MobileNeRF: Exploiting the Polygon Rasterization Pipeline for Efficient Neural Field Rendering on Mobile Architectures

(Supplementary Material)

A. More results

Check out our project page: https://mobile-nerf.github.io

B. Scene editing

Our representation is a textured mesh with baked lighting, and thus can be used in any application that combines, renders, or manipulates such meshes. Figure 9 (a) shows a simple example where meshes learned from four different sets of photos are composited into a single scene. The scene, rendered in 1920×1080 resolution without supersampling, runs at 150 FPS on the gaming laptop, and consumes 1.5 GB of GPU memory. Similarly Figure 9 (b)(c) show scenes where some parts or objects are edited or removed by manipulating the triangle meshes of the scenes in a 3D modeling software. The resulting renders do not account for differences in illumination between the captured photos or indirect illumination between different meshes. However, it suggests an easy way to create "photorealistic-looking" scenes from a library of objects captured using photos rather than painstaking 3D modeling.

https://youtu.be/kVy2W6afuyk shows three examples where we manipulate the learned NeRF objects interactively in real-time. We also highlight how easy it is to implement these operations with our mesh representation. In contrast, implementing those with classic NeRF is non-trivial.

In the first example, we render all 8 objects learned from the synthetic scenes at the same time, and we move the objects by using mouse to drag the objects. This is implemented by a single line of code with the *DragControls* class provided in the *threejs* library. *DragControls* is designed for manipulating meshes, which suits our needs exactly since our objects are meshes. We also cast real-time shadow of the objects by applying shadow mapping. This is implemented by having a directional light, an ambient light, and a plane below the objects to receive shadows. The drag control and the real-time shadow are also used in the following examples.

In the second example, we interactively deform the learned chair object to create new variations of chairs. To implement the deformation, we only need to deform the vertex positions of the meshes, and this is achieved by adding vertex deformation code in the vertex shader. Specifically, we implemented three operations: moving the chair up/down will lengthen or shorten its legs, moving the chair left/right will adjust its width, and moving the chair forward/backward will adjust the skew of its back.

In the third example, we render 9 ficus objects, which are considered "NeRF" objects, and a blue ball, which is a classic object with standard material used in classic rendering. We again change the vertex shader to make the leaves of the plants to be repelled by the blue ball.

C. Training

Our training stages are formalized as follows. In the first training stage, we optimize

$$\underset{\mathcal{V},\theta_{\mathcal{A}},\theta_{\mathcal{F}},\theta_{\mathcal{H}}}{\arg\min} \quad \mathcal{L}_{\mathbf{C}} + w_d \mathcal{L}_{\text{dist}} + \mathcal{L}_{\text{v}} \tag{18}$$

and

$$\underset{\mathcal{G}}{\arg\min} \quad \mathcal{L}_{\mathcal{G}}^{\text{bnd}} + w_{g1}\mathcal{L}_{\mathcal{G}}^{\text{sparse}} + w_{g2}\mathcal{L}_{\mathcal{G}}^{\text{smooth}}, \tag{19}$$

where $w_{g1} = w_{g2} = 10^{-5}$. w_d is set to 0.0 for synthetic 360° scenes, 0.01 for forward-facing scenes, and 0.001 for unbounded 360° scenes. In the second training stage, we optimize

$$\underset{\mathcal{V},\theta_{\mathcal{A}},\theta_{\mathcal{F}},\theta_{\mathcal{H}}}{\arg\min} \quad \mathcal{L}_{\mathbf{C}}^{\text{stage2}} + w_d \mathcal{L}_{\text{dist}} + \mathcal{L}_{\text{v}} \tag{20}$$

and Eq. 19. When the loss converges, we fix the weights of $\mathcal{V}$, $\theta_{\mathcal{A}}$, and $\mathcal{G}$ and optimize

$$\underset{\theta_{\mathcal{F}},\theta_{\mathcal{H}}}{\arg\min} \quad \mathcal{L}_{\mathbf{C}}^{\text{bin}}. \tag{21}$$

D. Network architectures

We adopt the MLP designed in NeRF as the network for both $\mathcal{A}$ and $\mathcal{F}$. We increase the hidden layer sizes from 256 to 384, since $\mathcal{A}$ and $\mathcal{F}$ are not used during inference, so we can afford more time on training. The small MLP $\mathcal{H}$ is the same as the small MLP used in SNeRG, with two hidden layers, each consisting 16 neurons.

