Plenoxels: Radiance Fields without Neural Networks

Supplementary Material

A. Overview

In the supplementary material, we include additional experimental details and present results and visualizations of further ablation studies. We also present full, per-scene quantitative and visual comparisons between our method and prior work. We encourage the reader to see the video for results of our method on a wide range of scenes.

B. Experimental Details

B.1. Implementation Details

As briefly discussed in Sec. 3.2, we use a simple data structure which consists of a data table in addition to a dense grid, where each cell is either NULL or a pointer into the data table. Each entry in the data table consists of the density value and the SH coefficients for each of the RGB color channels. NULL cells are considered to have all 0 values. This data structure allows for reasonably efficient trilinear interpolation both in the forward and backward passes while maintaining sparsity; due to the relatively large memory requirements to store the SH coefficients, gradients, and RMSProp running averages, the dense pointer grid is usually not dominant in size. Nevertheless, reading the pointers currently appears to take a significant amount of rendering time, and optimizations are likely possible.

Our main CUDA rendering and gradient kernels simultaneously parallelize across rays, colors, and SH coefficients. Each CUDA warp (32 threads) handles one ray, with threads processing one SH coefficient each; since coefficients are stored contiguously, this means access to global memory is highly coalesced. The SH coefficients are combined into colors using warp-level operations from NVIDIA CUB [25]. These features are particularly significant in the case of trilinear interpolation.

Note that in order to correctly perform trilinear color interpolation, instead of using the sigmoid function to ensure that predicted sample colors are always between 0 and 1 as in NeRF [28], we simply clip negative color values to 0 with a ReLU to preserve linearity as much as possible.

We use weight-based thresholding (as in PlenOctrees [59]) for the synthetic and real, 360° scenes, and density-based thresholding for the forward-facing scenes. The reason for this is that some content (especially at the edges) in the forward-facing scenes is not visible in most of the training views, so weight-based thresholding tends to prune these sparsely-supervised features.

We use a batch size of 5000 rays and optimize with RMSProp [11]. For σ we use the same delayed exponential learning rate schedule as Mip-NeRF [3], where the exponential is scaled by a learning rate of 30 (this is where the exponential would start, if not for the delay) and decays to 0.05 at step 250000, with an initial delay period of 15000 steps. For SH we use a pure exponential decay learning rate schedule, with an initial learning rate of 0.01 that decays to 5×10^{-6} at step 250000.

The TV losses are evaluated stochastically; they are applied only to 1% of all voxels in the grid in each step. Note that empty voxels can be selected, as their neighbors may not be empty. In practice, for performance reasons, we always apply the TV regularization on random contiguous segments of voxels (in the order that the pointer grid is stored). This is much faster to evaluate on the GPU due to locality. In all cases, the voxel differences in the TV loss defined below Eq. (3) is in practice normalized by the voxel resolution in each dimension, relative to 256 (for historical reasons):

$$\Delta_x((i,j,k),d) = \frac{|V_d(i+1,j,k) - V_d(i,j,k)|}{256/D_x} \qquad (6)$$

Where D_x is the grid resolution in the x dimension, and $V_d(i,j,k)$ is the dth value of voxel (i,j,k) (either density or a SH coefficient). We scale Δ_y, Δ_z analogously. Note that the same loss is applied in NDC and to the background model, except in the background model, the TV also wraps around the edges of the equirectangular image. For SH, empty grid cells and edges are considered to have the same value as the current cell (instead of 0) for purposes of TV.

B.2. Synthetic experiments

On the synthetic scenes, we found that our method performs nearly identically when TV regularization is present only in the first stage of optimization; turning off the regularization after pruning voxels and increasing resolution reduces our training time modestly. We suspect (see Tab. 3) this is due to the large number of training views (100) available for these scenes as well as the low level of noise; for the other datasets we retain TV regularization throughout optimization.

We start at resolution 256^3, prune and upsample to resolution 512^3 after 38400 steps (the equivalent of 3 epochs), and optimize for a total of 128000 steps (the equivalent of 10 epochs). We prune using a weight threshold of 0.256, and use λ_{TV} of 1×10^{-5} for σ and 1×10^{-3} for SH, only during the initial 38400 steps (and then turn off regularization after pruning and upsampling, for faster optimization).

