

THU-AM-009

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

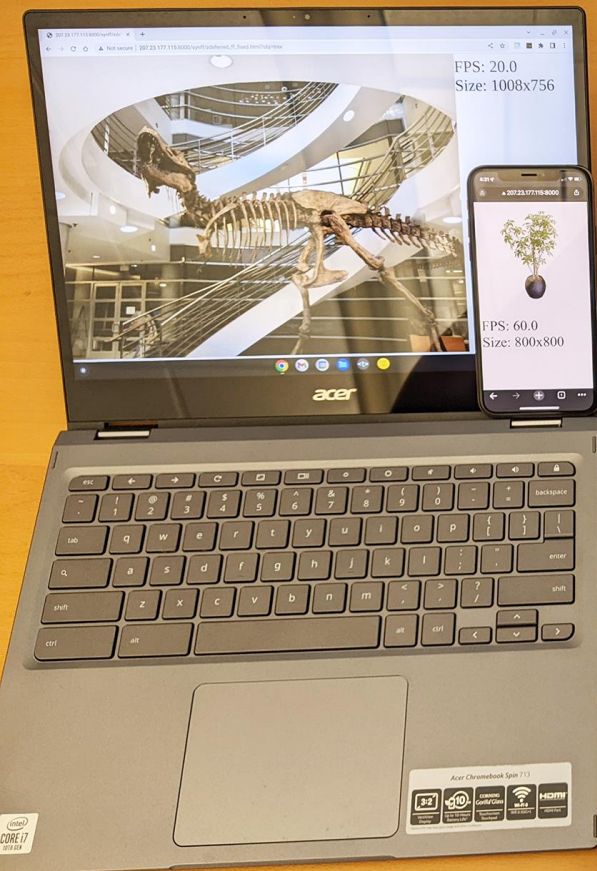
Zhiqin Chen*Simon Fraser University
Google Research**Thomas Funkhouser**

