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

Haithem Turki[1] Deva Ramanan[1,2] Mahadev Satyanarayanan[1]
[1]Carnegie Mellon University [3]Argo AI

Abstract

We explore how to leverage neural radiance fields (NeRFs) to build interactive 3D environments from large-scale visual captures spanning buildings or even multiple city blocks collected primarily from drone data. In contrast to the single object scenes against which NeRFs have been traditionally evaluated, this setting poses multiple challenges including (1) the need to incorporate thousands of images with varying lighting conditions, all of which capture only a small subset of the scene, (2) prohibitively high model capacity and ray sampling requirements beyond what can be naively trained on a single GPU, and (3) an arbitrarily large number of possible viewpoints that make it unfeasible to precompute all relevant information beforehand (as real-time NeRF renderers typically do). To address these challenges, we begin by analyzing visibility statistics for large-scale scenes, motivating a sparse network structure where parameters are specialized to different regions of the scene. We introduce a simple geometric clustering algorithm that partitions training images (or rather pixels) into different NeRF submodules that can be trained in parallel. We evaluate our approach across scenes taken from the Quad 6k and UrbanScene3D datasets as well as against our own drone footage and show a 3x training speedup while improving PSNR by over 11% on average. We subsequently perform an empirical evaluation of recent NeRF fast renderers on top of Mega-NeRF and introduce a novel method that exploits temporal coherence. Our technique achieves a 40x speedup over conventional NeRF rendering while remaining within 0.5 db in PSNR quality, exceeding the fidelity of existing fast renderers.

1. Introduction

Recent advances in neural rendering techniques have lead to significant progress towards photo-realistic novel view synthesis, a prerequisite towards many VR and AR applications. In particular, Neural Radiance Fields (NeRFs) [22] have attracted significant attention, spawning a wide range of follow-up works that improve upon various aspects of the original methodology.

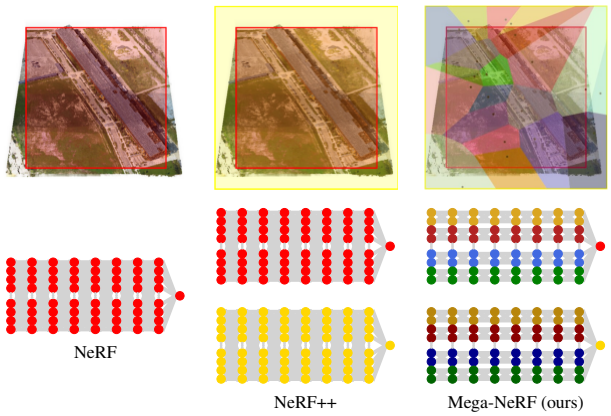


Figure 1. We scale neural reconstructions to massive urban scenes 1000x larger than prior work. To do so, we introduce innovations for scalable training and fast inference. The original NeRF encodes the entire scene within the weights of a single multi-layer perceptron (MLP). NeRF++ [38] models scenes by partitioning space into a foreground region containing all cameras (red) and an unbounded background (yellow), learning separate MLPs for each. Mega-NeRF (**ours**) further decomposes the foreground into a set of spatial cells, learning a separate NeRF for each. Each cell-specific submodule can be trained independently and in parallel by *sharding* input images and pixels relevant to that cell, as determined by a geometric ray clustering algorithm.

Scale. Simply put, our work explores 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 [15] 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 a 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-

	Resolution	# Images	# Pixels/Rays
Synthetic NeRF [22]	400 x 400	400	256,000,000
LLFF [21]	4032 x 3024	41	496,419,840
Light Field [37]	1280 x 720	214	195,910,200
Tanks and Temples [15]	1920 x 1080	283	587,658,240
Phototourism [13]	919 x 794	1708	1,149,113,846
Mill 19	4608 x 3456	1809	28,808,773,632
Quad 6k [4]	1708 x 1329	5147	11,574,265,679
UrbanScene3D [18]	5232 x 3648	2483	47,383,531,520

Table 1. 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 [38] and [35] respectively. For Phototourism we average across the scenes used in [19].

time [6], collected footage is typically reconstructed into 2D "birds-eye-view" maps that support post-hoc analysis [33]. 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." This requires us to train a usable model within a day. Secondly, as our datasets are orders of magnitude larger than previously evaluated datasets (Table 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. Because only a small fraction of the training images are visible from any particular scene point, we introduce a sparse network structure where parameters are specialized to different regions of the scene. We introduce a simple geometric clustering algorithm that partitions training images (or rather pixels) into different NeRF submodules that can be trained in parallel. We further exploit spatial locality at render time to implement a just-in-time visualization technique that allows for interactive fly-throughs of the captured environment.

	# Images	# Pixels/Rays	Scene Captured / Image
Synthetic - Chair	400	256,000,000	0.271
Synthetic - Drums	400	256,000,000	0.302
Synthetic - Ficus	400	256,000,000	0.582
Synthetic - Hotdog	400	256,000,000	0.375
Synthetic - Lego	400	256,000,000	0.205
Synthetic - Materials	400	256,000,000	0.379
Synthetic - Mic	400	256,000,000	0.518
Synthetic - Ship	400	256,000,000	0.483
T&T - Barn	384	796,262,400	0.135
T&T - Caterpillar	368	763,084,800	0.216
T&T - Family	152	315,187,200	0.284
T&T - Ignatius	263	545,356,800	0.476
T&T - Truck	250	518,400,000	0.225
Mill 19 - Building	1940	30,894,981,120	0.062
Mill 19 - Rubble	1678	26,722,566,144	0.050
Quad 6k	5147	11,574,265,679	0.010
UrbanScene3D - Residence	2563	51,162,236,928	0.059
UrbanScene3D - Sci-Art	2942	52,202,471,424	0.088
UrbanScene3D - Campus	1943	38,785,886,208	0.028

Table 2. Estimated fraction of the overall area captured per image across scenes from the Synthetic NeRF, Tanks and Temples (T&T), and 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.