B.3. Forward-facing experiments

For the forward-facing scenes we start at resolution $256 \times 256 \times 128$, prune and upsample to resolution $512 \times 512 \times 128$ at step 38400, prune and upsample to resolution $1408 \times 1156 \times 128$ at step 76800, and optimize for a total of 128000 steps. The final grid resolution is derived from the image resolution of the dataset, with some padding added on each side. We prune using a σ threshold of 5, use λ_{TV} of 5×10^{-4} for σ and 5×10^{-3} for SH, and use a sparsity penalty λ_s of 1×10^{-12} to encourage empty voxels.

While these TV parameters work well for the forward-facing NeRF scenes, more generally, we find that sometimes it is preferrable to use λ_{TV} 5×10^{-3} for density and 5×10^{-2} for SH, which reduces artifacts while blurring the scene more. This is used for some of the examples in the video, for example the piano. In general, since scenes differ significantly in content, camera noise, and actual scale, a hyperparameter sweep of the TV weights can be helpful, and using different TV values across the scenes would improve the metrics for the NeRF scenes as well.

B.4. 360° experiments

For the 360° scenes our foreground Plenoxel grid starts at resolution 128^3; we prune and upsample to 256^3, 512^3, and 640^3 with 25600 steps in between each upsampling. We optimize for a total of 102400 steps. We prune using a weight threshold of 1.28, and use λ_{TV} of 5×10^{-5} for σ and 5×10^{-3} for SH for the inner grid and λ_{TV} of 1×10^{-3} for both σ and SH for the 64 background grid layers of resolution 2048×1024. We use λ_s of 1×10^{-11} and λ_β of 1×10^{-5}. For simplicity of implementation, we did not use coarse-to-fine for the background and only use σ thresholding. We also do not use the delayed learning rate function for the background, opting instead to use an exponential decay to allow the background to optimize faster than the foreground at the beginning.

While the TV weights were fixed for these scenes, in general, a hyperparameter sweep of the TV weights can be helpful. For more general scenes, it is sometimes useful to use a near-bound on the camera rays (as in NeRF) to prevent floaters very close to the camera, or to only begin optimizing the foreground after, say, 1000 iterations. Further sparsity losses to encourage the weight distribution to be a delta function may also help.

C. Ablation Studies

We visualize ablations on the synthetic lego scene in Fig. 10. In addition to comparing nearest neighbor and trilinear interpolation, we also experimented with tricubic interpolation, which produces a function approximation that is both continuous (like trilinear interpolation) and smooth. However, we found tricubic interpolation offered negligible improvements compared to trilinear, in exchange for a substantial increase in computation (this increase in computation is why we do not include a full ablation table for tricubic interpolation).

LR Schedule	PSNR ↑	SSIM ↑	LPIPS ↓
Exp for SH, Delayed for σ [3]	30.57	0.950	0.065
Exp for SH and σ	30.58	0.950	0.066
Exp for SH, Constant for σ	30.37	0.948	0.068
Constant for SH and σ	30.13	0.945	0.075

Table 6. **Comparison of different learning rate schedules** for σ (voxel density) and spherical harmonics (SH), with fixed resolution 256^3 and RMSProp [11]. Results are averaged over the 8 synthetic scenes from NeRF [28]. Our method is robust to variations in learning rate schedule.