E. More details about texture images

Since the features to be stored are 8-dimensional, we use two PNG images to store them. Each PNG image has 4 channels, therefore two PNG images have a total of 8 channels to store 8-d features. To avoid having an extra image to store the binary alpha (opacity) channel, we squeeze the alpha channel into the first feature channel, so that the alpha is one when the first feature channel is non-zero, and zero

when the channel is zero. Since phones have a hardware constraint that the texture size must be a power of 2 and at most 4096×4096, we split the large texture images into multiple 4096×4096 texture images.

F. Quadrature details

The regular-grid mesh $\mathcal{M}$ provides an efficient way for computing intersections between a ray and the mesh of size $P \times P \times P$ in $O(P)$ complexity, as shown in Figure 5.

First, we compute the set of voxels that are intersected by the ray. This involves solving $3P$ ray-plane intersections and using those intersection points to obtain at most $3P$ intersected voxels. This step is shown in Figure 5a and Eq. 7.

Then, we use the acceleration grid $\mathcal{G} \in \mathbb{R}^{P \times P \times P}$ to prune voxels that are unlikely to contain geometry, with respect to a threshold $\tau_{\mathcal{G}} = 0.1$. This step is shown in Figure 5b and Eq. 8.

Finally, we compute intersections between the ray and the faces of $\mathcal{M}$ that are incident to the voxel's vertex to obtain the final set of quadrature points. This step is shown in Figure 5c and Eq. 9.

During the first quarter of the training iterations, $\mathcal{G}$ may not be accurate, therefore we will keep all $3P$ intersected voxels regardless of $\tau_{\mathcal{G}}$, and keep $3P$ intersection points (Recall that if the mesh grid is a regular grid, there are at most $3P$ intersections). Then in the next quarter, we will use $\mathcal{G}$ to remove empty voxels and keep at most $3P/2$ non-empty voxels and $3P/2$ intersection points that are closest to the camera. In the rest of the training, we will keep $3P/4$. We also double the training batch size each time we halve the number of intersections.

For the concentric boxes in unbounded 360° scenes, we will compute their intersections and keep all of them.

G. Initial meshes

In this section we detail the polygonal meshes used for synthetic 360°, forward-facing, and unbounded 360° scenes, see Fig. 4 for 2D illustrations.

We will call the coordinate system of a regular mesh grid in a unit cube centered at the origin as the normalized coordinates, and we can apply transformations to obtain the grids in the world coordinates for different types of scenes. In the following, we will denote points in the normalized coordinates as $\mathbf{p} \in [-0.5, 0.5]$ and points in the world coordinates as $\mathbf{p}'$.

For synthetic 360° scenes, we apply scaling to the grid to put the object inside the grid.

$$\mathbf{p}' = w\mathbf{p}, \tag{22}$$

where $w = 2.4$ or 3, depending on the size of the object. We use a grid size of $P = 128$.

In forward-facing scenes, we apply transformation to concentrate more voxels close to the camera, as shown in Fig. 4 (b).

$$\begin{cases} \mathbf{p}'_z &= \exp(w(\mathbf{p}_z + 0.5)), \\ \mathbf{p}'_x &= u\mathbf{p}_x\mathbf{p}'_z, \\ \mathbf{p}'_y &= v\mathbf{p}_y\mathbf{p}'_z, \end{cases} \tag{23}$$

where w is set to a value so that $\mathbf{p}'_z = 25$ when $\mathbf{p}_z = 0.5$; $u = v = 1.75$. We use a grid size of $P = 128$.