Prior art. Our approach of using "multiple" NeRF submodules is closely inspired by the recent work of KiloNeRF [25], which uses similar insights to enable real-time *inference* (or rendering) of an existing, pre-trained NeRF. However, even training a NeRF at our scene scale is essentially impossible with current pipelines. We demonstrate that modularity is vital for *training*, particularly when combined with an intelligent strategy for "sharding" training data into the appropriate modules via geometric clustering.

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

Fast rendering. Conventional NeRF rendering falls well below interactive thresholds. Plenoctree [35], SNeRG [11],

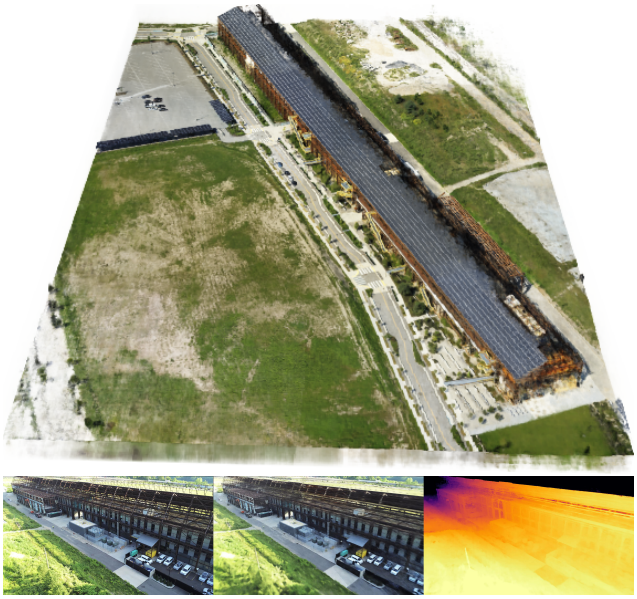


Figure 2. Visualization of Mill 19 by Mega-NeRF. The top panel shows a high-level 3D rendering of Mill 19 within our interactive visualizer. The bottom-left panel contains a ground truth image captured by our drone. The following two panels illustrate the model reconstruction along with the associated depth map.

and FastNeRF [10] 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.

KiloNeRF [25] first produces a standard NeRF model to use as a teacher to train a collection of much smaller MLPs which are then directly finetuned against the training data. Each small MLP is responsible for a subset of the scene and requires less computation than the original model, dramatically speeding up rendering. Although similar in spirit to Mega-NeRF's layer sparsity, KiloNeRF targets fast inferencing within bounded indoor scenes instead of end-to-end scalable model training. Its post-training steps incur significant overhead and increase processing time by over 2x.

Unbounded scenes. Although most NeRF-related work targets indoor areas, NeRF++ [38] 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 Section 3.1.

Figure 3. Novel viewpoint in our Rubble scene using traditional photogrammetry **(left)** and Mega-NeRF's renderer **(right)**. Mega-NeRF captures fine detail needed for immersive fly-throughs.

NeRF in the Wild [19] augments NeRF's model with an additional transient head and learned per-image embeddings to better explain photometric variance and transient occlusions across images. Although it does not explicitly target unbounded scenes, it achieves impressive results against outdoor sequences in the Phototourism [13] dataset. We adopt similar appearance embeddings for Mega-NeRF and quantify its impact in Section 4.2.

Training speed. Several works speed up model training by incorporating priors learned from similar datasets. PixelNeRF [36], IBRNet [31], and GRF [29] condition NeRF on predicted image features while Tancik et al. [26] 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 [27] subdivide a scene into multiple cells to filter out irrelevant geometry and speed up rendering. Funkhouser and Séquin [8] 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. Our work is inspired by previous large-scale reconstruction efforts based on classical Structure-from-Motion (SfM), in particular Agarwal et al's seminal "Building Rome in a Day," [3] which describes city-scale 3D reconstruction from internet-gathered data.

3. Approach

We first describe our overall model architecture in 3.1, then our training process in 3.2, and finally propose a novel renderer that exploits temporal coherence in 3.3.

3.1. Model Architecture

Background. We begin with a brief description of Neural Radiance Fields (NeRFs) [22]. 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 [22] 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 [19], 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 photometric variation 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:

$$f^{\mathbf{n}}(\mathbf{x}) = \sigma \tag{1}$$

$$f^{\mathbf{n}}(\mathbf{x}, \mathbf{d}, l^{(a)}) = \mathbf{c} \tag{2}$$

$$\text{where } \mathbf{n} = \underset{n \in \mathcal{N}}{\operatorname{argmin}} \left\| n - \mathbf{x} \right\|^2 \tag{3}$$

Centroid selection. Although we explored several methods, including k-means clustering and uncertainty-based partitioning as in [34], we ultimately found that tessellating the scene into a top-down 2D grid worked well in practice. This method is simple to implement, requires minimal preprocessing, and enables efficient assignment of point queries to centroids at inference time. As the variance in altitude between camera poses in our scenes is small relative to the differences in latitude and longitude, we fix the height of the centroids to the same value.

Foreground and background decomposition. Similar to NeRF++ [38], 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. Figure 4 illustrates the differences between both approaches.

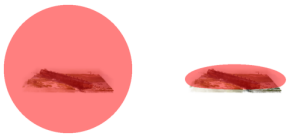


Figure 4. **Ray Bounds.** NeRF++ (**left**) samples within a unit sphere enclosing all camera poses to render its foreground component and uses a different methodology for the outer volume complement to efficiently render the background. Mega-NeRF (**right**) uses a similar background parameterization but models the foreground as an ellipsoid to achieve tighter bounds on the region of interest. It also uses camera altitude measurements to further constrain ray sampling and not needlessly query underground regions.

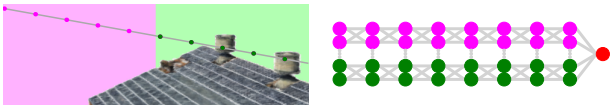


Figure 5. **Spatial assignment**. The depicted ray is used to train the submodules for both the magenta and green cells as we do not know the 3D location corresponding to the pixel a priori. At inference time, point queries within a cell are routed to the appropriate submodule.

