Towards City-Scale Neural Rendering

Haithem Turki
hturki@cs.cmu.edu

September 2023

Thesis Proposal

School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Thesis Committee

Deva Ramanan	Carnegie Mellon University (*chair*)
Shubham Tulsiani	Carnegie Mellon University
Jessica K. Hodgins	Carnegie Mellon University
Angjoo Kanazawa	University of California, Berkeley
Jonathan T. Barron	Google Research

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

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, from search and rescue to autonomous driving. Large-scale virtual reality, long the domain of science fiction, feels markedly more tangible.

This proposal aims to advance the frontier of large-scale neural rendering by building upon Neural Radiance Fields (NeRFs) [64], a family of methods attracting attention due to their state-of-the-art rendering quality and conceptual simplicity. As of July 2023, at least 3,000 papers have been proposed by research groups across the world across numerous use cases [75]. However, numerous 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 scenes, which limits their applicability. Another is quality, as NeRF assumes ideal viewpoint conditions that are unrealistic in practice and degrades when they are violated. Renderings are especially poor in under-observed regions. This is problematic for dynamic city-scale scenes where it is impossible to densely sample every location and time step. Speed is a third issue, as rendering falls below interactive thresholds. Current acceleration methods remain too slow or degrade quality at high resolution.

To address scaling, 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). We then extend our approach to build the largest dynamic NeRF representation to date. As a first step towards improving quality, we propose an anti-aliasing method with minimal performance overhead. To accelerate rendering, we improve sampling efficiency through a hybrid surface-volumetric approach 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 cannot capture). We finally propose to further improve quality in underobserved regions through diffusion models, which show promising results on single-object reconstructions.

Contents

Chapter 1

Introduction

The goal of this thesis is to develop the foundations needed for large-scale neural rendering. Given a set of images, camera poses, and auxiliary information (LiDAR, optical flow...), how do we efficiently train a high-quality representation? We evaluate possible solutions along three principal axes:

Scale. What is the scale, in terms of geographic coverage and pixel count, that our representations can capture? How do training and compute requirements evolve as we increase along those dimensions?

Quality. How realistic are visuals that our model generates? Is quality uniformly high within the rendered scene, or does it degrade in certain areas?

Speed. Once trained, how fast is our model to render?

1.1 Overview of Pre-Thesis Research

1.1.1 Scale

Chapter 2 (Mega-NeRF) explores how to scale NeRFs to larger settings. The original NeRF encodes the entire scene representation into a monolithic MLP. Increasing the model's capacity, which is needed to faithfully reconstruct larger-scale scenes, increases training and rendering time quadratically. We propose a spatial model decomposition that instead scales linearly with model capacity and use it to train representations far larger than prior work.

However, Mega-NeRF only handles static scenes, which limits its usefulness. Chapter 3 (SUDS) extends the previous chapter's methodology to target dynamic scenes. It proposes an efficient three-branch hash table representation to build a city-scale NeRF of Pittsburgh from 1.3 million frames across 1,700 videos, (to our knowledge) the largest dynamic NeRF built to date.

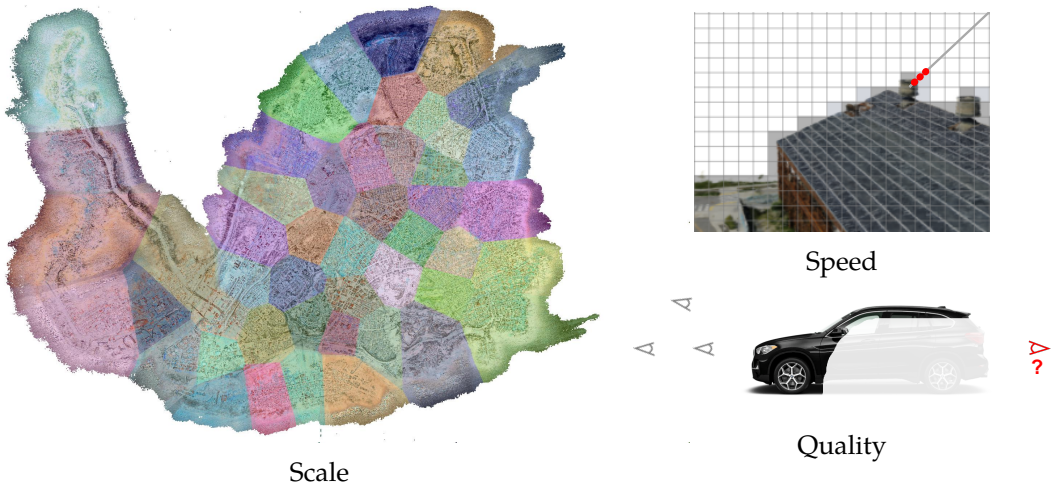


Figure 1.1: Our goal is to train city-scale representations in a scalable manner (**left**) that generate high-quality renderings from a wide range of camera trajectories (**bottom right**). Once trained, we aim to visualize our representation at interactive frame rates (**top right**).

1.1.2 Quality

Although Chapters 2 and 3 address how to scale NeRFs across large spatial footprints, rendering quality (especially for moving objects) remains significantly below photo-realistic thresholds. We begin addressing this in Chapter 4 by proposing a method (PyNeRF) that improves NeRF's performance with when trained against more freeform camera trajectories (such as when objects are viewed from difference distances). As our goal is to enable large-scale rendering, our method must incur as small a performance overhead as possible. Similar to the previous chapters, we do so through model decomposition but now partition across rendering scales instead of spatial footprint.

1.2 Proposed Work

1.2.1 Speed

Chapter 5 details our proposal to accelerate NeRF raycasting to the framerates needed to for high-resolution interactive rendering while maintaining NeRF's desirable continuous volumetric properties (which make it easy to store and simple to train). We wish to train a representation that encourages the model to represent as much of the world as possible through surfaces while maintaining the freedom to render transparency and finer details if needed. Once trained, we explore how to query the model as efficiently as possible.

1.2.2 Quality

We revisit quality in Chapter 6. The city-scale NeRF described in Chapter 3 generates plausible renderings near camera viewpoints close to training poses but degrades when rendering under-observed areas. Dynamic objects, such as cars which are captured from only a few viewpoints, fare especially poorly. We propose to extend recent efforts that use generative models to improve quality in under-observed regions and adapt them to our setting.

Chapter 2

Mega-NeRF: Scalable Construction of Large-Scale NeRFs for Virtual Fly-Throughs

2.1 Introduction

We first explore the scalability of NeRFs. The vast majority of existing methods explore single-object scenes, often captured indoors or from synthetic data. To our knowledge, Tanks and Temples [50] 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 [25], collected footage is typically reconstructed into 2D "birds-eye-view" maps that support post-hoc analysis [108]. 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 [1], "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

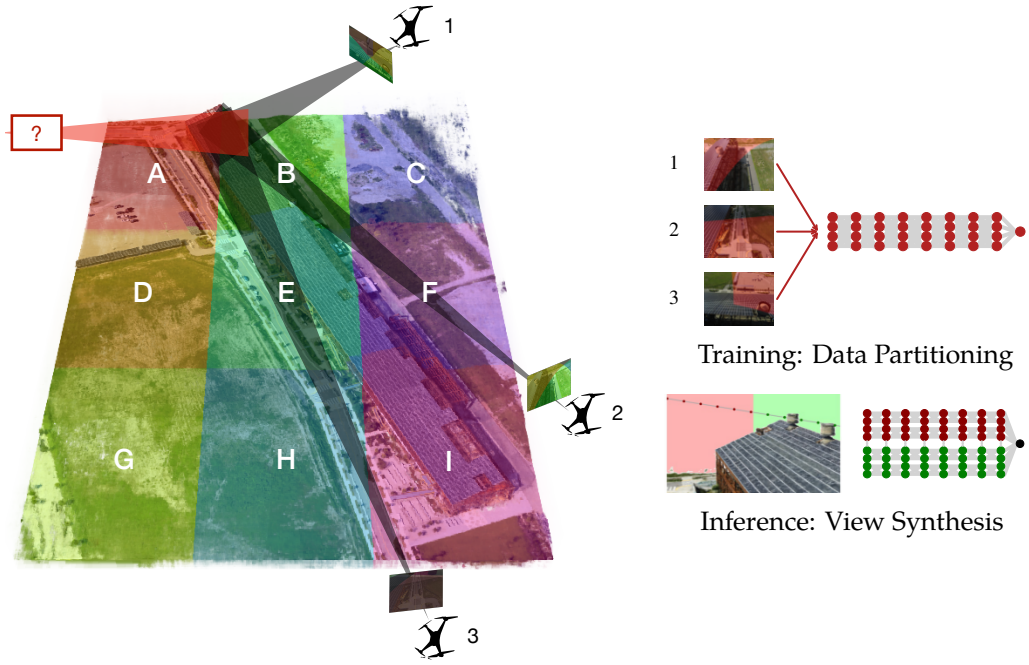


Figure 2.1: We scale neural reconstructions to massive urban scenes 1000x larger than prior work. To do so, Mega-NeRF decomposes a scene into a set of spatial cells (**left**), learning a separate NeRF submodule for each. We train each submodule with geometry-aware pixel-data partitioning, making use 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 10x. 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).

(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

	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.

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 [76] and KiloNeRF [78], 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 improve-

	Resolution	# Images	# Pixels/Rays
Synthetic NeRF [64]	400 x 400	400	256,000,000
LLFF [63]	4032 x 3024	41	496,419,840
Light Field [125]	1280 x 720	214	195,910,200
Tanks and Temples [50]	1920 x 1080	283	587,658,240
Phototourism [46]	919 x 794	1708	1,149,113,846
Mill 19	4608 x 3456	1809	28,808,773,632
Quad 6k [20]	1708 x 1329	5147	11,574,265,679
UrbanScene3D [58]	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 [126] and [122] respectively. For Phototourism we average across the scenes used in [61].

ments at training and 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 3x 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 [122], SNeRG [41], and FastNeRF [36] 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 [76] decomposes the scene into multiple cells via spatial Voronoi partitioning. Each cell is independently rendered using a smaller MLP, accelerating ren-

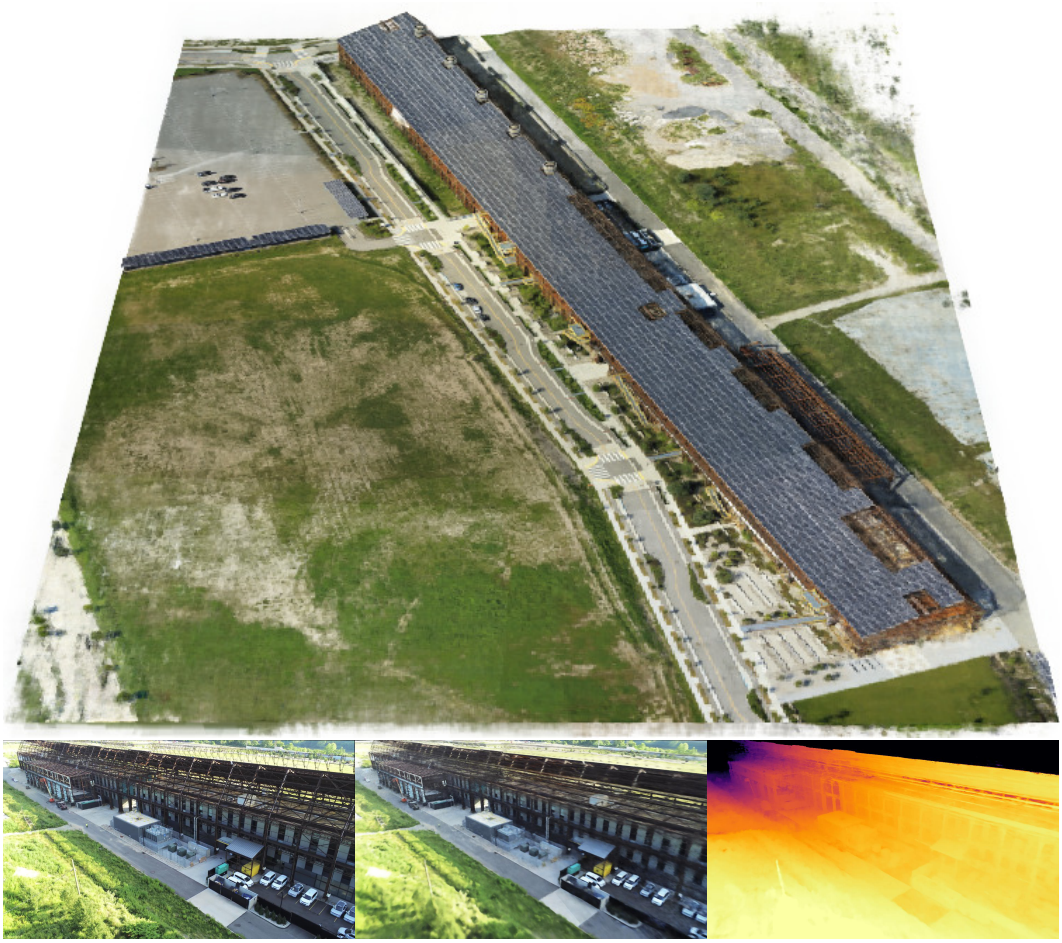


Figure 2.2: Visualization of Mill 19 by Mega-NeRF. The top panel shows a high-level 3D rendering of Mill 19 within our interactive visualizer. The bottom-left panel contains a ground truth image captured by our drone. The following two panels illustrate the model reconstruction along with the associated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

dering by 3x over NeRF. KiloNeRF [78] 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 2x for KiloNeRF. Training on all available data is prohibitive at our scale. Instead, our crucial insight is to geometrically partition training pixels into small data shards relevant for each submodule, which is essential for efficient training and high accuracy.

