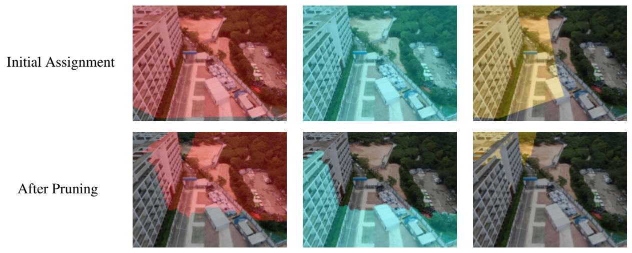

Figure 8. **Data pruning.** The initial assignment of pixels to cells is based purely on camera positions. We add each pixel to the training set of **all** cells it traverses, leading to overlap between sets **(top)**. After the model gains a 3D understanding of the scene, we can filter irrelevant pixels by instead assigning pixels based on camera ray intersection with solid surfaces **(bottom)**.

Supplemental Materials

A. 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. 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. 8).

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 5.

B. Scaling properties

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 16 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 Fig. 9. However, although increasing the channel count severely penalizes training and rendering speed, the number of submodules has less impact.

C. Dynamic octree generation

The maximum tree size used by Mega-NeRF-Dynamic is bounded by available GPU memory, which we set 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.

D. Baselines

Multi-view stereo. Although scaling dense multi-view stereo remains an open research problem [10], we optimize for the best possible reconstructions instead of training time for the purpose of our evaluation. We use COLMAP to generate meshes with Poisson surface reconstruction. We found that this method failed to generate reasonable results for scenes containing many sky pixels. We therefore use foreground masks generated from a trained Mega-NeRF model to mask background regions during surface reconstruction to achieve better results.

Stable View Synthesis. We train the network representing each scene from scratch for 600,000 iterations. Stable View Synthesis relies on a geometric scaffold containing depth information and we use the meshes generated from COLMAP for this purpose.

DeepView. We base our DeepView baseline on a publicly available implementation. We use 3 blocks of 24 channels and train our model for 200,000 iterations using random 200 x 100 crops of the input views. During training, we randomly sample nearby input views for a given target view as determined by the capture time of each image.

E. Limitations

Pose accuracy. 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. As alternatives to PixSFM [19], we experimented with refining our camera poses with BARF's [18] coarse-to-fine adjustment and Pix4DMapper [2], a commercial drone mapping solution. Our results were uniformly better with the PixSFM poses, with a PSNR gap of over 6 db relative to the second-best solution (Pix4DMapper). SC-NeRF [14] and GNeRF [22] are other recent alternatives that merit further exploration. Another hardware-based solution would be to use higher-accuracy RTK GPS modules when collecting footage.

	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 5. **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.

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

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).