3.2. Training Process

As each Mega-NeRF submodule is a self-contained MLP, we can train each entirely in parallel with no inter-subset communication. Furthermore, each training image captures only a small section of the overall scene as shown in Table 2. We therefore attempt to localize the data used to train each submodule by including only pixels that could plausibly depict the region of interest.

We partition our dataset at the pixel level as shown in Figure 6. For each training pixel, we sample points along the associated camera ray and register the cells it traverses as illustrated in Figure 5. If a camera ray traverses multiple cells, we include the pixel in all associated training sets.

We note that we could further deduplicate our training sets with additional 3D knowledge of the scene. The initial assignment of pixels to submodules is purely based on camera poses, irrespective of scene geometry. Once NeRF gains a coarse understanding of the scene, one could prune away irrelevant pixel/ray assignments that don't contribute to a particular submodule due to an intervening occluder. This could be implemented as a second partitioning after a small amount of initial training, incurring only a small inter-process communication cost. The analysis in Section A of the supplement suggests this reduces the amount of training data for each submodule by 2x while increasing accuracy.

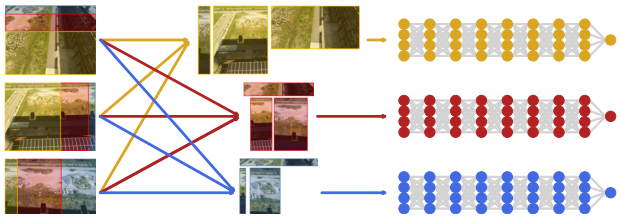


Figure 6. **Data partitioning.** We construct a distinct training set for each submodule consisting of training image pixels whose ray might traverse the cell it represents. Each submodule can then be trained entirely in parallel with no cross-cell communication.

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 Section 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 5 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 [35] 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 temporal coherence of interactive flythroughs; 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. Figure 7 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. Figure 8 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.

4. Experiments

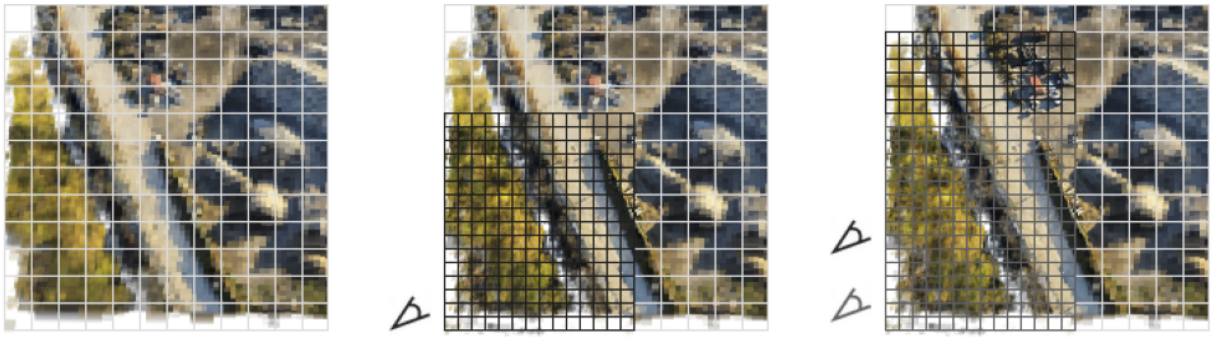
Our evaluation of Mega-NeRF is motivated by the following two questions. First, given a finite training budget, how accurately can Mega-NeRF capture a scene? Furthermore, after training, is it possible to render accurately at scale while minimizing latency?

Qualitative results. We present two sets of qualitative results. Figure 9 showcases Mega-NeRF's reconstruction quality relative 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. In several instances, Mega-NeRF presents sensible renderings where other approaches completely break down. Figure 10 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.

4.1. Evaluation protocols

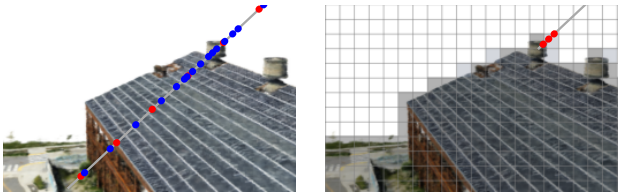
Datasets. We evaluate Mega-NeRF against multiple varied datasets. Our Mill 19 dataset consists of two scenes we recorded firsthand near a former industrial complex. Mill 19 - Building showcases footage captured in a grid pattern across a large $500 \times 250\ m^2$ area around an industrial building. 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 [4], a large-scale Structure-from-Motion dataset collected within the Cornell Universty Arts Quad, and several scenes from UrbanScene3D [18] which contain high-definition drone imagery of large-scale urban environments. We refine the initial GPS-derived camera poses of the raw images in the Mill 19 and UrbanScene3D datasets using Pix4DMapper [2]. We directly use the bundle-adjusted poses provided in Quad 6k. As in [19], we prefilter out a small number of images containing large occlusions.

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-



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

Figure 7. **Mega-NeRF-Dynamic.** Current renderers (such as Plenoctree [35]) 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.



Standard Hierachical Sampling Guided Sampling

Figure 8. **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.

precision training to further accelerate the process. We sample 1024 rays per batch and use the Adam optimizer [14] with an initial learning rate of 5×10^{-4} decaying exponentially to 5×10^{-5}. We employ the procedure described in [19] to finetune Mega-NeRF's appearance embeddings.

4.2. Scalable training

Baselines. We evaluate Mega-NeRF against the original NeRF [22] architecture, NeRF++ [38], and DeepView [7] as a non-neural radiance field-based alternative. 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 for fairness. We provide each implementation with the same amount of model capacity as Mega-NeRF by setting the MLP width to 2048 hidden units.

Metrics. We report quantitative results based on PSNR, SSIM [32], and the VGG implementation of LPIPS [39] provided at `https://github.com/richzhang/PerceptualSimilarity`. We also report training times as measured on a single machine with 8 V100 GPUs.

Results. We show results after 500,000 training iterations in Table 3 along with the time taken to finish training. Mega-NeRF outperforms the baselines across all metrics even after training the other approaches for longer periods.

