We sample random batches of training rays and optimize our color and distance fields by minimizing the photometric loss $\mathcal{L}_{\text{photo}}$ and Eikonal loss $\mathcal{L}_{\text{Eik}}$ along with interlevel loss $\mathcal{L}_{\text{prop}}$ [17] to train the proposal network:

$$\mathcal{L}(\boldsymbol{r}) \coloneqq \mathcal{L}_{\text{photo}}(\boldsymbol{r}) + \lambda_{\text{Eik}} \mathcal{L}_{\text{Eik}}(\boldsymbol{r}) + \mathcal{L}_{\text{prop}}(\boldsymbol{r}), \tag{4.8}$$

with $\lambda_{\text{Eik}} = 0.01$ in our experiments.

4.3.2 Finetuning

Adaptive surfaceness The first stage of our pipeline uses a global surfaceness value $\beta(\boldsymbol{x}) = \bar{\beta}$ for all $\boldsymbol{x}$, as in existing approaches [185, 208]. As $\bar{\beta}$ increases, the density σ_{SDF} in free-space areas converges to zero (Eq. (4.3)), reducing the required number of samples per ray. However, uniformly increasing this scene-wide parameter degrades the rendering quality near fine-grained and transparent structures (see Fig. 4.4).

We overcome this limitation by making $\beta(\boldsymbol{x})$ spatially adaptive via a 512^3 voxel grid. One possible approach is to directly optimize $\beta(\boldsymbol{x})$ via gradient descent, but we find that this overly relaxes the constraint on SDF correctness such that $f(\boldsymbol{x})$ predicts arbitrary density values as in the original NeRF. We instead rely on the Eikonal loss as a natural indicator of where the model cannot accurately reconstruct the scene via an SDF (and where we should therefore use a "softer" formulation). We collect per-sample triplets $(\boldsymbol{x}, \eta, w)$ rendered during the finetuning

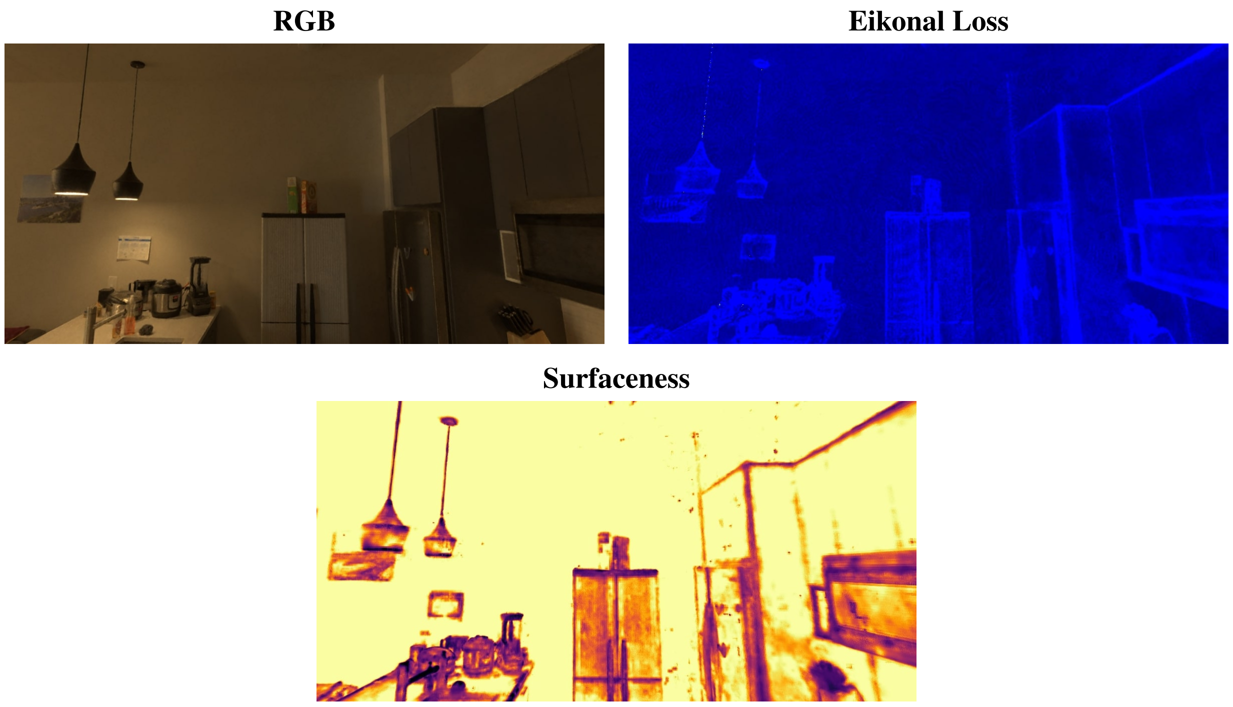


Figure 4.5: **Spatially adaptive surfaceness.** We make $\beta(\boldsymbol{x})$ spatially adaptive by means of a 512^3 voxel grid that we increase during the finetuning stage. We track Eikonal loss as we increase surfaceness as it is highest near object boundaries and semi-transparent surfaces (**top-right**, brighter = higher loss) that degrade when surfaceness is too high (Fig. 4.4). We stop increasing surfaceness in regions that cross a given threshold.

process, accumulate them over multiple training iterations (5,000), and partition them across the voxels of the surfaceness grid. Let $\Lambda_{\boldsymbol{v}}$ be the subset associated with voxel $\boldsymbol{v}$ corresponding to $\beta_{\boldsymbol{v}}$. We increase $\beta_{\boldsymbol{v}}$ by a fixed increment (100) if:

$$\frac{\sum_{(\boldsymbol{x},\eta,w)\in\Lambda_{\boldsymbol{v}}} w\eta(\|\nabla f(\boldsymbol{x})\| - 1)^2}{\sum_{(\ldots,w)\in\Lambda_{\boldsymbol{v}}} w} < \bar{\gamma}, \tag{4.9}$$

where $\bar{\gamma} := 0.25$ is a predefined threshold. Fig. 4.5 illustrates our approach.

Proposal network baking Although the proposal network allows us to quickly learn the scene geometry during the first stage of training, it is too expensive to evaluate in real time. We follow MERF's protocol [140] to bake the proposal network into a 1024^3 binary occupancy grid. We render all training rays and mark a voxel as occupied if there exists at least one sampled point $\boldsymbol{x}_i$ such that $\max(w_i, \sigma_i) > 0.005$. We finetune our model using the occupancy grid to prevent any loss in quality.

MLP distillation We find it important to use a large 256 channel-wide MLP to represent the signed distance f during the first training phase in order to learn accurate scene geometry. However, we later distill f into a smaller 16-wide network (f_{small}). We do so by sampling random rays from our training set for 5,000 iterations and minimizing the difference between $f(\boldsymbol{x}_i)$ and $f_{\text{small}}(\boldsymbol{x}_i)$ at every sampled point:

$$\mathcal{L}_{\text{dist}}(\boldsymbol{r}) := \sum_{i=0}^{N-1} |f(\boldsymbol{x}_i) - f_{\text{small}}(\boldsymbol{x}_i)|, \tag{4.10}$$

with a stop gradient applied to the outputs of f. We then discard the original SDF f and switch to using the distilled counterpart f_{small} for the rest of the finetuning stage.

4.3.3 Backgrounds

Many scenes we wish to reconstruct contain complex backgrounds that surface–volume methods struggle to replicate [95, 185, 208]. BakedSDF [209] defines a contraction space [17] in which the Eikonal loss of Eq. (4.1) is applied. However, we found this to negatively impact foreground quality. Other approaches use separate NeRF background models [223], which effectively doubles inference and memory costs, and makes them ill-suited for real-time rendering.

Relation between volumetric and surface-based NeRFs We discuss how to make a single MLP behave as an approximate SDF in the foreground and a volumetric model in the background. Both types of NeRF derive density σ by applying a non-linearity to the output of an MLP. Our insight is that although the original NeRF uses ReLU, any non-linear mapping to $\mathbb{R}^+$ may be used in practice, including our scaled CDF Ψ (β omitted without loss of generality). Since Ψ is invertible (as it is a CDF), $\sigma(\boldsymbol{x})$ and $\Psi(f(\boldsymbol{x}))$ are functionally equivalent as there exists an f such that $\Psi(f(\boldsymbol{x})) = \sigma(\boldsymbol{x})$ for any given point $\boldsymbol{x}$. Put otherwise, it is the Eikonal regularization that causes the divergence in behavior between both methods — in its absence, an "SDF" MLP is free to behave exactly as the density MLP in the original NeRF!

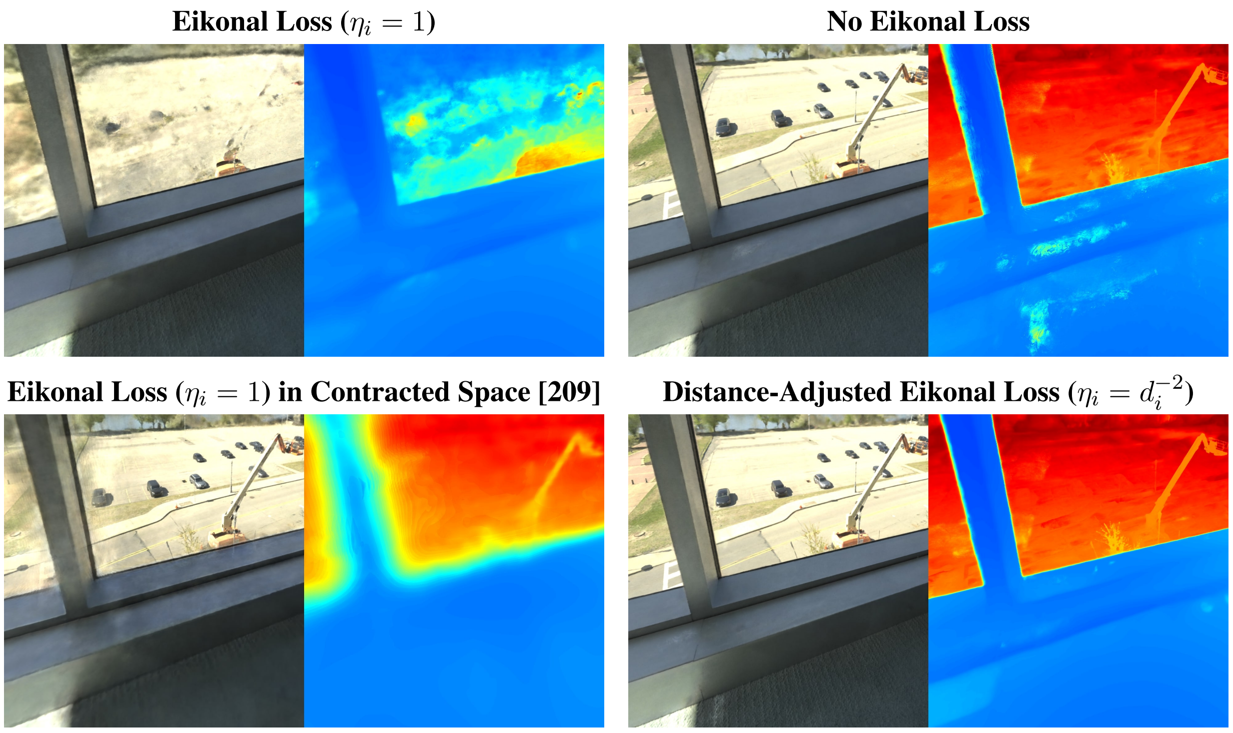

Eikonal Loss ($\eta_i = 1$)
No Eikonal Loss
Eikonal Loss ($\eta_i = 1$) in Contracted Space [209]
Distance-Adjusted Eikonal Loss ($\eta_i = d_i^{-2}$)

Figure 4.6: **Backgrounds.** Using standard Eikonal loss affects background reconstruction (**top-left**) while applying it in contracted space [209] affects the foreground (**bottom-left**). Omitting Eikonal loss entirely causes surface–volume methods to revert to NeRF's behavior, which improves background quality but degrades foreground surface reconstruction (**top-right**). By using distance-adjusted sample weights $\eta_i = d_i^{-2}$, we improve background reconstruction without impacting foreground quality (**bottom-right**).

Distance-adjusted loss We use a *distance-adjusted* Eikonal loss during training by using per-sample loss weights $\eta_i = \frac{1}{d_i^2}$ (where d_i is the metric distance along the ray of sample $\boldsymbol{x}_i$) instead of commonly-used uniform weights ($\eta_i = 1$) to downweight the loss applied to far-field regions. Intuitively, this encourages our method to behave as a valid SDF in the foreground (with well-defined surfaces) and more like NeRF in the background (to enable accurate reconstruction) without the need for separate foreground and background models. Fig. 4.6 and Table 4.6 illustrate the different approaches.

4.3.4 Real-Time Rendering

Texture storage Our architecture enables us to use lower-level optimizations. Methods such as iNGP [120] use concatenated multi-resolution features stored in hash tables. Since we use explicit 3D grids and triplanes, we can store our features as textures at render time, taking advantage of increased memory locality and texture interpolation hardware. As we sum our multi-resolution features during training, we optimize the number of texture fetches by storing pre-summed features g' at resolution level L (where we store $g'(\boldsymbol{v}) = \sum_{l=0}^{L} g(\boldsymbol{v}, l)$ for each texel in L). For a given sample $\boldsymbol{x}$ at render time, we obtain its anti-aliased feature by interpolating between the two levels implied by its pixel area $p(\boldsymbol{x})$, reducing the number of texture fetches to 8 queries per MLP evaluation from the original $3+3\times7=24$ (assuming three 3D grids and seven triplane levels), a $3\times$ reduction.

Sphere tracing Volumetric methods that use occupancy grids [e.g. 120, 140] sample within occupied voxels using a given step size. This hyperparameter must be carefully tuned to strike the proper balance between quality (not skipping thin surfaces) and performance (not excessively sampling empty space). Modeling an SDF allows us to sample more efficiently by advancing toward the predicted surface using *sphere tracing* [145]. At each sample point $\boldsymbol{x}_i$ and predicted surface distance $s = f(\boldsymbol{x}_i)$, we advance by $0.9s$ (chosen empirically to account for our model behaving as an approximate SDF) until hitting the surface (predicted as $s \leq 2\times10^{-4}$). We only perform sphere tracing where our model behaves as a valid SDF (determined by $\beta(\boldsymbol{x}_i) > 350$ in our experiments), and fall back to a predefined step size of 1 cm otherwise.

4.4 Experiments

As our goal is high-fidelity view synthesis at VR resolution ($\approx$ 4 megapixels), we primarily evaluate HybridNeRF against the Eyeful Tower dataset [201], which contains high-fidelity scenes designed for walkable VR (Section 4.4.2). We compare our work to a broader range of methods on additional datasets in Section 4.4.3. We ablate our design in Section 5.4.6.

4.4.1 Implementation

We train our models in the PyTorch framework [130] and implement our renderer in C++/CUDA. We parameterize unbounded scenes with MERF's piecewise-linear contraction [140] so that our renderer can query the occupancy grid via ray-AABB intersection. We train on each scene for

Table 4.1: **Eyeful Tower [201] results.** We omit 3DGS results for fisheye scenes as their implementation does not handle fisheye projection. Along with 3DGS and MERF, ours is the only to reach the 36 FPS target for VR along with a $>$1.5 dB PSNR improvement in quality.

	Pinhole			Fisheye			Overall			
	↑PSNR	↑SSIM	↓LPIPS	↑PSNR	↑SSIM	↓LPIPS	↑PSNR	↑SSIM	↓LPIPS	↑FPS
iNGP* [120]	27.35	0.826	0.361	33.32	0.938	0.155	30.06	0.877	0.267	4.55
VolSDF* [208]	27.10	0.856	0.310	34.09	0.951	0.116	30.28	0.899	0.222	15.29
MERF (pre-baking) [140]	26.44	0.831	0.506	31.18	0.922	0.549	28.59	0.872	0.526	18.11
MERF (baked) [140]	25.99	0.830	0.525	31.09	0.921	0.546	28.31	0.871	0.535	<u>60.18</u>
3D Gaussian splatting [81]	27.42	<u>0.877</u>	<u>0.291</u>	—	—	—	—	—	—	**138.22**
VR-NeRF [201]	<u>28.08</u>	0.834	0.326	<u>34.53</u>	0.951	0.130	31.01	0.888	0.237	6.05
Zip-NeRF [18]	**29.71**	0.868	0.305	34.19	**0.958**	**0.109**	**31.75**	<u>0.909</u>	<u>0.216</u>	<0.1
HybridNeRF	<u>29.07</u>	**0.880**	**0.268**	**34.57**	<u>0.952</u>	<u>0.115</u>	<u>31.57</u>	**0.913**	**0.198**	45.78

* Our implementation. VolSDF: with iNGP acceleration.

200,000 iterations (100,000 in each training stage) with 12,800 rays per batch using Adam [87] and a learning rate of 2.5×10^{-3}.

4.4.2 VR Rendering

Eyeful Tower dataset The dataset consists of room-scale captures, each containing high-resolution HDR images at 2K resolution, captured using a multi-view camera rig. Although care is taken to obtain the best quality images possible, inconsistencies still appear between images due to lighting changes and shadows from humans and the capture rig itself. We model as much of the dynamic range as possible by mapping colors in the PQ color space [161], as proposed in VR-NeRF [201], during training and tonemap to sRGB space during evaluation to compare against non-HDR baselines.

Baselines We compare HybridNeRF to baselines across the fidelity/speed spectrum. We benchmark several volumetric methods, including (1) iNGP [120], (2) VR-NeRF [201], which extends iNGP's [120] primitives to better handle HDR reconstruction, (3) Zip-NeRF [18], an anti-aliasing method that generates high-quality renderings at the cost of speed, and (4) MERF [140], a highly optimized method that uses sampling and memory layout optimizations to accelerate rendering. We also compare to VolSDF [208] as a hybrid surface–volume method similar to the first stage of our method. As the original VolSDF implementation uses large MLPs that are unsuitable for real-time rendering, we use an optimized version built on top of iNGP's acceleration primitives as a fairer comparison. We also benchmark 3D Gaussian splatting [81] as a non-neural approach that represents the current state of the art with across rendering quality and speed.

Metrics We report quantitative results based on PSNR, SSIM [187], and the AlexNet implementation of LPIPS [225] and measure frame rates rendered at 2K$\times$2K resolution on a single NVIDIA RTX 4090 GPU.

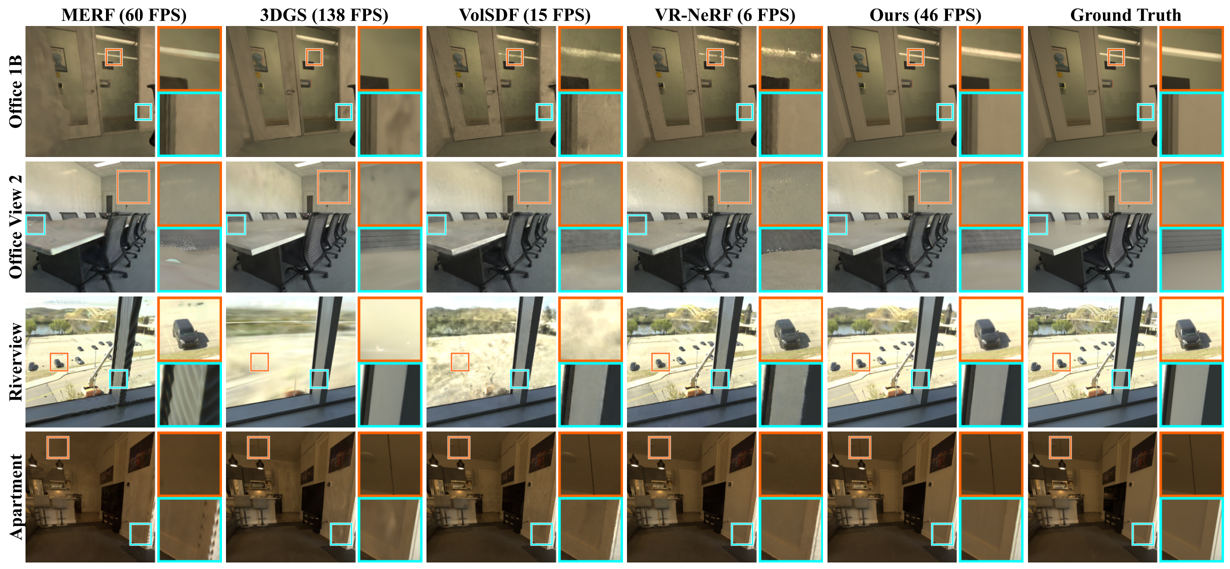


Figure 4.7: **Eyeful Tower [201].** HybridNeRF is the only method to accurately model reflections and shadows (**first two rows**), far-field content (**third row**) and fine structures (**bottom row**) at real-time frame rates at 2K×2K resolution.