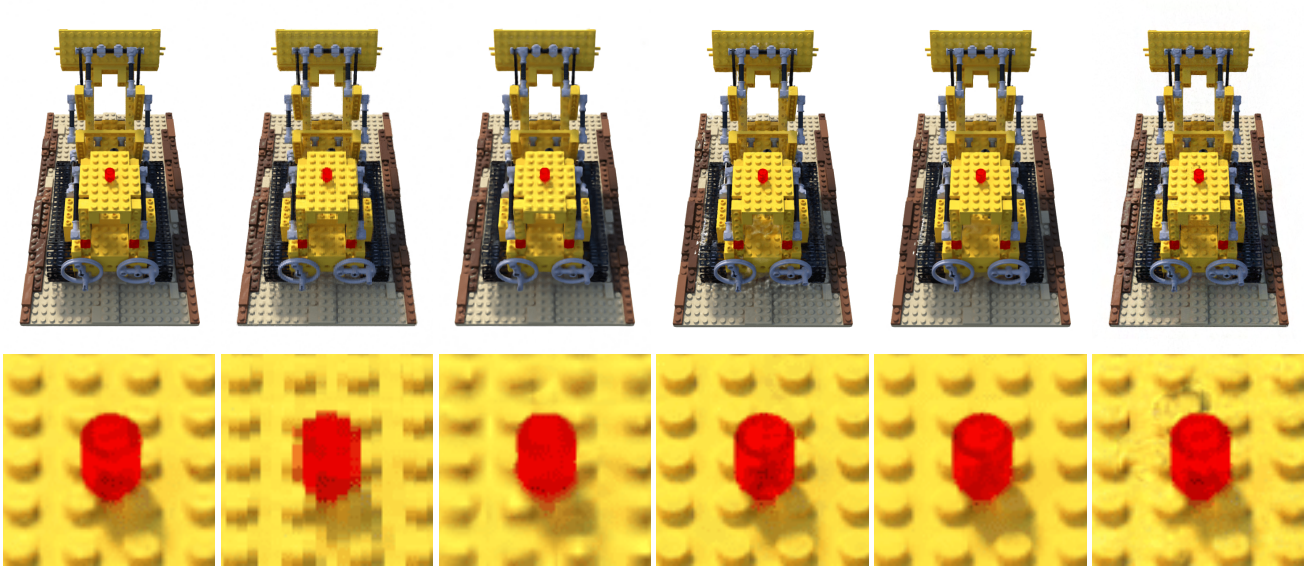
Optimizer	PSNR ↑	SSIM ↑	LPIPS ↓
RMSProp [11] for SH and σ	30.57	0.950	0.065
RMSProp for SH, SGD for σ	30.20	0.946	0.072
SGD for SH, RMSProp for σ	29.82	0.940	0.076
SGD for SH and σ	29.35	0.932	0.087

Table 7. **Comparison of different optimizers** for σ and SH, with fixed resolution 256^3. Results are averaged over the 8 synthetic scenes from NeRF [28]. Our method is robust to variations in optimizer, although there is a benefit to RMSProp particularly for optimizing the spherical harmonic coefficients.

Regularizer	PSNR ↑	SSIM ↑	LPIPS ↓
TV SH, TV σ, Sparsity	26.29	0.839	0.210
- Sparsity	26.31	0.839	0.210
- TV σ	25.25	0.807	0.226
- TV SH	25.80	0.814	0.234

Table 8. **Ablation over regularization.** Results are averaged over the 8 forward-facing scenes from NeRF, which are particularly sensitive to regularization due to the low number of training views. We find that the sparsity regularizer is not necessary for quality, but we retain it to reduce memory footprint. TV regularization is essential for σ but also important for spherical harmonics, as visualized in Fig. 3, even though this effect is not as pronounced in the PSNR metric. Without any TV regularization (on SH or σ), three of the eight scenes run out of memory on our GPU.

Tab. 6 and Tab. 7 show ablations over learning rate schedule and optimizer, respectively. We find that Plenoxel optimization is reasonably robust to both of these hyperparameters, although there is a noticeable improvement from using RMSProp compared to SGD, particularly for the spherical harmonic coefficients. Note that when comparing different learning rate schedules and optimizers, we tune the initial learning rate separately for each row to provide the best



(a) Trilinear, 256^3 (b) Nearest, 256^3 (c) Trilinear, 128^3 (d) 25 Images, Low TV (e) 25 Images, High TV (f) NV Formula

Figure 10. **Visual results of ablation studies** on the synthetic lego scene. Trilinear interpolation at resolution 256^3 is quite similar to our full model at resolution 512^3. Nearest neighbor interpolation shows clear voxel artifacts. Trilinear interpolation at lower resolution appears less detailed. Reducing the number of training views produces visual artifacts that are mostly resolved by increasing the TV regularization. Optimizing and rendering with the Neural Volumes [22] formula produces different visual artifacts.

results possible for each configuration.