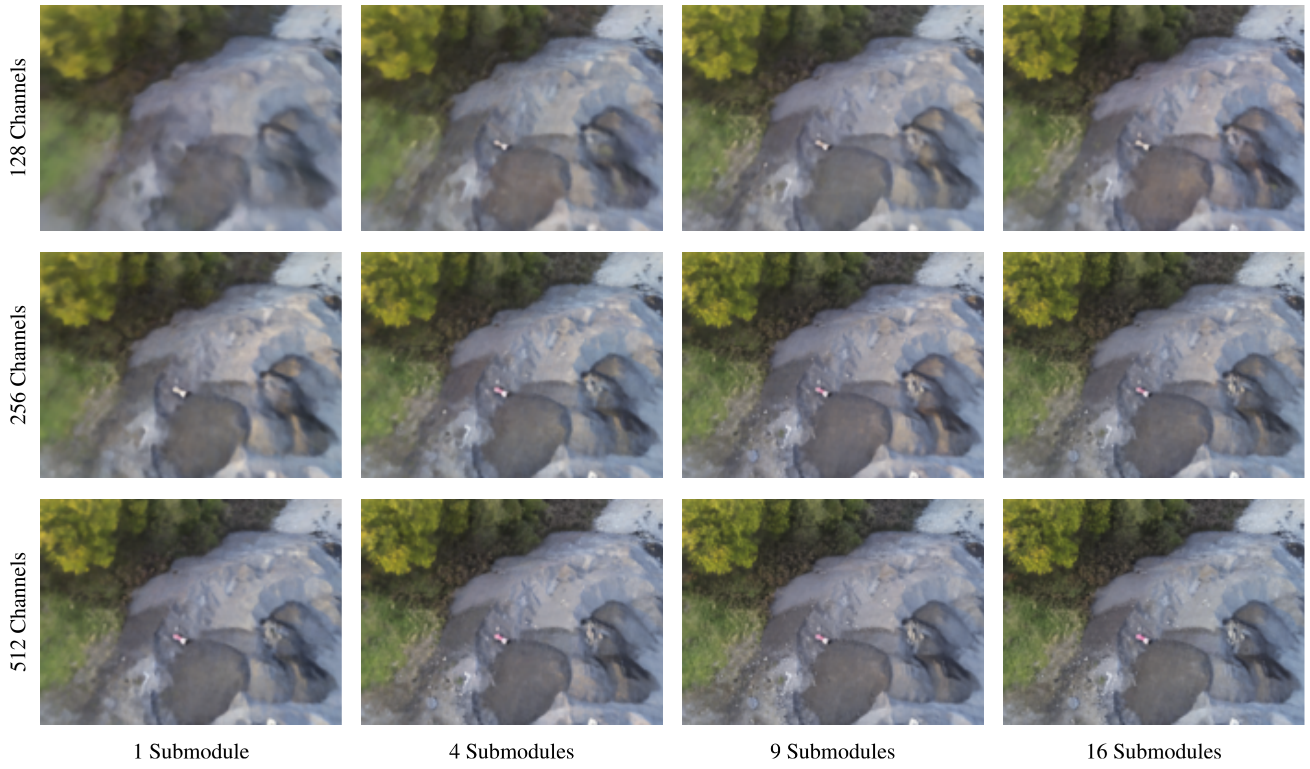

Figure 9. **Model scaling.** Example rendering within our Mill 19 - Rubble dataset across different numbers of submodules (**columns**) and channels per submodule (**rows**). Mega-NeRF generates increasingly photo-realistic renderings as capacity increases. Increasing the number of submodules increases the overall model capacity with little impact to training and inference time.

Dynamic objects. We 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 [37], Nerfies [26], NeRFlow [5], and DynamicMVS [11] focus on dynamism, but we theorize that scaling these approaches to larger urban scenes will require additional work.

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.

Rendering speed. While our renderer avoids the pitfalls of existing fast NeRF approaches, it does not quite reach the throughput needed for truly interactive applications. We

Figure 10. **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.

explored uncertainty-based methods as detailed in [44] to further improve sampling efficiency, but open challenges remain.

Training speed. 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 [46] and GRF [38] 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 [20], will serve as a valuable contribution.

F. 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 [39] 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.

G. Assets

Mill 19 dataset. We have publicly released 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 Figs. 10 and 11. We captured photos in the grid pattern illustrated in Fig. 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.

Figure 11. Our drone flying above the Mill 19 - Building scene.

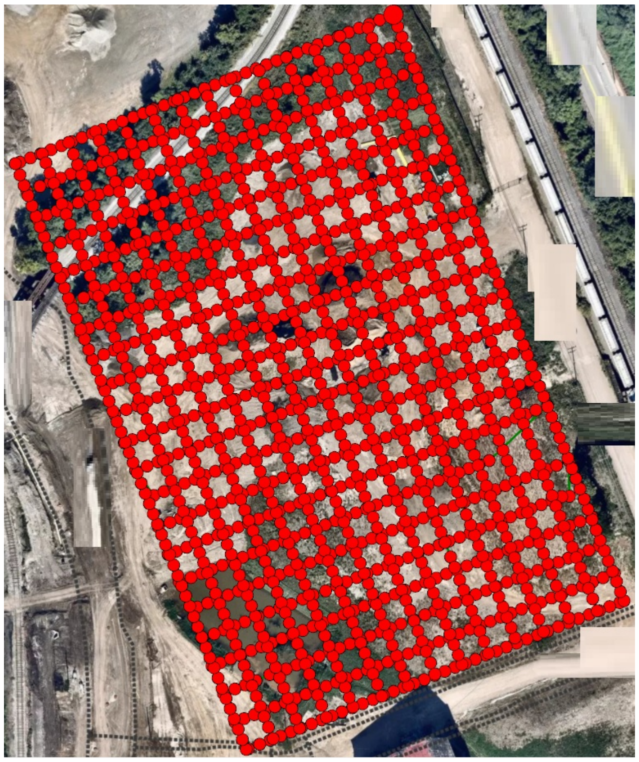

Figure 12. **Data collection**. We collect our poses for our Mill 19 dataset using a grid pattern as shown. Collecting footage across a 200,000 m^2 area takes approximately 2 hours.