Diagnostics. We compare Mega-NeRF to several ablations. Mega-NeRF-no-embed removes the appearance embeddings from the model structure. Mega-NeRF-no-bounds uses NeRF++'s unit sphere background/foreground partitioning instead of our formulation described in 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 per-region partitions. We limit training to 24 hours for expediency.

We present our results in Table 5. 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.

Scaling properties. We examine Mega-NeRF's scaling properties in Section A of the supplement.

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 4.2 to the extent possible. 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.

Metrics. We report the same perceptual metrics as in 4.2 and the time it takes to render a 720p image. We use generated background masks as Plenoctree and KiloNeRF assume bounded scenes. We also report any additional time needed to generate any auxiliary data structures used for rendering *beyond* the base model training time in the spirit of enabling fly-throughs within a day. As our renderer

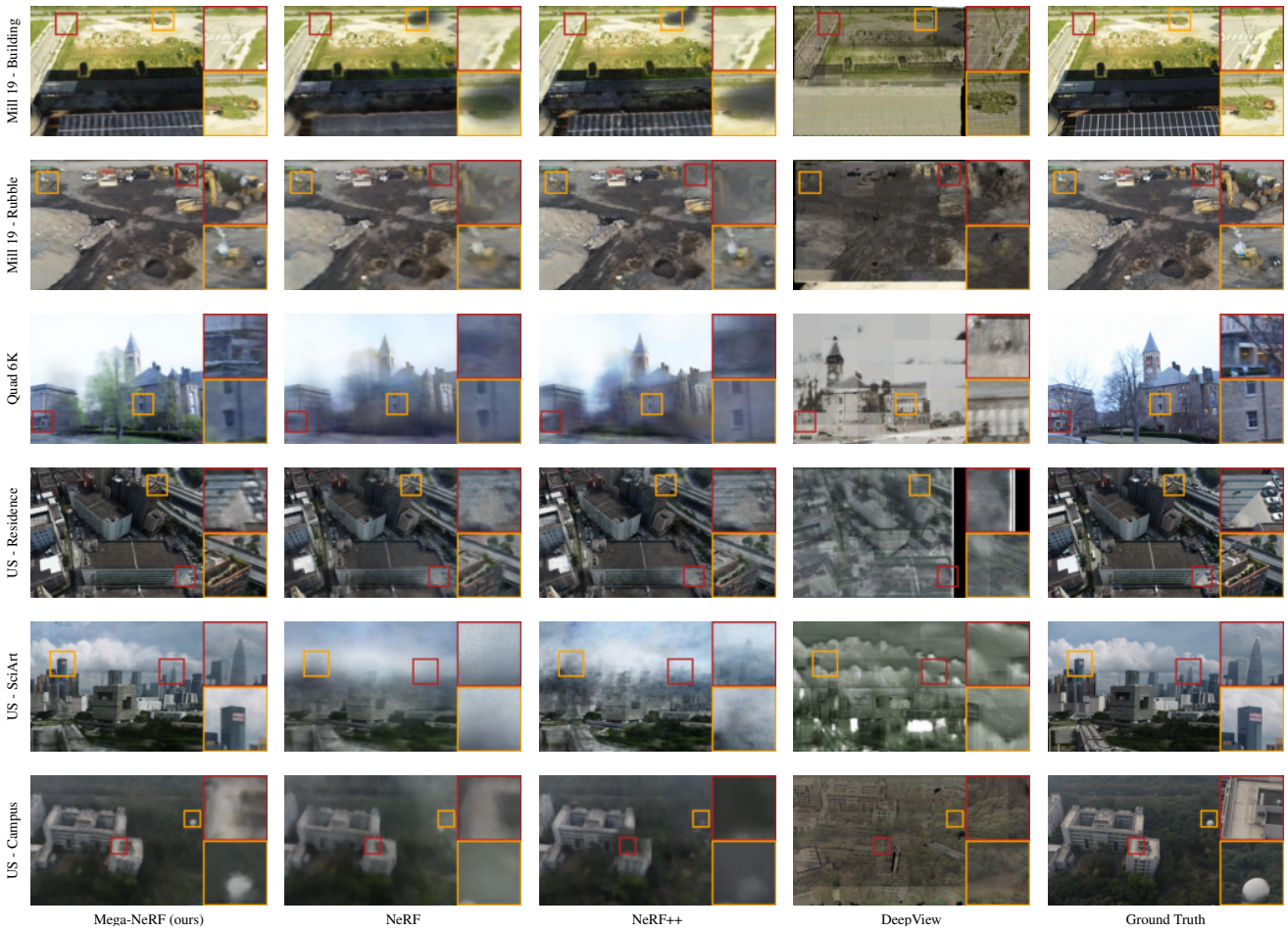


Figure 9. **Scalable training.** Mega-NeRF generates the highest-quality reconstructions while avoiding the numerous visual artifacts present in other approaches.

	Mill 19 - Building				Mill 19 - Rubble				Quad 6k			
	↑PSNR	↑SSIM	↓LPIPS	↓Time (h)	↑PSNR	↑SSIM	↓LPIPS	↓Time(h)	↑PSNR	↑SSIM	↓LPIPS	↓Time(h)
NeRF	18.21	0.406	0.665	59:51	20.30	0.433	0.656	60:21	16.29	0.395	0.704	62:48
NeRF++	18.70	0.444	0.596	89:02	20.65	0.474	0.588	90:42	16.43	0.396	0.698	90:34
DeepView	10.52	0.184	0.645	44:45	10.62	0.238	0.684	44:51	9.951	0.283	0.695	39:51
Mega-NeRF	**19.67**	**0.482**	**0.567**	**29:49**	**22.65**	**0.494**	**0.584**	**30:48**	**17.69**	**0.527**	**0.660**	**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	17.77	0.434	0.658	62:40	16.71	0.441	0.723	90:48	17.60	0.392	0.741	61:56
NeRF++	18.04	0.474	0.614	90:48	16.91	0.483	0.664	95:00	17.49	0.393	0.744	93:50
DeepView	11.50	0.119	0.674	51:46	10.28	0.289	0.784	47:54	13.26	0.237	**0.680**	52:48
Mega-NeRF	**20.68**	**0.535**	**0.580**	**27:20**	**19.12**	**0.550**	**0.622**	**27:39**	**18.75**	**0.429**	0.770	**29:03**

Table 3. **Scalable training.** We compare Mega-NeRF to NeRF, NeRF++, and DeepView after 500,000 iterations. Mega-NeRF consistently outperforms the baselines on all metrics even after allowing other approaches to train well beyond 24 hours.