Google Research

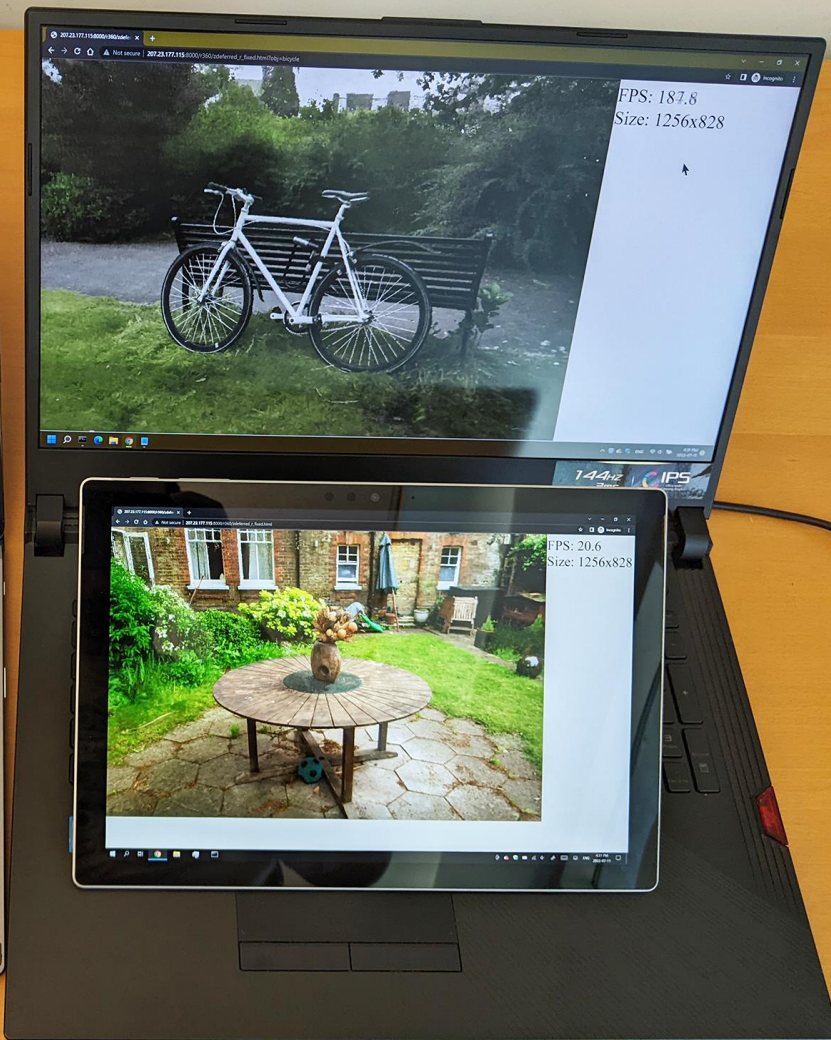
**Peter Hedman**

Google Research

**Andrea Tagliasacchi***Simon Fraser University
Google Research
University of Toronto



Chromebook
T-rex scene
1008x756, 20 FPS



Gaming laptop
Bicycle scene
1256x828, 187 FPS

iPhone XS
Ficus scene
800x800, 60FPS

Surface Pro 6
Garden scene
1256x828, 20 FPS

Triangle meshes as NeRF representation

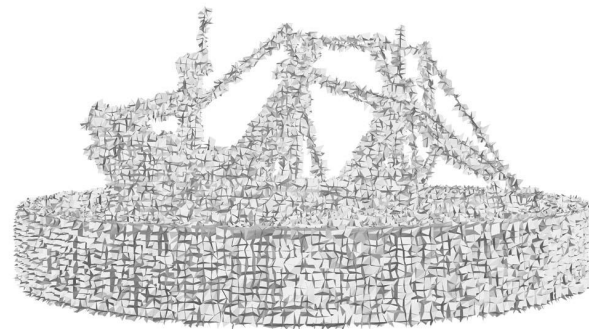
> Compatibility:

All GPUs in modern devices can render triangles.

> Speed:

GPUs are optimized to render triangles fast.

Extracted
triangle mesh
by our method



Rendered image
by our method

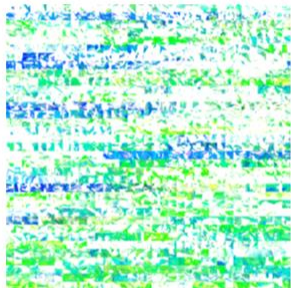


Overview - rendering

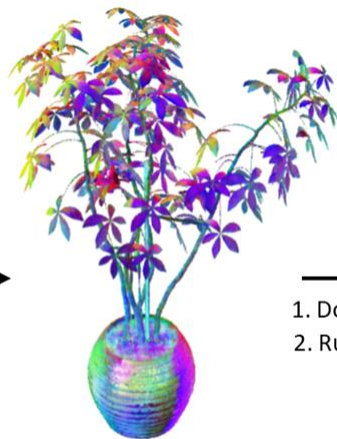
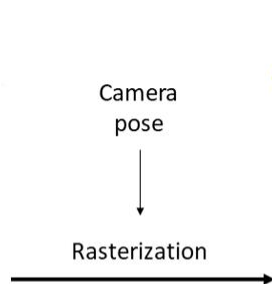


(a) Triangle mesh

+

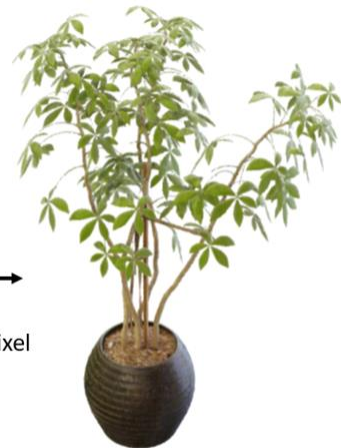
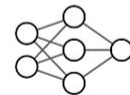