Results We summarize our results in Table 4.1 along with qualitative results in Fig. 4.7. VR-NeRF [201], iNGP [120], and Zip-NeRF [18] render well below real-time frame rates. Our VolSDF implementation, which uses the same primitives as iNGP, is 3× faster merely from the benefits of using a surface representation (and fewer samples per ray). MERF [140], as a volume representation, relies instead on precomputation to accelerate rendering by explicitly storing diffuse color and density outputs during its baking stage and using only a small MLP to model view-dependent effects. Although it reaches a high frame rate, it provides the least visually appealing results amongst our baselines. 3D Gaussian splatting [81] renders the fastest, but struggles with shadows and lighting changes across the training views and models them as unsightly floaters. Our method is the only to achieve both high quality and real-time frame rates.

4.4.3 Additional Comparisons

Datasets We evaluate HybridNeRF on MipNeRF-360 [17] as a highly-referenced dataset evaluated by many prior methods, and ScanNet++ [213] as a newer benchmark built from high-resolution captures of indoor scenes that are relevant to our goal of enabling immersive AR/VR applications. We test on all scenes in the former and a subset of the latter.

Baselines We compare HybridNeRF to a wide set of baselines on Mip-NeRF 360. We use the same set of baselines as in Section 4.4.2 for ScanNet++ and evaluate on the following 9 scenes: 5FB5D2DBF2, 8B5CAF3398, 39F36DA05B, 41B00FEDDB, 56A0EC536C, 98B4EC142F, B20A261FDF, F8F12E4E6B, FE1733741F. We undistort the fisheye DSLR captures to pinhole images using the official dataset toolkit [214] to facilitate comparisons against 3D Gaussian splatting [82] (whose implementation does not support fisheye projection). We use the official validation splits, which consist of

Table 4.2: **MipNeRF 360 [17].** Real-time methods are highlighted (**best**, second-best, *third-best*). Baseline numbers as published [28, 140, 209]. MobileNeRF [28] was not evaluated on indoor scenes. Our method performs similar to state-of-the-art real-time and offline methods.

	Outdoor			Indoor			Overall		
	↑PSNR	↑SSIM	↓LPIPS	↑PSNR	↑SSIM	↓LPIPS	↑PSNR	↑SSIM	↓LPIPS
NeRF [118]	21.46	0.458	0.515	26.84	0.790	0.370	23.85	0.606	0.451
NeRF++ [223]	22.76	0.548	0.427	28.05	0.836	0.309	25.11	0.676	0.375
SVS [142]	23.01	0.662	0.253	28.22	0.907	0.160	25.33	0.771	0.212
Mip-NeRF 360 [17]	*24.47*	0.691	0.283	31.72	0.917	0.180	27.69	0.791	0.237
iNGP [120]	22.90	0.566	0.371	29.15	0.880	0.216	25.68	0.706	0.302
Zip-NeRF [18]	**25.46**	**0.747**	**0.170**	**32.29**	**0.931**	*0.106*	**28.49**	**0.829**	**0.142**
Deep Blending [67]	21.54	0.524	0.364	26.40	0.844	0.261	23.70	0.666	0.318
MobileNeRF [28]	21.95	0.470	0.470	—	—	—	—	—	—
BakedSDF [209]	22.47	0.585	0.349	29.15	0.880	0.216	25.68	0.706	0.302
MERF [140]	23.19	0.616	0.343	27.80	0.855	0.271	25.24	0.722	0.311
3D Gaussian splatting [81]	24.13	*0.707*	0.211	30.94	0.927	**0.081**	27.16	*0.805*	0.153
HybridNeRF	24.73	0.716	*0.224*	*31.01*	*0.920*	0.095	*27.52*	0.806	*0.167*

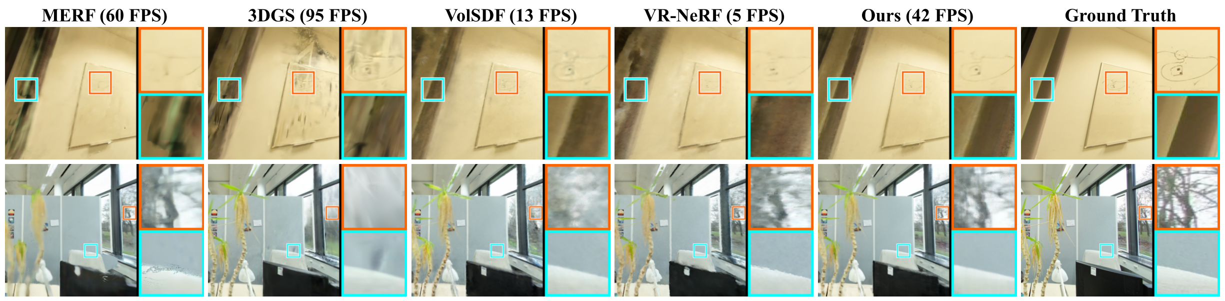


Figure 4.8: **ScanNet++ [213].** 3D Gaussian splatting [81] struggles with specular surfaces such as whiteboards (**above**) and far-field content (**below**). Our method performs best qualitatively while maintaining a real-time frame rate.

entirely novel trajectories that present a more challenging novel-view synthesis problem than the commonly used pattern of holding out every eighth frame [117].

Results We list results in Table 4.2 and Table 4.3. Our method performs comparably to the best on Mip-NeRF 360 across both real-time [81] and offline [17] methods. Although ScanNet++ [213] contains fewer lighting inconsistencies across training images than the Eyeful Tower dataset [201], 3D Gaussian splatting still struggles to reconstruct specular surfaces (whiteboards, reflective walls) and backgrounds (Table 4.3). Our method performs the best amongst real-time methods and comparably to Zip-NeRF [18], while rendering $>400\times$ faster.

4.4.4 Diagnostics

Ablations We ablate our design decisions by individually omitting the major components of our method, most notably: our distance-adjusted Eikonal loss, our adaptive surfaceness $\beta(\boldsymbol{x})$,

Table 4.3: **ScanNet++ [213] results.** Similar to Table 4.1, our method is the only to hit VR FPS rates along with 3DGS and MERF. Our quality is near-identical to Zip-NeRF while rendering $>400\times$ faster.

Method	↑PSNR	↑SSIM	↓LPIPS	↑FPS
iNGP* [120]	23.69	0.815	0.308	5.39
VolSDF* [208]	24.26	0.834	0.246	13.18
MERF (pre-baking) [140]	23.44	0.821	0.306	12.08
MERF (baked) [140]	23.19	0.820	0.308	<u>60.21</u>
3D Gaussian splatting [81]	23.76	0.830	0.248	**94.95**
VR-NeRF [201]	24.00	0.814	0.301	5.38
Zip-NeRF [18]	**24.79**	**0.863**	**0.216**	<0.1
HybridNeRF	<u>24.64</u>	<u>0.835</u>	<u>0.236</u>	41.90

* Our implementation. VolSDF: with iNGP acceleration.

MLP distillation, and hardware-accelerated textures (vs. iNGP [120] hash tables commonly used by other fast NeRF methods). We also measure the effect of using only 3D or triplane features (instead of using both).

Results We present results against the Eyeful Tower [201] in Table 5.7 and Table 4.5. Spatially adaptive surfaceness is crucial as using a global parameter degrades either speed (when β is optimized for quality) or rendering quality (when set for speed). Applying uniform Eikonal loss instead of our distance-adjusted variant degrades quality in unbounded scenes. Omitting the distillation process has a minor impact on quality relative to rendering speed. We note a similar finding when using iNGP [120] primitives instead of CUDA textures, which suggests that introducing hardware acceleration into these widely used primitives is a potential avenue for future research. Similar to MERF [140], we also note that using both low-resolution 3D features and high-resolution triplane improves rendering quality.

Geometric Reconstruction We evaluate geometric reconstruction on ScanNet++ [213] (which has "ground-truth" laser scan depth only for foreground pixels) in Table 4.6 for the strategies in Fig. 4.6. Using uniform Eikonal loss in contracted space degrades accuracy (0.419 m error vs 0.219 m for uniform world space and 0.221 m with our distance-adjusted method) and omitting Eikonal loss gives the worst results (0.996 m).

Color Distillation We distill the MLP used to represent distance from our 256-wide MLP to a 16-wide network during the finetuning stage (Section 4.3.2). As a final diagnostic, we investigate whether it is possible to further accelerate rendering by similarly distilling the color MLP. We found this to provide a significant boost in rendering speed (from 46 to 60 FPS) at the cost of a minor but statistically significant decrease in rendering quality (see Table 4.7). We observed

Table 4.4: **Diagnostics.** A global learned β ($\approx$ 200) produces the highest-quality renderings, but is slow to render as much of the scene is modeled volumetrically. Increasing β improves rendering speed but results in worse accuracy. Our full method (with spatially-varying $\beta(\boldsymbol{x})$) gets the best of both worlds. Other innovations such as distance-adjusted Eikonal loss are crucial for ensuring high accuracy for scenes with complex backgrounds. Finally, distillation and hardware acceleration come at a minor quality cost while doubling rendering speed.

Methods	$\beta(\boldsymbol{x})$	Dist.	Distill	Textures	↑PSNR	↑SSIM	↓LPIPS	↑FPS
w/ Global β (learned)	✗	✓	✓	✓	**31.76**	**0.923**	**0.188**	28.79
w/ Global β = 2000	✗	✓	✓	✓	27.16	0.835	0.345	**47.47**
w/o distance-adjusted Eik.	✓	✗	✓	✓	29.97	0.856	0.260	45.42
w/o MLP Distillation	✓	✓	✗	✓	<u>31.65</u>	0.915	<u>0.193</u>	35.25
w/o CUDA Textures	✓	✓	✓	✗	31.62	<u>0.921</u>	0.195	28.48
Full Method	✓	✓	✓	✓	31.57	0.913	0.198	<u>45.78</u>

Table 4.5: **Grid feature layout.** We measure the effect of using only 3D or triplane features on the Eyeful Tower dataset [201], and note a significant drop in quality when compared to using both.

	Pinhole			Fisheye			Overall		
	↑PSNR	↑SSIM	↓LPIPS	↑PSNR	↑SSIM	↓LPIPS	↑PSNR	↑SSIM	↓LPIPS
3D Only	27.10	0.832	0.410	32.17	0.928	0.187	29.41	0.875	0.308
Triplane Only	<u>28.24</u>	<u>0.843</u>	<u>0.312</u>	<u>33.16</u>	<u>0.938</u>	<u>0.150</u>	<u>30.47</u>	<u>0.886</u>	<u>0.238</u>
Both	**29.07**	**0.880**	**0.268**	**34.57**	**0.952**	**0.115**	**31.57**	**0.913**	**0.198**

Table 4.6: **Depth error on ScanNet++ [213].** Our distance-adjusted Eikonal loss degrades geometric reconstruction less than other alternatives used to render unbounded scenes.

Method	↓Distance (m)	↓Distance (%)
Uniform Eikonal loss (world space)	**0.219**	**8.56**
Uniform Eikonal loss (contracted space)	0.419	16.11
No Eikonal loss	0.996	29.93
Distance-adjusted Eikonal loss (ours)	<u>0.221</u>	<u>11.13</u>

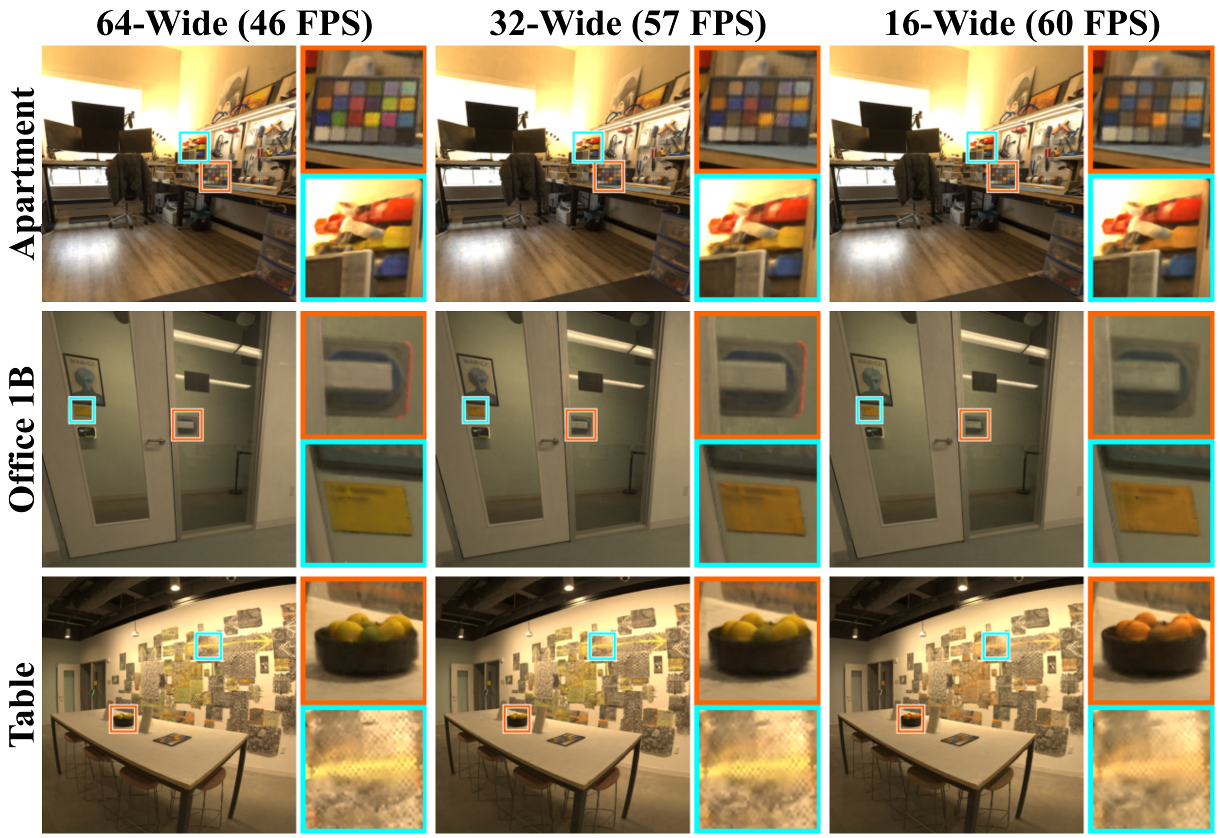


Figure 4.9: **Color Distillation.** Distilling the color MLP to a smaller width during the finetuning stage (Section 4.3.2) accelerates rendering at the cost of a minor decrease in quality. We observe largely similar results when decreasing the width to 32 channels, and more noticeable changes in color when further decreasing to 16.

Table 4.7: **Color distillation.** We evaluate the effect of color MLP distillation on the Eyeful Tower dataset [201], and find a significant increase in rendering speed at the cost of quality.

Color Width	↑PSNR	↑SSIM	↓LPIPS	↑FPS
16-wide (distilled)	30.88	0.888	0.236	**60.13**
32-wide (distilled)	<u>31.17</u>	<u>0.900</u>	<u>0.220</u>	<u>57.05</u>
64-wide (original)	**31.57**	**0.913**	**0.198**	45.78

qualitatively similar results when decreasing width from 64 to 32 channels with more notable changes in color when decreasing the width to 16 channels (see Fig. 4.9). As our initial results suggest that MLP evaluation remains a significant rendering bottleneck, replacing our scene-wide color MLP with a collection of smaller, location-specific MLPs, as suggested by KiloNeRF [139] and SMERF [42], is potential future work that could boost rendering speed at a smaller cost in quality.

4.5 Discussion

We present a hybrid surface–volume representation that combines the best of surface and volume-based rendering into a single model. We achieve state-of-the-art quality across several datasets while maintaining real-time frame rates at VR resolutions. Although we push the performance frontier of raymarching approaches, a significant speed gap remains next to splatting-based approaches [81]. Combining the advantages of our surface–volume representation with these methods would be a valuable next step.

4.5.1 Limitations

Memory Storing features in dense 3D grids and triplanes consumes significantly more memory than with hash tables [120]. Training is especially memory-intensive as intermediate activations must be stored for backpropagation along with per-parameter optimizer statistics. Storing features in a hash table during the training phase before "baking" them into explicit textures as in MERF [140] would ameliorate training-time consumption but not at inference time.

Training time. Although our training time is much faster than the original NeRF, it is about 2× slower than iNGP due to the additional backprogation needed for Eikonal regularization (in line with other "fast" surface approaches such as NeuS-facto [218]), and slower than 3D Gaussian splatting.

Chapter 5

Fast Anti-Aliasing for Neural Radiance Fields

The contents of this chapter were published as "PyNeRF: Pyramidal Neural Radiance Fields" in NeurIPS 2023

5.1 Introduction

After first discussing how to scale neural representations in Chapters 2 and 3, and how to accelerate rendering in Chapter 4, we now turn our attention to improving rendering quality. We first explore how to alleviate aliasing artifacts when training and rendering with freeform camera trajectories. As our goal is to efficiently render large-scale virtual worlds, we explore solutions that are compatible with the scaling and speed improvements described in previous chapters.

Although NeRF can provide high rendering quality, most NeRF methods assume that training and test-time cameras capture scene content from a roughly constant distance. Rendering quality degrades due to aliasing and excessive blurring when that assumption is violated. This is because NeRF raycasting is scale-unaware - it samples points along an infinitesimally thin ray and does not consider the area viewed by each pixel. Mip-NeRF [16], addresses the issue by projecting camera frustum volumes instead of point-sampling rays. However, it relies on the MLP representation used in the original NeRF and is thus slow and incompatible with accelerated grid-based NeRF implementations [26, 50, 120, 150], including those used by SUDS (Chapter 3) and HybridNeRF (Chapter 4). Zip-NeRF [18] instead uses a multisampling strategy to improve aliasing, and is compatible with grid methods, but training and rendering speed decreases with the number of additional samples.

Inspired by divide-and-conquer NeRF extensions [138, 139, 167, 178] and classical approaches such as Gaussian pyramids [12] and mipmaps [192], we propose a simple approach (PyNeRF [180]) that can easily be applied to any existing accelerated NeRF implementation. We train a pyramid of models at different scales, sample along camera rays (as in the original NeRF), and simply query coarser levels of the pyramid for samples that cover larger volumes (similar to voxel cone tracing [33]). Our method is simple to implement and significantly improves the rendering quality of fast rendering approaches with minimal performance overhead.