DONeRF [67] 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++ [126] 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 [61] 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 [46] dataset. We adopt similar appearance embeddings for Mega-NeRF and quantify its impact in Sec. 2.4.2.

Concurrent to us, Urban Radiance Fields [80] (URF), BungeeNeRF [113], and BlockNeRF [91] 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 BlockNeRF [91], 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. 3.1).

Training speed. Several works speed up model training by incorporating priors learned from similar datasets. PixelNeRF [121], IBRNet [105], and GRF [98] condition NeRF on predicted image features while Tancik et al. [92] 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 [95] subdivide a scene into cells to filter out irrelevant geometry and speed up rendering. Funkhouser and Séquin [30] 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," [6] 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) [64]. 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 $\mathbf{r}$ 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 [64] 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 [61], 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 $\mathbf{x}$, direction $\mathbf{d}$, and appearance embedding $l^{(a)}$ using the model weights $f^{\mathbf{n}}$ closest to the query point:

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

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

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

Centroid selection. Although we explored several methods, including k-means clustering and uncertainty-based partitioning as in [118], 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

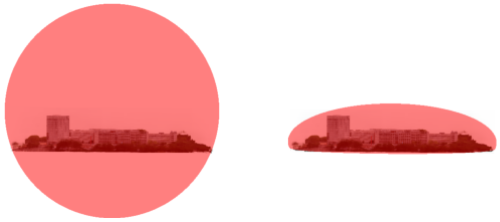


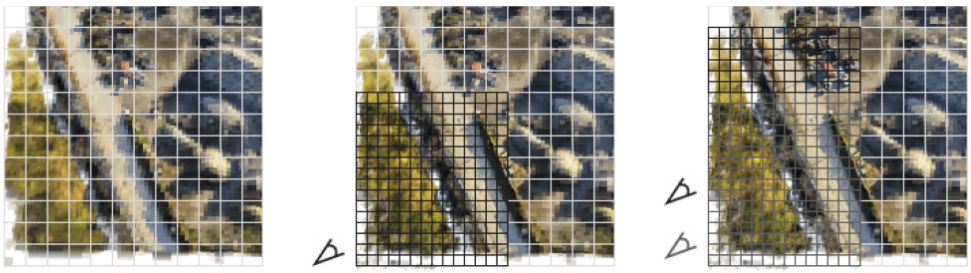
Figure 2.3: **Ray Bounds.** NeRF++ (**left**) samples within a unit sphere centered within and enclosing all camera poses to render its foreground component and uses a different methodology for the outer volume complement to efficiently render the background. Mega-NeRF (**right**) uses a similar background 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.

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++ [126], 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.

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. 3.1). In our experiments, this visibility partitioning reduces the size of each submodule's trainset by 10x 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



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

Figure 2.4: **Mega-NeRF-Dynamic.** Current renderers (such as Plenoctree [122]) 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.

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. 3.1, 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 2x. We provide details in Sec. 4.4.6.

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 [122] precomputes a cache of opacity and spherical harmonic coefficients into a sparse voxel 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 tem-

poral 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.

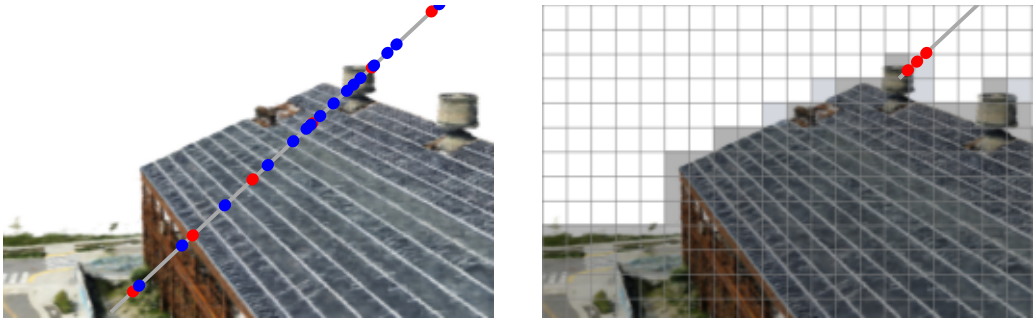
2.4 Experiments

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?

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 $500 \times 250\ m^2$ 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 [20], a large-scale Structure-from-Motion dataset col-



Standard Hierachical Sampling Guided Sampling

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.

lected within the Cornell Universty Arts Quad, and several scenes from UrbanScene3D [58] 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 [59]. We use a pretrained semantic segmentation model [26] 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 mixed-precision training to further accelerate the process. We sample 1024 rays per batch and use the Adam optimizer [49] with an initial learning rate of 5×10^{-4} decaying exponentially to 5×10^{-5}. We employ the procedure described in [61] to finetune Mega-NeRF's appearance embeddings.

2.4.2 Scalable training

Baselines. We evaluate Mega-NeRF against the original NeRF [64] architecture and NeRF++ [126]. We also evaluate our approach against Stable View Synthesis [81], an implementation of DeepView [27], and dense reconstructions from COLMAP [87], a traditional Multi-View Stereo approach, as non-neural radiance field-based alternatives.

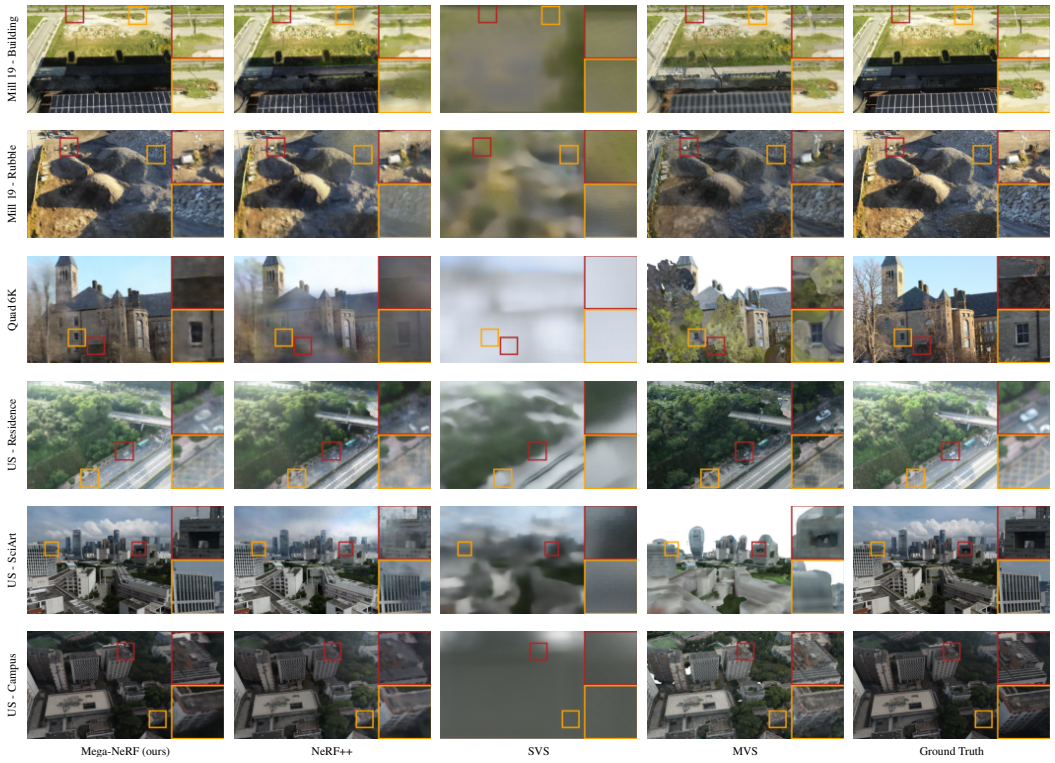


Figure 2.6: **Scalable training.** Mega-NeRF generates the best reconstructions while avoiding the artifacts present in the other approaches.

	Mill 19 - Building				Mill 19 - Rubble				Quad 6k			
	↑PSNR	↑SSIM	↓LPIPS	↓Time (h)	↑PSNR	↑SSIM	↓LPIPS	↓Time(h)	↑PSNR	↑SSIM	↓LPIPS	↓Time(h)
NeRF	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++	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	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	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	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	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++	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	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	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	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, NeRF++, Stable View Synthesis (SVS), DeepView, and Multi-View Stereo (MVS) after running each method to completion. Mega-NeRF consistently outperforms the baselines even after allowing other approaches to train well beyond 24 hours.

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 [106], and the VGG implementation of LPIPS [127]. 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 and KiloNeRF, in 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 [85] 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

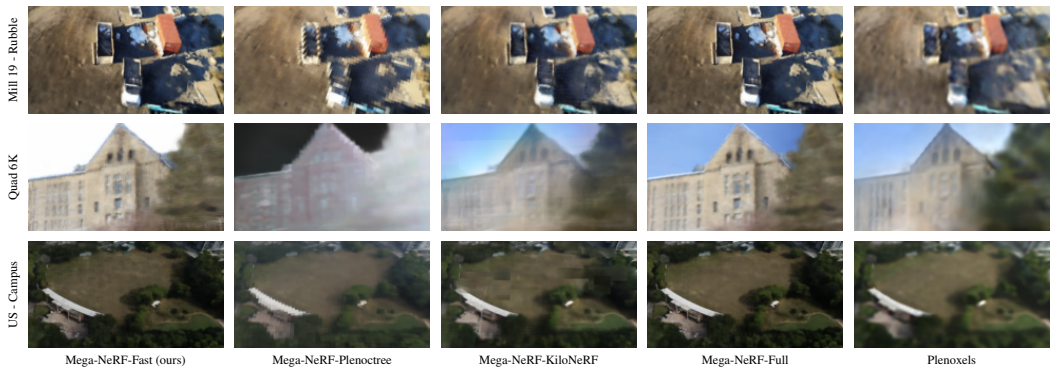


Figure 2.7: **Interactive rendering.** Plenoctree's approach causes significant voxelization and Plenoxel's renderings are blurry. KiloNeRF's results are crisper but capture less detail than Mega-NeRF-Dynamic and contain numerous visual artifacts.

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	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 40x relative to conventional ray sampling.

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 40x speedup. Mega-NeRF-Plenoctree and Mega-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 chan-

	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).

nels 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).

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

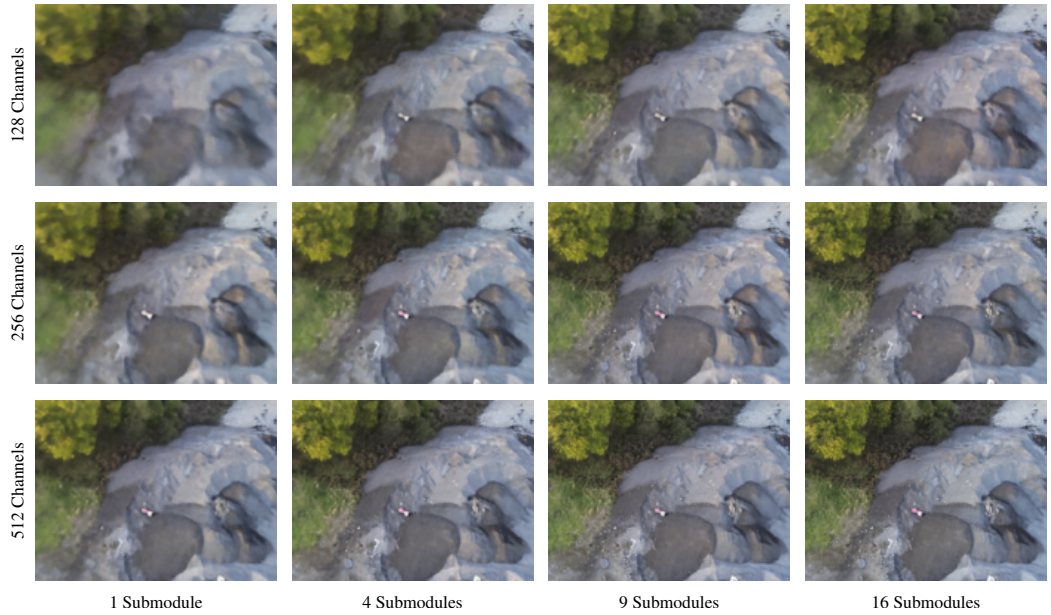


Figure 2.8: **Model scaling.** Example rendering within our Mill 19 - Rubble dataset across different numbers of submodules (**columns**) and channels per submodule (**rows**). Mega-NeRF generates increasingly photo-realistic renderings as capacity increases. Increasing the number of submodules increases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

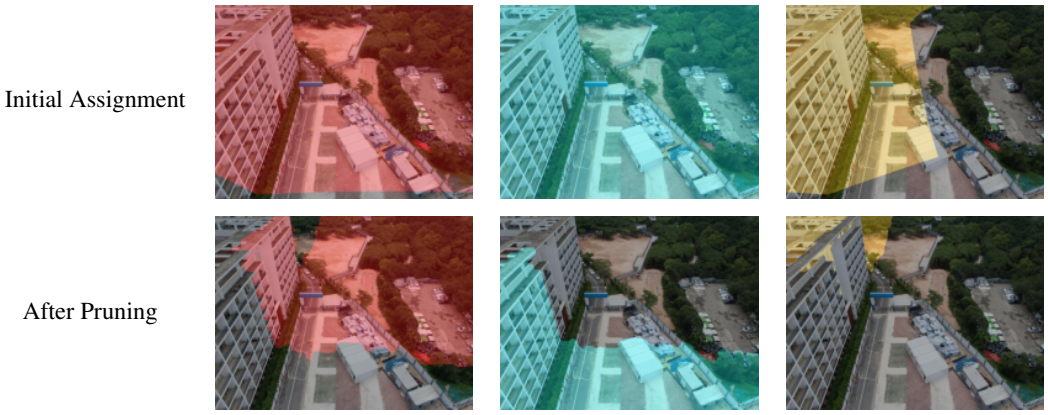


Figure 2.9: **Data pruning.** The initial assignment of pixels to cells is based purely on camera positions. We add each pixel to the training set of **all** cells it traverses, leading to overlap between sets (**top**). After the model gains a 3D understanding of the scene, we can filter irrelevant 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 2x while increasing accuracy for a fixed number of 500,000 iterations.