(b) Texture image
storing features
and opacity

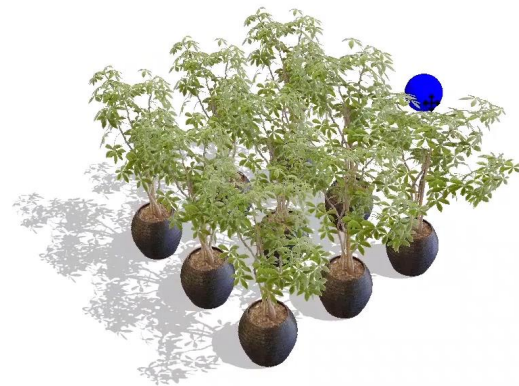


(c) "Feature image"
storing features and viewing
directions for each pixel

1. Downsampling for anti-aliasing
2. Running a small MLP for each pixel



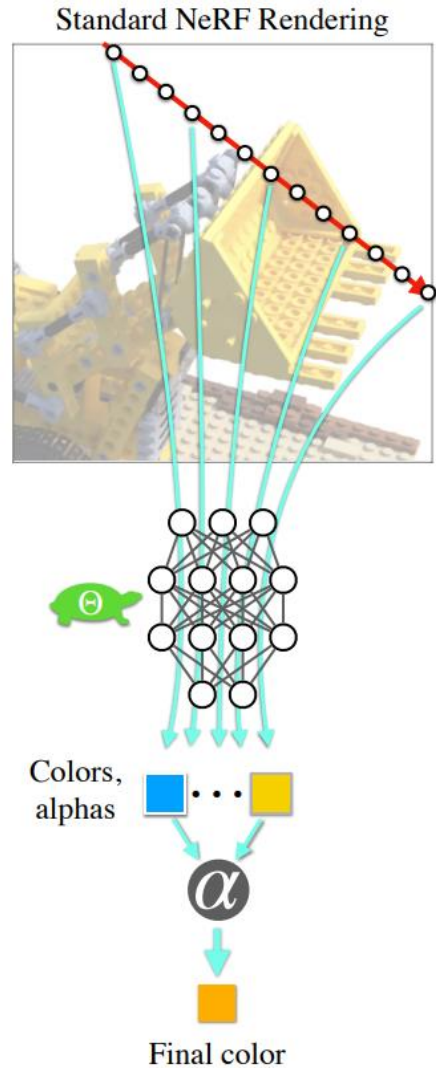
(d) Final output



Motivation - speed

Classic NeRF methods rely on volumetric rendering.

> Slow: for each pixel, many points need to be sampled along the ray and evaluated by the MLP.

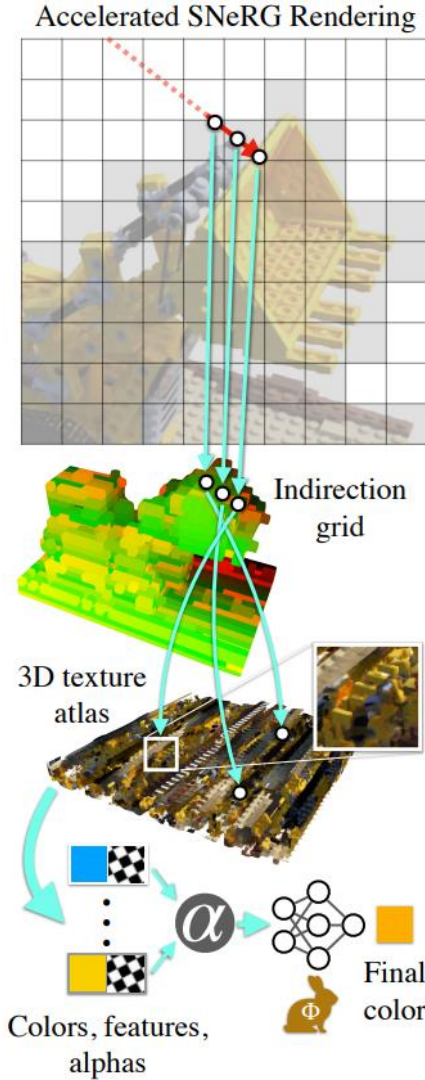


Motivation - GPU memory

Recent NeRF methods speed up inference by “baking” the MLP evaluation results into sparse 3D voxel grids.

E.g., SNeRG, PlenOctrees.

> Large: 3D textures or 3D structures have to be stored in GPU for fast accessing.