Contribution: Our primary contribution is a partitioning method that can be easily adapted

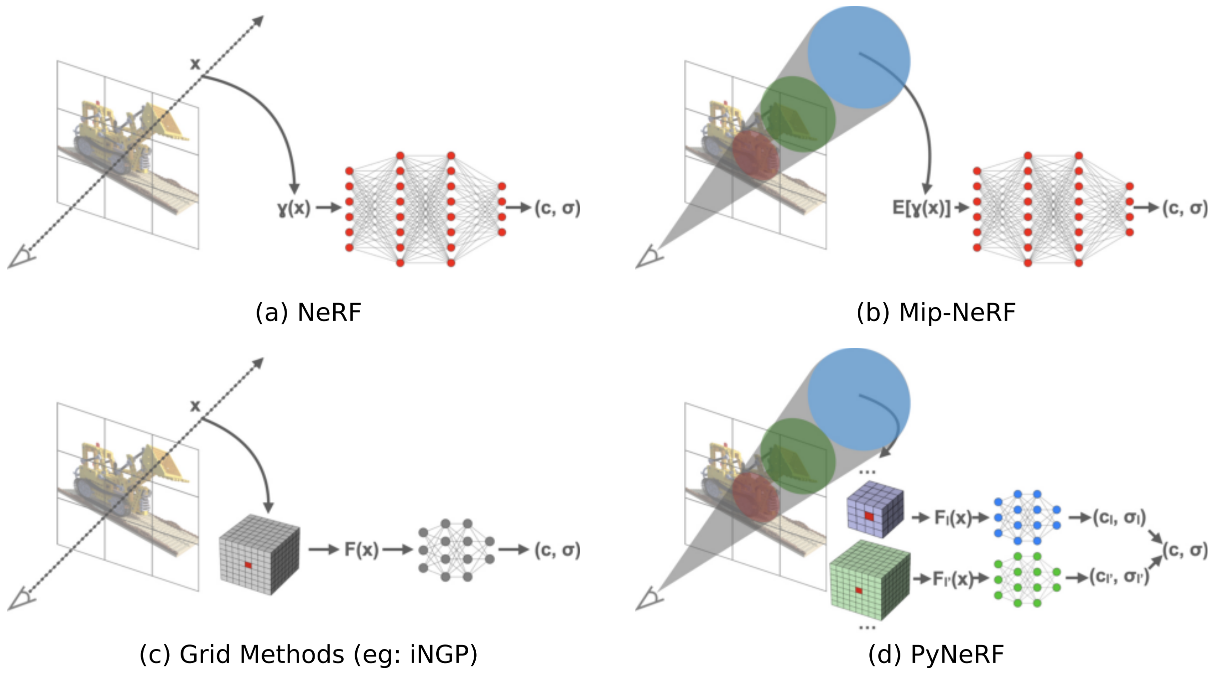


Figure 5.1: **Comparison of methods.** **(a)** NeRF traces a ray from the camera's center of projection through each pixel and samples points $\mathbf{x}$ along each ray. Sample locations are then encoded with a positional encoding to produce a feature $\gamma(\mathbf{x})$ that is fed into an MLP. **(b)** Mip-NeRF instead reasons about *volumes* by defining a 3D conical frustum per camera pixel. It splits the frustum into sampled volumes, approximates them as multivariate Gaussians, and computes the integral of the positional encodings of the coordinates contained within the Gaussians. Similar to NeRF, these features are then fed into an MLP. **(c)** Accelerated grid methods, such as iNGP, sample points as in NeRF, but do not use positional encoding and instead featurize each point by interpolating between vertices in a feature grid. These features are then passed into a much smaller MLP, which greatly accelerates training and rendering. **(d)** PyNeRF [180] also uses feature grids, but reasons about volumes by training separate models at different scales (similar to a mipmap). It calculates the area covered by each sample in world coordinates, queries the models at the closest corresponding resolutions, and interpolates their outputs.

to any existing grid-rendering approach. We present state-of-the-art reconstruction results against a wide range of datasets, including on novel scenes we designed that explicitly target common aliasing patterns. We evaluate different posssible architectures and demonstrate that our design choices provide a high level of visual fidelity while maintaining the rendering speed of fast NeRF approaches.

5.2 Related Work

The now-seminal Neural Radiance Fields (NeRF) paper [118] inspired a vast corpus of follow-up work. We discuss a non-exhaustive list of such approaches along axes relevant to our work.

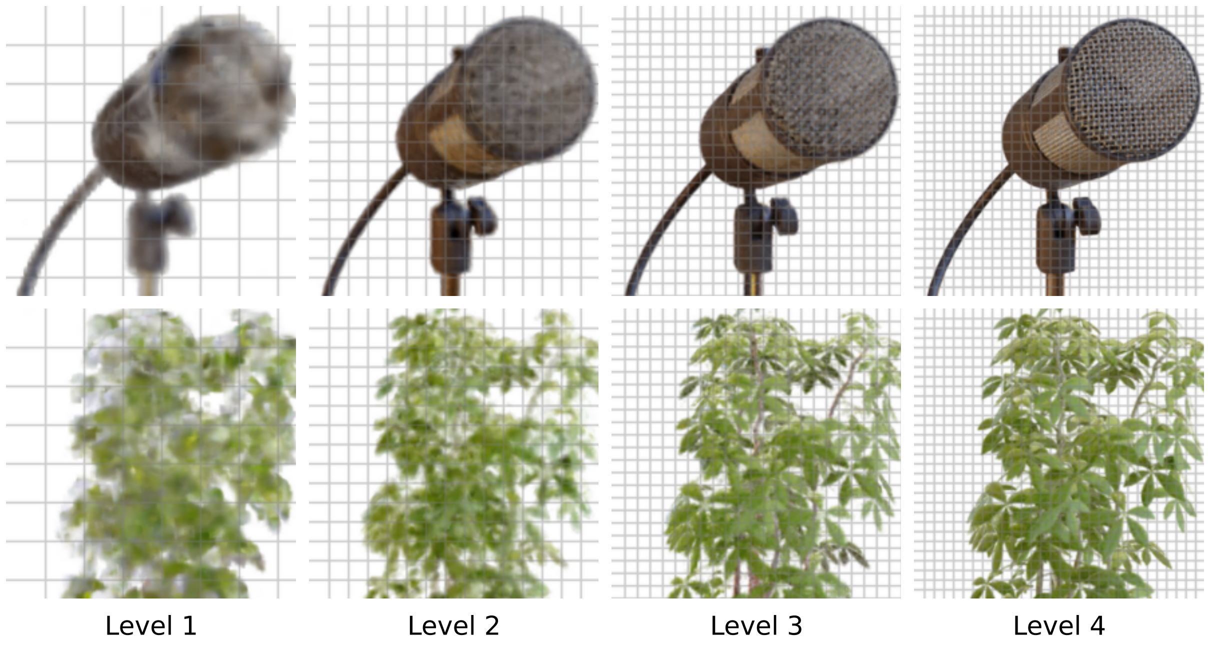


Figure 5.2: We visualize renderings from a pyramid of spatial grid-based NeRFs trained for different voxel resolutions. Models at finer pyramid levels tend to capture finer content.

Grid-based methods. The original NeRF took 1–2 days to train, with extensions for unbounded scenes [17, 223] taking longer. Once trained, rendering takes seconds per frame and is far below interactive thresholds. NSVF [106] combines NeRF's implicit representation with a voxel octree that allows for empty-space skipping and improves inference speeds by 10×. Follow-up works [58, 68, 216] further improve rendering to interactive speeds by storing precomputed model outputs into auxiliary grid structures that bypass the need to query the original model altogether at render time. Plenoxels [150] and DVGO [165] accelerate both training and rendering by directly optimizing a voxel grid instead of an MLP to train in minutes or even seconds. TensoRF [26] and K-Planes [50] instead use the product of low-rank tensors to approximate the voxel grid and reduce memory usage, while Instant-NGP [120] (iNGP) encodes features into a multi-resolution hash table. The main goal of our work is to combine the speed benefits of grid-based methods with an approach that maintains quality across different rendering scales.

Divide-and-conquer. Several works note the diminishing returns in using large networks to represent scene content, and instead render the area of interest with multiple smaller models. DeRF [138] and KiloNeRF [139] focus on inference speed while Mega-NeRF (Chapter 2), Block-NeRF [167], and SUDS (Chapter 3) use scene decomposition to efficiently train city-scale neural representations. Our method is similar in philosophy, although we partition across different resolutions instead of geographical area.

Aliasing. The original NeRF assumes that scene content is captured at roughly equidistant camera distances and emits blurry renderings when the assumption is violated. Mip-NeRF [16] reasons about the volume covered by each camera ray and proposes an integrated positional encoding that alleviates aliasing. Mip-NeRF 360 [17] extends the base method to unbounded scenes. Exact-NeRF [75] derives a more precise integration formula that better reconstructs far-away scene content. Bungee-NeRF [197] leverages Mip-NeRF and further adopts a coarse-to-fine training approach with residual blocks to train on large-scale scenes with viewpoint variation. LIRF [199] proposes a multiscale image-based representation that can generalize across scenes. The methods all build upon the original NeRF MLP model and do not readily translate to accelerated grid-based methods.

Concurrent work. Several contemporary efforts explore the intersection of anti-aliasing and fast rendering. Zip-NeRF [18] combines a hash table representation with a multi-sampling method that approximates the true integral of features contained within each camera ray's view frustum. Although it trains faster than Mip-NeRF, it is explicitly not designed for fast rendering as the multi-sampling adds significant overhead. Mip-VoG [72] downsamples and blurs a voxel grid according to the volume of each sample in world coordinates. We compare their reported numbers to ours in Section 5.4.2. Tri-MipRF [74] uses a similar prefiltering approach, but with triplanes instead of a 3D voxel grid.

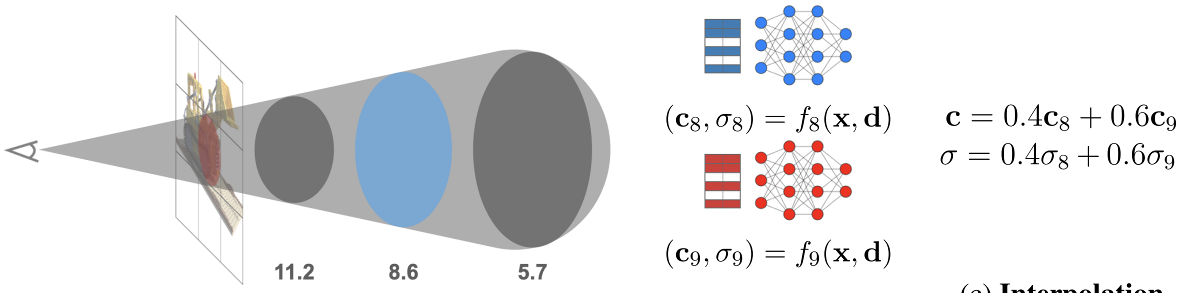
Classical methods. Similar to PyNeRF, classic image processing methods, such as Gaussian [12] and Laplacian [22] hierarchy, maintain a coarse-to-fine pyramid of different images at different resolutions. Compared to Mip-NeRF, which attempts to learn a single MLP model across all scales, one could argue that our work demonstrates that the classic pyramid approach can be efficiently adapted to neural volumetric models. In addition, our ray sampling method is similar to Crassin et al.'s approach [33], which approximates cone tracing by sampling along camera rays and querying different mipmap levels according the spatial footprint of each sample (stored as a voxel octree in their approach and as a NeRF model in ours).

5.3 Approach

5.3.1 Preliminaries

NeRF. NeRF [118] represents a scene within a continuous volumetric radiance field that captures geometry and view-dependent appearance. It encodes the scene within the weights of a multilayer perceptron (MLP). At render time, NeRF casts a camera ray $\mathbf{r}$ for each image pixel. NeRF samples multiple positions $\mathbf{x}_i$ along each ray and queries the MLP at each position (along with the ray viewing direction $\mathbf{d}$) to obtain density and color values σ_i and $\mathbf{c}_i$. To better capture high-frequency details, NeRF maps $\mathbf{x}_i$ and $\mathbf{d}$ through an L-dimensional positional encoding (PE) $\gamma(x) = [\sin(2^0\pi x), \cos(2^0\pi x), \ldots, \sin(2^L\pi x), \cos(2^L\pi x)]$ instead of directly using them as MLP inputs. It then composites a single color prediction $\hat{C}(\mathbf{r})$ per ray using numerical quadrature $\sum_{i=0}^{N-1} T_i(1-\exp(-\sigma_i\delta_i))\, c_i$, where $T_i = \exp(-\sum_{j=0}^{i-1}\sigma_j\delta_j)$ and δ_i is the distance between samples. The training process optimizes the model by sampling batches $\mathcal{R}$ of image pixels and minimizing the loss $\sum_{\mathbf{r}\in\mathcal{R}} \left\|C(\mathbf{r}) - \hat{C}(\mathbf{r})\right\|^2$. We refer the reader to Mildenhall et al. [118] for details.

Anti-aliasing. The original NeRF suffers from aliasing artifacts when reconstructing scene content observed at different distances or resolutions due to its reliance on point-sampled features. As these features ignore the volume viewed by each ray, different cameras viewing the same position from different distances may produce the same ambiguous feature. Mip-NeRF [16]


11.2
8.6
5.7
$(\mathbf{c}_8, \sigma_8) = f_8(\mathbf{x}, \mathbf{d})$
$(\mathbf{c}_9, \sigma_9) = f_9(\mathbf{x}, \mathbf{d})$
$\mathbf{c} = 0.4\mathbf{c}_8 + 0.6\mathbf{c}_9$
$\sigma = 0.4\sigma_8 + 0.6\sigma_9$

(a) **Point Sampling** (b) **Model Evaluation** (c) **Interpolation** bhaalbhansbb

Figure 5.3: **Overview. (a)** We sample frustums along the camera ray corresponding to each pixel and derive the scale of each sample according to its width in world coordinates. (**b**) We query the model heads closest to the scale of each sample. (**c**) We derive a single color and density value for each sample by interpolating between model outputs according to scale.

and variants instead reason about *volumes* by defining a 3D conical frustum per camera pixel. It featurizes intervals within the frustum with a integrated positional encoding (IPE) that approximates each frustum as a multivariate Gaussian to estimate the integral $\mathbb{E}[\gamma(x)]$ over the PEs of the coordinates within it.

Grid-based acceleration. Various methods [26, 50, 120, 150, 165] eschew NeRF's positional encoding and instead store learned features into a grid-based structure, e.g. implemented as an explicit voxel grid, hash table, or a collection of low-rank tensors. The features are interpolated based on the position of each sample and then passed into a hard-coded function or much smaller MLP to produce density and color, thereby accelerating training and rendering by orders of magnitude. However, these approaches all use the same volume-insensitive point sampling of the original NeRF and do not have a straightforward analogy to Mip-NeRF's IPE as they no longer use positional encoding.

5.3.2 Multiscale sampling

Assume that each sample $\mathbf{x}$ (where we drop the i index to reduce notational clutter) is associated with an integration volume. Intuitively, samples close to a camera correspond to small volumes, while samples far away from a camera correspond to large volumes (Fig. 5.3). Our crucial insight for enabling multiscale sampling with grid-based approaches is remarkably simple: *we train separate NeRFs at different voxel resolutions and simply use coarser NeRFs for samples covering larger volumes*. Specifically, we define a hierarchy of L resolutions that divide the world into voxels of length $1/N_0, ..., 1/N_{L-1}$, where $N_{l+1} = sN_l$ and s is a constant scaling factor. We also define a function $f_l(\mathbf{x}, \mathbf{d})$ at each level that maps from sample location $\mathbf{x}$ and viewing direction $\mathbf{d}$ to color $\mathbf{c}$ and density σ. f_l can be implemented by any grid-based NeRF; in our experiments, we use a hash table followed by small density and color MLPs, similar to iNGP. We further define a mapping function M that assigns the integration volume of sample $\mathbf{x}$ to the hierarchy level l. We explore different alternatives, but find that selecting the level whose

Algorithm 1 PyNeRF rendering function

```
Input: $m$ rays $\mathbf{r}$, $L$ pyramid levels, hierarchy mapping function $M$, base resolution $N_0$, scaling
  factor $s$
Output: $m$ estimated colors $\mathbf{c}$
  $\mathbf{x}, \mathbf{d}, P(\mathbf{x}) \leftarrow sample(\mathbf{r})$    ▷ Sample points $\mathbf{x}$ along each ray with direction $\mathbf{d}$ and area $P(\mathbf{x})$
  $M(P(\mathbf{x})) \leftarrow \log_s(P(\mathbf{x})/N_0)$    ▷ Eq. (5.1)
  $l \leftarrow \min(L-1, \max(0, \lceil M(P(\mathbf{x}))\rceil))$    ▷ Eq. (5.2)
  $w \leftarrow l - M(P(\mathbf{x}))$    ▷ Eq. (5.5)
  $model\_out \leftarrow zeros(len(\mathbf{x}))$    ▷ Zero-initialize model outputs for each sample $\mathbf{x}$
  for $i$ in unique($l$) do    ▷ Iterate over sample levels
    $model\_out[l=i]$ += $w[l=i] f_i(\mathbf{x}[l=i], \mathbf{d}[l=i])$
    $model\_out[l=i]$ += $(1-w)[l=i] f_{i-1}(\mathbf{x}[l=i], \mathbf{d}[l=i])$
  end for
  $\mathbf{c} \leftarrow composite(model\_out)$    ▷ Composite model outputs into per-ray color $\mathbf{c}$
  return $\mathbf{c}$
```

voxels project to the 2D pixel area $P(\mathbf{x})$ used to define the integration volume works well:

$$M(P(\mathbf{x})) = \log_s(P(\mathbf{x})/N_0) \tag{5.1}$$

$$l = \min(L-1, \max(0, \lceil M(P(\mathbf{x}))\rceil)) \tag{5.2}$$

$$\sigma, \mathbf{c} = f_l(\mathbf{x}, \mathbf{d}), \qquad \textbf{[GaussPyNeRF]} \tag{5.3}$$

where $\lceil \cdot \rceil$ is the ceiling function. Such a model can be seen as a (Gaussian) pyramid of spatial grid-based NeRFs (Fig. 5.2). If the final density and color were obtained by *summing* across different pyramid levels, the resulting levels would learn to specialize to residual or "band-pass" frequencies (as in a 3D Laplacian pyramid [22]):

$$\sigma, \mathbf{c} = \sum_{i=0}^{l} f_i(\mathbf{x}, \mathbf{d}). \qquad \textbf{[LaplacianPyNeRF]} \tag{5.4}$$

Our experiments show that such a representation is performant, but expensive since it requires l model evaluations per sample. Instead, we find a good tradeoff is to linearly interpolate between two model evaluations at the levels just larger than and smaller than the target integration volume:

$$\sigma, \mathbf{c} = w f_l(\mathbf{x}, \mathbf{d}) + (1-w) f_{l-1}(\mathbf{x}, \mathbf{d}), \quad \text{where} \quad w = l - M(P(\mathbf{x})). \quad \textbf{(Default) [PyNeRF]} \tag{5.5}$$

This adds the cost of only a *single* additional evaluation (increasing the overall rendering time from 0.0045 to 0.005 ms per pixel) while maintaining rendering quality (see Section 5.4.6). Our algorithm is summarized in Algorithm 1.

Matching areas vs volumes. One might suspect it may be better to select the voxel level l whose volume best matches the sample's 3D integration volume. We experimented with this, but found it more effective to match the projected 2D pixel area rather than volumes. Note that both approaches would produce identical results if the 3D volume was always a cube, but volumes may be elongated along the ray depending on the sampling pattern. Matching areas is preferable because most visible 3D scenes consist of empty space and surfaces, implying that when computing the composite color for a ray r, most of the contribution will come from a few samples $\mathbf{x}$ lying near the surface of intersection. When considering the target 3D integration volume associated with $\mathbf{x}$, most of the contribution to the final composite color will come from integrating along the 2D surface (since the rest of the 3D volume is either empty or hidden). This loosely suggests we should select levels of the voxel hierarchy based on (projected) area rather than volume.

Hierarchical grid structures. Our method can be applied to any accelerated grid method irrespective of the underyling storage. However, a drawback of this approach is an increased on-disk serialization footprint due to training a hierarchy of spatial grid NeRFs. A possible solution is to exploit hierarchical grid structures that already exist *within* the base NeRF. Note that multi-resolution grids such as those used by iNGP [120] or K-Planes [50] already define a scale hierarchy that is a natural fit for PyNeRF. Rather than learning a separate feature grid for each model in our pyramid, we can reuse the same multi-resolution features across levels (while still training different MLP heads).