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 4. Although Mega-NeRF-Plenoctree renders most quickly, voxelization has a large visual impact. Plenoctree's use of spherical harmonics generated inplausible shadings as shown in Figure 10. Mega-NeRF-KiloNeRF comes close to interactivity at 0.55 FPS but still suffers from blurriness and noticeable visual artifacts. Its knowledge distillation and finetuning processes also require over a day of additional processing. In contrast,

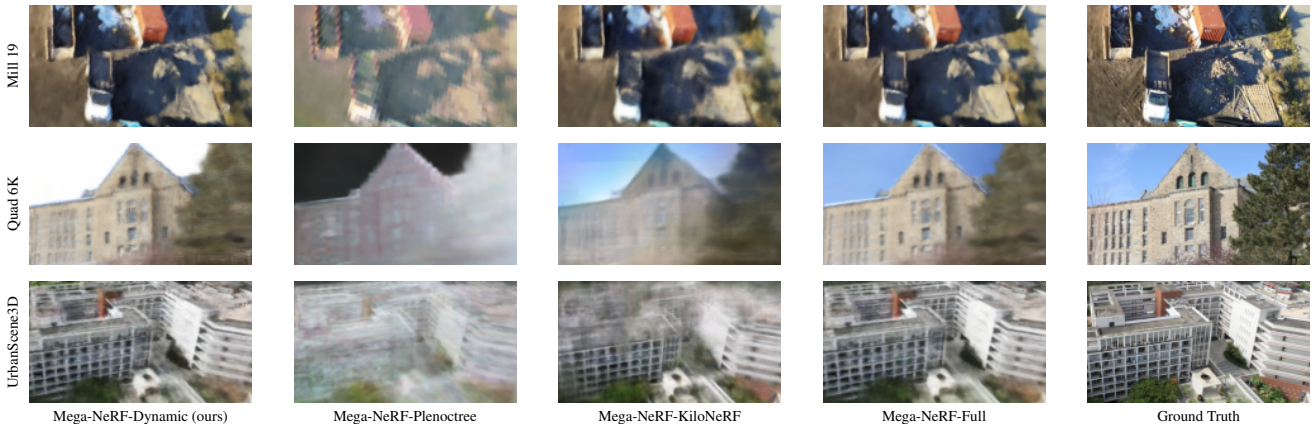


Figure 10. **Interactive rendering.** Plenoctree's approach causes significant voxelization and KiloNeRF's results suffer from blurriness and visual artifacts. Refer to Figure 15 in the supplement for results across all datasets.

best second-best	Mill 19					Quad 6k					UrbanScene3D				
	↑PSNR	↑SSIM	↓LPIPS	Train Time (h)	Render Time (s)	↑PSNR	↑SSIM	↓LPIPS	Train Time (h)	Render Time (s)	↑PSNR	↑SSIM	↓LPIPS	Train Time (h)	Render Time (s)
Mega-NeRF-Plenoctree	15.39	0.389	0.692	1:26	**0.086**	13.41	0.494	0.641	1:33	**0.030**	14.65	0.490	0.627	**1:07**	**0.047**
Mega-NeRF-KiloNeRF	20.24	0.460	0.580	30:03	1.534	17.34	0.504	0.576	27:33	2.469	19.87	0.574	0.510	34:00	1.977
Mega-NeRF-Full	**21.45**	**0.510**	0.548	-	311.14	**18.32**	**0.518**	0.570	-	518.86	**20.68**	**0.622**	0.489	-	423.14
Mega-NeRF-Initial	16.91	0.383	0.602	**1:08**	0.235	13.40	0.411	0.546	**1:31**	0.214	15.86	0.497	0.520	1:10	0.221
Mega-NeRF-Dynamic	21.21	0.497	**0.546**	**1:08**	10.617	17.78	0.485	**0.524**	**1:31**	7.986	20.19	0.594	**0.485**	1:10	9.719

Table 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, 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 0.5 db in PSNR quality while accelerating rendering by 40x relative to conventional ray sampling.

	Mill 19			Quad 6k			UrbanScene3D		
	↑PSNR	↑SSIM	↓LPIPS	↑PSNR	↑SSIM	↓LPIPS	↑PSNR	↑SSIM	↓LPIPS
Mega-NeRF-no-embed	19.45	0.457	0.600	15.99	0.453	0.684	17.36	0.460	0.683
Mega-NeRF-no-bounds	18.48	0.414	0.658	17.37	0.524	0.661	18.33	0.461	0.702
Mega-NeRF-dense	20.75	0.464	0.600	17.60	0.521	0.667	19.37	0.492	0.667
Mega-NeRF-joint	20.04	0.478	0.651	17.03	0.518	0.670	18.56	0.485	0.669
Mega-NeRF	**21.07**	**0.485**	**0.579**	**17.62**	**0.527**	**0.662**	**19.53**	**0.504**	**0.659**

Table 5. **Diagnostics.** We compare Mega-NeRF to various ablations after 24 hours of training. Each individual component contributes significantly to overall model performance.

Mega-NeRF-Dynamic remains within 0.5 db 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.

5. Conclusion

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.