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

submodule-specific data partitions. We 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 10x relative to fully-connected MLPs and is thus 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 3x while significantly improving reconstruction quality. Our empirical evaluation of existing fast renderers on top of Mega-NeRF suggests that interactive NeRF-based rendering at scale remains an open research question. 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 human-centered use cases. Several NeRF-related efforts, including NR-NeRF [97], Nerfies [71], NeRFlow [24], and DynamicMVS [33] 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 propose alternate solutions in Chapter 5.

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. Another option is to introduce conditional priors to avoid needing to train each scene from scratch. We explore how to best incorporate these in Chapter 6.

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 multiple efforts [57, 107, 44, 62, 19] 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.

Scale. Mega-NeRF explicitly targets urban-scale environments instead of smaller single-object settings. Our tests against scenes from the Synthetic-NeRF dataset suggests our ray bound strategy and per-image appearance embeddings do not harm quality but that our spatial partitioning strategy reduces PSNR by about 1 db relative to NeRF.

2.5.2 Societal impact

The capture of drone footage brings with it the possibility of inadvertently and accidentally capturing privacy-sensitive information such as people's faces and vehicle license plate numbers. Furthermore, what is considered sensitive and not can vary widely depending on the context.

One solution is the technique of "denaturing" first described by Wang et al [103] that allows for fine-grain policy guided removal of sensitive pixels at interactive frame rates. As denaturing can be done at full frame rate, preprocessing should not slow down training, although it is unclear what the impact of the altered pixels would have on the resulting model. We leave this to future work.

Chapter 3

SUDS: Scalable Urban Dynamic Scenes

3.1 Introduction

Scalable geometric reconstructions of cities have transformed our daily lives, with tools such as Google Maps and Streetview [9] becoming fundamental to how we navigate and interact with our environments. A watershed moment in the development of such technology was the ability to scale structure-from-motion (SfM) algorithms to city-scale footprints [7]. Since then, the advent of Neural Radiance Fields (NeRFs) [64] has transformed this domain by allowing for photorealistic interaction with a reconstructed scene via view synthesis.

The method described in Chapter 2 can be used to scale such representations to neighborhood-scale reconstructions for virtual drive-throughs 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.

Challenges. One possible solution is a dynamic NeRF that conditions on time or warps a canonical space with a time-dependent deformation [71]. 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 [35]. 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 sig-

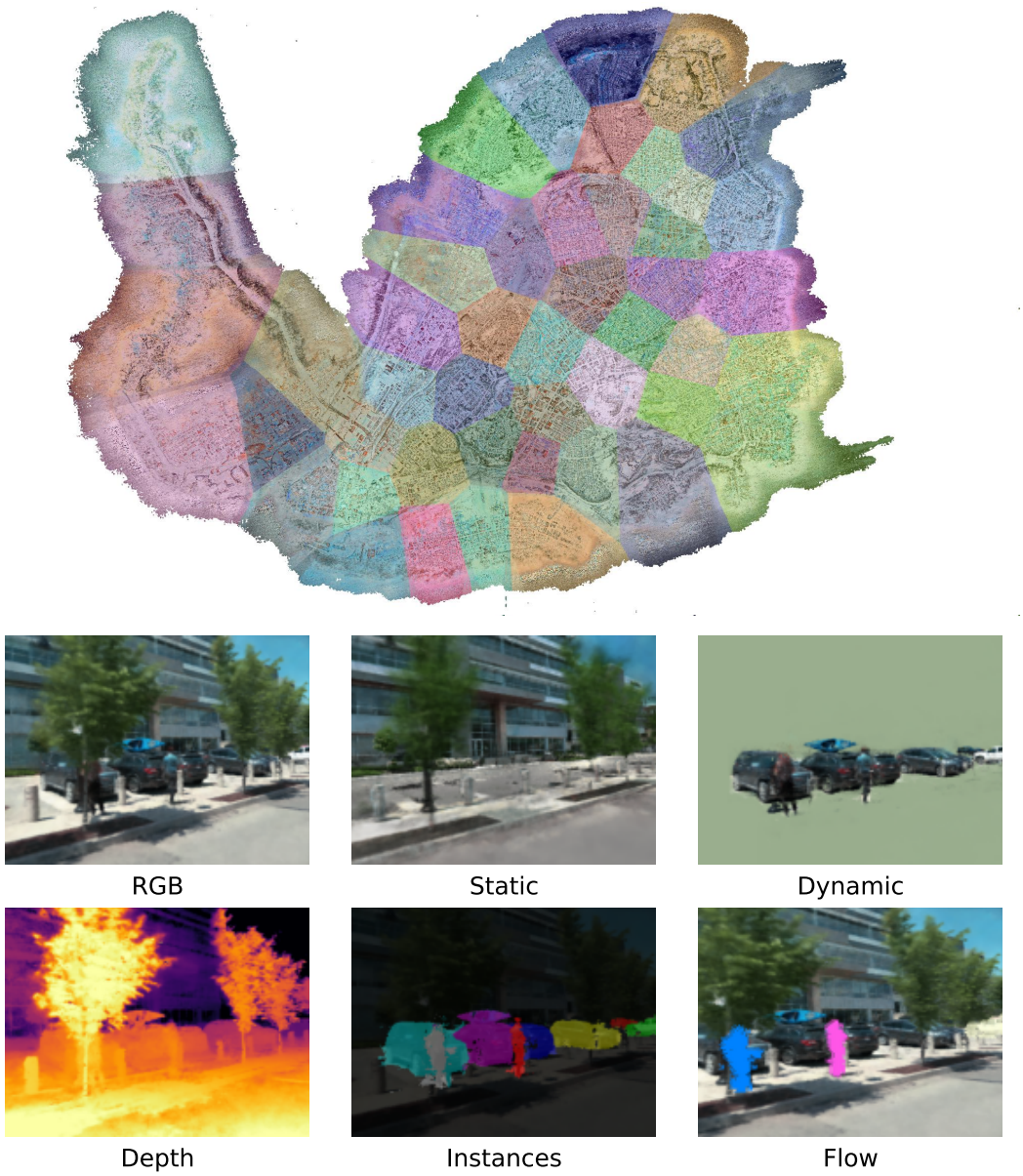
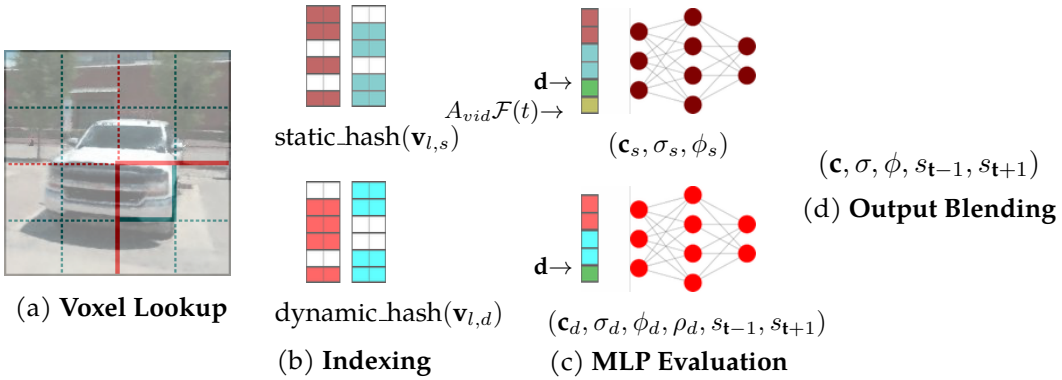


Figure 3.1: **SUDS.** We scale neural reconstructions to city scale by dividing the area into multiple cells and training hash table representations for each. We show our full city-scale reconstruction **above** and the derived representations **below.** Unlike prior methods, our approach handles dynamism 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.



(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.

nals, such as LiDAR depth measurements and optical flow. Other dynamic scene representations [52, 70] 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++ [126] and mip-NeRF 360 [11] 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 [113] introduced a coarse-to-fine approach that progressively adds more capacity to the network representation. Mega-NeRF (Chapter 2) and Block-NeRF [91] partition the scene spatially and train separate NeRFs for each partition. To model appearance variation, they incorporate per-image embeddings like NeRF-W [61]. 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 [55] and Space-time Neural Irradiance Fields [112] add time as an input to handle dynamic scenes. Similar to our work, NSFF [56], NeRFlow [24], and DyNeRF [34] incorporate 2D optical flow input and warping-based regularization losses to enforce plausible transitions between observed frames. Multiple methods [71, 74, 97, 72] instead disentangle scenes into a canonical template and per-frame deformation field. BANMo [117] 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 [55] and [117] use single video sequences.

While many of the previous works use segmentation data to factorize dynamic from static objects, D^2NeRF [111] does this automatically through regularization and explicitly handling shadows. Neural Groundplans [88] 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 [69, 70, 45, 116, 123, 124] represent scenes as the composition of per-object NeRF models and a background model. NSG [70] 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 [128] adds an extra head to NeRF that predicts extra semantic category logits for any 3D position. Panoptic-NeRF [29] and Panoptic Neural Fields [52] extend this to produce panoptic segmentations and the latter uses a similar bounding-box based object and background decomposition as

NSG. NeSF [102] 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 [51, 99] 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 [86] and DVGO [90] directly optimize a voxel representation instead of an MLP to train in minutes or even seconds. TensoRF [16] stores its representation as the outer product of low-rank tensors, reducing memory usage. Instant-NGP [66] 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 [23] and Dense Depth Priors [82] incorporate noisy point clouds obtained by structure from motion (SfM) in the loss function during optimization. Urban Radiance Fields [80] 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 [14]) 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 [64], 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 [64] 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 [34, 56, 111], 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 [126, 100, 80].

However, conventional NeRF training with MLPs is computationally prohibitive at our target scales. Inspired by Instant-NGP [66], 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. 4.1). 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 [66], 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 [117]):

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

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

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 [111]):

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

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

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

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 $\mathbf{d}$ [80, 40] and a video id **vid**.