Multi-resolution pixel input. One added benefit of the above is that one can train with multiscale training data, which is particularly helpful for learning large, city-scale NeRFs [167, 178, 179, 197, 202]. For such scenarios, even storing high-resolution pixel imagery may be cumbersome. In our formulation, one can store low-resolution images and quickly train a coarse scene representation. The benefits are multiple. Firstly, divide-and-conquer approaches such as Mega-NeRF (Chapter 3) partition large scenes into smaller cells and train using different training pixel/ray subsets for each (to avoid training on irrelevant data). However, in the absence of depth sensors or a priori 3D scene knowledge, Mega-NeRF is limited in its ability to prune irrelevant pixels/rays (due to intervening occluders) which empirically bloat the size of each training partition by 2× (Chapter 2). With our approach, we can learn a coarse 3D knowledge of the scene on downsampled images and then filter higher-resolution data partitions more efficiently. Once trained, lower-resolution levels can also serve as an efficient initialization for finer layers. In addition, many contemporary NeRF methods use occupancy grids [120] or proposal networks [17] to generate refined samples near surfaces. We can quickly train these along with our initial low-resolution model and then use them to train higher-resolution levels in a sample-efficient manner. We show in our experiments that such course-to-fine multiscale training can speed up convergence (Section 5.4.5).

Unsupervised levels. A naive implementation of our method will degrade when zooming in and out of areas that have not been seen at training time. Our implementation mitigates this by maintaining an auxiliary data structure (similar to an occupancy grid [120]) that tracks the coarsest and finest levels queried in each region during training. We then use the structure at inference time to only query levels that were supervised during training.

Table 5.1: **Synthetic results.** PyNeRF outperforms all baselines and trains over 60× faster than Mip-NeRF. Both PyNeRF and Mip-NeRF properly reconstruct the brick wall in the Blender-A dataset, but Mip-NeRF fails to accurately reconstruct checkerboard patterns.

	Multiscale Blender [16]					Blender-A				
	$\uparrow$PSNR	$\uparrow$SSIM	$\downarrow$LPIPS	$\downarrow$Avg Error	$\downarrow$Train Time (h)	$\uparrow$PSNR	$\uparrow$SSIM	$\downarrow$LPIPS	$\downarrow$Avg Error	$\downarrow$ Train Time (h)
Plenoxels [150]	24.98	0.843	0.161	0.080	0:28	18.13	0.511	0.523	0.190	**0:40**
K-Planes [50]	29.88	0.946	0.058	0.022	0:32	21.17	0.593	0.641	0.405	1:22
TensoRF [26]	30.04	0.948	0.056	0.021	0:27	27.01	0.785	0.197	0.054	1:20
iNGP [120]	30.21	0.958	0.040	0.022	**0:20**	20.85	0.767	0.244	0.089	<u>0:56</u>
Nerfacto [168]	29.56	0.947	0.051	0.022	<u>0:25</u>	27.46	0.796	0.195	<u>0.053</u>	1:07
Mip-VoG [72]	30.42	0.954	0.053	—	—	—	—	—	—	—
Mip-NeRF [16]	<u>34.50</u>	<u>0.974</u>	<u>0.017</u>	<u>0.009</u>	29:49	<u>31.33</u>	<u>0.894</u>	<u>0.098</u>	0.063	30:12
PyNeRF	**34.78**	**0.976**	**0.015**	**0.008**	<u>0:25</u>	**41.99**	**0.986**	**0.007**	**0.004**	1:10

Table 5.2: **Synthetic results across downsampling levels.** We average results across Multiscale Blender [16] and Blender-A and list metrics for each downsampling level. All PyNeRF variants outperform their baselines by a wide margin.

	$\uparrow$PSNR				$\uparrow$SSIM				$\downarrow$LPIPS				
	Full Res.	1/2 Res.	1/4 Res.	1/8 Res.	Full Res.	1/2 Res.	1/4 Res.	1/8 Res.	Full Res.	1/2 Res.	1/4 Res.	1/8 Res.	$\downarrow$Avg Error
Plenoxels [150]	22.61	23.68	24.54	23.62	0.767	0.768	0.784	0.789	0.307	0.265	0.200	0.161	0.102
K-Planes [50]	25.14	27.03	30.26	30.75	0.807	0.840	0.896	0.925	0.225	0.163	0.085	0.053	0.046
TensoRF [26]	25.93	28.12	31.46	30.97	0.865	0.893	0.921	0.930	0.169	0.112	0.064	0.056	0.042
iNGP [120]	26.90	29.14	30.89	28.49	0.865	0.905	0.947	0.947	0.152	0.095	0.047	0.054	0.032
Nerfacto [168]	25.35	27.26	29.78	29.09	0.809	0.840	0.893	0.917	0.214	0.158	0.094	0.068	0.049
Mip-NeRF [16]	32.07	33.65	34.76	35.00	0.952	0.959	0.961	0.960	0.048	0.036	0.028	0.021	0.020
SUDS	**33.18**	**35.83**	**37.59**	**38.29**	**0.964**	**0.977**	**0.984**	**0.989**	<u>0.030</u>	**0.013**	**0.007**	**0.004**	**0.008**
SUDS-K-Planes	<u>33.12</u>	35.18	36.45	36.94	<u>0.963</u>	0.973	0.980	0.985	**0.028**	<u>0.014</u>	0.009	<u>0.005</u>	**0.008**
SUDS-TensoRF	32.94	<u>35.34</u>	<u>36.92</u>	<u>37.46</u>	0.959	<u>0.974</u>	<u>0.982</u>	<u>0.987</u>	0.033	<u>0.014</u>	<u>0.008</u>	<u>0.005</u>	**0.008**

5.4 Experiments

We first evaluate PyNeRF's performance by measuring its reconstruction quality on bounded synthetic (Section 5.4.2) and unbounded real-world (Section 5.4.3) scenes. We demonstrate PyNeRF's generalizability by evaluating it on additional NeRF backbones (Section 5.4.4) and then explore the convergence benefits of using multiscale training data in city-scale reconstruction scenarios (Section 5.4.5). We ablate our design decisions in Section 5.4.6.

5.4.1 Experimental Setup

Training. We implement PyNeRF on top of the Nerfstudio library [168] and train on each scene with 8,192 rays per batch by default for 20,000 iterations on the Multiscale Blender and Mip-NeRF 360 datasets, and 50,000 iterations on the Boat dataset and Blender-A. We train a hierarchy of 8 PyNeRF levels backed by a single multi-resolution hash table similar to that used by iNGP [120] in Section 5.4.2 and Section 5.4.3 before evaluating additional backbones in Section 5.4.4. We use 4 features per level with a hash table size of 2^{20} by default, which we found to give the best quality-performance trade-off on the A100 GPUs we use in our experiments. Each PyNeRF uses a 64-channel density MLP with one hidden layer followed by a 128-channel color MLP with two hidden layers. We use similar model capacities in our baselines for fairness.

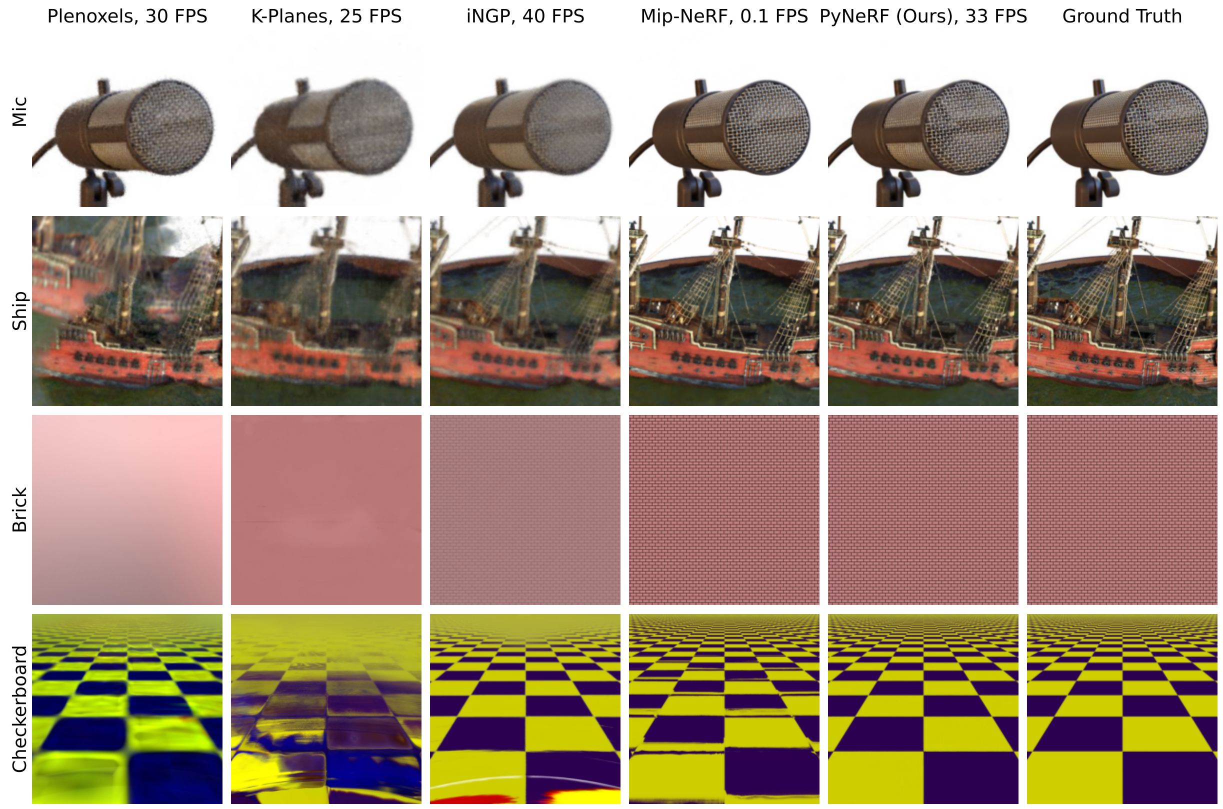


Figure 5.4: **Synthetic results.** PyNeRF and Mip-NeRF provide comparable results on the first three scenes that are crisper than those of the other fast renderers. Mip-NeRF does not accurately render the tiles in the last row while PyNeRF recreates them near-perfectly.

We sample rays using an occupancy grid [120] on the Multiscale Blender dataset, and with a proposal network [17] on all others. We use gradient scaling [**?**] to improve training stability in scenes with that capture content at close distance (Blender-A and Boat). We parameterize unbounded scenes with Mip-NeRF 360's contraction method.

Metrics. We report quantitative results based on PSNR, SSIM [187], and the AlexNet implementation of LPIPS [225], along with the training time in hours as measured on a single A100 GPU. For ease of comparison, we also report the "average" error metric proposed by Mip-NeRF [16] composed of the geometric mean of $\mathrm{MSE} = 10^{-\mathrm{PSNR}/10}$, $\sqrt{1 - \mathrm{SSIM}}$, and LPIPS.

5.4.2 Synthetic Reconstruction

Datasets. We evaluate PyNeRF on the Multiscale Blender dataset proposed by Mip-NeRF along with our own Blender scenes (which we name "Blender-A") intended to further probe the anti-aliasing ability of our approach (by reconstructing a slanted checkerboard and zooming into a brick wall).

Baselines. We compare PyNeRF to several fast-rendering approaches, namely Instant-NGP [120] and Nerfacto [168], which store features within a multi-resolution hash table, Plenoxels [150] which optimizes an explicit voxel grid, and TensoRF [26] and K-Planes [50], which rely on low-rank tensor decomposition. We also compare our Multiscale Blender results to those reported by Mip-VoG [72], a contemporary fast anti-aliasing approach, and to Mip-NeRF [16] on both datasets.

Results. We summarize our results in Table 5.1 and Table 5.2. We show qualitative examples in Fig. 5.4. PyNeRF outperforms all fast rendering approaches as well as Mip-VoG by a wide margin and is slightly better than Mip-NeRF on Multiscale Blender while training over 60× faster. Both PyNeRF and Mip-NeRF properly reconstruct the brick wall in the Blender-A dataset, but Mip-NeRF fails to accurately reconstruct checkerboard patterns.

5.4.3 Real-World Reconstruction

Datasets. We evaluate PyNeRF on the Boat scene of the ADOP [146] dataset, which to our knowledge is one of the only publicly available unbounded real-world captures that captures its primary object of interest from different camera distances. For further comparison, we construct a multiscale version of the outdoor scenes in the Mip-NeRF 360 [17] dataset using the same protocol as Multiscale Blender [16].

Baselines. We compare PyNeRF to the same fast-rendering approaches as in Section 5.4.2, along with two unbounded Mip-NeRF variants: Mip-NeRF 360 [17] and Exact-NeRF [75]. We report numbers on each variant with and without generative latent optimization [113] to account for lighting changes.

Results. We summarize our results in Table 5.3 and Table 5.4 along with qualitative results in Fig. 5.5. Once again, PyNeRF outperforms all baselines, trains 40× faster than Mip-NeRF 360, and 100× faster than Exact-NeRF (the next best alternatives).

5.4.4 Additional Backbones

Methods. We demonstrate how PyNeRF can be applied to any grid-based NeRF method by evaluating it with K-Planes [50] and TensoRF [26] in addition to our default iNGP-based implementatino. We take advantage of the inherent multi-resolution structure of iNGP and K-Planes by reusing the same feature grid across PyNeRF levels and train a separate feature grid per level in our TensoRF variant.

Results. We train the PyNeRF variants along with their backbones across the datasets described in Section 5.4.2 and Section 5.4.3, and summarize the results in Table 5.5. All PyNeRF variants show clear improvements over their base methods.

5.4.5 City-Scale Convergence

Dataset. We evaluate PyNeRF's convergence properties on the the Argoverse 2 [193] Sensor dataset (to our knowledge, the largest city-scale dataset publicly available). We select the largest overlapping subset of logs and filter out moving objects through a pretrained segmentation model [29]. The resulting training set contains 400 billion rays across 150K video frames.

Table 5.3: **Real-world results.** PyNeRF outperforms all baselines in PSNR and average error, and trains 40× faster than Mip-NeRF 360 and 100× faster than Exact-NeRF (the next best methods).

	Boat [146]					Mip-NeRF 360 [17]				
	↑PSNR	↑SSIM	↓LPIPS	↓Avg Error	↓Train Time (h)	↑PSNR	↑SSIM	↓LPIPS	↓Avg Error	↓ Train Time (h)
Plenoxels [150]	17.05	0.505	0.617	0.185	2:14	21.88	0.606	0.524	0.117	1:00
K-Planes [50]	18.00	0.501	0.590	0.168	2:41	21.53	0.577	0.500	0.120	1:08
TensoRF [26]	14.75	0.398	0.630	0.234	2:30	18.07	0.439	0.677	0.181	1:07
iNGP [120]	15.34	0.433	0.646	0.222	**1:42**	21.14	0.568	0.521	0.126	**0:40**
Nerfacto [168]	19.27	0.570	0.425	0.135	2:12	22.47	0.616	0.431	0.105	1:02
Mip-NeRF 360 w/ GLO [17]	20.03	0.595	**0.416**	0.124	37:28	22.76	**0.664**	**0.342**	0.095	37:35
Mip-NeRF 360 w/o GLO [17]	15.92	0.480	0.501	0.194	37:10	22.70	**0.664**	**0.342**	0.095	37:22
Exact-NeRF w/ GLO [75]	20.21	**0.601**	0.425	0.123	109:11	21.40	0.619	0.416	0.121	110:06
Exact-NeRF w/o GLO [75]	16.33	0.489	0.510	0.187	107:52	22.56	0.619	0.410	0.121	108:11
PyNeRF	**20.43**	**0.601**	0.422	**0.121**	2:12	**23.09**	0.654	0.358	**0.094**	1:00

Table 5.4: **Real-world results across downsampling level.** We average results across Boat [146] and Mip-NeRF 360 [17]. As in Table 5.2, all PyNeRF variants improve significantly upon their baselines.

	↑PSNR				↑SSIM				↓LPIPS				
	Full Res.	1/2 Res.	1/4 Res.	1/8 Res.	Full Res.	1/2 Res.	1/4 Res.	1/8 Res.	Full Res.	1/2 Res.	1/4 Res.	1/8 Res.	↓Avg Error
Plenoxels [150]	20.69	20.70	20.98	21.93	0.627	0.543	0.547	0.640	0.661	0.607	0.525	0.364	0.128
K-Planes [50]	20.53	20.55	20.84	21.85	0.618	0.525	0.512	0.602	0.655	0.587	0.488	0.328	0.128
TensoRF [26]	17.31	17.33	17.49	17.96	0.548	0.431	0.367	0.384	0.748	0.714	0.662	0.552	0.190
iNGP [120]	19.53	19.83	16.06	20.86	0.598	0.504	0.489	0.574	0.670	0.610	0.517	0.351	0.146
Nerfacto [168]	21.37	21.42	21.81	23.15	0.629	0.558	0.575	0.688	0.594	0.512	0.389	0.226	0.110
Mip-NeRF 360 w/ GLO [17]	21.73	21.72	22.13	23.65	**0.650**	**0.597**	**0.628**	**0.736**	**0.518**	**0.427**	**0.309**	**0.165**	0.100
Mip-NeRF 360 w/o GLO [17]	21.01	21.00	21.39	22.88	0.634	0.580	0.610	0.718	0.529	0.441	0.323	0.179	0.111
Exact-NeRF w/ GLO [75]	20.72	20.73	21.04	22.34	0.637	0.571	0.583	0.674	0.559	0.478	0.378	0.237	0.121
Exact-NeRF w/o GLO [75]	20.98	20.97	21.34	22.80	0.635	0.578	0.604	0.710	0.548	0.451	0.339	0.192	0.113
SUDS	**22.05**	**22.16**	**22.56**	**23.84**	0.645	0.591	0.620	0.725	0.535	0.441	0.316	0.184	**0.098**
SUDS-K-Planes	21.47	21.49	21.87	23.18	0.633	0.570	0.591	0.694	0.563	0.478	0.362	0.217	0.108
SUDS-TensoRF	20.82	20.89	21.25	22.48	0.594	0.521	0.528	0.630	0.648	0.558	0.438	0.284	0.122

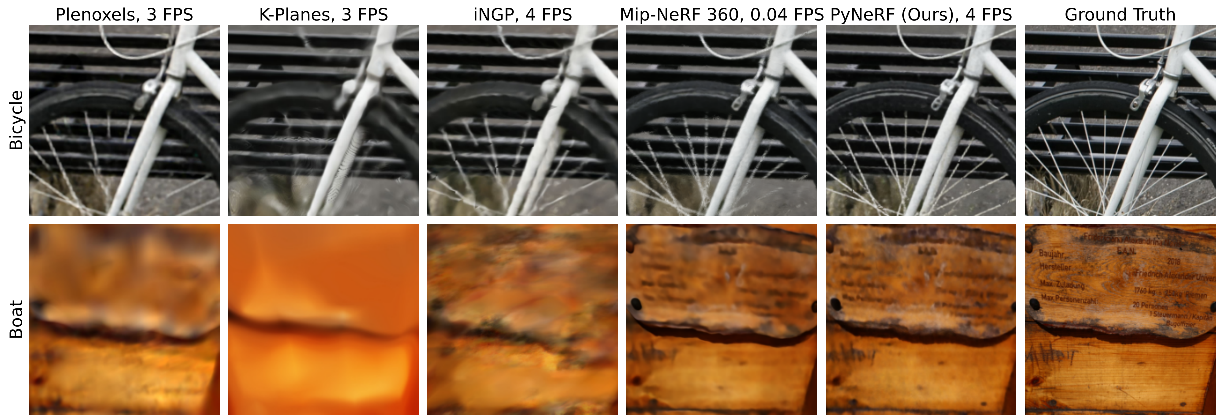


Figure 5.5: **Real-world results.** PyNeRF reconstructs higher-fidelity details (such as the spokes on the bicycle and the lettering within the boat) than other methods.