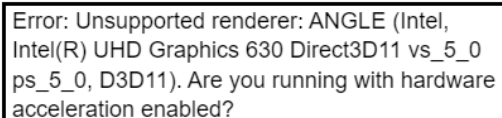


[1] (SNeRG) Baking Neural Radiance Fields for Real-Time View Synthesis. Hedman et al. ICCV 2021.

[2] PlenOctrees for Real-time Rendering of Neural Radiance Fields. Yu et al. ICCV 2021.

Motivation - compatibility

Most NeRF methods need cuda and high-end accelerators.



Error: Unsupported renderer: ANGLE (Intel, Intel(R) UHD Graphics 630 Direct3D11 vs_5_0 ps_5_0, D3D11). Are you running with hardware acceleration enabled?

Frames per second: 125.52

SNeRG

10:18

📶 🔋 100%

Real-time Online Demo

We're excited to present a live demo that works in modern browsers. Click on one of the scenes below to open the demo app.

Note: Our full models are on the order of 2GB in size; for online viewing, the PlenOctrees used are *lower resolution, quantized* versions of 34-125MB, losing approximately 0.5-1.5 dB in PSNR.

Unfortunately, mobile and tablet devices are not currently supported due to WebGL compatibility issues. We hope to support this in the future.

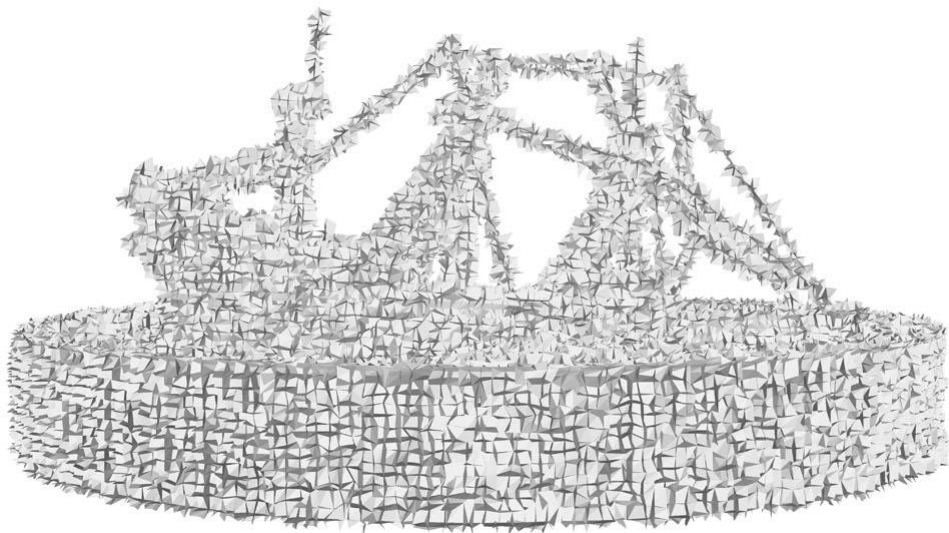
PlenOctrees

Our method

We use textured triangle meshes as the NeRF representation.

- > Compatibility: all GPUs in modern devices can render triangles.
- > Speed: GPUs are optimized to render triangles fast.
- > Memory: storing 2D textures consumes much less memory than storing 3D textures.

Triangle meshes as NeRF representation



Extracted triangle mesh
by our method



Rendered image
by our method



▲ 207.23.177.115:8000



FPS: 30.7

Size: 980x660

5:03



▲ 207.23.177.115:8000



FPS: 44.9

Size: 800x800

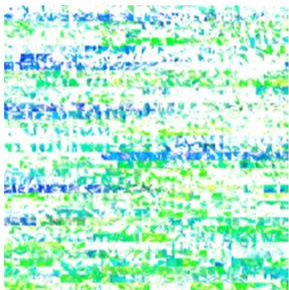
View dependent colors

> Store 8-d features instead of 3-d RGB colors in the texture image.

> Use a tiny MLP running in a GLSL fragment shader to produce the output color.