In unbounded 360° scenes, we assume the cameras are inside the unit cube in the normalized coordinates, therefore we do not apply transformations. However, to model the surrounding environments, we add a set of $L + 1$ concentric boxes around the regular grid. The boxes have fixed positions and geometry, and their distances to the center are given by

$$d_i = (\exp(\frac{wi}{L}) + w - 1)/2w, \tag{24}$$

where i ranges from 0 to L. w is set to a value so that $d_L = 8$, therefore $d_i \in [0.5, 8]$. We use a grid size of $P = 128$, and $L = 64$.

H. Per-Scene metrics

We provide per-scene breakdown for the quality metrics in Table 8 9 10 12 13 14 16 17 18. We provide per-scene breakdown for rendering speed and storage cost in Table 11 15 19, where OOM (out-of-memory) indicates the device cannot run a testing scene due to GPU memory issues, and ICP (incompatible) indicates the device cannot run the method due to compatibility issues. The GPU memory and disk storage were tested on the Desktop.

For Surface Pro 6, Gaming laptop, and Desktop, we disable frame-rate limiting from vertical synchronization by starting the Chrome browser with the following arguments:

```
--disable-frame-rate-limit
--disable-gpu-vsync
```

However, for phones and Chromebook, we did not find a way to easily disable vertical synchronization, therefore the FPS is capped at 60.

The models in our online demo (https://mobile-nerf.github.io) are the same as the ones used in our paper. The rendered images of our method are nearly identical whether they are rendered in Python (for computing quantitative metrics) or web browsers, see Figure 10. If one overlays the difference image and the rendered image, one can find that the few very different pixels are all on the boundary of a part, which indicates that they are likely caused by precision errors in rasterization.

	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Mean
NeRF [33]	33.00	25.01	30.13	36.18	32.54	29.62	32.91	28.65	31.00
JAXNeRF [14]	33.88	25.08	30.51	36.91	33.24	30.03	34.52	29.07	31.65
SNeRG [21]	33.24	24.57	29.32	34.33	33.82	27.21	32.60	27.97	30.38
Ours	34.09	25.02	30.20	35.46	34.18	26.72	32.48	29.06	30.90

Table 8. **PSNR**↑ on **Synthetic** 360° **scenes**.

	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Mean
NeRF [33]	0.967	0.925	0.964	0.974	0.961	0.949	0.980	0.856	0.947
JAXNeRF [14]	0.974	0.927	0.967	0.979	0.968	0.952	0.987	0.865	0.952
SNeRG [21]	0.975	0.929	0.967	0.971	0.973	0.938	0.982	0.865	0.950
Ours	0.978	0.927	0.965	0.973	0.975	0.913	0.979	0.867	0.947

Table 9. **SSIM**↑ on **Synthetic** 360° **scenes**.

	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Mean
NeRF [33]	0.046	0.091	0.044	0.121	0.050	0.063	0.028	0.206	0.081
JAXNeRF [14]	0.027	0.070	0.033	0.030	0.030	0.048	0.013	0.156	0.051
SNeRG [21]	0.025	0.061	0.028	0.043	0.022	0.052	0.016	0.156	0.050
Ours	0.025	0.077	0.048	0.050	0.025	0.092	0.032	0.145	0.062

Table 10. **LPIPS**↓ on **Synthetic** 360° **scenes**.

	SNeRG [21]								
	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Mean
iPhone XS	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	-
Pixel 3	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	-
Surface Pro 6	ICP	ICP	ICP	ICP	ICP	ICP	ICP	ICP	-
Chromebook	28.06	OOM	OOM	26.11	27.08	16.48	26.99	11.01	22.62
Gaming laptop	4.94	10.27	OOM	8.10	9.41	2.05	21.65	1.69	8.30
Gaming laptop 🔌	37.66	51.06	OOM	45.52	60.20	13.81	87.67	11.17	43.87
Desktop 🔌	120.70	147.72	81.88	436.05	232.03	92.45	507.54	39.73	207.26
GPU memory	1254.00	4729.00	8243.00	1253.00	1253.00	1253.00	1251.00	2422.00	2707.25
Disk storage	141.00	44.00	43.00	67.00	114.00	134.00	22.00	129.00	86.75