Table 5.5: **Additional backbones.** We train the PyNeRF variants along with their backbones across the datasets described in Section 5.4.2 and Section 5.4.3 All PyNeRF variants outperform their baselines by a wide margin.

	Synthetic				Real-World			
	↑PSNR	↑SSIM	↓LPIPS	↓Avg Error	↑PSNR	↑SSIM	↓LPIPS	↓Avg Error
iNGP [120]	28.86	0.916	0.087	0.032	19.94	0.541	0.537	0.146
K-Planes [50]	27.90	0.865	0.131	0.047	20.54	0.553	0.520	0.136
TensoRF [26]	29.12	0.902	0.100	0.042	17.21	0.421	0.696	0.200
PyNeRF	**36.22**	**0.979**	**0.013**	**0.004**	**22.65**	**0.645**	**0.369**	**0.098**
PyNeRF-K-Planes	35.42	0.975	<u>0.014</u>	<u>0.005</u>	<u>22.00</u>	<u>0.622</u>	<u>0.405</u>	<u>0.108</u>
PyNeRF-TensoRF	<u>35.67</u>	<u>0.976</u>	0.015	<u>0.005</u>	21.35	0.568	0.482	0.122

Methods. We use SUDS (Chapter 3) as the backbone model in our experiments. We begin training our method on 8× downsampled images (containing 64× fewer rays) for 5,000 iterations and then on progressively higher resolutions (downsampled to 4×, 2×, and 1×) every 5,000 iterations hereafter. We compare to the original SUDS method as a baseline.

Metrics. We report the evolution of the quality metrics used in Section 5.4.2 and Section 5.4.3 over the first four hours of the training process.

Results. We summarize our results in Table 5.6. PyNeRF converges more rapidly than the SUDS baseline, achieving the same rendering quality at 2 hours as SUDS after 4.

5.4.6 Diagnostics

Methods. We validate our design decisions by testing several variants. We ablate our MLP-level interpolation described in Eq. (5.5) and compare it to the GausssPyNeRF and LaplacianPyNeRF variants described in Section 5.3.2 along with another that instead interpolates the learned grid feature vectors (which avoids the need for an additional MLP evaluation per sample). As increased storage footprint is a potential drawback method, we compare our default strategy of

Table 5.6: **City-scale convergence.** We track rendering quality over the first four hours of training. PyNeRF achieves the same rendering quality as SUDS 2× faster.

↑ **PSNR**

Time (h)	1:00	2:00	3:00	4:00
SUDS (Chapter 3)	16.01	17.41	18.08	18.53
PyNeRF	**17.17**	**18.44**	**18.59**	**18.73**

↑ **SSIM**

Time (h)	1:00	2:00	3:00	4:00
SUDS (Chapter 3)	0.570	0.600	0.602	0.606
PyNeRF	**0.614**	**0.618**	**0.619**	**0.621**

↓ **LPIPS**

Time (h)	1:00	2:00	3:00	4:00
SUDS (Chapter 3)	0.531	0.496	0.470	0.466
PyNeRF	**0.521**	**0.485**	**0.469**	**0.465**

↓ **Avg Error**

Time (h)	1:00	2:00	3:00	4:00
SUDS (Chapter 3)	0.182	0.160	0.150	0.145
PyNeRF	**0.165**	**0.146**	**0.144**	**0.142**

Table 5.7: **Diagnostics.** The rendering quality of our interpolation method is near-identical to the full residual approach while training 3× faster, and is significantly better than other alternatives. Reusing the same feature grid across levels performs comparably to storing separate hash tables per level while training faster.

Method	Our Interp.	Shared Features	2D Area	↑PSNR	↑SSIM	↓LPIPS	↓ Avg Error	↓ Train Time (h)
GaussPyNeRF (Eq. 5.3)	✗	✓	✓	28.72	0.803	0.201	0.056	**0:43**
LaplacianPyNeRF (Eq. 5.4)	✗	✓	✓	**29.48**	**0.813**	**0.190**	**0.052**	2:44
Feature grid interpolation	✗	✗	✓	28.45	0.767	0.244	0.070	<u>0:46</u>
Separate hash tables	✓	✗	✓	29.41	**0.813**	0.196	0.054	0:52
Levels w/ 3D Volumes	✓	✓	✗	29.19	0.811	0.184	0.054	0:48
PyNeRF	✓	✓	✓	<u>29.44</u>	<u>0.812</u>	<u>0.191</u>	<u>0.053</u>	0:48

sharing the same multi-resolution feature grid across PyNeRF levels to the naive implementation that trains a separate grid per level. We also explore using 3D sample volumes instead of projected 2D pixel areas to determine voxel levels l.

Results. We train our method and variants as described in Section 5.4.2 and Section 5.4.3, and summarize the results (averaged across datasets) in Table 5.7. Our proposed interpolation method strikes a good balance — its performance is near-identical to the full LaplacianPyNeRF approach while training 3× faster (and is significantly better than the other interpolation methods). Our strategy of reusing the same feature grid across levels performs comparably to the naive implementation while training faster due to fewer feature grid lookups. Using 2D pixel areas instead of 3D volumes to determine voxel level l provides an improvement.

Single-scale datasets. Although PyNeRF is designed for scenarios that capture scene content at different distances, we also evaluate it on the original Synthetic-NeRF [118] dataset where the camera distance remains constant to see whether we regress quality in these settings. As shown in Table 5.8, PyNeRF performs similarly to existing SOTA.

Table 5.8: **Single-scale results.** We evaluate PyNeRF on single-scale Blender [118]. PyNeRF performs comparably to existing state-of-the-art.

PSNR	Lego	Mic	Materials	Chair	Hotdog	Ficus	Drums	Ship	Mean
K-Planes [50]	35.38	33.27	29.57	33.88	36.19	30.81	<u>25.62</u>	30.16	31.86
TensoRF [26]	35.14	25.70	**33.69**	**37.03**	36.04	29.77	24.64	30.12	31.52
iNGP [120]	<u>35.67</u>	**36.85**	29.60	35.71	<u>37.37</u>	<u>33.95</u>	25.44	30.29	<u>33.11</u>
Nerfacto [168]	34.84	33.58	26.50	34.48	37.07	30.66	23.63	**30.95**	31.46
SUDS	**36.63**	<u>36.39</u>	<u>29.92</u>	<u>35.76</u>	**37.64**	**34.29**	**25.80**	<u>30.64</u>	**33.38**

5.5 Discussion

We propose a method that significantly improves the anti-aliasing properties of fast volumetric renderers. Our approach can be easily applied to any existing grid-based NeRF, and although simple, provides state-of-the-art reconstruction results against a wide variety of datasets (while training 60–100× faster than existing anti-aliasing methods). We propose several synthetic scenes that model common aliasing patterns as few existing NeRF datasets cover these scenarios in practice. Creating and sharing additional real-world captures would likely facilitate further research.

5.5.1 Limitations

Although our method generalizes to any grid-based method (Section 5.4.4), it requires a larger on-disk serialization footprint due to training a hierarchy of spatial grid NeRFs. This can be mitigated by reusing the same feature grid when the underlying backbone uses a multi-resolution feature grid [50, 120], but this is not true of all methods [26, 150].

Chapter 6

Conclusion and Future Work

In this thesis, we advance the frontier of large-scale neural rendering along multiple dimensions. We start by addressing scalable reconstruction, first via neighborhood-scale static reconstructions, and then by designing dynamic representations of entire cities. We then introduce a method that enables high-fidelity real-time rendering at VR resolution, and begin to address the topic of quality via a fast anti-aliasing method. However, much work remains to be done. We list several directions of particular interest below.

6.1 Integrating Learned Priors

Although Chapter 5 makes a first attempt at improving quality via efficient anti-aliasing, NeRF rendering quality generally remains far below photorealistic thresholds, especially in dynamic settings. One problem is that as NeRF methods are usually optimized on a per-scene basis, they are unable to hallucinate accurate color and geometry in under-observed regions. This becomes a limiting factor when reconstructing large-scale dynamic worlds where it is impossible to densely sample every location at every time step.

A potential solution to generating plausible renderings in sparse-view situations is to introduce additional regularization terms into the optimization process. RegNeRF [124], FreeNeRF [206], and SimpleNeRF [162] add handcrafted losses that discourage degenerate solutions in few-shot settings, while DS-NeRF [37] and Dense Depth Priors [143] use depth as an additional source of information. PixelNeRF [215] adopts a more data-driven approach by aggregrating features obtained from a convolutional feature extractor trained across different datasets. While it can generate plausible geometry across different datasets, it tends to bias towards blurry renderings, especially in ambiguous regions far from source inputs.

More recent data-driven efforts [107, 151, 194, 228] leverage the popularity of diffusion models for image generation [71, 144, 163]. These approaches use score distillation sampling (SDS) [132] or similar methods that allow a 2D diffusion model to act as a critic and supervise the optimization of a 3D representation. ReconFusion [194] stands out as a current state-of-the-art method that presents results that are far visually appealing than those of prior methods that use handcrafted regularization. However, although the use of diffusion models bears considerable promise, current approaches suffer from several shortcomings. First, methods such as ReconFusion aggregate multi-view information (used to condition the diffusion model) via mechanisms that are slow and extremely memory-intensive to compute (such as PixelNeRF [215] renderings). Second, using SDS to optimize a 3D representation is relatively slow and expensive, as it requires repeatedly rendering sampled cameras across the scene that are then processed by the diffusion model. This becomes increasingly expensive as the size of the scene increases, and becomes even more intractable when optimizing dynamic 4D representations. Finally, as Poole et al note [132], score distillation sampling (and variants) can generate oversaturated and oversmoothed renderings. As SDS optimizes for modes of a distribution, it sometimes encourages monochrome or grey renderings even when the underlying generative model produces diverse images. Can we do better?

6.1.1 3D Predictions via RGB-D Diffusion Models

Most existing works rely on 2D diffusion models that are designed to generate RGB images. Leveraging these priors for 3D reconstruction therefore requires a translation step, most often via score distillation sampling. However, as the authors of MVD-Fusion [73] observe, RGB-D predictions can be leveraged as 3D outputs. Within the context of 3D scene reconstruction, an RGB-D rendering for a given posed target view can interpreted as point cloud or 3D Gaussian [81] outputs, enabling the diffusion model to directly make predictions in 3D!

How to best train a RGB-D diffusion model remains a research challenge. MVD-Fusion [73] limits its scope to synthetic object-centric renderings from Objaverse [35], limiting its applicability to real-world scenes. LDM3D [164] trains an RGB-D diffusion model on real-world data with depth maps obtained from a pre-trained monocular depth estimator [136]. However, this provides depth estimates with unknown scale and shift factors that are inconsistent across predictions, limiting its applicability to multi-view reconstruction.

Multi-view depth estimators traditionally perform poorly on data outside of their training domains, and so cannot be directly applied to arbitrary real-world scenes [156]. However, the recent DUSt3R [190] model, which can be used to infer multi-view consistent depth (amongst other applications), shows promising results across a variety of datasets. From our preliminary investigation on the DL3DV-10K dataset [105], it is potentially robust enough to enable us to train a real-world RGB-D diffusion model.

6.1.2 Proposed Approach

We train a RGB-D diffusion model conditioned via pathways similar to those proposed by ReconFusion [194]. We generate renderings for a target view by conditioning on neighboring input views. Instead of fusing input image data via a learned PixelNeRF [215] model suggested by ReconFusion, we use RGB pixels of the neighboring views and depth estimates from DUSt3R (which are consistent across the input views) to infer pixel-aligned Gaussians that we volumetrically render from the target view as in [80]. We found rendering via these off-the-shelf components to be much faster and more memory-efficient than with PixelNeRF ($>10\times$ in our experiments). We also use CLIP embeddings of the input views via cross-attention as in prior work [107, 151, 194]. Fig. 6.1 illustrates our approach.

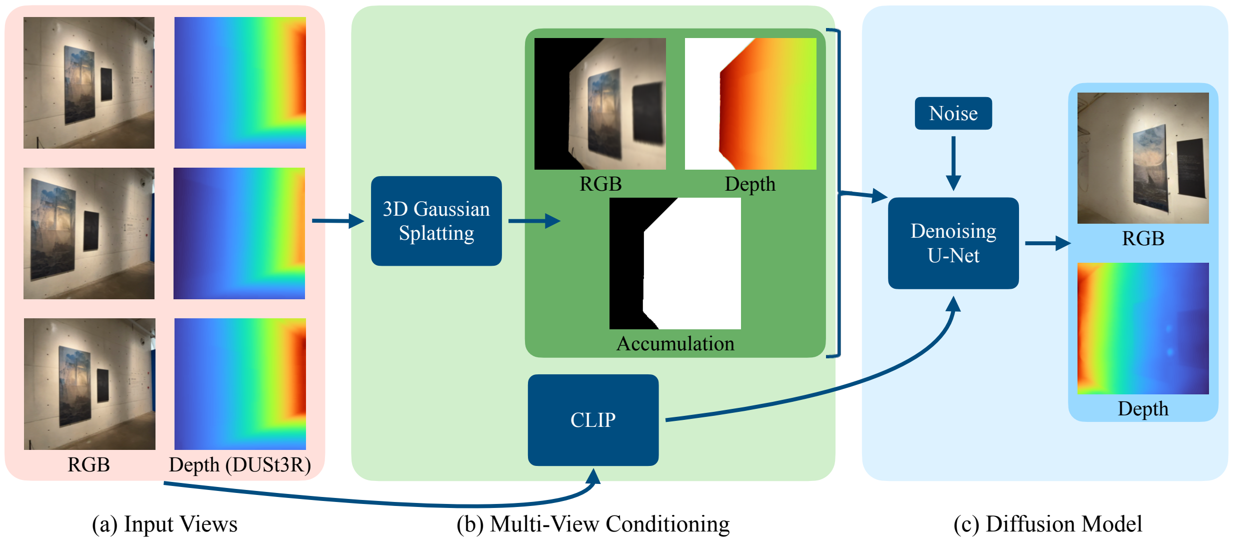


Figure 6.1: **Multiview RGB-D Diffusion.** We generate renderings for a target view by conditioning on neighboring input views **(a)**. We use the RGB pixels of the input views and depth estimates from DUSt3R [190] to infer pixel-aligned 3D gaussians [81] that we volumetrically render from the target view as in [80]. We condition the diffusion model by channel-concatenating the rendered color, depth, and accumulation, and use CLIP embeddings of the input views via cross-attention **(b)**. We generate RGB-D predictions for the target view via multi-step denoising **(c)**.

6.1.3 Challenges

Using the diffusion model. Once trained, how best to use an RGB-D diffusion model to optimize a 3D scene representation remains a challenge. One solution is to use SDS as in prior work [107, 151, 194], with the added benefit that our model can be used to supervise both color and depth renderings, which may allow for quicker convergence. However, this would have our method inherit all of the drawbacks of SDS as described at the beginning of Section 6.1.

As the outputs of our RGB-D diffusion model can be used to infer pixel-aligned 3D gaussians, it is theoretically possible to optimize a gaussian-based scene representation by directly projecting the diffusion model outputs into the scene. Once projected as 3D gaussians, these outputs can be combined with the original set of gaussians and used to condition subsequent model evaluations, allowing us to complete the scene in a feed-forward autoregressive manner. The main outstanding challenge is how to best ensure that model outputs are consistent with the rest of the scene (which the conditioning in Fig. 6.1 encourages but does not guarantee).

Depth estimation. The quality of our proposed method is heavily dependent on accurate depth estimates. Although our preliminary experiments suggest that DUSt3R exhibits stronger generalization than prior methods, its predictions remain imperfect. The original DUSt3R paper derives multi-view consistent depth by minimizing the reprojection error of pairwise pointmaps via backpropagation. We discuss how to infer pixel-aligned 3D gaussians that can be volumetrically rendered from these depth predictions in Section 6.1.2 - the photometric loss of these renderings could be used to evaluate and further optimize the accuracy of the depth estimates. Per-scene volumetric optimization has traditionally been regarded as too expensive to perform

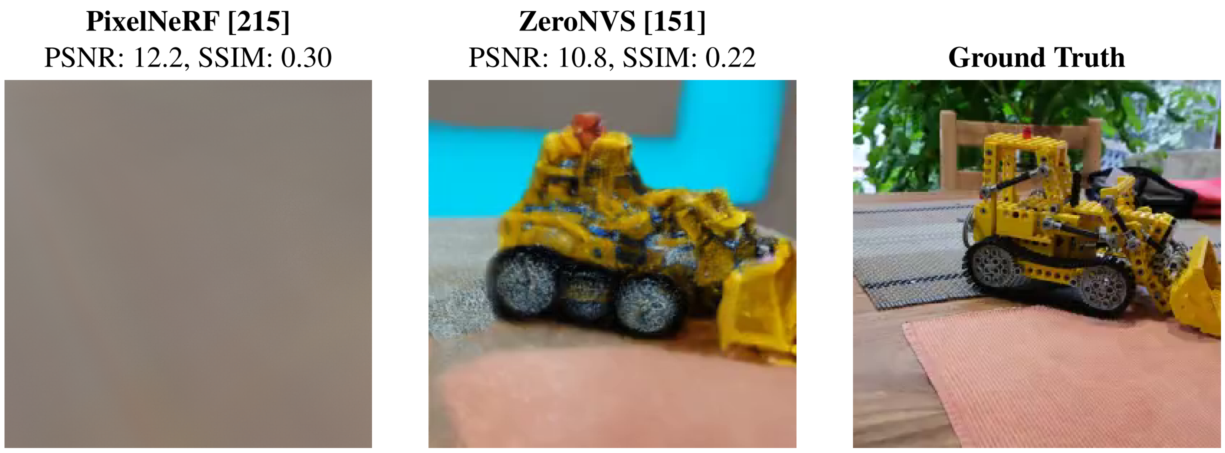


Figure 6.2: **Limitations of pixel-wise metrics.** Although ZeroNVS's [151] rendering is far more semantically meaningful than that of PixelNeRF [215], it scores more poorly with regards to PSNR and SSIM. Images courtesy of [151].

at scale - however [46] suggests that the combination of DUSt3R and 3D Gaussian representations [81] may now make this feasible.

Evaluation metrics. Finding metrics that can adequately quantify the impact of generative model is an open research question. As Sargent, Chen, et al note [25, 151], raw pixel metrics such as PSNR and SSIM that are commonly used to evaluate NeRF methods are poorly suited for generative solutions. They favor blurry estimates and can score monochromatic backgrounds over more semantically meaningful renderings that are slightly misaligned (Fig. 6.2). Perceptual measures such as LPIPS [224] and DISTS [40] that rely on learned feature extractors can more meaningfully evaluate the texture and structure similarity of generated renderings relative to ground truth targets. However, their utility is limited to evaluating the reconstruction quality of renderings that cover regions close to training data. These metrics become less meaningful when evaluating the *generative* capabilities of diffusion-based solutions and whether they generate plausible renderings far from from source views.

Fréchet Inception Distance (FID) [70] and Kernel Inception Distance (KID) [20] have traditionally been used to evaluate the fidelity and diversity of generative models such as GANs [63], but require thousands of samples to evaluate and are often too expensive to compute for SDS-based approaches [151]. The speed benefits of our proposed approach might enable their use, but further work remains to validate this hypothesis and whether these metrics are practical.

6.2 Accurate Camera Estimation

As mentioned in Chapter 2, NeRF rendering quality is heavily correlated with the accuracy of the camera intrinsics and extrinsics associated with the training images. To date, the vast majority of NeRF-related works treat camera parameter estimation as a separate pre-processing step, often obtained via incremental (or "classical") Structure-from-Motion (SfM) pipelines such as COLMAP [154] and Metashape [14]. Recent methods such as Pixel-Perfect SfM [104] (which we use for pose estimation in Chapter 2) further improve accuracy by aligning deep features, but rely on the same bundle adjustment [175] pipeline as classical approaches. Although these pipelines work well with small-scale, densely sampled, static captures, they fail to reliably infer accurate poses in sparse view settings, struggle with moving objects, and often slow down when processing over a thousand images.