Tab. 8 shows ablation over regularization, for the forward-facing scenes. We find that TV regularization is important for these scenes, likely due to their low number of training images. Regularization on density has a quantitatively larger effect than regularization on spherical harmonics, but both are important for avoiding visual artifacts (see Fig. 3).

Tab. 5 compares the performance of Plenoxels when trained with the rendering formula used in NeRF (originally from Max [24]) and when trained with the rendering formula used in Neural Volumes [22]. The Max formula is defined in Eq. (1) and rewritten here in a slightly more convenient format:

$$T_i = \exp\left(-\sum_{j=1}^{i-1} \sigma_j \delta_j\right) \tag{7}$$

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} (T_i - T_{i+1})\mathbf{c}_i \tag{8}$$

The Neural Volumes formula can be written as:

$$T_i = \min\left\{1,\ \sum_{j=1}^{i-1} \exp(-\delta_i \sigma_i)\right\} \tag{9}$$

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} (T_i - T_{i+1})\mathbf{c}_i \tag{10}$$

where $\exp(-\sigma_i)$ is modeled directly rather than modeling σ_i and then exponentiating (we write it in this format to make the comparison to the Max formula more clear).

These formulas only differ in their definition of the transmittance T_i. In particular, the Neural Volumes formula treats the fraction of the ray contributed by sample i as a function of the density and sampling distance of sample i only, unless the ray is already fully occluded before it exits sample i. In contrast, the contribution of sample i in the Max formula depends on the density of sample i as well as the densities of all preceding samples along the ray. In essence, opacity in the Neural Volumes formula is absolute and ray-independent (except for clipping the total contribution to 1), whereas opacity in the Max formula denotes the fraction of incoming light that each sample absorbs, a ray-dependent quantity. As we show in Tab. 5, the Max formula results in substantially better performance; we suspect this difference is due to its more physically-accurate modeling of transmittance.

D. Per-Scene Results

D.1. Synthetic, Bounded Scenes

Full, per-scene results for the 8 synthetic scenes from NeRF are presented in Tab. 10 and Fig. 11. Note that the values for JAXNeRF are from our own rerunning with centered pixels (we ran JAXNeRF in parallel across 4 GPUs and multiplied the times by 4 to account for this parallelization).

D.2. Real, Forward-Facing Scenes

Full, per-scene results for the 8 forward-facing scenes from NeRF are presented in Tab. 11. Note that the values for JAXNeRF are from our own rerunning with centered pixels (we ran JAXNeRF in parallel across 4 GPUs and multiplied the times by 4 to account for this parallelization).

D.3. Real, 360° Scenes

Full, per-scene results for the four 360° scenes from Tanks and Temples [16] are presented in Tab. 9. Note that the values for NeRF++ appear slightly different from the paper; we re-evaluated the metrics independently using VGG LPIPS and standard SSIM, from rendered images shared by the original authors.

PSNR ↑

	M60	Playground	Train	Truck	Mean
Ours	17.93	23.03	17.97	22.67	20.40
NeRF++ [60]	18.49	22.93	17.77	22.77	20.49

SSIM ↑

	M60	Playground	Train	Truck	Mean
Ours	0.687	0.712	0.629	0.758	0.696
NeRF++	0.650	0.672	0.558	0.712	0.648

LPIPS ↓

	M60	Playground	Train	Truck	Mean
Ours	0.439	0.435	0.443	0.364	0.420
NeRF++	0.481	0.477	0.531	0.424	0.478

Optimization Time ↓

	M60	Playground	Train	Truck	Mean
Ours	25.5m	26.3m	29.5m	28.0m	27.3m

Table 9. **Full results on 360° scenes.**

PSNR ↑

	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Mean
Ours	33.98	25.35	31.83	36.43	34.10	29.14	33.26	29.62	31.71
NV [22]	28.33	22.58	24.79	30.71	26.08	24.22	27.78	23.93	26.05
JAXNeRF [8, 28]	34.20	25.27	31.15	36.81	34.02	30.30	33.72	29.33	31.85