References

[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. 2

[2] Pix4dmapper. https://www.pix4d.com/product/pix4dmapper-photogrammetry-software. Accessed: 2021-11-01. 5

[3] 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. 3

[4] David Crandall, Andrew Owens, Noah Snavely, and Daniel Huttenlocher. Discrete-continuous optimization for large-scale structure from motion. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011. 2, 5, 11

[5] Yilun Du, Yinan Zhang, Hong-Xing Yu, Joshua B. Tenenbaum, and Jiajun Wu. Neural radiance flow for 4d view synthesis and video processing. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021. 11

[6] 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. 2018. 2

[7] 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 *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2367–2376, 2019. 6

[8] 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. 3

[9] Chen Gao, Ayush Saraf, Johannes Kopf, and Jia-Bin Huang. Dynamic view synthesis from dynamic monocular video. In *Proceedings of the IEEE International Conference on Computer Vision*, 2021. 11

[10] Stephan J Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin. Fastnerf: High-fidelity neural rendering at 200fps. *arXiv preprint arXiv:2103.10380*, 2021. 3

[11] Peter Hedman, Pratul P. Srinivasan, Ben Mildenhall, Jonathan T. Barron, and Paul Debevec. Baking neural radiance fields for real-time view synthesis. *ICCV*, 2021. 2

[12] Yoonwoo Jeong, Seokjun Ahn, Christopher Choy, Animashree Anandkumar, Minsu Cho, and Jaesik Park. Self-calibrating neural radiance fields. In *ICCV*, 2021. 11

[13] Yuhe Jin, Dmytro Mishkin, Anastasiia Mishchuk, Jiri Matas, Pascal Fua, Kwang Yi, and Eduard Trulls. Image matching across wide baselines: From paper to practice, 03 2020. 2, 3

[14] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014. 6

[15] 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. 1, 2

[16] Chen-Hsuan Lin, Wei-Chiu Ma, Antonio Torralba, and Simon Lucey. Barf: Bundle-adjusting neural radiance fields. In *IEEE International Conference on Computer Vision (ICCV)*, 2021. 11

[17] Philipp Lindenberger, Paul-Edouard Sarlin, Viktor Larsson, and Marc Pollefeys. Pixel-Perfect Structure-from-Motion with Featuremetric Refinement. In *ICCV*, 2021. 11

[18] Yilin Liu, Fuyou Xue, and Hui Huang. Urbanscene3d: A large scale urban scene dataset and simulator. 2021. 2, 5, 11

[19] Ricardo Martin-Brualla, Noha Radwan, Mehdi S. M. Sajjadi, Jonathan T. Barron, Alexey Dosovitskiy, and Daniel Duckworth. NeRF in the Wild: Neural Radiance Fields for Unconstrained Photo Collections. In *CVPR*, 2021. 2, 3, 4, 5, 6

[20] 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 *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021. 11

[21] 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. 2

[22] 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. 1, 2, 3, 4, 6

[23] 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. 11

[24] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. 11

[25] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps, 2021. 2, 3

[26] 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. 3

[27] 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. 3

[28] 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 *IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2021. 11

[29] Alex Trevithick and Bo Yang. Grf: Learning a general radiance field for 3d scene representation and rendering. In *arXiv:2010.04595*, 2020. 3, 11

[30] 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. 11

[31] 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. 3

[32] 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. 6

[33] William T. Weldon and Joseph Hupy. Investigating methods for integrating unmanned aerial systems in search and rescue operations. *Drones*, 4(3), 2020. 2

[34] 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. *CoRR*, abs/2105.09103, 2021. 4, 11

[35] 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. 2, 5, 6

[36] 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. 3, 11

[37] 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. 2

[38] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. Nerf++: Analyzing and improving neural radiance fields, 2020. 1, 2, 3, 4, 6

[39] 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. 6

Supplemental Materials

A. Scaling properties

Model scaling. We further 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 25 submodules and 128, 256, and 512 channels respectively. We summarize our findings in Table 6. Increasing the model capacity along either dimension improves rendering quality, as depicted in Figure 11. However, although increasing the channel count severely penalizes training and rendering speed, the number of submodules has little impact.

Cell repartitioning. 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, Section 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.

We provide some preliminary insight into the impact of this optimization by constructing repartitioned datasets using trained Mega-NeRF models. We then compare the reconstruction quality of a model trained against this repartitioned set against an equivalent model trained against the original partitions. We summarize our findings in Table 7. We also measure the average fraction of pixels each submodule is trained against relative to the total number in the scene.

B. Limitations

Although our work presents a first step towards scaling NeRF to handle large-scale view synthesis, several obstacles remain ahead of deploying them in practice. Pose accuracy is arguably the largest limiting factor. The initial models we trained using raw camera poses collected from standard drone GPS and IMU sensors were extremely blurry. One hardware solution would be to use higher-accuracy RTK GPS modules when collecting footage. We also tested BARF's [16] coarse-to-fine adjustment method to improve our poses but found that the results were generally inferior to those provided by Pix4DMapper. SCNeRF [12], GNeRF [20], and PixSFM [17] are all recent alternatives that merit further exploration.

While our renderer avoids the pitfalls of existing fast NeRF approaches, it does not quite reach the throughput needed for truly interactive applications. We explored uncertainty-based methods as detailed in [34] to further improve sampling efficiency, but open challenges remain.

We also did not explicitly address dynamic scenes within our work, a relevant factor for many human-centered use cases. Several recent NeRF-related efforts, including NR-NeRF [28], Nerfies [23], NeRFlow [5], and DynamicMVS [9] focus on dynamism, but we theorize that scaling these approaches to larger urban scenes will require additional work.

Finally, although our training process is several factors quicker than previous works, NeRF training time remains a significant bottleneck towards rapid model deployment. Recent methods such as pixelNeRF [36] and GRF [29] that introduce conditional priors would likely complement our efforts but necessitate gathering similar data to the scenes that we target. We hope that our Mill 19 dataset, in addition to existing collections such as UrbanScene3D [18], will serve as a valuable contribution.

C. 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.

We are exploring the technique of "denaturing" first described by Wang et al [30] 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 plan on investigating this further in the future.

D. Assets

Mill 19 dataset. We plan on contributing our Mill 19 dataset along with our calibrated poses to the wider research community. We collected our data using the Parrot ANAFI drone pictured in Figures 13 and 14. We captured photos in the grid pattern illustrated in Figure 12. In addition to FAA approval, we also received permission from the city to fly in the area and ensured that no non-consenting people were captured in the data.