Around the time that we first explored static reconstruction of city blocks (Chapter 2), several NeRF efforts [30, 76, 100, 114, 189] emerged that attempted to jointly optimize camera parameters during NeRF optimization, but we found that these methods tend to fall into local optima and provide results that are lacking relative to offline SfM approaches such as Pixel-Perfect SfM. More recently, a multitude of promising follow-up methods have emerged, some of which directly target NeRF reconstruction [27, 80, 115, 129, 160, 176] and some of which address camera estimation from a more general perspective [19, 99, 184, 190, 221, 222].

Once again, DUSt3R [190] stands out due to its robustness, rapid inference time, and ability to predict dense depth (which is useful for many neural reconstruction-related tasks such as that described in Section 6.1). First attempts at leveraging DUSt3R for neural reconstruction [46] show promise, but work remains. Extending DUSt3R to better support large image collections (from its current bottleneck of several hundred images), handle dynamic scenes, and directly integrate 3DGS-style [81] volumetric rendering are all worthwhile future directions.

6.3 Simulation Frameworks

The scope of this thesis concerns itself with large-scale neural rendering, which needs to be integrated into broader graphics and simulation frameworks in many use cases. Although the implicit neural representations described in previous chapters have many advantages, such as compactness and ease of optimization, they are difficult to integrate into external pipelines for two reasons. First, although they can be rendered in real-time on a desktop workstation via the method described in Chapter 4, they require a large amount of GPU memory to do so, which makes it difficult to accommodate other simulation routines, especially on lower-end devices. Second, as implicit representations backed by neural networks, they must be rendered separately from other commonly used graphics structures (such as meshes) and are often difficult to manipulate, especially in dynamic 4D scenes. Properly integrating them into external lighting representations and physics engines can be extremely challenging.

The recent explosion of interest in explicit, rasterization-based techniques such as 3D Gaussian splatting [81] potentially solves both of these issues. Although their rendering quality is currently slightly worse than those of state-of-the-art implicit methods (Chapter 4), many ongoing efforts aim to minimize the quality gap [21, 49, 79, 109, 210]. Importantly, gaussian splatting methods are extremely fast to render ($>$200 FPS [45]), can be designed to be reasonably memory-efficient [92, 119], and can reason about dynamics in a more interpretable manner than black-box neural methods [110].

Initial results suggest that gaussian representations can be indeed be efficiently relighted [148] and integrated into physics engines [198] for object-scale scenes. A logical next step would be to evaluate whether this holds for larger, unbounded outdoor scenes. Validating that this is the case would imply that the city-scale simulations we allude to in Chapter 3 are close to becoming a reality. A follow-up question would then arise — would it then be possible to train higher-level planning algorithms (such as those targeted by reinforcement learning) in a truly end-to-end manner via closed-loop simulation?

Bibliography

[1] Open3d oriented bounding box implementation. http://www.open3d.org/docs/latest/python_api/open3d.geometry.OrientedBoundingBox.html#open3d.geometry.OrientedBoundingBox.create_from_axis_aligned_bounding_box. Accessed: 2022-11-06. 3.4.2

[2] Gen 3.6 search and rescue. https://www.faa.gov/air_traffic/publications/atpubs/aip_html/part1_gen_section_3.6.html. Accessed: 2021-10-15. 2.1

[3] Autovrse training solutions. https://training.autovrse.in. Accessed: 2024-04-07. 1.1

[4] Meta codec avatars. https://tech.facebook.com/reality-labs/2019/3/codec-avatars-facebook-reality-labs. Accessed: 2024-04-07. 1.1

[5] Ekto dk1 experience. https://youtu.be/uONsZHtpRV8?si=1_hCKlIGhV5oCgnr. Accessed: 2024-04-08. 1.1

[6] Scikit incremental pca. https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.IncrementalPCA.html. Accessed: 2022-10-29. 3.4.1

[7] Unreal: Virtual reality training and its impact on the trades. https://www.leecontracting.com/virtual-reality-training-impact-on-trades. Accessed: 2024-04-07. 1.1

[8] Kwea123's nsff implementation. https://github.com/kwea123/nsff_pl. Accessed: 2022-10-29. 3.3.3

[9] Reconstructing the real world in drive sim with ai. https://blogs.nvidia.com/blog/drive-sim-neural-reconstruction-engine. Accessed: 2024-04-07. 1.1

[10] The role of vr training in upskilling and reskilling workers. https://www.strivr.com/blog/vr-upskilling-reskilling-workers. Accessed: 2024-04-07. 1.1

[11] Waymo's next chapter in san francisco. https://waymo.com/blog/2023/08/waymos-next-chapter-in-san-francisco. Accessed: 2024-04-07. 1.1

[12] Edward Adelson, Charles Anderson, James Bergen, Peter Burt, and Joan Ogden. Pyramid

methods in image processing. *RCA Eng.*, 29, 11 1983. 5.1, 5.2

[13] Sameer Agarwal, Yasutaka Furukawa, Noah Snavely, Ian Simon, Brian Curless, Steven M. Seitz, and Richard Szeliski. Building rome in a day. *Commun. ACM*, 54(10):105–112, oct 2011. ISSN 0001-0782. doi: 10.1145/2001269.2001293. URL `https://doi.org/10.1145/2001269.2001293`. 2.2

[14] Agisoft, LLC. Metashape 2.0, 2023. 6.2

[15] Shir Amir, Yossi Gandelsman, Shai Bagon, and Tali Dekel. Deep vit features as dense visual descriptors. *arXiv preprint arXiv:2112.05814*, 2021. 3.4, 3.4.1

[16] Jonathan T. Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P. Srinivasan. Mip-NeRF: A multiscale representation for anti-aliasing neural radiance fields. In *ICCV*, 2021. (document), 1.2, 1.6, 1.3.4, 5.1, 5.2, 5.3.1, 5.1, 5.2, 5.4.1, 5.4.2, 5.4.3

[17] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *CVPR*, 2022. (document), 3.2, 3.4.1, 4.1, 4.2, 4.3.1, 4.3.3, 4.4.3, 4.2, 4.4.3, 5.2, 5.2, 5.3.2, 5.4.1, 5.4.3, 5.3, 5.4

[18] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Zip-NeRF: Anti-aliased grid-based neural radiance fields. In *ICCV*, 2023. 1.3.4, 4.1, 4.4.2, 4.4.2, 4.2, 4.4.3, 4.3, 5.1, 5.2

[19] Axel Barroso-Laguna, Sowmya Munukutla, Victor Prisacariu, and Eric Brachmann. Matching 2d images in 3d: Metric relative pose from metric correspondences. In *CVPR*, 2024. 6.2

[20] Mikołaj Bińkowski, Dougal J. Sutherland, Michael Arbel, and Arthur Gretton. Demystifying MMD GANs. In *ICLR*, 2018. 6.1.3

[21] Samuel Rota Bulò, Lorenzo Porzi, and Peter Kontschieder. Revising densification in gaussian splatting, 2024. 6.3

[22] P. Burt and E. Adelson. The laplacian pyramid as a compact image code. *IEEE Transactions on Communications*, 31(4):532–540, 1983. doi: 10.1109/TCOM.1983.1095851. 5.2, 5.3.2

[23] Ang Cao and Justin Johnson. HexPlane: A fast representation for dynamic scenes. 2023. 4.3.1

[24] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. 2021. 3.3.1, 3.3.2

[25] Eric R Chan, Koki Nagano, Matthew A Chan, Alexander W Bergman, Jeong Joon Park, Axel Levy, Miika Aittala, Shalini De Mello, Tero Karras, and Gordon Wetzstein. Generative novel view synthesis with 3d-aware diffusion models. In *ICCV*, pages 4217–4229, 2023. 6.1.3

[26] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. In *ECCV*, 2022. 1.2, 1.3.3, 3.2, 4.2, 5.1, 5.2, 5.3.1, 5.1, 5.2, 5.4.2, 5.4.4, 5.3, 5.4, 5.5, 5.8, 5.5.1

[27] Yu Chen and Gim Hee Lee. Dbarf: Deep bundle-adjusting generalizable neural radiance fields. In *CVPR*, pages 24–34, 2023. 6.2

[28] Zhiqin Chen, Thomas Funkhouser, Peter Hedman, and Andrea Tagliasacchi. Mobilenerf: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures. In *CVPR*, 2023. 1.2, 1.3.3, 4.2, 4.2

[29] Bowen Cheng, Maxwell D Collins, Yukun Zhu, Ting Liu, Thomas S Huang, Hartwig Adam, and Liang-Chieh Chen. Panoptic-deeplab: A simple, strong, and fast baseline for bottom-up panoptic segmentation. In *CVPR*, 2020. 5.4.5

[30] Shin-Fang Chng, Sameera Ramasinghe, Jamie Sherrah, and Simon Lucey. Gaussian activated neural radiance fields for high fidelity reconstruction and pose estimation. In *ECCV*, page 264–280, Berlin, Heidelberg, 2022. Springer-Verlag. ISBN 978-3-031-19826-7. doi: 10.1007/978-3-031-19827-4_16. URL `https://doi.org/10.1007/978-3-031-19827-4_16`. 2.5.1, 6.2

[31] Ernest Cline. *Ready Player One*. Crown, first edition, 2011. (document), 1.1

[32] David Crandall, Andrew Owens, Noah Snavely, and Daniel Huttenlocher. Discrete-continuous optimization for large-scale structure from motion. In *CVPR*, 2011. 1.3.1, **??**, 2.4.1

[33] Cyril Crassin, Fabrice Neyret, Miguel Sainz, Simon Green, and Elmar Eisemann. Interactive indirect illumination using voxel cone tracing: A preview. In *Symposium on Interactive 3D Graphics and Games*, 2011. 5.1, 5.2

[34] Paul Debevec, Yizhou Yu, and George Borshukov. Efficient view-dependent image-based rendering with projective texture-mapping. In *Rendering Techniques' 98: Proceedings of the Eurographics Workshop in Vienna, Austria, June 29—July 1, 1998 9*, pages 105–116. Springer, 1998. 1.1

[35] Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3d objects. In *CVPR*, pages 13142–13153, 2023. 6.1.1

[36] Tali Dekel, Shaul Oron, Michael Rubinstein, Shai Avidan, and William T. Freeman. Best-buddies similarity for robust template matching. In *CVPR*, pages 2021–2029, 2015. doi: 10.1109/CVPR.2015.7298813. 3.4.2

[37] Kangle Deng, Andrew Liu, Jun-Yan Zhu, and Deva Ramanan. Depth-supervised NeRF: Fewer views and faster training for free. In *CVPR*, June 2022. 3.2, 3.3.3, 6.1

[38] Valentin Deschaintre, Miika Aittala, Fredo Durand, George Drettakis, and Adrien Bousseau. Single-image svbrdf capture with a rendering-aware deep network. *ACM Transactions on Graphics (ToG)*, 37(4):1–15, 2018. 1.1

[39] Valentin Deschaintre, Miika Aittala, Frédo Durand, George Drettakis, and Adrien Bousseau. Flexible svbrdf capture with a multi-image deep network. In *Computer graphics forum*, volume 38, pages 1–13. Wiley Online Library, 2019. 1.1

[40] Keyan Ding, Kede Ma, Shiqi Wang, and Eero P Simoncelli. Image quality assessment: Unifying structure and texture similarity. *IEEE transactions on pattern analysis and ma-*

chine intelligence, 44(5):2567–2581, 2020. 6.1.3

[41] Yilun Du, Yinan Zhang, Hong-Xing Yu, Joshua B. Tenenbaum, and Jiajun Wu. Neural radiance flow for 4d view synthesis and video processing. In *ICCV*, 2021. 1.3.2, 2.5.1, 3.2, 3.3.2

[42] Daniel Duckworth, Peter Hedman, Christian Reiser, Peter Zhizhin, Jean-François Thibert, Mario Lučić, Richard Szeliski, and Jonathan T. Barron. SMERF: Streamable memory efficient radiance fields for real-time large-scene exploration. arXiv:2312.07541, 2023. 4.2, 4.4.4

[43] SM Ali Eslami, Danilo Jimenez Rezende, Frederic Besse, Fabio Viola, Ari S Morcos, Marta Garnelo, Avraham Ruderman, Andrei A Rusu, Ivo Danihelka, Karol Gregor, et al. Neural scene representation and rendering. *Science*, 360(6394):1204–1210, 2018. 1.1

[44] J. Eyerman, G. Crispino, A. Zamarro, and R Durscher. Drone efficacy study (DES): Evaluating the impact of drones for locating lost persons in search and rescue events. *Brussels, Belgium: DJI and European Emergency Number Association*, 2018. 2.1

[45] Zhiwen Fan, Kevin Wang, Kairun Wen, Zehao Zhu, Dejia Xu, and Zhangyang Wang. Lightgaussian: Unbounded 3d gaussian compression with 15x reduction and 200+ fps, 2023. 6.3

[46] Zhiwen Fan, Wenyan Cong, Kairun Wen, Kevin Wang, Jian Zhang, Xinghao Ding, Danfei Xu, Boris Ivanovic, Marco Pavone, Georgios Pavlakos, Zhangyang Wang, and Yue Wang. Instantsplat: Unbounded sparse-view pose-free gaussian splatting in 40 seconds, 2024. 6.1.3, 6.2

[47] L-CCGP Florian and Schroff Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. In *Conference on Computer Vision and Pattern Recognition (CVPR). IEEE/CVF*, 2017. 2.4.1

[48] 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. **??**, **??**, 2.3, 2.4.2

[49] Linus Franke, Darius Rückert, Laura Fink, and Marc Stamminger. Trips: Trilinear point splatting for real-time radiance field rendering. *Computer Graphics Forum*, 43(2), 2024. doi: https://doi.org/10.1111/cgf.15012. 6.3

[50] Sara Fridovich-Keil, Giacomo Meanti, Frederik Rahbæk Warburg, Benjamin Recht, and Angjoo Kanazawa. K-planes: Explicit radiance fields in space, time, and appearance. In *CVPR*, 2023. 1.2, 1.3.3, 4.2, 4.3.1, 5.1, 5.2, 5.3.1, 5.3.2, 5.1, 5.2, 5.4.2, 5.4.4, 5.3, 5.4, 5.5, 5.8, 5.5.1

[51] Xiao Fu, Shangzhan Zhang, Tianrun Chen, Yichong Lu, Lanyun Zhu, Xiaowei Zhou, Andreas Geiger, and Yiyi Liao. Panoptic nerf: 3d-to-2d label transfer for panoptic urban scene segmentation. In *International Conference on 3D Vision (3DV)*, 2022. 3.2

[52] Thomas A. Funkhouser and Carlo H. Séquin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*,

SIGGRAPH '93, page 247–254, New York, NY, USA, 1993. Association for Computing Machinery. ISBN 0897916018. doi: 10.1145/166117.166149. URL https://doi.org/10.1145/166117.166149. 2.2

[53] Thomas A Funkhouser, Carlo H Sequin, and Seth J Teller. Management of large amounts of data in interactive building walkthroughs. In *Proceedings of the 1992 symposium on Interactive 3D graphics*, pages 11–20, 1992. 3.3.2

[54] Adrien Gaidon, Qiao Wang, Yohann Cabon, and Eleonora Vig. Virtual worlds as proxy for multi-object tracking analysis. In *CVPR*, pages 4340–4349, 2016. 3.4.3

[55] Chen Gao, Ayush Saraf, Johannes Kopf, and Jia-Bin Huang. Dynamic view synthesis from dynamic monocular video. In *ICCV*, 2021. 1.3.2, 3.2, 3.3.2, 3.3.2, 3.3.3, 3.3.3

[56] Chen Gao, Ayush Saraf, Johannes Kopf, and Jia-Bin Huang. Dynamic view synthesis from dynamic monocular video. In *ICCV*, 2021. 2.5.1

[57] Hang Gao, Ruilong Li, Shubham Tulsiani, Bryan Russell, and Angjoo Kanazawa. Monocular dynamic view synthesis: A reality check. In *NeurIPS*, 2022. 3.1

[58] Stephan J Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin. Fastnerf: High-fidelity neural rendering at 200fps. pages 14346–14355, 2021. 1.2, 1.3.3, 2.2, 4.2, 5.2

[59] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *CVPR*, 2012. 3.4.3

[60] S. George, T. Eiszler, R. Iyengar, H. Turki, Z. Feng, J. Wang, P. Pillai, and M. Satyanarayanan. Openrtist: End-to-end benchmarking for edge computing. *IEEE Pervasive Computing*, 19(04):10–18, oct 2020. ISSN 1558-2590. doi: 10.1109/MPRV.2020.3028781. 5

[61] Shilpa George, Haithem Turki, Ziqiang Feng, Deva Ramanan, Padmanabhan Pillai, and Mahadev Satyanarayanan. *Low-Bandwidth Self-Improving Transmission of Rare Training Data*. Association for Computing Machinery, New York, NY, USA, 2023. ISBN 9781450399906. URL https://doi.org/10.1145/3570361.3613300. 3

[62] William Gibson. *Neuromancer*. Ace, first edition, 1986. (document), 1.1

[63] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *NeurIPS*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.NeurIPS.cc/paper_files/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf. 6.1.3

[64] Yuan-Chen Guo, Yan-Pei Cao, Chen Wang, Yu He, Ying Shan, Xiaohu Qie, and Song-Hai Zhang. VMesh: Hybrid volume-mesh representation for efficient view synthesis. 2023. 4.2

[65] Kunal Gupta, Miloš Hašan, Zexiang Xu, Fujun Luan, Kalyan Sunkavalli, Xin Sun, Manmohan Chandraker, and Sai Bi. MCNeRF: Monte Carlo rendering and denoising for real-time NeRFs. 2023. 1.2, 1.3.3, 4.2

[66] Zekun Hao, Arun Mallya, Serge Belongie, and Ming-Yu Liu. GANcraft: Unsupervised 3D Neural Rendering of Minecraft Worlds. In *ICCV*, 2021. 3.3.2

[67] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. Deep blending for free-viewpoint image-based rendering. *ACM Trans. Graph.*, 37(6), dec 2018. ISSN 0730-0301. doi: 10.1145/3272127.3275084. URL `https://doi.org/10.1145/3272127.3275084`. 4.2

[68] Peter Hedman, Pratul P. Srinivasan, Ben Mildenhall, Jonathan T. Barron, and Paul Debevec. Baking neural radiance fields for real-time view synthesis. *ICCV*, 2021. 1.2, 1.3.3, 2.2, 4.2, 5.2

[69] Philipp Henzler, Niloy Mitra, and Tobias Ritschel. Escaping plato's cave using adversarial training: 3d shape from unstructured 2d image collections. In *ICCV*. IEEE, 2019. 1.1

[70] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *NeurIPS*, 30, 2017. 6.1.3

[71] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *NeurIPS*, 33:6840–6851, 2020. 6.1

[72] Dongting Hu, Zhenkai Zhang, Tingbo Hou, Tongliang Liu, Huan Fu, and Mingming Gong. Multiscale representation for real-time anti-aliasing neural rendering. arxiv:2304.10075, 2023. 5.2, 5.1, 5.4.2

[73] Hanzhe Hu, Zhizhuo Zhou, Varun Jampani, and Shubham Tulsiani. Mvd-fusion: Single-view 3d via depth-consistent multi-view generation. In *CVPR*, 2024. 6.1.1

[74] Wenbo Hu, Yuling Wang, Lin Ma, Bangbang Yang, Lin Gao, Xiao Liu, and Yuewen Ma. Tri-miprf: Tri-mip representation for efficient anti-aliasing neural radiance fields. In *ICCV*, 2023. 5.2

[75] Brian K. S. Isaac-Medina, Chris G. Willcocks, and Toby P. Breckon. Exact-NeRF: An exploration of a precise volumetric parameterization for neural radiance fields. *CVPR*, 2023. 5.2, 5.4.3, 5.3, 5.4