SSIM ↑

	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Mean
Ours	0.977	0.933	0.976	0.980	0.975	0.949	0.985	0.890	0.958
NV	0.916	0.873	0.910	0.944	0.880	0.888	0.946	0.784	0.893
JAXNeRF	0.975	0.929	0.970	0.978	0.970	0.955	0.983	0.868	0.954

LPIPS ↓

	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Mean
Ours	0.031	0.067	0.026	0.037	0.028	0.057	0.015	0.134	0.049
NV	0.109	0.214	0.162	0.109	0.175	0.130	0.107	0.276	0.160
JAXNeRF	0.036	0.085	0.037	0.074	0.068	0.057	0.023	0.192	0.072

Optimization Time ↓

	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Mean
Ours	9.6m	9.8m	8.8m	12.5m	10.8m	11.0m	8.2m	18.0m	11.1m
JAXNeRF	37.8h	37.8h	37.7h	38.0h	26.0h	38.1h	37.8h	26.0h	34.9h

Table 10. **Full results on synthetic scenes.**

(a) Ground Truth (b) Neural Volumes (c) JAXNeRF (d) Plenoxels

Figure 11. **Synthetic scenes.** We show a random view from each of the synthetic scenes, comparing the ground truth, Neural Volumes [22], JAXNeRF [8, 28], and our Plenoxels.

(e) Ground Truth (f) Neural Volumes (g) JAXNeRF (h) Plenoxels

Figure 11. **Synthetic scenes.** We show a random view from each of the synthetic scenes, comparing the ground truth, Neural Volumes [22], JAXNeRF [8, 28], and our Plenoxels.

PSNR ↑

	Fern	Flower	Fortress	Horns	Leaves	Orchids	Room	T-Rex	Mean
Ours	25.46	27.83	31.09	27.58	21.41	20.24	30.22	26.48	26.29
LLFF [27]	28.42	22.85	19.52	29.40	18.52	25.46	24.15	24.70	24.13
JAXNeRF [8, 28]	25.20	27.80	31.57	27.70	21.10	20.37	32.81	27.12	26.71

SSIM ↑

	Fern	Flower	Fortress	Horns	Leaves	Orchids	Room	T-Rex	Mean
Ours	0.832	0.862	0.885	0.857	0.760	0.687	0.937	0.890	0.839
LLFF	0.932	0.753	0.697	0.872	0.588	0.844	0.857	0.840	0.798
JAXNeRF	0.798	0.840	0.890	0.840	0.703	0.649	0.952	0.890	0.820