(a) Triangle mesh

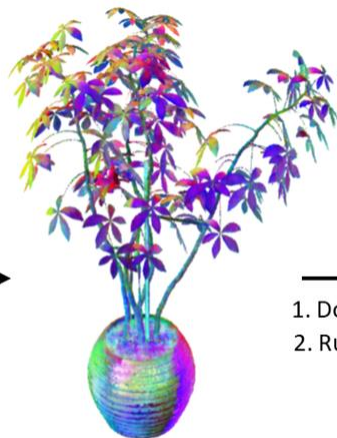


(b) Texture image
storing features
and opacity

+

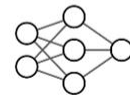
Camera
pose

Rasterization



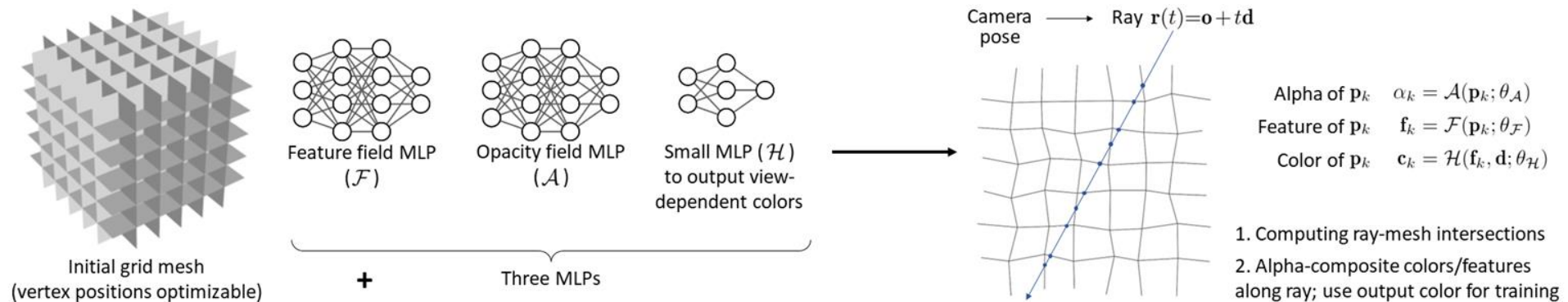
(c) "Feature image"
storing features and viewing
directions for each pixel