[76] Yoonwoo Jeong, Seokjun Ahn, Christopher Choy, Animashree Anandkumar, Minsu Cho, and Jaesik Park. Self-calibrating neural radiance fields. In *ICCV*, 2021. 2.5.1, 6.2

[77] Zhang Jiakai, Liu Xinhang, Ye Xinyi, Zhao Fuqiang, Zhang Yanshun, Wu Minye, Zhang Yingliang, Xu Lan, and Yu Jingyi. Editable free-viewpoint video using a layered neural representation. In *ACM SIGGRAPH*, 2021. 3.2

[78] Yuhe Jin, Dmytro Mishkin, Anastasiia Mishchuk, Jiri Matas, Pascal Fua, Kwang Moo Yi, and Eduard Trulls. Image matching across wide baselines: From paper to practice. *International Journal of Computer Vision*, 129(2):517–547, 2021. **??**, 2.2

[79] Jaewoo Jung, Jisang Han, Honggyu An, Jiwon Kang, Seonghoon Park, and Seungryong Kim. Relaxing accurate initialization constraint for 3d gaussian splatting. *arXiv preprint arXiv:2403.09413*, 2024. 6.3

[80] Nikhil Keetha, Jay Karhade, Krishna Murthy Jatavallabhula, Gengshan Yang, Sebastian Scherer, Deva Ramanan, and Jonathon Luiten. Splatam: Splat, track & map 3d gaussians

for dense rgb-d slam. In *CVPR*, 2024. 6.1.2, 6.1, 6.2

[81] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), July 2023. URL `https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/`. (document), 1.3.3, 4.1, 4.2, 4.1, 4.4.2, 4.4.2, 4.2, 4.8, 4.4.3, 4.3, 4.5, 6.1.1, 6.1, 6.1.3, 6.2, 6.3

[82] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3D Gaussian splatting for real-time radiance field rendering. 42(4):139:1–14, 2023. doi: 10.1145/3592433. 4.4.3

[83] Bernhard Kerbl, Andreas Meuleman, Georgios Kopanas, Michael Wimmer, Alexandre Lanvin, and George Drettakis. A hierarchical 3d gaussian representation for real-time rendering of very large datasets. *ACM Transactions on Graphics*, 43(4), July 2024. URL `https://repo-sam.inria.fr/fungraph/hierarchical-3d-gaussians/`. 1.3.1

[84] Justin Kerr, Letian Fu, Huang Huang, Yahav Avigal, Matthew Tancik, Jeffrey Ichnowski, Angjoo Kanazawa, and Ken Goldberg. Evo-nerf: Evolving nerf for sequential robot grasping of transparent objects. In *6th annual conference on robot learning*, 2022. 1.1

[85] Leonid Keselman and Martial Hebert. Approximate differentiable rendering with algebraic surfaces. In *ECCV*, 2022. 4.2

[86] Leonid Keselman and Martial Hebert. Flexible techniques for differentiable rendering with 3D Gaussians. arXiv:2308.14737, 2023. 4.2

[87] DP Kingma, J Ba, Y Bengio, and Y LeCun. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations*, 2015. 2.4.1, 3.4.1, 4.4.1

[88] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics*, 36(4), 2017. 2.1, **??**

[89] Sosuke Kobayashi, Eiichi Matsumoto, and Vincent Sitzmann. Decomposing nerf for editing via feature field distillation. In *NeurIPS*, volume 35, 2022. 3.2, 3.3.2

[90] Abhijit Kundu, Kyle Genova, Xiaoqi Yin, Alireza Fathi, Caroline Pantofaru, Leonidas Guibas, Andrea Tagliasacchi, Frank Dellaert, and Thomas Funkhouser. Panoptic Neural Fields: A Semantic Object-Aware Neural Scene Representation. In *CVPR*, 2022. (document), 1.3.2, 3.1, 3.2, **??**, 3.4, 3.4.3

[91] Andreas Kurz, Thomas Neff, Zhaoyang Lv, Michael Zollhöfer, and Markus Steinberger. Adanerf: Adaptive sampling for real-time rendering of neural radiance fields. 2022. 1.2, 1.3.3, 4.2

[92] Joo Chan Lee, Daniel Rho, Xiangyu Sun, Jong Hwan Ko, and Eunbyung Park. Compact 3d gaussian representation for radiance field. In *CVPR*, 2024. 6.3

[93] Aviad Levis, Pratul P Srinivasan, Andrew A Chael, Ren Ng, and Katherine L Bouman. Gravitationally lensed black hole emission tomography. In *CVPR*, pages 19841–19850, 2022. 1.1

[94] Tianye Li, Mira Slavcheva, Michael Zollhoefer, Simon Green, Christoph Lassner, Changil Kim, Tanner Schmidt, Steven Lovegrove, Michael Goesele, and Zhaoyang Lv. Neural 3d video synthesis. In *CVPR*, 2022. 3.2

[95] Zhaoshuo Li, Thomas Müller, Alex Evans, Russell H. Taylor, Mathias Unberath, Ming-Yu Liu, and Chen-Hsuan Lin. Neuralangelo: High-fidelity neural surface reconstruction. 2023. doi: 10.1109/CVPR52729.2023.00817. 4.3.3

[96] Zhengqi Li, Simon Niklaus, Noah Snavely, and Oliver Wang. Neural scene flow fields for space-time view synthesis of dynamic scenes. In *CVPR*, 2021. 1.3.2, 3.2, 3.3.2, 3.3.2, 3.3.3, 3.3.3, 3.3.3

[97] Zhengqin Li, Kalyan Sunkavalli, and Manmohan Chandraker. Materials for masses: Svbrdf acquisition with a single mobile phone image. In *ECCV*, 2018. 1.1

[98] Jiabin Liang, Lanqing Zhang, Zhuoran Zhao, and Xiangyu Xu. Infnerf: Towards infinite scale nerf rendering with o(log n) space complexity, 2024. 1.3.1

[99] Amy Lin, Jason Y Zhang, Deva Ramanan, and Shubham Tulsiani. RelPose++: Recovering 6d poses from sparse-view observations. In *International Conference on 3D Vision (3DV)*, 2024. 6.2

[100] Chen-Hsuan Lin, Wei-Chiu Ma, Antonio Torralba, and Simon Lucey. Barf: Bundle-adjusting neural radiance fields. In *ICCV*, 2021. 2.5.1, 6.2

[101] Chen-Hsuan Lin, Jun Gao, Luming Tang, Towaki Takikawa, Xiaohui Zeng, Xun Huang, Karsten Kreis, Sanja Fidler, Ming-Yu Liu, and Tsung-Yi Lin. Magic3d: High-resolution text-to-3d content creation. In *CVPR*, 2023. 1.1

[102] Jiaqi Lin, Zhihao Li, Xiao Tang, Jianzhuang Liu, Shiyong Liu, Jiayue Liu, Yangdi Lu, Xiaofei Wu, Songcen Xu, Youliang Yan, and Wenming Yang. Vastgaussian: Vast 3d gaussians for large scene reconstruction. In *CVPR*, 2024. 1.3.1

[103] Liqiang Lin, Yilin Liu, Yue Hu, Xingguang Yan, Ke Xie, and Hui Huang. Capturing, reconstructing, and simulating: the urbanscene3d dataset. In *ECCV*, 2022. 1.3.1, 1.3.2, **??**, 2.4.1

[104] Philipp Lindenberger, Paul-Edouard Sarlin, Viktor Larsson, and Marc Pollefeys. Pixel-Perfect Structure-from-Motion with Featuremetric Refinement. In *ICCV*, 2021. 2.4.1, 6.2

[105] Lu Ling, Yichen Sheng, Zhi Tu, Wentian Zhao, Cheng Xin, Kun Wan, Lantao Yu, Qianyu Guo, Zixun Yu, Yawen Lu, et al. Dl3dv-10k: A large-scale scene dataset for deep learning-based 3d vision. In *CVPR*, 2024. 6.1.1

[106] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. *NIPS*, 2020. 5.2

[107] Ruoshi Liu, Rundi Wu, Basile Van Hoorick, Pavel Tokmakov, Sergey Zakharov, and Carl Vondrick. Zero-1-to-3: Zero-shot one image to 3d object. In *ICCV*, 2023. 6.1, 6.1.2, 6.1.3

[108] Shaohui Liu, Yinda Zhang, Songyou Peng, Boxin Shi, Marc Pollefeys, and Zhaopeng Cui. DIST: Rendering deep implicit signed distance function with differentiable sphere tracing. 2020. 4.1

[109] Wenkai Liu, Tao Guan, Bin Zhu, Lili Ju, Zikai Song, Dan Li, Yuesong Wang, and Wei Yang. Efficientgs: Streamlining gaussian splatting for large-scale high-resolution scene representation, 2024. 6.3

[110] Jonathon Luiten, Georgios Kopanas, Bastian Leibe, and Deva Ramanan. Dynamic 3d gaussians: Tracking by persistent dynamic view synthesis. In *3DV*, 2024. 6.3

[111] Li Ma, Vasu Agrawal, Haithem Turki, Changil Kim, Chen Gao, Pedro Sander, Michael Zollhöfer, and Christian Richardt. Specnerf: Gaussian directional encoding for specular reflections. In *CVPR*, 2024. 2

[112] Stephen Robert Marschner. *Inverse rendering for computer graphics*. Cornell University, 1998. 1.1

[113] 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 *CVPR*, 2021. 2.2, 2.2, 2.3.1, 2.4.1, 3.2, 5.4.3

[114] Quan Meng, Anpei Chen, Haimin Luo, Minye Wu, Hao Su, Lan Xu, Xuming He, and Jingyi Yu. GNeRF: GAN-based Neural Radiance Field without Posed Camera. In *ICCV*, 2021. 2.5.1, 6.2

[115] Andreas Meuleman, Yu-Lun Liu, Chen Gao, Jia-Bin Huang, Changil Kim, Min H Kim, and Johannes Kopf. Progressively optimized local radiance fields for robust view synthesis. In *CVPR*, pages 16539–16548, 2023. 6.2

[116] Zhenxing Mi and Dan Xu. Switch-nerf: Learning scene decomposition with mixture of experts for large-scale neural radiance fields. In *ICLR*, 2023. 1.3.1, 1.3.2

[117] Ben Mildenhall, Pratul P. Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. *ACM Transactions on Graphics (TOG)*, 2019. **??**, 4.4.3

[118] 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 *ECCV*, 2020. (document), 1.1, 1.2, 1.6, 1.3.4, **??**, 2.3.1, **??**, **??**, 2.3, 2.4.2, 3.3.2, 3.3.3, **??**, **??**, **??**, 3.4.3, 4.3.1, 4.2, 5.2, 5.3.1, 5.4.6, 5.8

[119] Wieland Morgenstern, Florian Barthel, Anna Hilsmann, and Peter Eisert. Compact 3d scene representation via self-organizing gaussian grids, 2024. 6.3

[120] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4): 102:1–102:15, July 2022. doi: 10.1145/3528223.3530127. URL `https://doi.org/10.1145/3528223.3530127`. (document), 1.2, 1.3.2, 1.3.3, 1.6, 3.2, 3.3.2, 4.2, 4.3.1, 4.3.4, 4.3.4, 4.1, 4.4.2, 4.4.2, 4.2, 4.3, 4.4.4, 4.4.4, 4.5.1, 5.1, 5.2, 5.3.1, 5.3.2, 5.1, 5.2, 5.4.1, 5.4.2, 5.3, 5.4, 5.5, 5.8, 5.5.1

[121] Thomas Neff, Pascal Stadlbauer, Mathias Parger, Andreas Kurz, Joerg H. Mueller, Chakravarty R. Alla Chaitanya, Anton S. Kaplanyan, and Markus Steinberger. DONeRF: Towards Real-Time Rendering of Compact Neural Radiance Fields using Depth

Oracle Networks. *Computer Graphics Forum*, 40(4), 2021. ISSN 1467-8659. doi: 10.1111/cgf.14340. URL `https://doi.org/10.1111/cgf.14340`. 1.2, 1.3.3, 2.2, 4.2

[122] Thu Nguyen-Phuoc, Chuan Li, Lucas Theis, Christian Richardt, and Yong-Liang Yang. Hologan: Unsupervised learning of 3d representations from natural images. In *ICCV*, pages 7588–7597, 2019. 1.1

[123] Michael Niemeyer and Andreas Geiger. Giraffe: Representing scenes as compositional generative neural feature fields. In *CVPR*, 2021. 3.2

[124] Michael Niemeyer, Jonathan T. Barron, Ben Mildenhall, Mehdi S. M. Sajjadi, Andreas Geiger, and Noha Radwan. Regnerf: Regularizing neural radiance fields for view synthesis from sparse inputs. In *CVPR*, 2022. 6.1

[125] Michael Oechsle, Songyou Peng, and Andreas Geiger. UNISURF: Unifying neural implicit surfaces and radiance fields for multi-view reconstruction. In *ICCV*, 2021. 1.3.3, 4.2, 4.3.1

[126] Julian Ost, Fahim Mannan, Nils Thuerey, Julian Knodt, and Felix Heide. Neural scene graphs for dynamic scenes. In *CVPR*, pages 2856–2865, June 2021. (document), 1.3.2, 2.5.1, 3.1, 3.2, 3.7, **??**, **??**, **??**, 3.4, 3.4.3

[127] Keunhong Park, Utkarsh Sinha, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Steven M. Seitz, and Ricardo Martin-Brualla. Nerfies: Deformable neural radiance fields. *ICCV*, 2021. 2.5.1, 3.1, 3.2

[128] Keunhong Park, Utkarsh Sinha, Peter Hedman, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Ricardo Martin-Brualla, and Steven M. Seitz. Hypernerf: A higher-dimensional representation for topologically varying neural radiance fields. *ACM Trans. Graph.*, 40(6), dec 2021. 3.2

[129] Philipp; Mildenhall Ben; Barron Jonathan T.; Martin-Brualla Ricardo Park, Keunhong; Henzler. Camp: Camera preconditioning for neural radiance fields. *ACM Trans. Graph.*, 2023. 6.2

[130] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. pages 8024–8035, 2019. 4.4.1

[131] M. Piala and R. Clark. Terminerf: Ray termination prediction for efficient neural rendering. In *2021 International Conference on 3D Vision (3DV)*, pages 1106–1114, Los Alamitos, CA, USA, dec 2021. IEEE Computer Society. doi: 10.1109/3DV53792.2021.00118. URL `https://doi.ieeecomputersociety.org/10.1109/3DV53792.2021.00118`. 1.2, 1.3.3, 4.2

[132] Ben Poole, Ajay Jain, Jonathan T. Barron, and Ben Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. In *ICLR*, 2023. 1.1, 6.1

[133] Charalambos Poullis, Andrew Gardner, and Paul Debevec. Photogrammetric Modeling

and Image-based Rendering for Rapid Virtual Environement Creation. In *Proceedings of the 24th Army Science Conference*, Orlando, FL, December 2004. 1.1

[134] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-NeRF: Neural Radiance Fields for Dynamic Scenes. In *CVPR*, 2020. 3.2

[135] Ravi Ramamoorthi. Nerfs: The search for the best 3d representation, 2023. (document), 1.1

[136] René Ranftl, Alexey Bochkovskiy, and Vladlen Koltun. Vision transformers for dense prediction. In *ICCV*, pages 12179–12188, October 2021. 6.1.1

[137] D. Rebain, W. Jiang, S. Yazdani, K. Li, K. Yi, and A. Tagliasacchi. Derf: Decomposed radiance fields. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14148–14156, Los Alamitos, CA, USA, jun 2021. IEEE Computer Society. doi: 10.1109/CVPR46437.2021.01393. URL `https://doi.ieeecomputersociety.org/10.1109/CVPR46437.2021.01393`. 2.1, 2.2

[138] D. Rebain, W. Jiang, S. Yazdani, K. Li, K. Yi, and A. Tagliasacchi. DeRF: Decomposed radiance fields. In *CVPR*, 2021. 5.1, 5.2

[139] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. In *ICCV*, pages 14335–14345, 2021. 2.1, 2.2, 2.4.3, 4.4.4, 5.1, 5.2

[140] Christian Reiser, Richard Szeliski, Dor Verbin, Pratul P. Srinivasan, Ben Mildenhall, Andreas Geiger, Jonathan T. Barron, and Peter Hedman. MERF: Memory-efficient radiance fields for real-time view synthesis in unbounded scenes. 42(4):89:1–12, 2023. doi: 10.1145/3592426. 4.2, 4.3.2, 4.3.4, 4.4.1, 4.1, 4.4.2, 4.4.2, 4.2, 4.3, 4.4.4, 4.5.1

[141] Konstantinos Rematas, Andrew Liu, Pratul P. Srinivasan, Jonathan T. Barron, Andrea Tagliasacchi, Tom Funkhouser, and Vittorio Ferrari. Urban radiance fields. 2022. 1.2, 1.3.2, 2.2, 3.2, 3.3.2, 3.3.2, 3.3.3

[142] Gernot Riegler and Vladlen Koltun. Stable view synthesis. In *CVPR*, 2021. **??**, **??**, 2.3, 2.4.2, 4.2

[143] Barbara Roessle, Jonathan T. Barron, Ben Mildenhall, Pratul P. Srinivasan, and Matthias Nießner. Dense depth priors for neural radiance fields from sparse input views. In *CVPR*, June 2022. 3.2, 6.1

[144] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, pages 10684–10695, June 2022. 6.1

[145] Radu Alexandru Rosu and Sven Behnke. PermutoSDF: Fast multi-view reconstruction with implicit surfaces using permutohedral lattices. 2023. doi: 10.1109/CVPR52729.2023.00818. 4.3.4

[146] Darius Rückert, Linus Franke, and Marc Stamminger. Adop: Approximate differentiable one-pixel point rendering. *ACM Trans. Graph.*, 41(4):99:1–14, jul 2022. ISSN 0730-0301. (document), 5.4.3, 5.3, 5.4

[147] Darius Rückert, Yuanhao Wang, Rui Li, Ramzi Idoughi, and Wolfgang Heidrich. Neat:

Neural adaptive tomography. *ACM Transactions on Graphics (TOG)*, 41(4):1–13, 2022. 1.1

[148] Shunsuke Saito, Gabriel Schwartz, Tomas Simon, Junxuan Li, and Giljoo Nam. Relightable gaussian codec avatars. In *CVPR*, 2024. 6.3

[149] Sara Fridovich-Keil and Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *CVPR*, 2022. 2.4.3, **??**

[150] Sara Fridovich-Keil and Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *CVPR*, 2022. 3.2, 5.1, 5.2, 5.3.1, 5.1, 5.2, 5.4.2, 5.3, 5.4, 5.5.1

[151] Kyle Sargent, Zizhang Li, Tanmay Shah, Charles Herrmann, Hong-Xing Yu, Yunzhi Zhang, Eric Ryan Chan, Dmitry Lagun, Li Fei-Fei, Deqing Sun, and Jiajun Wu. ZeroNVS: Zero-shot 360-degree view synthesis from a single real image. In *CVPR*, June 2024. 6.1, 6.1.2, 6.1.3, **??**, 6.2

[152] M. Satyanarayanan, T. Eiszler, J. Harkes, H. Turki, and Z. Feng. Edge computing for legacy applications. *IEEE Pervasive Computing*, 19(04):19–28, oct 2020. ISSN 1558-2590. doi: 10.1109/MPRV.2020.3026229. 6

[153] Mahadev Satyanarayanan, Ziqiang Feng, Shilpa George, Jan Harkes, Roger Iyengar, Haithem Turki, and Padmanabhan Pillai. Accelerating silent witness storage. *IEEE Micro*, 42(6):39–47, 2022. doi: 10.1109/MM.2022.3193048. 4

[154] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 6.2

[155] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise view selection for unstructured multi-view stereo. In *European Conference on Computer Vision (ECCV)*, 2016. **??**, **??**, 2.3, 2.4.2

[156] Philipp Schröppel, Jan Bechtold, Artemij Amiranashvili, and Thomas Brox. A benchmark and a baseline for robust multi-view depth estimation. In *Proceedings of the International Conference on 3D Vision (3DV)*, 2022. 6.1.1