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:

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

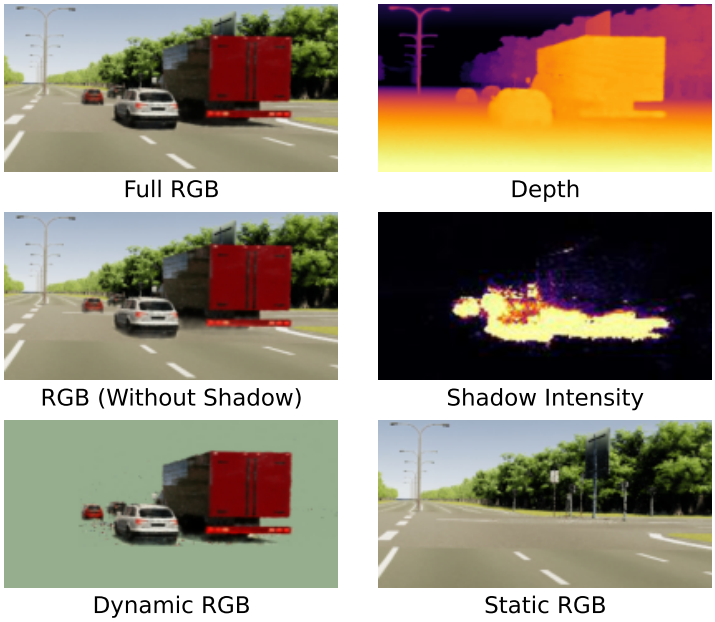
$$\begin{aligned}\mathbf{c}(\mathbf{x}, \mathbf{t}, \mathbf{vid}, \mathbf{d}) = &\frac{\sigma_s}{\sigma}(1 - \rho_d)\mathbf{c}_s(\mathbf{x}, \mathbf{d}, A_{vid}\mathcal{F}(t)) \\ &+ \frac{\sigma_d}{\sigma}\mathbf{c}_d(\mathbf{x}, \mathbf{t}, \mathbf{vid}, \mathbf{d})\end{aligned} \tag{3.10}$$

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 **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:

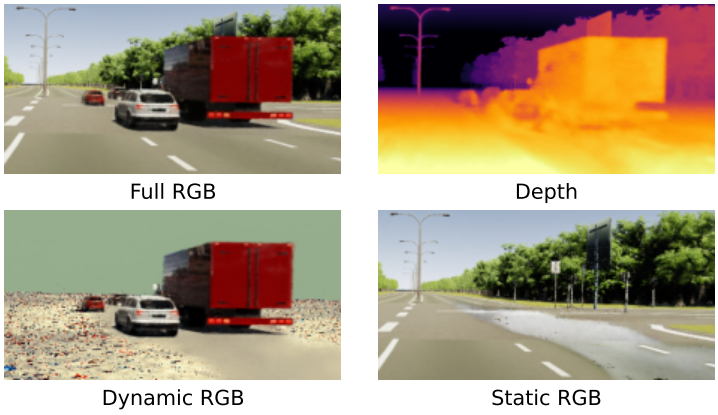
$$\begin{aligned}\hat{C}(\mathbf{r}, \mathbf{t}, \mathbf{vid}) = &\int_0^{+\infty} T(t)\sigma(\mathbf{r}(t), \mathbf{t}, \mathbf{vid})\mathbf{c}(\mathbf{r}(t), \mathbf{t}, \mathbf{vid}, \mathbf{d})dt \\ &+ T(+\infty)\mathbf{c}_e(\mathbf{d}, \mathbf{vid}),\end{aligned} \tag{3.11}$$

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

Feature distillation. We build semantic awareness into SUDS to enable the open-world tasks described in Sec. 3.4.2. Similar to recent work [51, 99], we distill the outputs of a self-supervised 2D feature extractor, namely DINO [14], 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



(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**).

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 [56, 24, 34], 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 4.5).

Spatial partitioning. We scale our representation to arbitrarily large environments by decomposing the scene into individually trained models [100, 91], each with its own static, dynamic, and far-field branch. Intuitively, the reconstruction for neighborhood X can be done largely independantly 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 [31]. Mega-NeRF includes only those datapoints whose associated camera rays intersect the spatial cell. 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

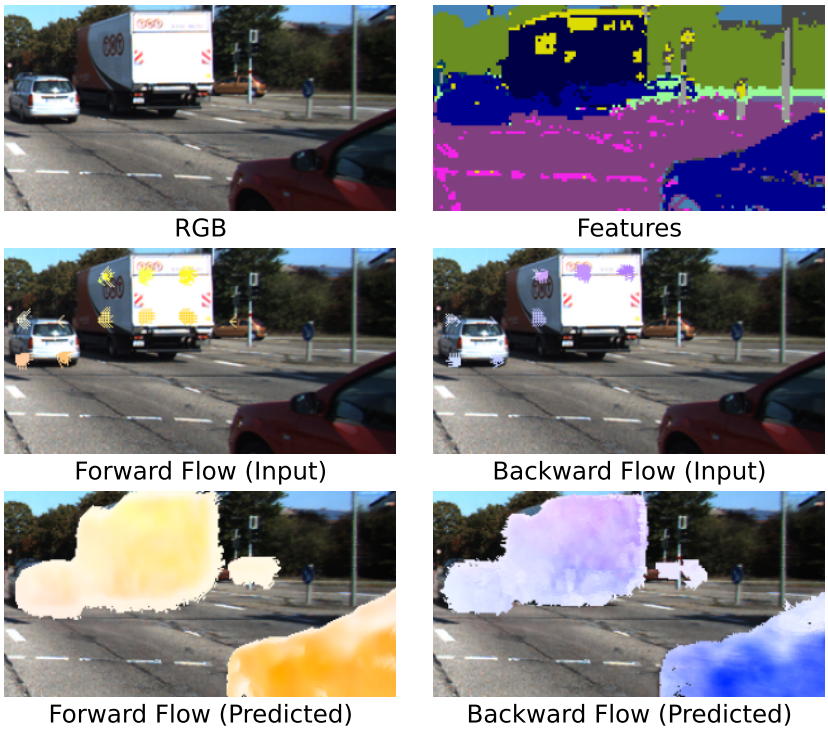


Figure 3.4: **Scene Flow.** We minimize the photometric and feature-metric loss of warped renderings relative to ground truth inputs (**top**). We use 2D optical flow from off-the-shelf estimators or sparse correspondences computed directly from 2D DINO features [8] (**middle**) to supervise our flow predictions (**bottom**).

minimizing the following loss:

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

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 [64]. 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

$D(\mathbf{r})$ [23, 80]:

$$\mathcal{L}_d(\mathbf{r}) = \left\| D(\mathbf{r}) - \hat{D}(\mathbf{r}) \right\|^2 \tag{3.17}$$

$$\text{where } \hat{D}(\mathbf{r}) = \int_0^{+\infty} T(s)\sigma(\mathbf{r}(s))ds \tag{3.18}$$

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:

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

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:

$$\sigma^w_{t'}(\mathbf{x} + s_{t'}, \mathbf{t} + t', \mathbf{vid}) \in \mathbb{R} \tag{3.20}$$

$$\mathbf{c}^w_{t'}(\mathbf{x} + s_{t'}, \mathbf{t} + t', \mathbf{vid}, \mathbf{d}) \in \mathbb{R}^3 \tag{3.21}$$

$$\Phi^w_{t'}(\mathbf{x} + s_{t'}, \mathbf{t} + t', \mathbf{vid}) \in \mathbb{R}^C \tag{3.22}$$

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 [56, 34]. As in NSFF [56], 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 [2] 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:

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

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

Flow regularization. As in prior work [56, 34] 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 [56]:

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

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 [111]:

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

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 [111].

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

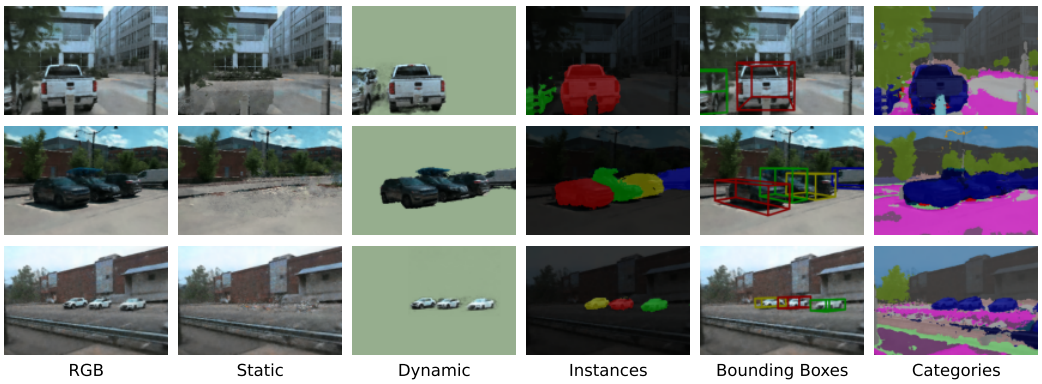


Figure 3.5: **City-1M.** We demonstrate SUDS's capabilities on multiple downstream tasks, including instance segmentation and 3D bounding box estimation without any labeled data (by just making use of geometric clustering). In the last column, we show category-level semantic classification by matching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

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. 4.4.6.

3.4.1 Experimental Setup

2D feature extraction. We use Amir et al's feature extractor implementation [8] 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 [4].

Flow supervision. We explored using an estimator trained on synthetic data [94] in addition to directly computing 2D correspondences from DINO itself [8]. 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 [11]. We use Adam [49] with a learning rate of 5×10^{-3} decaying to 5×10^{-4}.

Metrics. We report quantitative results based on PSNR, SSIM [106], and the AlexNet implementation of LPIPS [127].

	Mega-NeRF (Chapter 2)	Mega-NeRF-T	Mega-NeRF-A	SUDS
PSNR ↑	16.42	16.46	16.70	**21.67**
SSIM ↑	0.493	0.493	0.493	**0.562**
LPIPS ↓	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.

	≤ 15k	15-30k	30-45k	≥ 45k
↑PSNR	22.86	21.99	21.35	20.75
↑SSIM	0.583	0.569	0.557	0.538
↓LPIPS	0.516	0.545	0.564	0.578

Images

	≤ 60	60-90	90-120	≥ 120
↑PSNR	22.47	21.72	21.68	21.11
↑SSIM	0.587	0.556	0.559	0.555
↓LPIPS	0.526	0.557	0.557	0.565

Videos

	≤ 2 km^2	2-3 km^2	3-4 km^2	≥ 4 km^2
↑PSNR	22.73	21.47	21.53	22.18
↑SSIM	0.609	0.556	0.561	0.557
↓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 sub-linearly 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 [94] 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.

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 [70] renders ghosting artifacts near areas of movement. Our method forecasts plausible trajectories and generates higher-quality renderings.

Instance segmentation. We derive the instance count as in prior work [88] by sampling dynamic density values σ_d, projecting those above a given threshold onto a discretized ground 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 [3].

Tracking. We can compute mask and keypoint-level correspondences across frames after detecting instances (Sec. 3.4.2) by using Best-Buddies similarity [22] 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

	KITTI - 75%			KITTI - 50%			KITTI - 25%		
	↑PSNR	↑SSIM	↓LPIPS	↑PSNR	↑SSIM	↓LPIPS	↑PSNR	↑SSIM	↓LPIPS
NeRF [64]	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 [70]	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 [64]	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 [70]	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 [89]	NeRF [64]	NeRF + Time	NSG [70]	PNF [52]	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 [70, 52].

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.

3.4.3 KITTI Benchmarks

Baselines. We compare SUDS to SRN [89], the original NeRF implementation [64], a variant of NeRF taking time as an additional input, NSG [70], and PNF [52]. 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 [37] subsequences and experimental setup as prior work [70, 52]. 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.

	↑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.

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 [32] 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.

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 4.5. 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 training and rendering speeds, while faster than Mega-NeRF (Chapter 2), are still prohibitively expensive. We address these shortcomings in the next chapters. 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 preprocessing, 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.

3.5.2 Societal Impact

As SUDS attempts to model dynamic urban scenes with pedestrians and vehicles, our approach carries surveillance and privacy concerns related to the intentional or inadvertent capture or privacy-sensitive information such as human faces and vehicle license plate numbers. As we distill semantic knowledge into SUDS, we

are able to (imperfectly) filter out either entire categories (people) or components (faces) at render time. However this information would still reside in the model itself. This could in turn be mitigated by preprocessing the input data used to train the model.

Chapter 4

PyNeRF: Pyramidal Neural Radiance Fields

4.1 Introduction

After discussing how to scale neural representations in Chapters 2 and 3, we now turn our attention to quality. Although NeRF provides state-of-the-art rendering quality, it assumes invariants that are often infeasible in real-world scenarios. As an example that we focus on in this chapter, it assumes that scene content is equidistant from the camera. Rendering quality degrades due to aliasing and excessive blurring when that assumption is violated. As an additional constraint relevant to city-scale rendering, we want quality improvements to incur as little overhead as possible, and be compatible with modern NeRF acceleration techniques such as the grid-based representation described in Chapter 3.

Solutions such as Mip-NeRF [10] address aliasing by projecting camera frustum volumes instead of point-sampling rays. However, these anti-aliasing methods rely on the base NeRF MLP representation (and are thus slow) and are incompatible with grid representations due to their reliance on non-grid-based inputs.

Inspired by divide-and-conquer NeRF extensions [77, 78, 100, 91] and classical approaches such as Gaussian pyramids [5] and mipmaps [109], we propose a simple approach 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 [21]). 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 to any existing fast-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

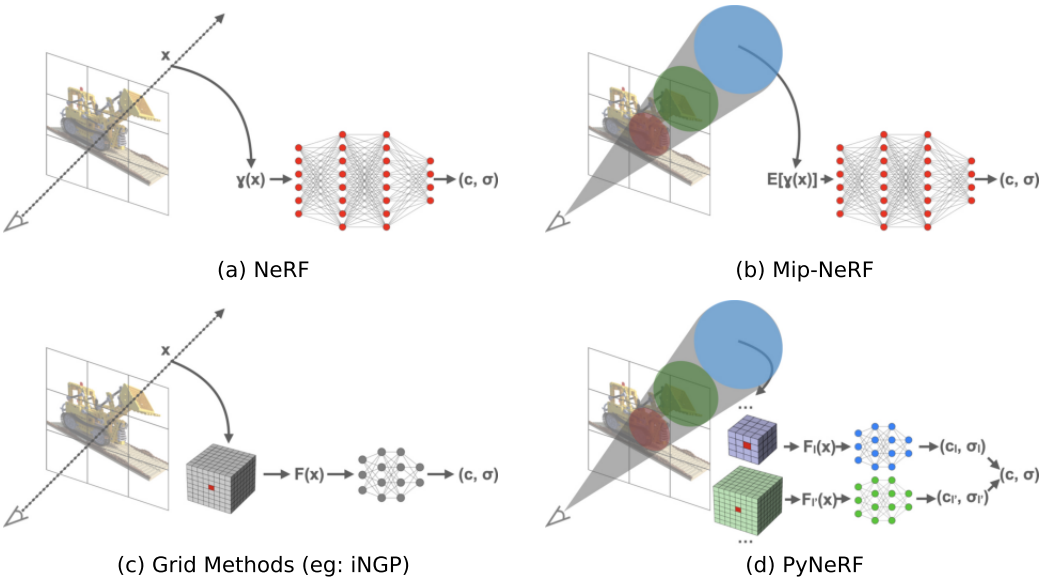


Figure 4.1: **Comparison of methods.** (**a**) NeRF traces a ray from the camera's center of projection through each pixel and samples points $\mathbf{x}$ along each ray. Sample locations are then encoded with a positional encoding to produce a feature $\gamma(\mathbf{x})$ that is fed into an MLP. (**b**) 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 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.

posssible architectures and demonstrate that our design choices provide a high level of visual fidelity while maintaining the rendering speed of fast NeRF approaches.