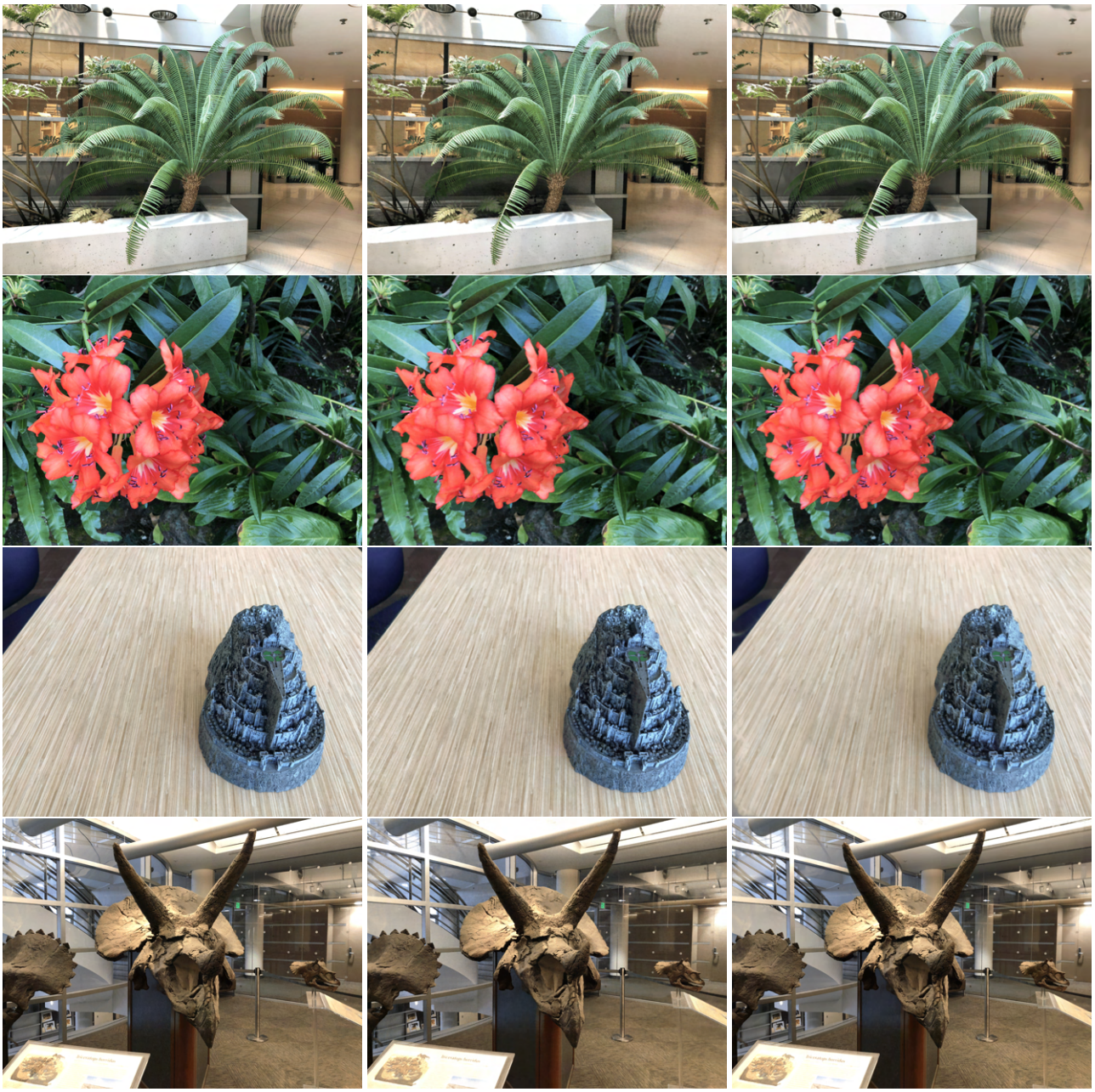
LPIPS ↓

	Fern	Flower	Fortress	Horns	Leaves	Orchids	Room	T-Rex	Mean
Ours	0.224	0.179	0.180	0.231	0.198	0.242	0.192	0.238	0.210
LLFF	0.155	0.247	0.216	0.173	0.313	0.174	0.222	0.193	0.212
JAXNeRF	0.272	0.198	0.151	0.249	0.305	0.307	0.164	0.235	0.235