[157] Prafull Sharma, Ayush Tewari, Yilun Du, Sergey Zakharov, Rares Andrei Ambrus, Adrien Gaidon, William T. Freeman, Fredo Durand, Joshua B. Tenenbaum, and Vincent Sitzmann. Neural groundplans: Persistent neural scene representations from a single image, 2023. URL `https://openreview.net/forum?id=Pza24zf9FpS`. 3.2, 3.4.2

[158] Harry Shum and Sing Bing Kang. Review of image-based rendering techniques. In *Visual Communications and Image Processing 2000*, volume 4067, pages 2–13. SPIE, 2000. 1.1

[159] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. In *NeurIPS*, 2019. 1.1, **??**, 3.4.3

[160] Cameron Smith, David Charatan, Ayush Tewari, and Vincent Sitzmann. Flowmap: High-quality camera poses, intrinsics, and depth via gradient descent. In *CVPR*, 2024. 6.2

[161] SMPTE. High dynamic range electro-optical transfer function of mastering reference

displays. SMPTE Standard ST 2084:2014, Society of Motion Picture and Television Engineers, 2014. 4.4.2

[162] Nagabhushan Somraj, Adithyan Karanayil, and Rajiv Soundararajan. SimpleNeRF: Regularizing sparse input neural radiance fields with simpler solutions. In *SIGGRAPH Asia*, December 2023. doi: 10.1145/3610548.3618188. 6.1

[163] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *International Conference on Learning Representations*, 2021. 6.1

[164] Gabriela Ben Melech Stan, Diana Wofk, Scottie Fox, Alex Redden, Will Saxton, Jean Yu, Estelle Aflalo, Shao-Yen Tseng, Fabio Nonato, Matthias Muller, and Vasudev Lal. Ldm3d: Latent diffusion model for 3d, 2023. 6.1.1

[165] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In *CVPR*, 2022. 3.2, 5.2, 5.3.1

[166] 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 *CVPR*, 2021. 2.2

[167] Matthew Tancik, Vincent Casser, Xinchen Yan, Sabeek Pradhan, Ben Mildenhall, Pratul P. Srinivasan, Jonathan T. Barron, and Henrik Kretzschmar. Block-nerf: Scalable large scene neural view synthesis. In *CVPR*, pages 8248–8258, June 2022. 1.2, 1.3.2, 2.2, 3.2, 3.3.2, 5.1, 5.2, 5.3.2

[168] Matthew Tancik, Ethan Weber, Evonne Ng, Ruilong Li, Brent Yi, Justin Kerr, Terrance Wang, Alexander Kristoffersen, Jake Austin, Kamyar Salahi, Abhik Ahuja, David McAllister, and Angjoo Kanazawa. Nerfstudio: A modular framework for neural radiance field development. In *ACM SIGGRAPH 2023 Conference Proceedings*, SIGGRAPH '23, 2023. 1.2, 1.3.3, 5.1, 5.2, 5.4.1, 5.4.2, 5.3, 5.4, 5.8

[169] Matthew Tancik, Ethan Weber, Evonne Ng, Ruilong Li, Brent Yi, Terrance Wang, Alexander Kristoffersen, Jake Austin, Kamyar Salahi, Abhik Ahuja, David Mcallister, Justin Kerr, and Angjoo Kanazawa. Nerfstudio: A modular framework for neural radiance field development. pages 72:1–12, 2023. doi: 10.1145/3588432.3591516. 4.3.1

[170] Zachary Teed and Jia Deng. Raft: Recurrent all-pairs field transforms for optical flow (extended abstract). In *IJCAI*, 2021. 3.4.1, 3.4.2

[171] Seth J. Teller and Carlo H. Séquin. Visibility preprocessing for interactive walkthroughs. In *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '91, page 61–70, New York, NY, USA, 1991. Association for Computing Machinery. ISBN 0897914368. doi: 10.1145/122718.122725. URL `https://doi.org/10.1145/122718.122725`. 2.2

[172] Ayush Tewari, Ohad Fried, Justus Thies, Vincent Sitzmann, Stephen Lombardi, Kalyan Sunkavalli, Ricardo Martin-Brualla, Tomas Simon, Jason Saragih, Matthias Nießner, Rohit Pandey, Sean Fanello, Gordon Wetzstein, Jun-Yan Zhu, Christian Theobalt, Maneesh Agrawala, Eli Shechtman, Dan B Goldman, and Michael Zollhöfer. State of the art on neural rendering, 2020. 1.1

[173] Edgar Tretschk, Ayush Tewari, Vladislav Golyanik, Michael Zollhöfer, Christoph Lassner, and Christian Theobalt. Non-rigid neural radiance fields: Reconstruction and novel view synthesis of a dynamic scene from monocular video. In *ICCV*, 2021. 2.5.1, 3.2

[174] Alex Trevithick and Bo Yang. Grf: Learning a general radiance field for 3d representation and rendering. In *ICCV*, pages 15182–15192, 2021. 2.2

[175] Bill Triggs, Philip F McLauchlan, Richard I Hartley, and Andrew W Fitzgibbon. Bundle adjustment—a modern synthesis. In *Vision Algorithms: Theory and Practice: International Workshop on Vision Algorithms Corfu, Greece, September 21–22, 1999 Proceedings*, pages 298–372. Springer, 2000. 6.2

[176] Prune Truong, Marie-Julie Rakotosaona, Fabian Manhardt, and Federico Tombari. Sparf: Neural radiance fields from sparse and noisy poses. In *CVPR*, pages 4190–4200, 2023. 6.2

[177] Vadim Tschernezki, Iro Laina, Diane Larlus, and Andrea Vedaldi. Neural Feature Fusion Fields: 3D distillation of self-supervised 2D image representation. In *Proceedings of the International Conference on 3D Vision (3DV)*, 2022. 3.2, 3.3.2

[178] Haithem Turki, Deva Ramanan, and Mahadev Satyanarayanan. Mega-nerf: Scalable construction of large-scale nerfs for virtual fly-throughs. In *CVPR*, pages 12922–12931, June 2022. (document), 1.3, 1.3.1, 1.3.2, 1.3.3, 2.1, 3.3.2, 3.3.2, 5.1, 5.3.2

[179] Haithem Turki, Jason Y Zhang, Francesco Ferroni, and Deva Ramanan. SUDS: Scalable urban dynamic scenes. In *CVPR*, 2023. (document), 1.1, 1.4, 1.3.2, 1.3.3, 1.3.4, 3.1, 5.3.2

[180] Haithem Turki, Michael Zollhöfer, Christian Richardt, and Deva Ramanan. Pynerf: Pyramidal neural radiance fields. In *Thirty-Seventh Conference on Neural Information Processing Systems*, 2023. (document), 1.6, 1.3.4, 5.1, 5.1

[181] Haithem Turki, Vasu Agrawal, Samuel Rota Bulò, Lorenzo Porzi, Peter Kontschieder, Deva Ramanan, Michael Zollhöfer, and Christian Richardt. Hybridnerf: Efficient neural rendering via adaptive volumetric surfaces. In *CVPR*, 2024. (document), 1.5, 1.3.3, 1.3.4, 4.1

[182] Suhani Vora*, Noha Radwan*, Klaus Greff, Henning Meyer, Kyle Genova, Mehdi S. M. Sajjadi, Etienne Pot, Andrea Tagliasacchi, and Daniel Duckworth. Nesf: Neural semantic fields for generalizable semantic segmentation of 3d scenes. *Transactions on Machine Learning Research*, 2022. https://openreview.net/forum?id=ggPhsYCsm9. 3.2

[183] Angtian Wang, Peng Wang, Jian Sun, Adam Kortylewski, and Alan Yuille. VoGE: A differentiable volume renderer using Gaussian ellipsoids for analysis-by-synthesis. In *ICLR*, 2023. 4.2

[184] Jianyuan Wang, Nikita Karaev, Christian Rupprecht, and David Novotny. Vggsfm: Visual geometry grounded deep structure from motion. In *CVPR*, 2024. 6.2

[185] Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. *NeurIPS*, 2021. 1.3.3, 4.1, 4.2, 4.3.1, 4.3.2, 4.3.3

[186] Qianqian Wang, Zhicheng Wang, Kyle Genova, Pratul Srinivasan, Howard Zhou,

Jonathan T. Barron, Ricardo Martin-Brualla, Noah Snavely, and Thomas Funkhouser. Ibrnet: Learning multi-view image-based rendering. In *CVPR*, 2021. 2.2

[187] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13 (4):600–612, 2004. doi: 10.1109/TIP.2003.819861. 2.4.2, 3.4.1, 4.4.2, 5.4.1

[188] Zian Wang, Tianchang Shen, Merlin Nimier-David, Nicholas Sharp, Jun Gao, Alexander Keller, Sanja Fidler, Thomas Müller, and Zan Gojcic. Adaptive shells for efficient neural radiance field rendering. 42(6):259:1–15, 2023. doi: 10.1145/3618390. 4.2

[189] Zirui Wang, Shangzhe Wu, Weidi Xie, Min Chen, and Victor Adrian Prisacariu. NeRF$--$: Neural radiance fields without known camera parameters. *arXiv preprint arXiv:2102.07064*, 2021. 2.5.1, 6.2

[190] Wang, Shuzhe and Leroy, Vincent and Cabon, Yohann and Chidlovskii, Boris and Revaud Jerome. DUSt3R: Geometric 3D Vision Made Easy. In *CVPR*, 2024. 6.1.1, 6.1, 6.2

[191] William T. Weldon and Joseph Hupy. Investigating methods for integrating unmanned aerial systems in search and rescue operations. *Drones*, 4(3), 2020. ISSN 2504-446X. doi: 10.3390/drones4030038. URL `https://www.mdpi.com/2504-446X/4/3/38`. 2.1

[192] Lance Williams. Pyramidal parametrics. *Computer Graphics*, 17:1–11, July 1983. 1.3.4, 5.1

[193] Benjamin Wilson, William Qi, Tanmay Agarwal, John Lambert, Jagjeet Singh, Siddhesh Khandelwal, Bowen Pan, Ratnesh Kumar, Andrew Hartnett, Jhony Kaesemodel Pontes, Deva Ramanan, Peter Carr, and James Hays. Argoverse 2: Next generation datasets for self-driving perception and forecasting. In *NeurIPS Datasets and Benchmarks*, 2021. 5.4.5

[194] Rundi Wu, Ben Mildenhall, Philipp Henzler, Keunhong Park, Ruiqi Gao, Daniel Watson, Pratul P. Srinivasan, Dor Verbin, Jonathan T. Barron, Ben Poole, and Aleksander Holynski. Reconfusion: 3d reconstruction with diffusion priors. In *CVPR*, June 2024. 6.1, 6.1.2, 6.1.3

[195] Tianhao Wu, Fangcheng Zhong, Andrea Tagliasacchi, Forrester Cole, and Cengiz Oztireli. D^{2}nerf: Self-supervised decoupling of dynamic and static objects from a monocular video. In *NeurIPS*, 2022. 3.2, 3.3.2, 3.3.2, 3.3.3, 3.3.3

[196] Wenqi Xian, Jia-Bin Huang, Johannes Kopf, and Changil Kim. Space-time neural irradiance fields for free-viewpoint video. In *CVPR*, pages 9421–9431, 2021. 1.3.2, 3.2

[197] Yuanbo Xiangli, Linning Xu, Xingang Pan, Nanxuan Zhao, Anyi Rao, Christian Theobalt, Bo Dai, and Dahua Lin. Bungeenerf: Progressive neural radiance field for extreme multi-scale scene rendering. In *ECCV*, 2022. 1.2, 1.3.2, 2.2, 3.2, 5.2, 5.3.2

[198] Tianyi Xie, Zeshun Zong, Yuxing Qiu, Xuan Li, Yutao Feng, Yin Yang, and Chenfanfu Jiang. Physgaussian: Physics-integrated 3d gaussians for generative dynamics. In *CVPR*, 2024. 6.3

[199] Huang Xin, Zhang Qi, Feng Ying, Li Xiaoyu, Wang Xuan, and Wang Qing. Local implicit ray function for generalizable radiance field representation. In *CVPR*, 2023. 5.2

[200] Linning Xu, Vasu Agrawal, William Laney, Tony Garcia, Aayush Bansal, Changil Kim, Samuel Rota Bulò, Lorenzo Porzi, Peter Kontschieder, Aljaž Božič, Dahua Lin, Michael Zollhöfer, and Christian Richardt. VR-NeRF: High-fidelity virtualized walkable spaces. In *SIGGRAPH Asia*, 2023. 1.1, 1.3.3

[201] Linning Xu, Vasu Agrawal, William Laney, Tony Garcia, Aayush Bansal, Changil Kim, Samuel Rota Bulò, Lorenzo Porzi, Peter Kontschieder, Aljaž Božič, Dahua Lin, Michael Zollhöfer, and Christian Richardt. VR-NeRF: High-fidelity virtualized walkable spaces. 2023. doi: 10.1145/3610548.3618139. (document), 3, 4.3.1, 4.4, 4.1, 4.4.2, 4.4.2, 4.7, 4.4.2, 4.4.3, 4.3, 4.4.4, 4.5, 4.7

[202] Linning Xu, Yuanbo Xiangli, Sida Peng, Xingang Pan, Nanxuan Zhao, Christian Theobalt, Bo Dai, and Dahua Lin. Grid-guided neural radiance fields for large urban scenes. In *CVPR*, 2023. 1.3.1, 1.3.2, 5.3.2

[203] Bangbang Yang, Yinda Zhang, Yinghao Xu, Yijin Li, Han Zhou, Hujun Bao, Guofeng Zhang, and Zhaopeng Cui. Learning object-compositional neural radiance field for editable scene rendering. In *ICCV*, October 2021. 3.2

[204] Gengshan Yang, Minh Vo, Neverova Natalia, Deva Ramanan, Vedaldi Andrea, and Joo Hanbyul. Banmo: Building animatable 3d neural models from many casual videos. In *CVPR*, 2022. 3.2, 3.3.2

[205] Guo-Wei Yang, Wen-Yang Zhou, Hao-Yang Peng, Dun Liang, Tai-Jiang Mu, and Shi-Min Hu. Recursive-NeRF: An efficient and dynamically growing nerf. *arXiv preprint arXiv:2105.09103*, 2021. 2.3.1

[206] Jiawei Yang, Marco Pavone, and Yue Wang. Freenerf: Improving few-shot neural rendering with free frequency regularization. 2023. 6.1

[207] Ze Yang, Yun Chen, Jingkang Wang, Sivabalan Manivasagam, Wei-Chiu Ma, Anqi Joyce Yang, and Raquel Urtasun. Unisim: A neural closed-loop sensor simulator. In *CVPR*, 2023. 1.1

[208] Lior Yariv, Jiatao Gu, Yoni Kasten, and Yaron Lipman. Volume rendering of neural implicit surfaces. 2021. (document), 1.3.3, 4.2, 4.1, 4.1, 4.2, 4.3.1, 4.3.1, 4.3.2, 4.3.3, 4.1, 4.4.2, 4.3

[209] Lior Yariv, Peter Hedman, Christian Reiser, Dor Verbin, Pratul P. Srinivasan, Richard Szeliski, Jonathan T. Barron, and Ben Mildenhall. Bakedsdf: Meshing neural sdfs for real-time view synthesis. *arXiv*, 2023. (document), 1.3.3, 4.1, 4.2, 4.3.3, **??**, 4.6, 4.2

[210] Zongxin Ye, Wenyu Li, Sidun Liu, Peng Qiao, and Yong Dou. Absgs: Recovering fine details for 3d gaussian splatting, 2024. 6.3

[211] Lin Yen-Chen, Pete Florence, Jonathan T. Barron, Alberto Rodriguez, Phillip Isola, and Tsung-Yi Lin. iNeRF: Inverting neural radiance fields for pose estimation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021. 1.1

[212] Lin Yen-Chen, Pete Florence, Jonathan T. Barron, Tsung-Yi Lin, Alberto Rodriguez, and Phillip Isola. NeRF-Supervision: Learning dense object descriptors from neural radiance fields. In *IEEE Conference on Robotics and Automation (ICRA)*, 2022. 1.1

[213] Chandan Yeshwanth, Yueh-Cheng Liu, Matthias Nießner, and Angela Dai. ScanNet++: A high-fidelity dataset of 3D indoor scenes. 2023. (document), 4.4.3, 4.8, 4.4.3, 4.3, 4.4.4, 4.6

[214] Chandan Yeshwanth, Yueh-Cheng Liu, Matthias Nießner, and Angela Dai. ScanNet++ Toolkit. `https://github.com/scannetpp/scannetpp`, 2023. Accessed: 2023-11-01. 4.4.3

[215] A. Yu, V. Ye, M. Tancik, and A. Kanazawa. pixelnerf: Neural radiance fields from one or few images. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4576–4585, Los Alamitos, CA, USA, jun 2021. IEEE Computer Society. doi: 10.1109/CVPR46437.2021.00455. URL `https://doi.ieeecomputersociety.org/10.1109/CVPR46437.2021.00455`. 2.2, 6.1, 6.1.2, **??**, 6.2

[216] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. PlenOctrees for real-time rendering of neural radiance fields. In *ICCV*, 2021. 1.2, 1.3.3, 2.2, 2.2, 2.3.3, 2.4, 2.4.3, 4.2, 5.2

[217] Hong-Xing Yu, Leonidas J. Guibas, and Jiajun Wu. Unsupervised discovery of object radiance fields. In *International Conference on Learning Representations*, 2022. 3.2

[218] Zehao Yu, Anpei Chen, Bozidar Antic, Songyou Peng, Apratim Bhattacharyya, Michael Niemeyer, Siyu Tang, Torsten Sattler, and Andreas Geiger. SDFStudio: A unified framework for surface reconstruction, 2022. URL `https://github.com/autonomousvision/sdfstudio`. 4.5.1

[219] Wentao Yuan, Zhaoyang Lv, Tanner Schmidt, and Steven Lovegrove. Star: Self-supervised tracking and reconstruction of rigid objects in motion with neural rendering. In *CVPR*, pages 13144–13152, 2021. 3.2

[220] Kaan Yücer, Alexander Sorkine-Hornung, Oliver Wang, and Olga Sorkine-Hornung. Efficient 3D object segmentation from densely sampled light fields with applications to 3D reconstruction. *ACM Transactions on Graphics*, 35(3), 2016. **??**

[221] Jason Y. Zhang, Deva Ramanan, and Shubham Tulsiani. RelPose: Predicting probabilistic relative rotation for single objects in the wild. In *European Conference on Computer Vision (ECCV)*, 2022. 6.2

[222] Jason Y Zhang, Amy Lin, Moneish Kumar, Tzu-Hsuan Yang, Deva Ramanan, and Shubham Tulsiani. Cameras as rays: Pose estimation via ray diffusion. In *International Conference on Learning Representations (ICLR)*, 2024. 6.2

[223] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. Nerf++: Analyzing and improving neural radiance fields. *arXiv:2010.07492*, 2020. (document), 2.2, 2.2, 2.3.1, **??**, **??**, 2.3, 2.4.2, 3.2, 3.3.2, 4.3.3, 4.2, 5.2

[224] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. 2018. doi: 10.1109/CVPR.2018.00068. 6.1.3

[225] 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. 2.4.2, 3.4.1, 4.4.2, 5.4.1

[226] Yuqi Zhang, Guanying Chen, and Shuguang Cui. Efficient large-scale scene representation with a hybrid of high-resolution grid and plane features, 2023. 1.3.1

[227] Shuaifeng Zhi, Tristan Laidlow, Stefan Leutenegger, and Andrew Davison. In-place scene labelling and understanding with implicit scene representation. In *ICCV*, 2021. 3.2

[228] Zhizhuo Zhou and Shubham Tulsiani. Sparsefusion: Distilling view-conditioned diffusion for 3d reconstruction. In *CVPR*, 2023. 6.1