1. Downsampling for anti-aliasing
2. Running a small MLP for each pixel



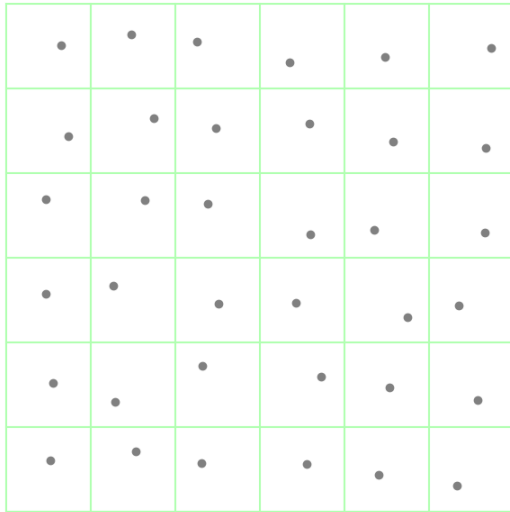
(d) Final output

Training - stage 1

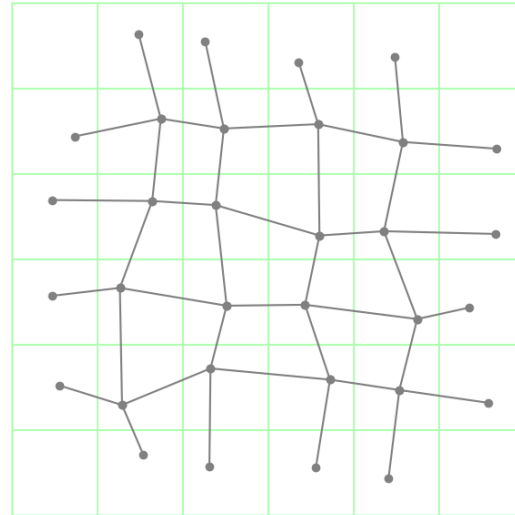


Initial grid mesh

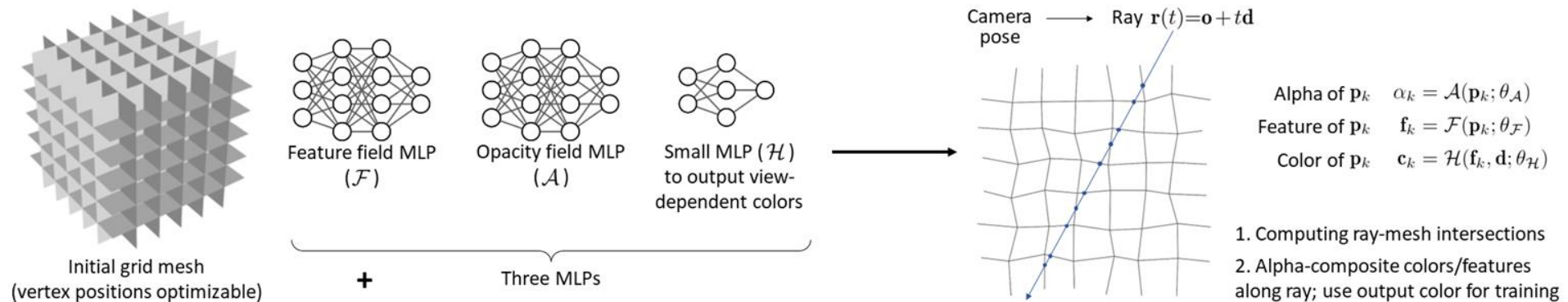
1. Store a grid of vertices



2. Connect adjacent vertices to form faces



Training - stage 1



$$\mathbf{C}(\mathbf{r}) = \sum_{k=1}^K T_k \alpha_k \mathbf{c}_k, \quad T_k = \prod_{l=1}^{k-1} (1 - \alpha_l)$$

$$\mathcal{L}_{\mathbf{C}} = \mathbb{E}_{\mathbf{r}} \|\mathbf{C}(\mathbf{r}) - \mathbf{C}_{\text{gt}}(\mathbf{r})\|_2^2.$$

Training - stage 2

1. Binarization

$$\alpha_k = \mathcal{A}(\mathbf{p}_k; \theta_{\mathcal{A}}) \quad \mathcal{A} : \mathbb{R}^3 \rightarrow [0, 1]$$

$$\hat{\alpha}_k \in \{0, 1\}$$

$$\hat{\alpha}_k = \alpha_k + \nabla [\mathbb{1}(\alpha_k > 0.5) - \alpha_k]$$

Training - stage 2

1. Binarization



2. Supersampling
(for antialiasing)

(for antialiasing)



*Actually, we average pixel features instead of pixel colors. Please refer to the paper for more details.

Ground truth

With SS

Without SS

Training - stage 3

Extract the mesh

> store visible triangles in OBJ files.

Bake textures

> store the features and alpha into PNG texture images.

Cache the neural renderer

> store the weights of the view-dependent MLP into a JSON file

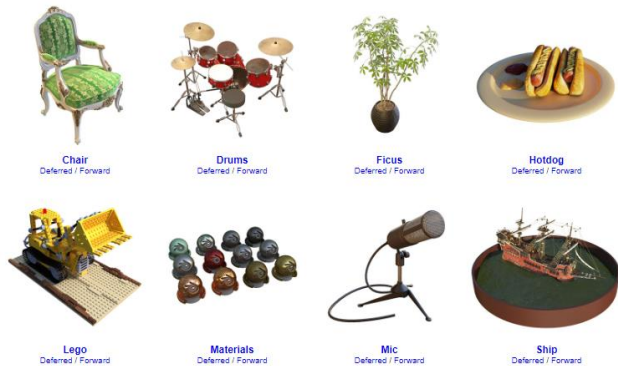
Real-time NeRF viewer

HTML+JavaScript+WebGL using Three.js.