4.2 Related Work

We discuss a non-exhaustive list of such approaches along axes relevant to our work.

Grid-based methods. The original NeRF took 1–2 days to train, with extensions for unbounded scenes [126, 11] taking longer. Once trained, rendering takes sec-

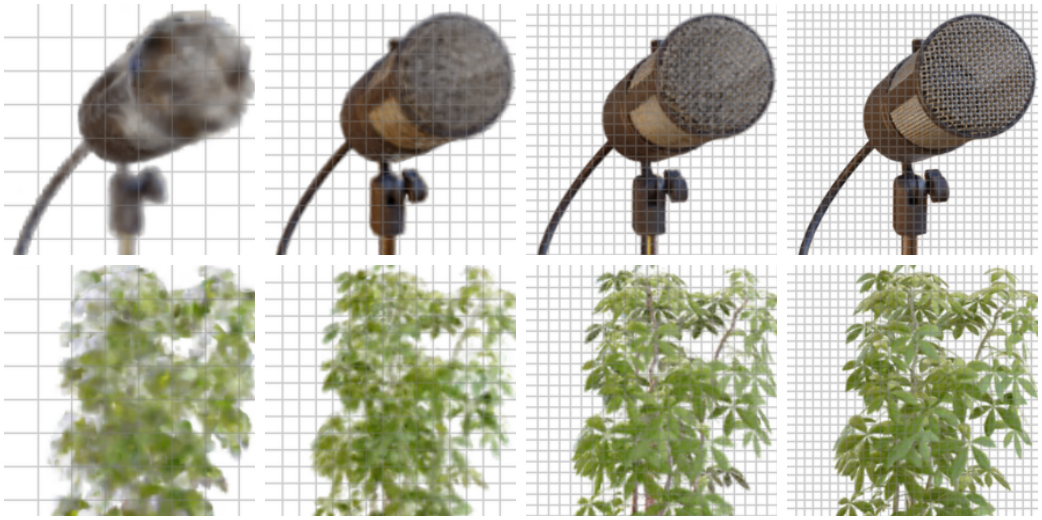


Figure 4.2: We visualize renderings from a pyramid of spatial grid-based NeRFs trained for different voxel resolutions. Models at finer pyramid levels tend to capture finer content.

onds per frame and is far below interactive thresholds. NSVF [60] combines NeRF's implicit representation with a voxel octree that allows for empty-space skipping and improves inference speeds by 10×. Follow-up works [122, 36, 41] 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 [86] and DVGO [90] accelerate both training and rendering by directly optimizing a voxel grid instead of an MLP to train in minutes or even seconds. TensoRF [16] and K-Planes [28] instead use the product of low-rank tensors to approximate the voxel grid and reduce memory usage, while Instant-NGP [66] (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 [77] and KiloNeRF [78] focus on inference speed while Mega-NeRF (Chapter 2), Block-NeRF [91], 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 [10] reasons about the volume covered by each camera ray and proposes an integrated positional encoding that alleviates aliasing. Mip-NeRF 360 [11] extends the base method to unbounded scenes. Exact-NeRF [43] derives a more precise integration formula that better reconstructs far-away scene content. Bungee-NeRF [113] 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 [114] 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.

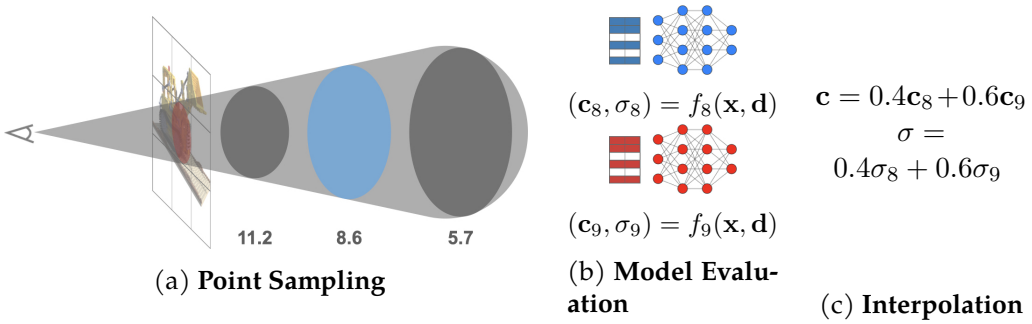
Concurrent work. Several contemporary efforts explore the intersection of anti-aliasing and fast rendering. Zip-NeRF [12] combines a hash table representation with a multi-sampling method that approximates the true integral of features contained within each camera ray's view frustum. Although it trains faster than Mip-NeRF, it is explicitly not designed for fast rendering as the multi-sampling adds significant overhead. Mip-VoG [42] downsamples and blurs a voxel grid according to the volume of each sample in world coordinates. We compare their reported numbers to ours in Sec. 4.4.2.

Classical methods. Similar to PyNeRF, classic image processing methods, such as Gaussian [5] and Laplacian [13] 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 [21], 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).

4.3 Approach

4.3.1 Preliminaries

NeRF. NeRF [64] 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



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

Figure 4.3: **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.

sampling batches $\mathcal{R}$ of image pixels and minimizing the loss $\sum_{\mathbf{r}\in\mathcal{R}} \|C(\mathbf{r}) - \hat{C}(\mathbf{r})\|^2$. We refer the reader to mildenhall2020nerf 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 [10] 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 [86, 66, 90, 16, 28] 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.

4.3.2 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. 4.3). 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 voxels project to the 2D pixel area $P(\mathbf{x})$ used to define the integration volume works well:

$$M(P(\mathbf{x})) = \log_s(P(\mathbf{x})/N_0) \tag{4.1}$$

$$l = \min(L-1, \max(0, M(P(\mathbf{x})))) \tag{4.2}$$

$$\sigma, \mathbf{c} = f_l(\mathbf{x}, \mathbf{d}), \qquad \textbf{[GaussPyNeRF]} \tag{4.3}$$

where $\cdot$ is the ceiling function. Such a model can be seen as a (Gaussian) pyramid of spatial grid-based NeRFs (Fig. 4.2). 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 [13]):

$$\sigma, \mathbf{c} = \sum_{i=0}^{l} f_i(\mathbf{x}, \mathbf{d}). \qquad \textbf{[LaplacianPyNeRF]} \tag{4.4}$$

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:

$$\sigma, \mathbf{c} = w f_l(\mathbf{x}, \mathbf{d}) + (1-w) f_{l-1}(\mathbf{x}, \mathbf{d}), \quad \text{where} \quad w = l - M(P(\mathbf{x})). \quad \textbf{(Default) [PyNeRF]} \tag{4.5}$$

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 Sec. 4.4.6). 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

Require: m rays $\mathbf{r}$, L pyramid levels, hierarchy mapping function M, base resolution N_0, scaling factor s
Ensure: 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. (4.1)
$l \leftarrow \min(L-1, \max(0, M(P(\mathbf{x}))))$ ▷ eq. (4.2)
$w \leftarrow l - M(P(\mathbf{x}))$ ▷ eq. (4.5)
$model\_out \leftarrow zeros(len(\mathbf{x}))$ ▷ Zero-initialize model outputs for each sample $\mathbf{x}$
for i in unique(l) **do** ▷ Iterate over sample levels
  $model\_out[l=i] \mathrel{+}= w[l=i] f_i(\mathbf{x}[l=i], \mathbf{d}[l=i])$
  $model\_out[l=i] \mathrel{+}= (1-w)[l=i] f_{i-1}(\mathbf{x}[l=i], \mathbf{d}[l=i])$
end for
$\mathbf{c} \leftarrow composite(model\_out)$ ▷ Composite model outputs into per-ray color $\mathbf{c}$
return $\mathbf{c}$

Algorithm 1: PyNeRF rendering function

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.

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 [100, 91, 113, 101, 115]. 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 2) 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×. 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 or proposal networks 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 (Sec. 4.4.4).

4.3.3 Space efficiency

A downside of our method is an increased serialization footprint due to training a hierarchy of spatial grid NeRFs. We describe several mitigations and measure their impact in Sec. 4.4.5:

Saving memory on coarser levels. The learned feature grids used in fast NeRF methods usually comprise the bulk of their memory footprint. In the case of iNGP [66] (which we use as the backbone model in our experiments), the feature grid is stored as a multi-resolution hash table. One possible optimization is to use smaller hash tables for lower-resolution levels. Assume that S_l is the hash table size of the finest-resolution level l. By setting $S_{l-1} = S_l/2$, we can reduce the overall storage cost of our pyramid from lS_l (when each level has the same capacity) to less than $2S_l$.

Space-efficient grid implementations. Storing features in an explicit voxel grid is expensive at high resolutions as the space complexity is $\mathcal{O}(n^3)$ for spatial scenes and $\mathcal{O}(n^4)$ spatio-temporal scenes. iNGP [66] instead uses hash tables that are smaller that the explicit grids. As an alternative, TensoRF [16] uses CANDECOMP/PARAFAC (CP) [15] or Vector-Matrix (VM) decomposition to store features in a highly space-efficient manner. We test both decomposition methods as alternatives to our default hash table encoding.

Existing grid hierarchy. Note that multi-resolution grids such as those used by iNGP [66] or K-Planes [28] already define a scale hierarchy that is a natural fit for SUDS. 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).

4.4 Experiments

We first evaluate PyNeRF's performance by measuring its reconstruction quality on bounded synthetic (Sec. 4.4.2) and unbounded real-world (Sec. 4.4.3) scenes. We then explore the convergence benefits of using multiscale training data in city-scale reconstruction scenarios (Sec. 4.4.4) and space efficiency strategies (Sec. 4.4.5). We finally ablate our design decisions in Sec. 4.4.6.

4.4.1 Experimental Setup

Training. We implement PyNeRF on top of the Nerfstudio library [93] and train on each scene for 20,000 iterations with 8,192 rays per batch. We sample rays using an occupancy grid as proposed by iNGP [66] on the Multiscale Blender dataset, and with a proposal sampling method similar to Mip-NeRF 360 [11] on all others. We parameterize unbounded scenes with Mip-NeRF 360's contraction method. We use Adam [49] with a learning rate of 10^{-2} that is cosine decayed to 5×10^{-4}.

Table 4.1: **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 [10]					Blender-A				
	↑PSNR	↑SSIM	↓LPIPS	↓Avg Error	↓Train Time (h)	↑PSNR	↑SSIM	↓LPIPS	↓Avg Error	↓ Train Time (h)
Plenoxels [86]	24.98	0.843	0.161	0.080	0:28	18.13	0.511	0.523	0.190	**0:20**
K-Planes [28]	29.88	0.946	0.058	0.022	0:32	18.55	0.368	0.641	0.251	0:38
TensoRF [16]	30.51	0.957	0.038	0.017	0:27	25.33	0.730	0.265	0.070	0:32
iNGP [66]	30.21	0.958	0.040	0.022	**0:20**	20.85	0.767	0.244	0.089	0:29
Mip-VoG [42]	30.42	0.954	0.053	—	—	—	—	—	—	—
Mip-NeRF [10]	34.50	0.974	0.017	0.009	29:49	31.33	0.894	0.098	0.063	30:12
PyNeRF	**34.57**	**0.976**	**0.013**	**0.008**	0:25	**40.14**	**0.989**	**0.013**	**0.006**	0:29

Metrics. We report quantitative results based on PSNR, SSIM [106], and the AlexNet implementation of LPIPS [127], 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 [10] composed of the geometric mean of $\text{MSE} = 10^{-\text{PSNR}/10}$, $\sqrt{1-\text{SSIM}}$, and LPIPS.

4.4.2 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). We encourage reviewers to view our supplementary videos and will release our dataset upon acceptance.

Baselines. We compare PyNeRF to several fast-rendering approaches, namely Instant-NGP [66], Plenoxels [86] which optimizes an explicit voxel grid, and TensoRF [16] and K-Planes [28], which rely on low-rank tensor decomposition. We also compare our Multiscale Blender results to those reported by Mip-VoG [42], a contemporary fast anti-aliasing approach, and to Mip-NeRF [10] on both datasets.