Third-party assets. The main third-party assets we used were the UrbanScene3D [18] dataset, the Quad 6k dataset [4], and Pytorch [24], all of which are cited in our main paper. Pytorch uses a BSD license and the UrbanScene3D dataset is freely available for research and education use only. The Quad 6k dataset does not include an explicit license but is freely available at http://vision.soic.indiana.edu/projects/disco/.

E. Additional results

We include additional interactive rendering results across all datasets in Figure 15 to complement those shown in Figure 10.

	1 Submodule					4 Submodules					9 Submodules					25 Submodules				
	↑PSNR	↑SSIM	↓LPIPS	Train Time (h)	Render Time (s)	↑PSNR	↑SSIM	↓LPIPS	Train Time (h)	Render Time (s)	↑PSNR	↑SSIM	↓LPIPS	Train Time (h)	Render Time (s)	↑PSNR	↑SSIM	↓LPIPS	Train Time (h)	Render Time (s)
128 Channels	21.15	0.422	0.701	**18:54**	**5.090**	21.65	0.444	0.664	**18:56**	**5.222**	21.90	0.462	0.637	**19:01**	**5.239**	21.89	0.486	0.603	**19:02**	**5.263**
256 Channels	21.81	0.449	0.649	28:54	8.069	22.34	0.478	0.602	29:09	8.199	22.62	0.502	0.574	29:13	8.224	22.79	0.530	0.541	29:14	8.241
512 Channels	**22.34**	**0.476**	**0.605**	52:33	15.827	**22.88**	**0.515**	**0.555**	52:34	15.966	**23.02**	**0.538**	**0.527**	53:36	15.999	**23.24**	**0.572**	**0.492**	53:45	15.994

Table 6. **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).

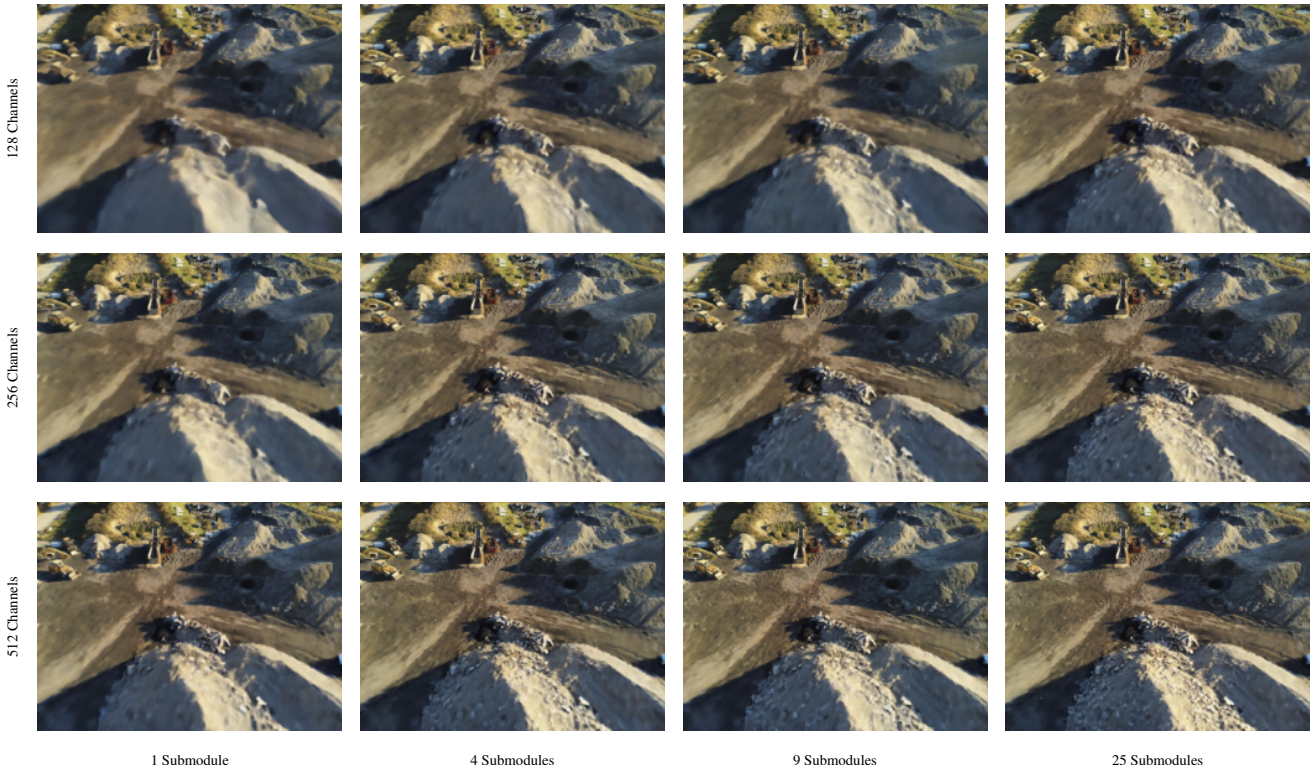


Figure 11. **Model scaling.** Example rendering within our Mill 19 - Rubble dataset across different numbers of submodules (**columns**) and channels per submodule (**rows**). Mega-NeRF generates increasingly photo-realistic renderings as capacity increases. Increasing the number of submodules increases the overall model capacity without penalizing training time or inference time.

	4 Submodules				9 Submodules				25 Submodules			
	↑PSNR	↑SSIM	↓LPIPS	↓Pixels	↑PSNR	↑SSIM	↓LPIPS	↓Pixels	↑PSNR	↑SSIM	↓LPIPS	↓Pixels
Original Data	22.34	0.478	0.602	0.389	22.62	0.502	0.574	0.218	22.79	0.530	0.541	0.107
Repartitioned Data	**22.56**	**0.492**	**0.585**	**0.300**	**22.75**	**0.522**	**0.546**	**0.142**	**23.04**	**0.565**	**0.500**	**0.057**

Table 7. **Pixel Repartitioning.** 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 repartition pixels. 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.