Optimization Time ↓

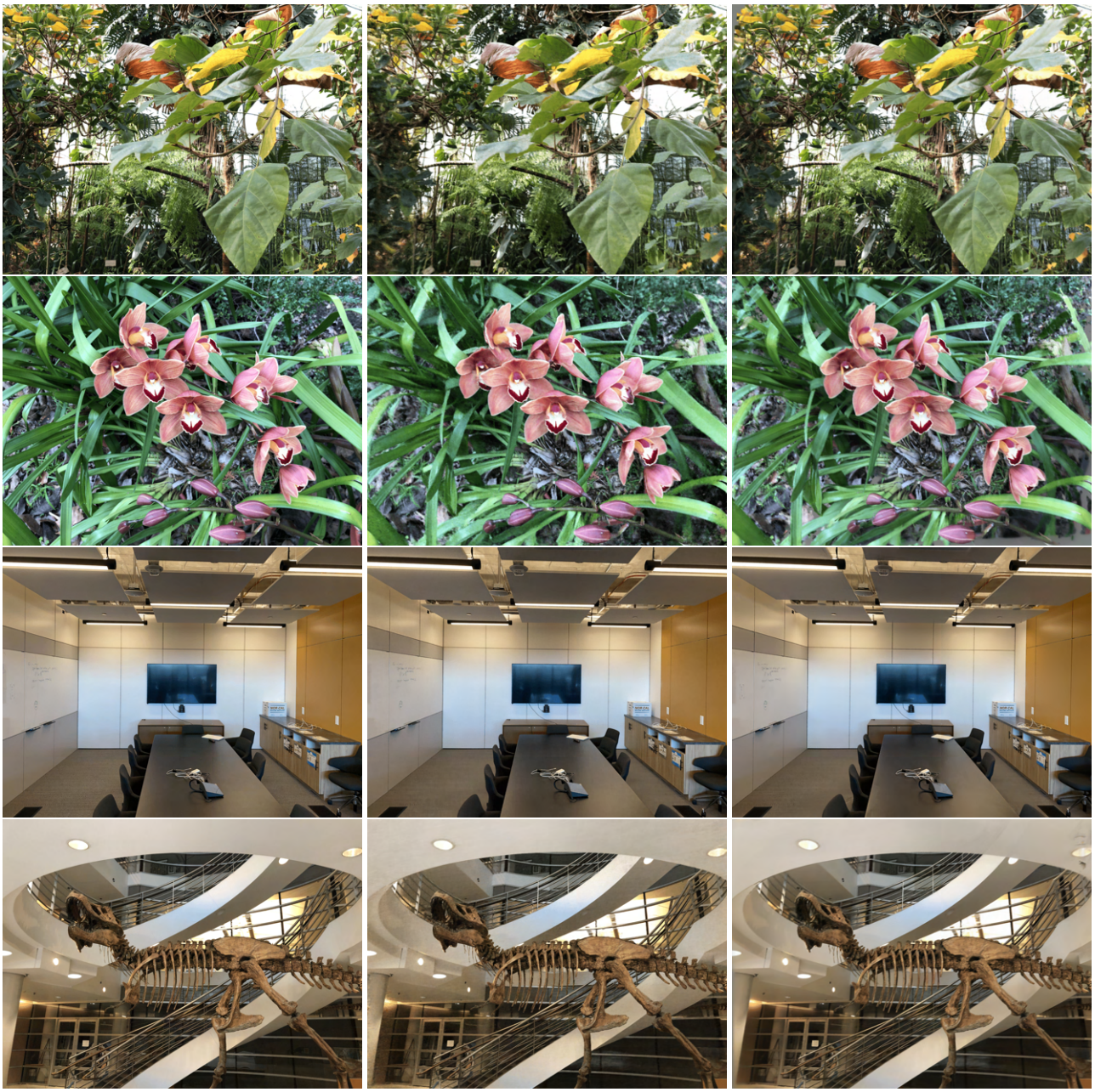
	Fern	Flower	Fortress	Horns	Leaves	Orchids	Room	T-Rex	Mean
Ours	23.7m	22.0m	31.2m	26.3m	13.3m	23.4m	28.8m	24.8m	24.2m
JAXNeRF	38.9h	38.8h	38.6h	38.7h	38.8h	38.7h	39.1h	38.6h	38.8h