Results. We summarize our results in Tab. 4.1 and show qualitative examples in Fig. 4.4. 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.

4.4.3 Real-World Reconstruction

Datasets. We evaluate PyNeRF on the Boat scene of the ADOP [84] 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.

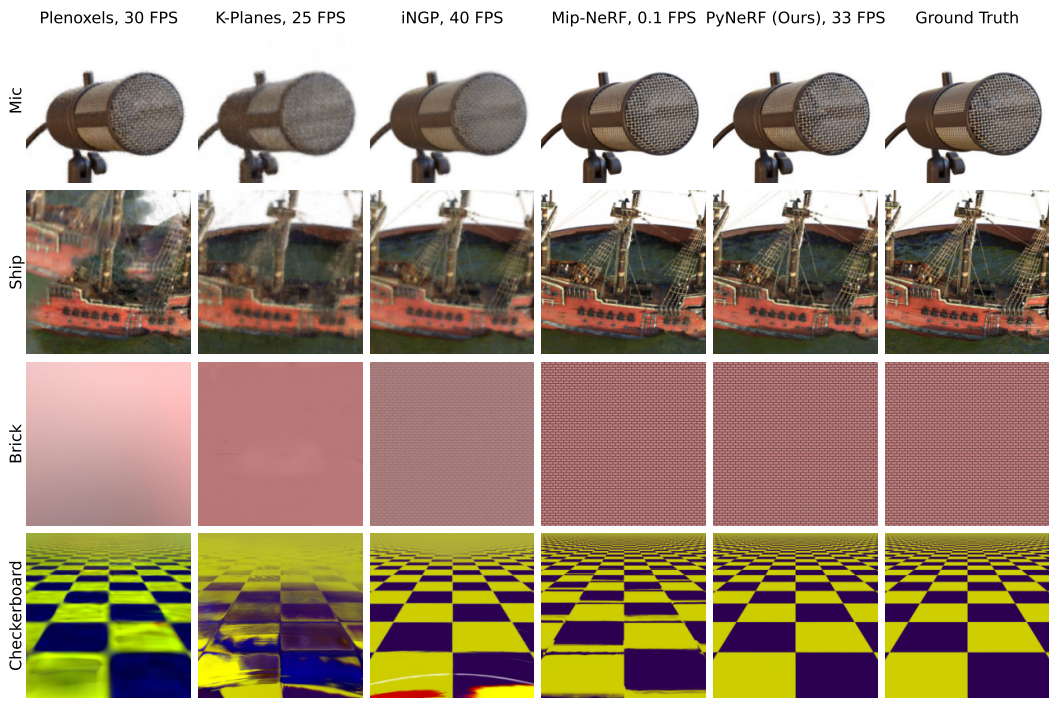


Figure 4.4: **Synthetic results.** PyNeRF and Mip-NeRF provide comparable results on the first three scenes that are crisper than those of the other fast renderers. Mip-NeRF does not accurately render the tiles in the last row while PyNeRF recreates them near-perfectly.

For further comparison, we construct a multiscale version of the outdoor scenes in the Mip-NeRF 360 [11] dataset using the same protocol as Multiscale Blender [10].

Baselines. We compare PyNeRF to the same fast-rendering approaches as in Sec. 4.4.2, along with two unbounded Mip-NeRF variants: Mip-NeRF 360 [11] and Exact-NeRF [43]. We report numbers on each variant with and without generative latent optimization [61] to account for lighting changes.

Results. We summarize our results in Tab. 4.2 along with qualitative results in Fig. 4.5. Once again, PyNeRF outperforms all baselines, trains 40× faster than Mip-NeRF 360, and 110× faster than Exact-NeRF (the next best alternatives).

4.4.4 City-Scale Convergence

Dataset. We evaluate PyNeRF's convergence properties on the the Argoverse 2 [110] Sensor dataset (to our knowledge, the largest city-scale dataset available). We select the largest overlapping subset of logs and filter out moving objects through a pretrained segmentation model [18]. The resulting training set contains 400 billion rays across 150K video frames.

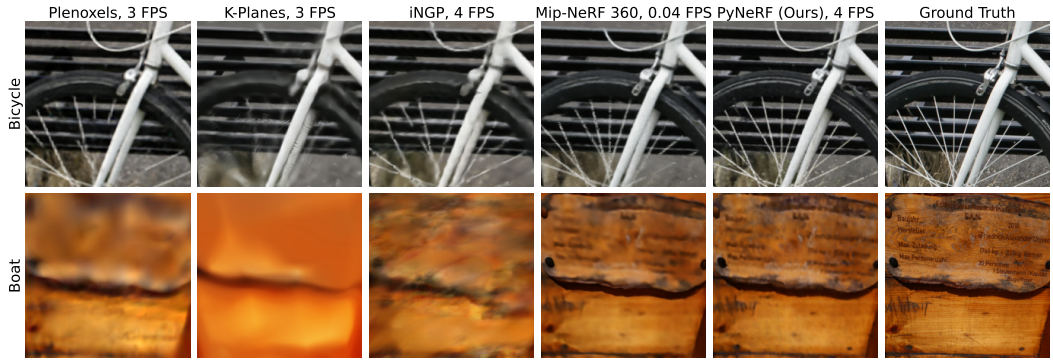


Figure 4.5: **Real-world results.** PyNeRF reconstructs higher-fidelity details (such as the spokes on the bicycle and the lettering within the boat) than other methods.

Table 4.2: **Real-world results.** PyNeRF outperforms all baselines in PSNR and average error, and trains 40× faster than Mip-NeRF 360 and 110× faster than Exact-NeRF (the next best methods).

	Boat [84]					Mip-NeRF 360 [11]				
	↑PSNR	↑SSIM	↓LPIPS	↓Avg Error	↓Train Time (h)	↑PSNR	↑SSIM	↓LPIPS	↓Avg Error	↓ Train Time (h)
Plenoxels [86]	17.05	0.505	0.617	0.185	1:23	21.88	0.606	0.524	0.117	1:00
K-Planes [28]	18.00	0.501	0.590	0.168	1:09	21.62	0.572	0.519	0.121	1:08
TensoRF [16]	14.75	0.398	0.630	0.234	1:05	14.75	0.368	0.742	0.248	1:07
iNGP [66]	15.34	0.433	0.646	0.222	**0:36**	21.14	0.568	0.521	0.126	**0:40**
Mip-NeRF 360 w/ GLO [11]	20.03	0.595	**0.416**	0.124	37:28	22.00	0.615	0.400	0.108	37:35
Mip-NeRF 360 w/o GLO [11]	15.92	0.480	0.501	0.194	37:10	21.74	0.612	0.403	0.110	37:22
Exact-NeRF w/ GLO [43]	20.21	**0.601**	0.425	0.123	109:11	22.03	0.626	**0.398**	0.106	110:06
Exact-NeRF w/o GLO [43]	16.33	0.489	0.510	0.187	107:52	21.48	0.620	0.418	0.121	108:11
PyNeRF	**21.07**	0.586	0.454	**0.118**	0:57	**22.61**	**0.635**	0.399	**0.102**	1:00

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 Sec. 4.4.2 and Sec. 4.4.3 over the first four hours of the training process.

Results. We summarize our results in Tab. 4.3. PyNeRF converges more rapidly than the SUDS baseline, achieving the same rendering quality at 2 hours as SUDS after 4.

4.4.5 Space Efficiency

We measure the perceptual quality and memory footprint of the strategies defined in Sec. 4.3.3 across the datasets defined in Sec. 4.4.2 and Sec. 4.4.3 and compare to our default SUDS method and the base TensoRF-CP, TensoRF-VM, and iNGP implementations. We summarize our results in Tab. 4.4. Using smaller tables for coarser

Table 4.3: **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**

levels reduces the overall model size by over 2× but comes at a 0.5 db cost in PSNR. Sharing the feature grid across levels reduces the overall size even further (by 4×) without a noticeable degradation in quality. Using TensoRF instead of iNGP hash tables also reduces disk space, especially with CP decomposition (300× reduction) although this is offset by a significant decrease in quality. All PyNeRF implementations perform significantly better than their base models.

4.4.6 Diagnostics

Methods. We validate our design decisions by testing several variants. We ablate our MLP-level interpolation described in eq. (4.5) and compare it to the GausssPyNeRF and LaplacianPyNeRF variants described in Sec. 4.3.2 along with another that instead interpolates the learned grid feature vectors (which avoids the need for an additional MLP evaluation per sample). We also explore using 3D sample volumes instead of projected 2D pixel areas to determine voxel levels l, and using supersampling instead of model partitioning altogether.

Results. We train our method and variants as described in Sec. 4.4.2 and Sec. 4.4.3, and summarize the results (averaged across datasets) in Tab. 4.5. Our proposed interpolation method strikes a good balance — its performance is near-identical to the full LaplacianPyNeRF approach while training 3× faster

Table 4.4: **Space Efficiency.** Using smaller tables for coarser levels or sharing the feature grid across hierarchy levels both decrease model size, with the latter strategy resulting in no apparent degradation in quality. Using TensoRF as the backbone instead of iNGP also reduces model size but reduces quality significantly. All PyNeRF implementations perform better than the base models.

Strategy	↑PSNR	↑SSIM	↓LPIPS	↓Avg Error	↓ Model Size (MB)
SUDS - smaller coarse tables	29.06	0.773	0.236	0.068	410
SUDS - TensoRF-CP	22.19	0.591	0.524	0.141	3
SUDS - TensoRF-VM	23.88	0.642	0.412	0.118	531
SUDS - shared iNGP table	**29.49**	0.778	**0.230**	**0.066**	222
TensoRF-CP [16]	21.63	0.580	0.529	0.150	**0.4**
TensoRF-VM [16]	22.81	0.588	0.461	0.132	67
iNGP [66]	23.96	0.697	0.354	0.096	214
SUDS	**29.49**	**0.780**	0.233	**0.066**	912

Table 4.5: **Diagnostics.** The rendering quality of our interpolation method is near-identical to the full residual approach while training 3× faster, and significantly better than other alternatives.

Ablation	↑PSNR	↑SSIM	↓LPIPS	↓Avg Error	↓ Training Time (h)
GaussPyNeRF (Eq. 4.3)	28.69	0.761	0.248	0.071	**0:47**
LaplacianPyNeRF (Eq. 4.4)	**29.51**	**0.780**	**0.230**	**0.065**	2:44
Feature grid interpolation	28.67	0.767	0.244	0.070	0:49
Levels w/ 3D Volumes	28.97	0.767	0.243	0.072	0:53
Super-sampling (8×)	24.62	0.680	0.332	0.103	5:36
PyNeRF	29.49	**0.780**	0.233	0.066	0:52

while providing a 1 dB improvement in PSNR relative to the other strategies and adding only a small overhead in training time. Using 2D pixel areas instead of 3D volumes to determine voxel level l provides a meaningful improvement. The super-sampling approach performs worst across all metrics.

4.5 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 NeRF representation, and although simple, provides state-of-the-art reconstruction results against a wide variety of datasets (while training 60–110× 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.

4.5.1 Limitations

A limitation of our method is that performance will degrade when zooming in and out of areas that have not been seen at training time. "Zoom-out" (rendering far-away views) can be handled by simulating far-away views during training by simply adding downsampled training images. "Zoom-in" (rendering nearby views) is fundamentally challenging since one cannot easily simulate such views without hallucination-based methods such as super-resolution. Instead, we can force renderings to query only the finest level that was supervised at train-time (by maintaining an occupancy grid-like data structure), which results in the same blurry artifacts or typical NeRF approaches.

4.5.2 Societal Impact

Our method facilitates the rapid construction of high-quality neural representations in a resource efficient manner. As such, the risks inherent to our work is similar to those of other neural rendering papers, namely privacy and security concerns related to the intentional or inadvertent capture or privacy-sensitive information such as human faces and vehicle license plate numbers. Many recent approaches [128, 51, 99, 101, 48] distill semantics into NeRF's representation, which may be used to filter out sensitive information at render time. However this information would still reside in the model itself. This could in turn be mitigated by preprocessing the input data used to train the model [103].

Chapter 5

Proposed Work: Fast Rendering via Hybrid Surface-Volume Representations

5.1 Introduction

Interactivity is essential to large-scale rendering as it is impossible to pre-compute all possible viewpoints of interest beforehand. Many existing approaches bake a trained NeRF into a finite-resolution structure. As mentioned in Chapter 2, these scale poorly against large scale scenes and either render low-quality views when zoomed in or have prohibitive memory requirements. Other methods try to directly accelerate volumetric rendering, but either do not meet the real-time requirements of VR applications (60 FPS at 2K resolution) or come with quality tradeoffs.

Our approach to this problem is simple - although NeRF is a volumetric representation, most of the world is composed of surfaces. Therefore, our model should be able to render most of the world with few samples. However, as NeRF is volumetric, it has the freedom to arbitrarily allocate density behind apparent surfaces. As an example, it often "cheats" when modeling difficult-to-model effects such as wall reflections by assigning density behind these effects. The ray marching process therefore then samples many times per ray, requiring many model evaluations.