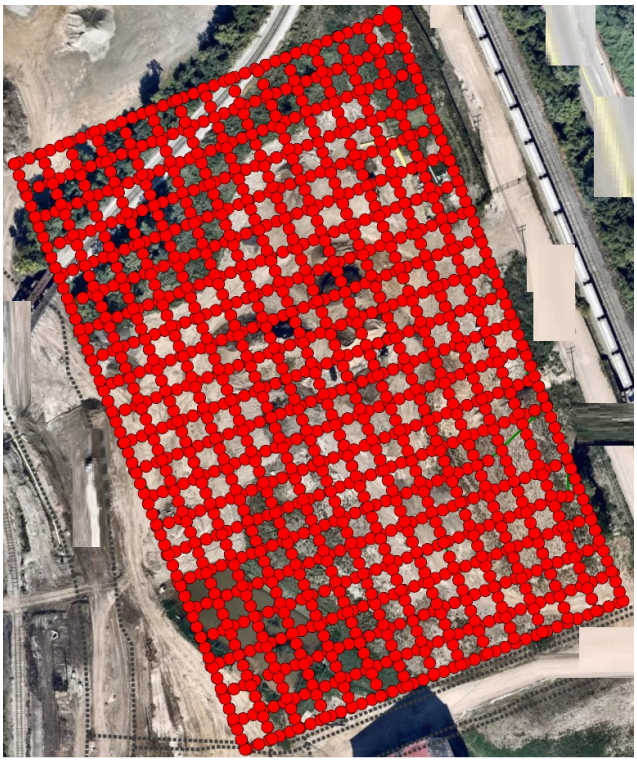


Figure 12. **Data collection**. We collect our poses for our Mill 19 dataset using a grid pattern as shown. Collecting footage across a 200,000 m^2 area takes approximately 2 hours.

Figure 13. **Parrot ANAFI drone**. The drone used to collect data for the Mill 19 dataset. The drone comes equipped with a 4K Camera, GPS, and an inertial measurement unit (IMU) from which we derive initial camera poses.

Figure 14. Our drone flying above the Mill 19 - Building scene.

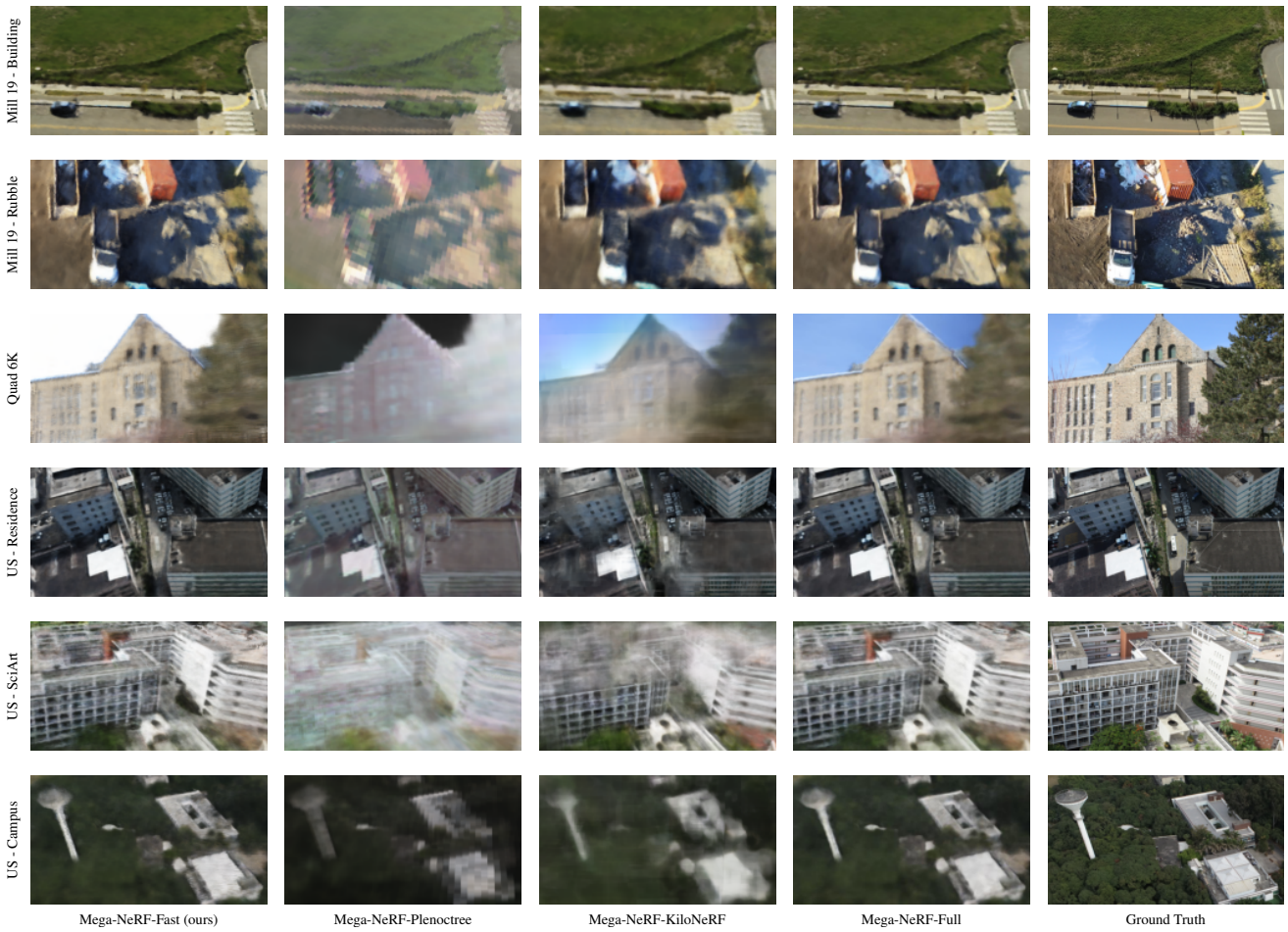


Figure 15. **Interactive rendering.** Plenoctree's approach causes significant voxelization and KiloNeRF's results suffer from blurriness and visual artifacts.