Table 11. **Full results on forward-facing scenes.**



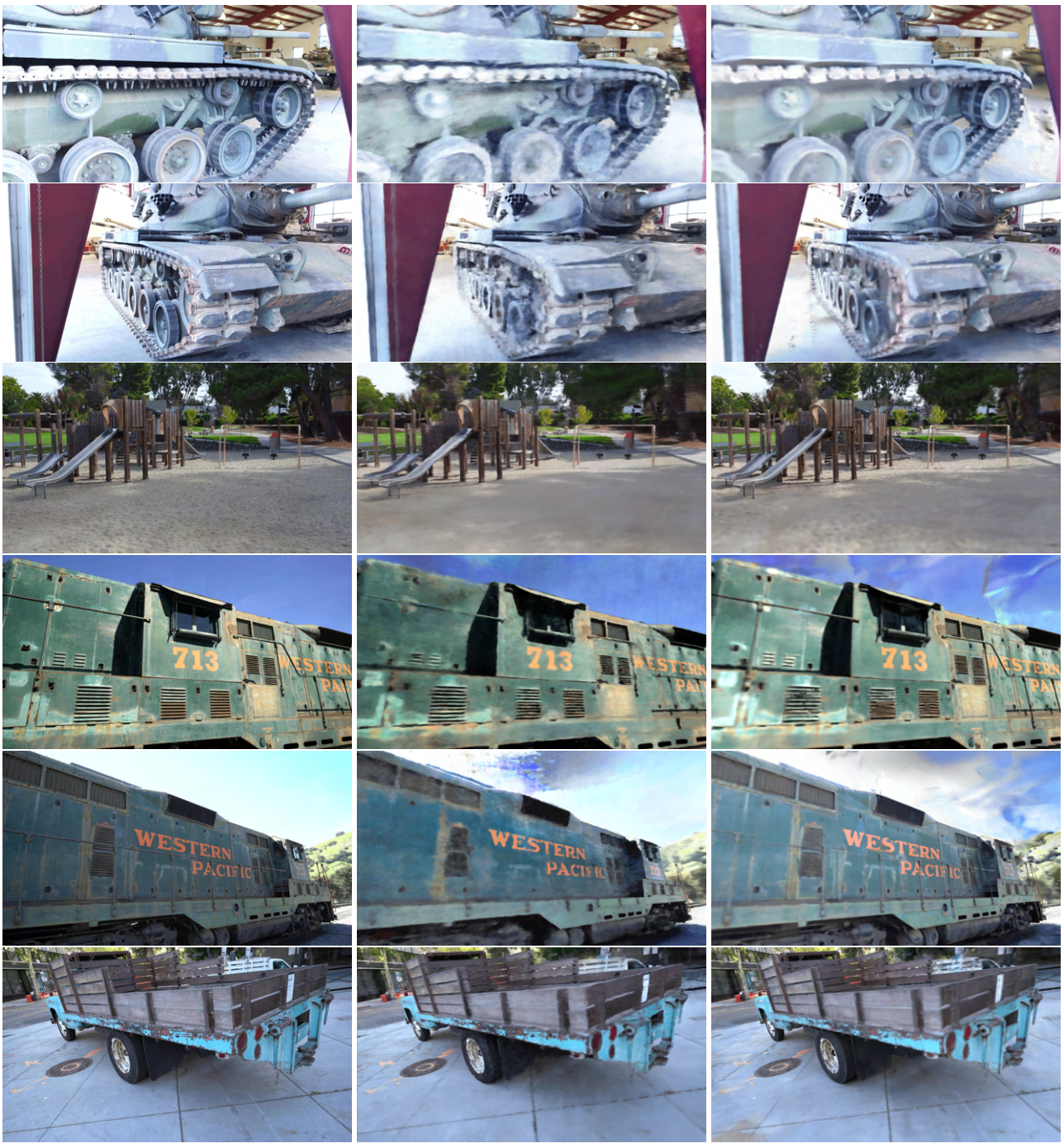
(a) Ground Truth (b) JAXNeRF (c) Plenoxels

Figure 12. **Forward-facing scenes.** We show a random view from each of the forward-facing scenes, comparing the ground truth, JAXNeRF [8, 28], and our Plenoxels.



(d) Ground Truth (e) JAXNeRF (f) Plenoxels

Figure 12. **Forward-facing scenes.** We show a random view from each of the forward-facing scenes, comparing the ground truth, JAXNeRF [8, 28], and our Plenoxels. Note that these two methods have different behaviors in unsupervised regions (*e.g.* the bottom right in the orchids view): JAXNeRF fills in plausible textures whereas Plenoxels default to gray.



(a) Ground Truth (b) NeRF++ (c) Plenoxels

Figure 13. 360° **scenes.** We show a random view from each of the Tanks and Temples scenes, comparing the ground truth, NeRF++ [60], and our Plenoxels. We include two random views each for the M60 and train scenes, since the playground and truck scenes were shown in the main text.