Recent recent surface reconstruction methods [119, 104] adopt a hybrid volumetric-hybrid approach that begin by training a volumetric model (as in NeRF), but gradually encourage the model to behave as much like an SDF as possible. We believe that they can help accelerate neural rendering through two insights. First, as these methods encourage the model to bias towards surfaces, they are more compact than pure NeRF approaches. Second, as these methods derive density as a function of distance from surfaces, we are able to derive efficient sampling bounds in a mathematically robust manner. When combined, these two insights promise to greatly accelerate NeRF rendering while maintaining rendering

quality.

5.2 Related work

We summarize a non-exhaustive list of approaches relevant to our work:

Voxel baking. Some of the earliest NeRF acceleration methods store precomputed non-view dependent model outputs into a finite-resolution structures such as voxel octrees [122, 41, 36]. They bypass the original model entirely at render time by computing the final view-dependent radiance using a much smaller MLP or spherical harmonics. As shown in Chapter 2, although they achieve interactivity, they suffer from the finite capacity of the caching structure and cannot capture low-level details at scale.

Feature grids. Recent methods use a hybrid approach that combine a learned feature grid with a much smaller MLP than the original NeRF [16, 28, 66]. Instant-NGP [66] (iNGP), arguably the most popular of these methods, encodes features into a multi-resolution hash table. Although these hybrid representations speed up rendering dramatically, they cannot reach the level needed for truly interactive rendering alone, as even iNGP reaches only 5 FPS on Tanks and Temples [54]. MERF [79] comes closest through a baking pipeline that makes use of various sampling and memory layout optimizations. We use several of their optimizations in our implementation.

Meshes. Recent approaches [120, 39] train a surface representation similar to ours but then bake it into a mesh structure that handles view-dependence appearances. This mesh is further optimized and simplified, which introduces significant complexity to the training pipeline. Mobile-NeRF [17] alternatively represents the scene as a triangle mesh textured by deep features. Similar to earlier voxel-based approaches, these methods bypass the original model entirely at render time and suffer from the same finite-resolution shortfalls. Furthermore, they struggle to handle transparency and view-dependent effects, which are especially noticeable at VR resolution.

Sample efficiency. Several approaches [67, 73, 53, 11] accelerate rendering by intelligently placing far fewer samples along each ray than the original hierarchical sampling. 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 worse quality-performance tradeoff curve than our proposed approach.

Gaussian Splatting. 3D Gaussian splatting [47] has most recently emerged as the new state-of-the-art approach pushing the frontier of rendering speed and quality. This approach optimizes a discrete set of 3D gaussians through NeRF's volume rendering equation and achieves high visual quality while rendering at over 100 fps

at 1080p resolution. While impressive, the number of Gaussians needed to represent high-resolution or city-scale scenes remains an open question.

5.3 Method

The goal of our method is to first train a continuous neural representation that defines surfaces as compactly as possible while maintaining the ability to model fine structures and transparent effects (Sec. 5.3.1). We then discuss how to efficiently query the trained representation (Sec. 5.3.2).

5.3.1 Training

Preliminaries. Our model combines the acceleration benefits of grid-based approaches [66] with the properties of surface representations [119, 104]. We refer the reader to Chapters 3 and 4 for an overview of grid-based methods and use the same denisty parameterization as VolSDF [119] by training a signed distance function $f(\mathbf{x})$ such that the underlying surface of the scene is the zero-level set of f:

$$S = \{x : f(x) = 0\} \ . \tag{5.1}$$

We render the scene using NeRF volumetric rendering by defining volume density $\tau(\mathbf{x})$ and opacity $\sigma(\mathbf{x})$ as:

$$\tau(\mathbf{x}) = \frac{1}{\beta}\Psi_\beta(f(\mathbf{x})) \tag{5.2}$$

$$\sigma(\mathbf{x}) = 1 - e^{-\tau(\mathbf{x})t}, \tag{5.3}$$

where t is the distance between ray samples and Ψ_β is the CDF of a zero-mean Laplace distribution with scale parameter $\beta > 0$:

$$\Psi_\beta(s) = \begin{cases} \frac{1}{2}\exp(\frac{-s}{\beta}) & \text{if } s > 0 \\ 1 - \frac{1}{2}\exp(\frac{s}{\beta}) & \text{if } s \leq 0 \end{cases} \tag{5.4}$$

We encourage $f(\mathbf{x})$ to behave as a valid signed function by regularizing with the Eikonal loss term:

$$\mathcal{L}_{\text{eikonal}} = E_x[(\|\nabla f(\mathbf{x})\| - 1)^2] \, . \tag{5.5}$$

Surface parameterization. Prior work treats β as a scene-wide hyperparameter that is either trainable or explicitly scheduled during the training process. Note that as β approaches 0, the volumetric density converges towards an indicator function that returns $\frac{1}{\beta}$ inside any object and 0 in free space (Fig. 5.2). We therefore wish for

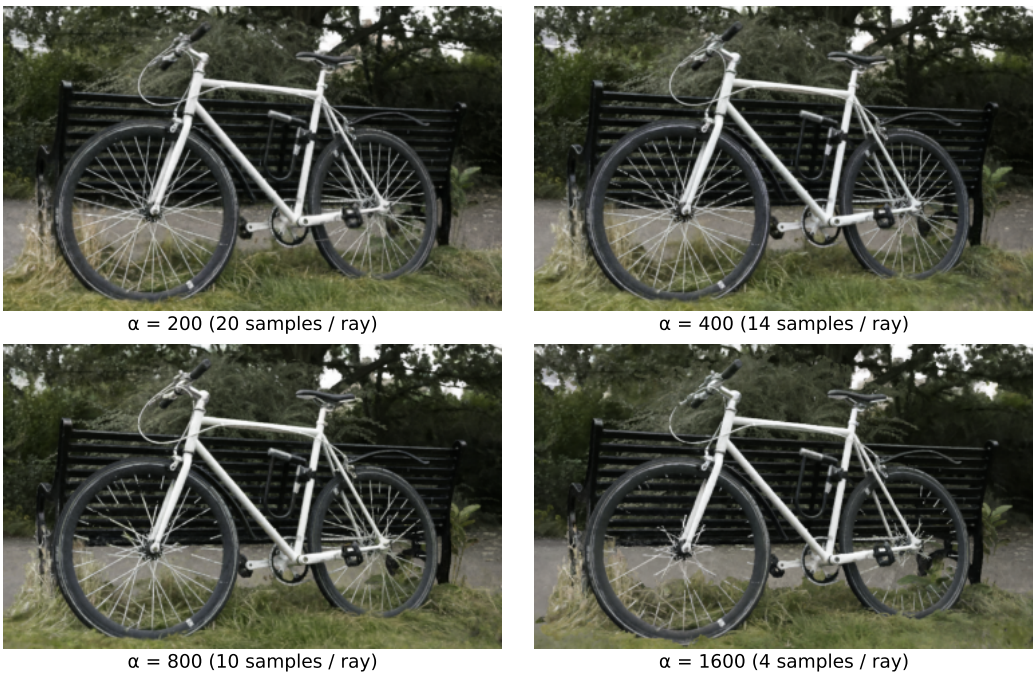


Figure 5.1: **Surfaces.** We render VolSDF [119] using different values of $\alpha = \frac{1}{\beta}$. It exhibits SDF-like behavior as α increases, reducing the number of required samples per ray. However, setting this scene-wide hyperparameter too aggressively hinders the model's ability to reconstruct fine details.

β to be as low as possible to improve sample efficiency. However, setting this parameter too aggressively reduces rendering quality and hinders the model's ability to capture fine geometry (Fig. 5.1).

Solutions such as VMesh [39] train entirely separate volumetric and surface-based networks that are alpha-composited to render a single pointwise color prediction. Although this allows for more accurate rendering of fine details, it requires sampling two separate networks, potentially increasing train and render times by 2x. We propose an alternate solution where our network predicts per-location β in addition to signed distance. This gives our model the freedom to model difficult surfaces in a less constrained manner than previous surface-based approaches while maintaining their otherwise desirable compactness properties.

5.3.2 Rendering

Occupancy grid. Once trained, our aim is to render as efficiently as possible and therefore avoid needlessly querying low-density space. Similar to MERF [79], we use a proposal network during training which we bake into a hierarchical occu-

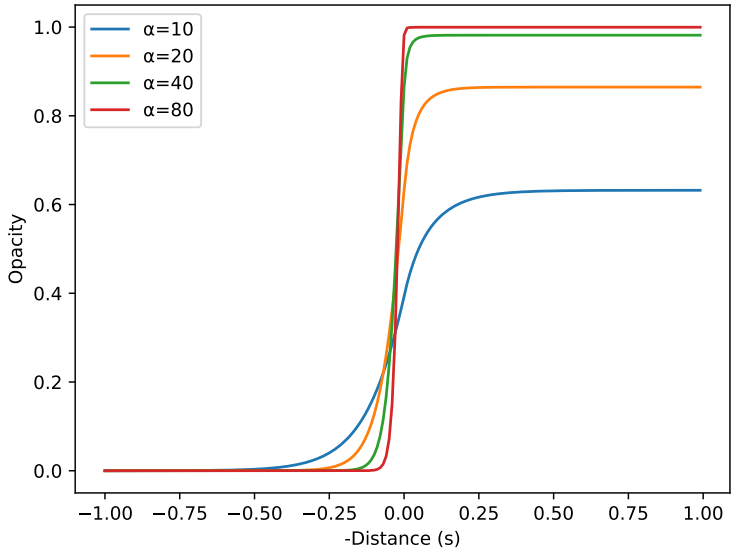


Figure 5.2: **Distance to Density.** We plot opacity as a function of surface distance using different values for $\alpha = \frac{1}{\beta}$. As β decreases, the volumetric density converges towards an indicator function that returns $\frac{1}{\beta}$ inside any object and 0 in free space.

pancy grid containing only part of the scene with opacity values above a defined threshold σ_c. We use the same piecewise-projective contraction method and empty space skipping to query the occupancy grids as efficiently as possible.

Surface-guided sampling. After reaching an occupied voxel, sampling within the voxel remains an open question. Existing approaches use a constant or exponentially increasing step size that must be carefully tuned to avoid aliasing artifacts or excessive model queries. However, as we model density as a function of signed distance, we can directly predict where to next sample along the ray. For a given opacity threshold σ_c, fixed render step size t, and existing ray sample with predicted distance s, one need simply predict the next sample distance d such as the predicted surface distance $s - d$ remains above the opacity threshold:

$$\alpha\Psi_\beta(s-d) = -\frac{\log(1-\sigma_c)}{t},. \tag{5.6}$$

Note that although this method works well for most surfaces, it fails to converge near discontinuities (5-10% in our experiments). We therefore track convergence and fall back to standard volumetric rendering if convergence does not improve after a fixed number of iterations.

5.4 Evaluation

We plan to evaluate our method against a soon-to-be-released dataset of indoor scenes captured in HDR at 4K resolution. Each scene contains 1,000-2,000 images with complex view-dependent effects. We will compare our method against existing state-of-the-art methods including a baked mesh baseline, 3D Gaussian Splatting [47], and Instant NGP [66].

Chapter 6

Proposed Work: Generative Models for Urban Scene Completion

6.1 Introduction

NeRF can generate highly photorealistic renderings of camera viewpoints close to training poses but generalize poorly to larger viewpoint changes. The problem is especially acute in dynamic 4D settings such as that described in Chapter 3 where the training data captures only a subset of the scene at any given time step. In the case of moving objects, we might only view the object of interest from a single viewpoint (such as the back of a car). The ability to generate plausible renderings of the entire object would greatly enhance use cases such as re-simulation for autonomous driving along more immersive general free viewpoint rendering.

Various approaches improve NeRF's extrapolation capabilities through hand-

Figure 6.1: **Moving objects.** Moving objects are especially challenging to render plausibly when seen from a limited set of training viewpoints.

crafted or data-driven priors. Reg-NeRF [68] regularizes geometry and appearance of patches rendered from unobserved viewpoints and an annealed ray sampling strategy to prevent divergence during training. Manhattan SDF [38] uses a pretrained segmentation network along with Manhattan-world assumption to regularize the geometry of floor and wall regions. PixelNeRF [121] conditions NeRF on image inputs using convolutional features. GANeRF [83] uses an adversarial discriminator to generate sharper renderings and remove floater artifacts in under-observed areas.

Most recently, Tewari et al [96], SparseFusion [129], and DiffRF [65], all propose combining denoising diffusion models with a 3D neural representation to generate plausible view-consistent renderings from sparse views. Although results are encouraging, they are solely evaluated on static, single-object scenes. We aim to extend their work to less constrained urban settings.

As a tractable starting point, we propose a compositional scene representation similar to that proposed by Neural Scene Graphs [70] where the background and each foreground object is modeled with a separate NeRF/diffusion model pairing. Assuming that the background is constant, can we generate plausible renderings of a given object at an arbitrary timestep and viewpoint? Rigid, symmetrical objects such as vehicles will likely be easier to model than deformable pedestrians. Assuming the initial experiment is successful, our next goal would be to derive a suitable model representation and training strategy for city-scale scenes, as representing each object with a separate NeRF is infeasible. Learning category-specific codes and model distillation are possible solutions.