	Ours								
	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Mean
iPhone XS	60.00	60.00	60.00	60.00	50.10	54.65	60.00	42.37	55.89
Pixel 3	41.68	38.71	43.09	35.59	29.56	32.35	52.65	23.52	37.14
Surface Pro 6	83.40	83.15	99.34	64.01	57.11	58.80	130.62	42.76	77.40
Chromebook	60.00	60.00	60.00	53.24	47.51	51.04	60.00	37.56	53.67
Gaming laptop	186.03	183.04	231.01	156.08	118.27	129.80	332.10	89.74	178.26
Gaming laptop 🔌	657.77	656.22	643.32	649.58	566.39	618.98	648.88	412.70	606.73
Desktop 🔌	810.99	789.30	882.23	707.27	629.70	659.95	970.35	509.48	744.91
GPU memory	451.00	590.00	450.00	456.00	723.00	721.00	322.00	594.00	538.38
Disk storage	107.00	120.00	80.00	88.00	199.00	191.00	50.00	171.00	125.75

Table 11. **Rendering speed** in frames per second (FPS), and **GPU memory and disk storage** in MB, on **Synthetic** 360° **scenes**.

	Room	Fern	Leaves	Fortress	Orchids	Flower	Trex	Horns	Mean
NeRF [33]	32.70	25.17	20.92	31.16	20.36	27.40	26.80	27.45	26.50
JAXNeRF [14]	33.30	24.92	21.24	31.78	20.32	28.09	27.43	28.29	26.92
SNeRG [21]	30.04	24.85	20.01	30.91	19.73	27.00	25.80	26.71	25.63
Ours	31.28	24.59	20.54	30.82	19.66	27.05	26.26	27.09	25.91

Table 12. **PSNR**↑ on **Forward-facing scenes**.

	Room	Fern	Leaves	Fortress	Orchids	Flower	Trex	Horns	Mean
NeRF [33]	0.948	0.792	0.690	0.881	0.641	0.827	0.880	0.828	0.811
JAXNeRF [14]	0.958	0.806	0.717	0.897	0.657	0.850	0.902	0.863	0.831
SNeRG [21]	0.936	0.802	0.696	0.889	0.655	0.835	0.882	0.852	0.818
Ours	0.943	0.808	0.711	0.891	0.647	0.839	0.900	0.864	0.825

Table 13. **SSIM**↑ on **Forward-facing scenes**.

	Room	Fern	Leaves	Fortress	Orchids	Flower	Trex	Horns	Mean
NeRF [33]	0.178	0.280	0.316	0.171	0.321	0.219	0.249	0.268	0.250
JAXNeRF [14]	0.086	0.207	0.247	0.108	0.266	0.156	0.143	0.173	0.173
SNeRG [21]	0.133	0.198	0.252	0.125	0.255	0.167	0.157	0.176	0.183
Ours	0.143	0.202	0.245	0.115	0.277	0.163	0.147	0.169	0.183

Table 14. **LPIPS**↓ on **Forward-facing scenes**.

	SNeRG [21]								
	Room	Fern	Leaves	Fortress	Orchids	Flower	Trex	Horns	Mean
iPhone XS	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	-
Pixel 3	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	-
Surface Pro 6	ICP	ICP	ICP	ICP	ICP	ICP	ICP	ICP	-
Chromebook	9.75	6.02	OOM	9.68	OOM	5.12	8.68	OOM	7.85
Gaming laptop	7.77	1.28	0.80	8.46	1.14	0.67	4.72	4.18	3.63
Gaming laptop 🔌	52.40	14.45	6.15	54.47	12.43	8.77	32.87	26.51	26.01
Desktop 🔌	110.36	28.18	13.54	122.91	17.59	15.96	62.65	34.46	50.71
GPU memory	3594.00	3585.00	4729.00	3595.00	5903.00	3593.00	3595.00	5903.00	4312.13
Disk storage	149.00	288.00	408.00	162.00	704.00	321.00	251.00	415.00	337.25