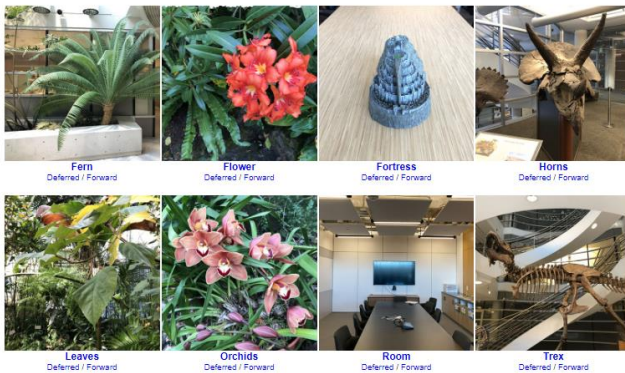
Online demo:

<https://mobile-nerf.github.io>

-- Synthetic 360° scenes --



-- Forward-facing scenes --



-- Unbounded 360° scenes --



Results - testing devices

Device	Type	OS	GPU	Power
iPhone XS	Phone	iOS 15	Integrated GPU	6W
Pixel 3	Phone	Android 12	Integrated GPU	9W
Surface Pro 6	Tablet	Windows 10	Integrated GPU	15W
Chromebook	Laptop	Chrome OS	Integrated GPU	15W
Gaming laptop	Laptop	Windows 11	NVIDIA RTX 2070	115W
Desktop	PC	Ubuntu 16.04	NVIDIA RTX 2080 Ti	250W

Table 1. **Hardware specs** – of the devices used in our rendering experiments. The power is the max GPU power for discrete NVIDIA cards, and the combined max CPU and GPU power for integrated GPUs.

Results - GPU memory and disk storage

Device	Type	OS	GPU	Power
iPhone XS	Phone	iOS 15	Integrated GPU	6W
Pixel 3	Phone	Android 12	Integrated GPU	9W
Surface Pro 6	Tablet	Windows 10	Integrated GPU	15W
Chromebook	Laptop	Chrome OS	Integrated GPU	15W
Gaming laptop	Laptop	Windows 11	NVIDIA RTX 2070	115W
Desktop	PC	Ubuntu 16.04	NVIDIA RTX 2080 Ti	250W

Table 1. **Hardware specs** – of the devices used in our rendering experiments. The power is the max GPU power for discrete NVIDIA cards, and the combined max CPU and GPU power for integrated GPUs.

Dataset Method	Synthetic 360°		Forward-facing		Unbounded 360°
	Ours	SNeRG	Ours	SNeRG	Ours
GPU memory	538.38	2707.25	759.25	4312.13	1162.20
Disk storage	125.75	86.75	201.50	337.25	344.60

Table 3. **Resources** – memory and disk storage (MB).

Results - rendering speed

Device	Type	OS	GPU	Power
iPhone XS	Phone	iOS 15	Integrated GPU	6W
Pixel 3	Phone	Android 12	Integrated GPU	9W
Surface Pro 6	Tablet	Windows 10	Integrated GPU	15W
Chromebook	Laptop	Chrome OS	Integrated GPU	15W
Gaming laptop	Laptop	Windows 11	NVIDIA RTX 2070	115W
Desktop	PC	Ubuntu 16.04	NVIDIA RTX 2080 Ti	250W

Table 1. **Hardware specs** – of the devices used in our rendering experiments. The power is the max GPU power for discrete NVIDIA cards, and the combined max CPU and GPU power for integrated GPUs.

Dataset Method	Synthetic 360°		Forward-facing		Unbounded 360°
	Ours	SNeRG	Ours	SNeRG	Ours
GPU memory	538.38	2707.25	759.25	4312.13	1162.20
Disk storage	125.75	86.75	201.50	337.25	344.60

Table 3. **Resources** – memory and disk storage (MB).

Dataset Method	Synthetic 360°		Forward-facing		Unbounded 360°
	Ours	SNeRG	Ours	SNeRG	Ours
iPhone XS	55.89	0.0 $\frac{8}{8}$	27.19 $\frac{2}{8}$	0.0 $\frac{8}{8}$	22.20 $\frac{4}{5}$
Pixel 3	37.14	0.0 $\frac{8}{8}$	12.40	0.0 $\frac{8}{8}$	9.24
Surface Pro 6	77.40	Unsupported	21.51	Unsupported	19.44
Chromebook	53.67	22.62 $\frac{2}{8}$	19.44	7.85 $\frac{3}{8}$	15.28
Gaming laptop	178.26	8.30 $\frac{1}{8}$	57.72	3.63	55.32
Gaming laptop †	606.73	43.87 $\frac{1}{8}$	250.17	26.01	192.59
Desktop †	744.91	207.26	349.34	50.71	279.70