Chapter 7

Thesis Timeline

Fall - Winter 2023 (September 2023 - January 2024): Fast Rendering via Hybrid Surface-Volume Representations
Spring 2024 (February - May 2024): Generative Models for Urban Scene Completion
Summer 2024: Thesis Writing
September 2024: Thesis Defense

Bibliography

[1] 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. 4

[2] Kwea123's nsff implementation. https://github.com/kwea123/nsff_pl. Accessed: 2022-10-29. 32

[3] 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. 36

[4] Scikit incremental pca. https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.IncrementalPCA.html. Accessed: 2022-10-29. 34

[5] Edward Adelson, Charles Anderson, James Bergen, Peter Burt, and Joan Ogden. Pyramid methods in image processing. *RCA Eng.*, 29, 11 1983. 41, 44

[6] 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. 9

[7] Sameer Agarwal, Yasutaka Furukawa, Noah Snavely, Ian Simon, Brian Curless, Steven M Seitz, and Richard Szeliski. Building rome in a day. *Communications of the ACM*, 54(10):105–112, 2011. 22

[8] Shir Amir, Yossi Gandelsman, Shai Bagon, and Tali Dekel. Deep vit features as dense visual descriptors. *arXiv preprint arXiv:2112.05814*, 2021. 31, 34

[9] Dragomir Anguelov, Carole Dulong, Daniel Filip, Christian Frueh, Stéphane Lafon, Richard Lyon, Abhijit Ogale, Luc Vincent, and Josh Weaver. Google street view: Capturing the world at street level. *Computer*, 43(6):32–38, 2010. 22

[10] 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. 41, 44, 45, 49, 50

[11] 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. 25, 34, 42, 44, 48, 50, 51, 56

[12] 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. 44

[13] P. Burt and E. Adelson. The laplacian pyramid as a compact image code. *IEEE Transactions on Communications*, 31(4):532–540, 1983. 44, 46

[14] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. 2021. 26, 28

[15] J. Douglas Carroll and Jih Jie Chang. Analysis of individual differences in multidimensional scaling via an n-way generalization of "eckart-young" decomposition. *Psychometrika*, 35:283–319, 1970. 48

[16] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. In *ECCV*, 2022. 26, 43, 45, 48, 49, 51, 53, 56

[17] 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. 56

[18] 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. 50

[19] 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. 21

[20] David Crandall, Andrew Owens, Noah Snavely, and Daniel Huttenlocher. Discrete-continuous optimization for large-scale structure from motion. In *CVPR*, 2011. 7, 13

[21] 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. 41, 44

[22] 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. 36

[23] Kangle Deng, Andrew Liu, Jun-Yan Zhu, and Deva Ramanan. Depth-supervised NeRF: Fewer views and faster training for free. In *CVPR*, June 2022. 26, 32

[24] 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. 21, 25, 30

[25] 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. 4

[26] 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. 14

[27] 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. 14

[28] 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. 43, 45, 48, 49, 51, 56

[29] 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. 25

[30] 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. 9

[31] 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. 30

[32] Adrien Gaidon, Qiao Wang, Yohann Cabon, and Eleonora Vig. Virtual worlds as proxy for multi-object tracking analysis. In *CVPR*, pages 4340–4349, 2016. 38

[33] Chen Gao, Ayush Saraf, Johannes Kopf, and Jia-Bin Huang. Dynamic view synthesis from dynamic monocular video. In *ICCV*, 2021. 21

[34] Chen Gao, Ayush Saraf, Johannes Kopf, and Jia-Bin Huang. Dynamic view synthesis from dynamic monocular video. In *ICCV*, 2021. 25, 27, 30, 32, 33

[35] Hang Gao, Ruilong Li, Shubham Tulsiani, Bryan Russell, and Angjoo Kanazawa. Monocular dynamic view synthesis: A reality check. In *NeurIPS*, 2022. 22

[36] Stephan J Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin. Fastnerf: High-fidelity neural rendering at 200fps. pages 14346–14355, 2021. 7, 43, 56

[37] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *CVPR*, 2012. 37

[38] Haoyu Guo, Sida Peng, Haotong Lin, Qianqian Wang, Guofeng Zhang, Hujun Bao, and Xiaowei Zhou. Neural 3d scene reconstruction with the manhattan-world assumption. In *CVPR*, 2022. 62

[39] 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. 56, 58

[40] Zekun Hao, Arun Mallya, Serge Belongie, and Ming-Yu Liu. GANcraft: Unsupervised 3D Neural Rendering of Minecraft Worlds. In *ICCV*, 2021. 28

[41] Peter Hedman, Pratul P. Srinivasan, Ben Mildenhall, Jonathan T. Barron, and Paul Debevec. Baking neural radiance fields for real-time view synthesis. *ICCV*, 2021. 7, 43, 56

[42] 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. 44, 49

[43] 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. 44, 50, 51

[44] Yoonwoo Jeong, Seokjun Ahn, Christopher Choy, Animashree Anandkumar, Minsu Cho, and Jaesik Park. Self-calibrating neural radiance fields. In *ICCV*, 2021. 21

[45] 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. 25

[46] 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. 7, 9

[47] 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. 56, 60

[48] Justin Kerr, Chung Min Kim, Ken Goldberg, Angjoo Kanazawa, and Matthew Tancik. Lerf: Language embedded radiance fields. 2023. 54

[49] DP Kingma, J Ba, Y Bengio, and Y LeCun. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations*, 2015. 14, 34, 48

[50] 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. 4, 7

[51] Sosuke Kobayashi, Eiichi Matsumoto, and Vincent Sitzmann. Decomposing nerf for editing via feature field distillation. In *Advances in Neural Information Processing Systems*, volume 35, 2022. 26, 28, 54

[52] 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. 24, 25, 37

[53] Andreas Kurz, Thomas Neff, Zhaoyang Lv, Michael Zollhöfer, and Markus Steinberger. Adanerf: Adaptive sampling for real-time rendering of neural radiance fields. 2022. 56

[54] Sicheng Li, Hao Li, Yue Wang, Yiyi Liao, and Lu Yu. Steernerf: Accelerating nerf rendering via smooth viewpoint trajectory. In *CVPR*, 2023. 56

[55] 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. 25

[56] Zhengqi Li, Simon Niklaus, Noah Snavely, and Oliver Wang. Neural scene flow fields for space-time view synthesis of dynamic scenes. In *CVPR*, 2021. 25, 27, 30, 32, 33

[57] Chen-Hsuan Lin, Wei-Chiu Ma, Antonio Torralba, and Simon Lucey. Barf: Bundle-adjusting neural radiance fields. In *ICCV*, 2021. 21

[58] Liqiang Lin, Yilin Liu, Yue Hu, Xingguang Yan, Ke Xie, and Hui Huang. Capturing, reconstructing, and simulating: the urbanscene3d dataset. In *ECCV*, 2022. 7, 14

[59] Philipp Lindenberger, Paul-Edouard Sarlin, Viktor Larsson, and Marc Pollefeys. Pixel-Perfect Structure-from-Motion with Featuremetric Refinement. In *ICCV*, 2021. 14

[60] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. *NIPS*, 2020. 43

[61] 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. 7, 9, 10, 14, 25, 50

[62] 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. 21

[63] 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. 7

[64] 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. I, 7, 10, 14, 22, 26, 31, 37, 44

[65] Norman Müller, Yawar Siddiqui, Lorenzo Porzi, Samuel Rota Bulo, Peter Kontschieder, and Matthias Nießner. Diffrf: Rendering-guided 3d radiance field diffusion. In *CVPR*, 2023. 62

[66] 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. 26, 27, 43, 45, 48, 49, 51, 53, 56, 57, 60

[67] 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. 8, 56

[68] 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. 62

[69] Michael Niemeyer and Andreas Geiger. Giraffe: Representing scenes as compositional generative neural feature fields. In *CVPR*, 2021. 25

[70] Julian Ost, Fahim Mannan, Nils Thuerey, Julian Knodt, and Felix Heide. Neural scene graphs for dynamic scenes. In *CVPR*, pages 2856–2865, June 2021. 24, 25, 36, 37, 62

[71] 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. 21, 22, 25

[72] 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. 25

[73] 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. 56

[74] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-NeRF: Neural Radiance Fields for Dynamic Scenes. In *CVPR*, 2020. 25

[75] Ravi Ramamoorthi. Nerfs: The search for the best 3d representation, 2023. I

[76] 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. 6, 7

[77] D. Rebain, W. Jiang, S. Yazdani, K. Li, K. Yi, and A. Tagliasacchi. DeRF: Decomposed radiance fields. In *CVPR*, 2021. 41, 43

[78] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14335–14345, 2021. 6, 8, 41, 43

[79] 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. *SIGGRAPH*, 2023. 56, 58

[80] Konstantinos Rematas, Andrew Liu, Pratul P. Srinivasan, Jonathan T. Barron, Andrea Tagliasacchi, Tom Funkhouser, and Vittorio Ferrari. Urban radiance fields. 2022. 9, 26, 27, 28, 32

[81] Gernot Riegler and Vladlen Koltun. Stable view synthesis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2021. 14

[82] 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. 26

[83] Barbara Roessle, Norman Müller, Lorenzo Porzi, Samuel Rota Bulò, Peter Kontschieder, and Matthias Nießner. Ganerf: Leveraging discriminators to optimize neural radiance fields, 2023. 62

[84] Darius Rückert, Linus Franke, and Marc Stamminger. Adop: Approximate differentiable one-pixel point rendering. *ACM Trans. Graph.*, 41(4):99:1–14, jul 2022. 49, 51

[85] Sara Fridovich-Keil and Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *CVPR*, 2022. 16

[86] Sara Fridovich-Keil and Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *CVPR*, 2022. 26, 43, 45, 49, 51

[87] 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. 14

[88] 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. 25, 36

[89] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. In *Advances in Neural Information Processing Systems*, 2019. 37

[90] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In *CVPR*, 2022. 26, 43, 45

[91] 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. 9, 25, 30, 41, 43, 47

[92] 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. 9

[93] 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. 48

[94] Zachary Teed and Jia Deng. Raft: Recurrent all-pairs field transforms for optical flow (extended abstract). In *IJCAI*, 2021. 34, 35

[95] 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. 9

[96] Ayush Tewari, Tianwei Yin, George Cazenavette, Semon Rezchikov, Joshua B. Tenenbaum, Frédo Durand, William T. Freeman, and Vincent Sitzmann. Diffusion with forward models: Solving stochastic inverse problems without direct supervision. In *arXiv*, 2023. 62

[97] 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. 21, 25

[98] Alex Trevithick and Bo Yang. Grf: Learning a general radiance field for 3d representation and rendering. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15182–15192, 2021. 9

[99] 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. 26, 28, 54

[100] 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. 27, 30, 41, 47

[101] Haithem Turki, Jason Y Zhang, Francesco Ferroni, and Deva Ramanan. SUDS: Scalable urban dynamic scenes. In *CVPR*, 2023. 47, 54

[102] 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. 26

[103] Junjue Wang, Brandon Amos, Anupam Das, Padmanabhan Pillai, Norman Sadeh, and Mahadev Satyanarayanan. A scalable and privacy-aware iot service for live video analytics. In *Proceedings of the 8th ACM on Multimedia Systems Conference*, MMSys'17, page 38–49, New York, NY, USA, 2017. Association for Computing Machinery. 21, 54

[104] 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. 55, 57

[105] 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. 9

[106] 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. 16, 34, 49

[107] 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. 21

[108] William T. Weldon and Joseph Hupy. Investigating methods for integrating unmanned aerial systems in search and rescue operations. *Drones*, 4(3), 2020. 4

[109] Lance Williams. Pyramidal parametrics. *Computer Graphics*, 17:1–11, July 1983. 41

[110] 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. 50

[111] Tianhao Wu, Fangcheng Zhong, Andrea Tagliasacchi, Forrester Cole, and Cengiz Oztireli. D^2nerf: Self-supervised decoupling of dynamic and static objects from a monocular video. In *Advances in Neural Information Processing Systems*, 2022. 25, 27, 28, 33

[112] 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. 25

[113] 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. 9, 25, 44, 47

[114] 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. 44

[115] 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. 47

[116] 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. 25

[117] 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. 25, 27

[118] 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. 10

[119] Lior Yariv, Jiatao Gu, Yoni Kasten, and Yaron Lipman. Volume rendering of neural implicit surfaces. 2021. 55, 57, 58

[120] 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. 56

[121] 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. 9, 62

[122] 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. 7, 12, 43, 56

[123] Hong-Xing Yu, Leonidas J. Guibas, and Jiajun Wu. Unsupervised discovery of object radiance fields. In *International Conference on Learning Representations*, 2022. 25

[124] 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. 25

[125] 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. 7

[126] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. Nerf++: Analyzing and improving neural radiance fields. *arXiv:2010.07492*, 2020. 7, 9, 11, 14, 25, 27, 42

[127] 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. 16, 34, 49

[128] Shuaifeng Zhi, Tristan Laidlow, Stefan Leutenegger, and Andrew Davison. In-place scene labelling and understanding with implicit scene representation. In *ICCV*, 2021. 25, 54

[129] Zhizhuo Zhou and Shubham Tulsiani. Sparsefusion: Distilling view-conditioned diffusion for 3d reconstruction. In *CVPR*, 2023. 62