	Ours								
	Room	Fern	Leaves	Fortress	Orchids	Flower	Trex	Horns	Mean
iPhone XS	29.82	25.10	OOM	30.02	OOM	26.28	26.30	25.59	27.19
Pixel 3	13.57	12.74	8.66	14.69	10.77	12.98	13.07	12.71	12.40
Surface Pro 6	22.92	20.32	13.84	29.13	17.10	22.30	22.53	23.92	21.51
Chromebook	20.70	18.95	14.65	23.16	16.79	20.06	20.08	21.12	19.44
Gaming laptop	64.27	55.88	37.11	76.29	48.72	60.60	59.65	59.26	57.72
Gaming laptop 🔌	281.01	252.70	170.66	303.77	222.54	260.44	258.45	251.82	250.17
Desktop 🔌	377.87	352.01	254.51	397.00	323.54	367.68	359.68	362.46	349.34
GPU memory	610.00	610.00	1143.00	473.00	1276.00	611.00	604.00	747.00	759.25
Disk storage	127.00	147.00	353.00	89.00	372.00	151.00	162.00	211.00	201.50

Table 15. **Rendering speed** in frames per second (FPS), and **GPU memory and disk storage** in MB, on **Forward-facing scenes**.

	Bicycle	Flower	Garden	Stump	Treehill	Mean
JAXNeRF [14]	21.76	19.40	23.11	21.73	21.28	21.46
NeRF++ [52]	22.64	20.31	24.32	24.34	22.20	22.76
Ours	21.70	18.86	23.54	23.95	21.72	21.95

Table 16. **PSNR**↑ on **Unbounded** 360° **scenes**.

	Bicycle	Flower	Garden	Stump	Treehill	Mean
JAXNeRF [14]	0.455	0.376	0.546	0.453	0.459	0.458
NeRF++ [52]	0.526	0.453	0.635	0.594	0.530	0.548
Ours	0.426	0.321	0.599	0.556	0.450	0.470

Table 17. **SSIM**↑ on **Unbounded** 360° **scenes**.

	Bicycle	Flower	Garden	Stump	Treehill	Mean
JAXNeRF [14]	0.536	0.529	0.415	0.551	0.546	0.515
NeRF++ [52]	0.455	0.466	0.331	0.416	0.466	0.427
Ours	0.513	0.526	0.358	0.430	0.522	0.470

Table 18. **LPIPS**↓ on **Unbounded** 360° **scenes**.

	Ours					
	Bicycle	Flower	Garden	Stump	Treehill	Mean
iPhone XS	OOM	OOM	22.20	OOM	OOM	22.20
Pixel 3	9.44	8.61	10.49	8.54	9.12	9.24
Surface Pro 6	20.24	19.12	21.67	18.21	17.97	19.44
Chromebook	15.89	14.72	16.56	14.23	15.02	15.28
Gaming laptop	55.62	59.18	58.19	51.73	51.89	55.32
Gaming laptop 🔌	195.63	194.66	204.31	178.89	189.46	192.59
Desktop 🔌	280.24	282.02	295.74	265.90	274.58	279.70
GPU memory	1350.00	1081.00	808.00	1082.00	1490.00	1162.20
Disk storage	400.00	294.00	239.00	337.00	453.00	344.60

Table 19. **Rendering speed** in frames per second (FPS), and **GPU memory and disk storage** in MB, on **Unbounded** 360° **scenes**.

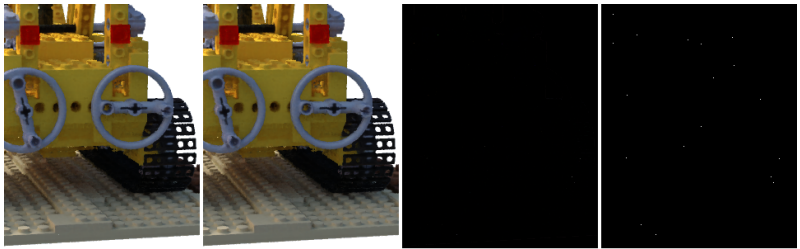

Figure 10. Comparison between images rendered in Python and in a web browser. Image pixel value range is 0-255. Zoom in for details.