Third-party assets. The main third-party assets we used were the UrbanScene3D [20] dataset, the Quad 6k dataset [4], and Pytorch [27], all of which are cited in our main paper. Pytorch uses a BSD license and the Urban-

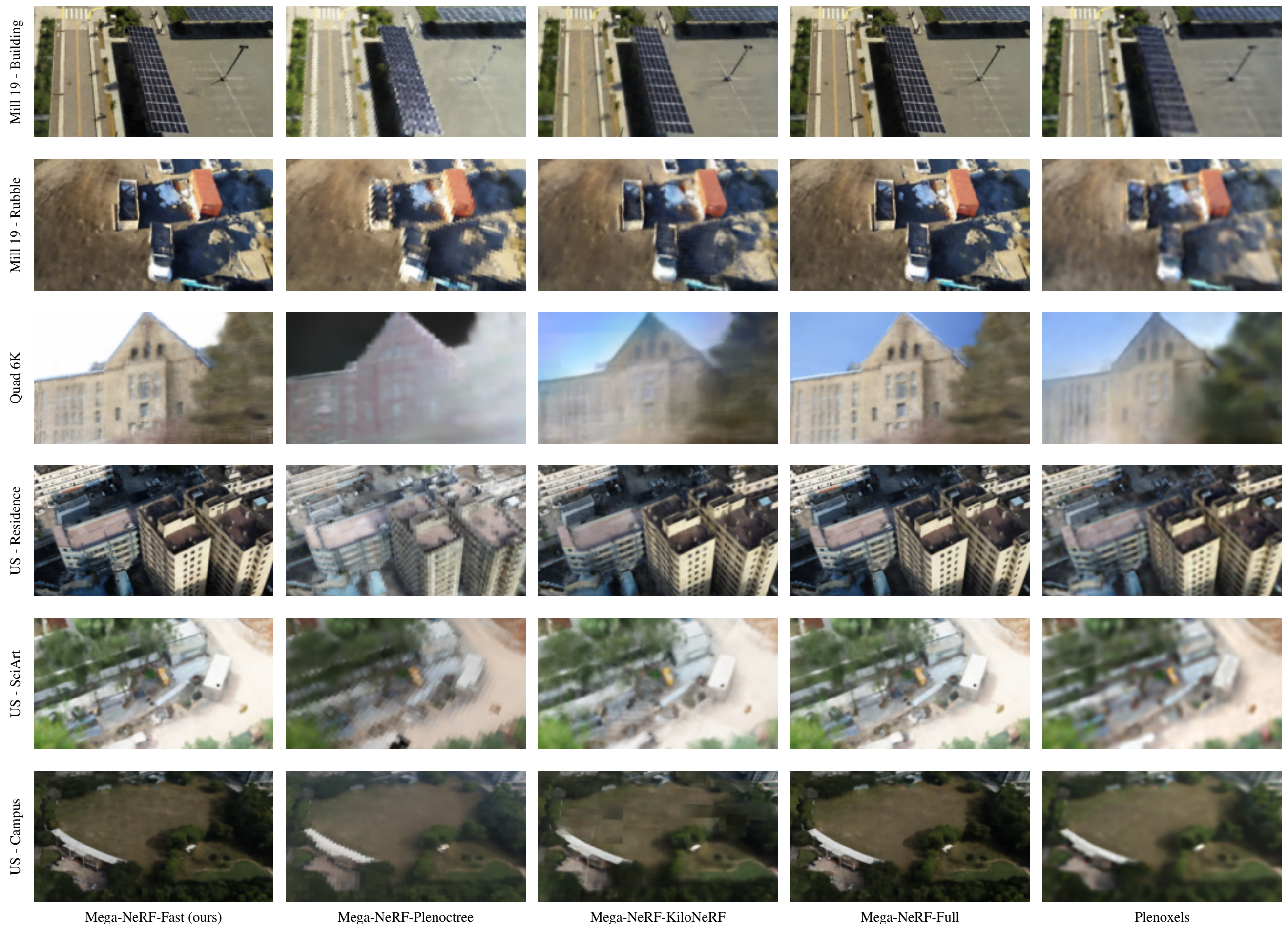

Figure 13. Additional interactive rendering results.

Scene3D 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/.

H. Dataset statistics

Visibility. We generate the visibility statistics in Table 1 by first training a NeRF model for each scene. As in Sec. A, we compute and store opacity values into a fixed resolution structure and project camera rays for all images in the scene. We then measure the proportion of surface voxels each image's rays intersects relative to the total number in the scene.

Additional datasets. We provide top-level statistics for commonly used view synthesis datasets in Table 7 to complement those in Table 1.

I. Additional results

We include additional interactive rendering results across all datasets in Fig. 13 to complement those in Fig. 7.

	Resolution	# Images	# Pixels/Rays
Synthetic NeRF [24]	400 x 400	400	256,000,000
LLFF [23]	4032 x 3024	41	496,419,840
Light Field [47]	1280 x 720	214	195,910,200
Tanks and Temples [17]	1920 x 1080	283	587,658,240
Phototourism [15]	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 [20]	5232 x 3648	3824	74,102,106,112

Table 7. 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 [48] and [45] respectively. For Phototourism we average across the scenes used in [21].