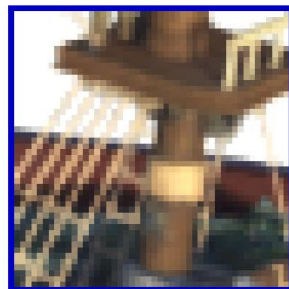
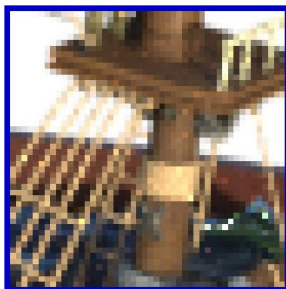
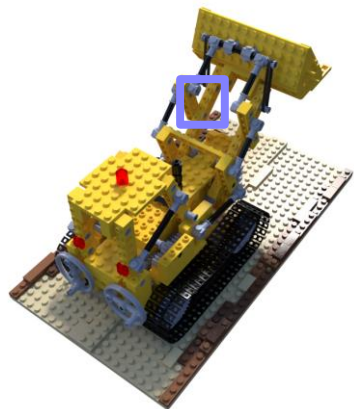
Table 2. **Rendering speed** – on various devices in frames per second (FPS). The devices are on battery, except for the gaming laptop and the desktop which are plugged in, indicated with a †. The mobile devices (first four rows) have almost identical rendering speed when plugged in. With the notation $\frac{M}{N}$ we indicate that M out of N testing scenes failed to run due to out-of-memory errors.

Results - rendering quality

	Synthetic 360°			Forward-facing			Unbounded 360°		
	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
NeRF	31.00	0.947	0.081	26.50	0.811	0.250	-	-	-
JAXNeRF	31.65	0.952	0.051	26.92	0.831	0.173	21.46	0.458	0.515
NeRF++	-	-	-	-	-	-	22.76	0.548	0.427
SNeRG	30.38	0.950	0.050	25.63	0.818	0.183	-	-	-
Ours	30.90	0.947	0.062	25.91	0.825	0.183	21.95	0.470	0.470

Table 4. **Quantitative Analysis** – For NeRF and NeRF++, we dash entries where the original papers did not report quantitative performance.

Visual results

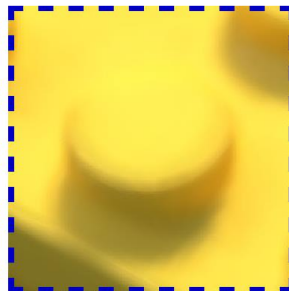


(a) Ground truth

(b) SNeRG

(c) Our method

Visual results



(a) Ground truth

(b) SNeRG

(c) Our method

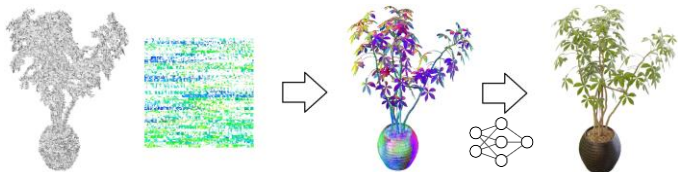
[New] Shader code optimization

MLP shader optimizations suggested by Noeri Huisman

1. Inject network weights directly into the shader source code.
2. Use mat4 and vec3 multiplications in all operations.
3. Forward rendering instead of deferred rendering.

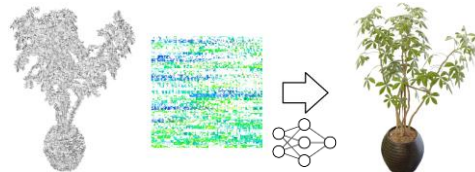
* See the supplementary material of our paper on arXiv for more details.

Rendering speed improvements (tested on 5 real unbounded scenes with a mobile phone)



Deferred rendering

26 FPS \rightarrow 35 FPS (\uparrow 35%)



Forward rendering

26 FPS \rightarrow 84 FPS (\uparrow 223%)

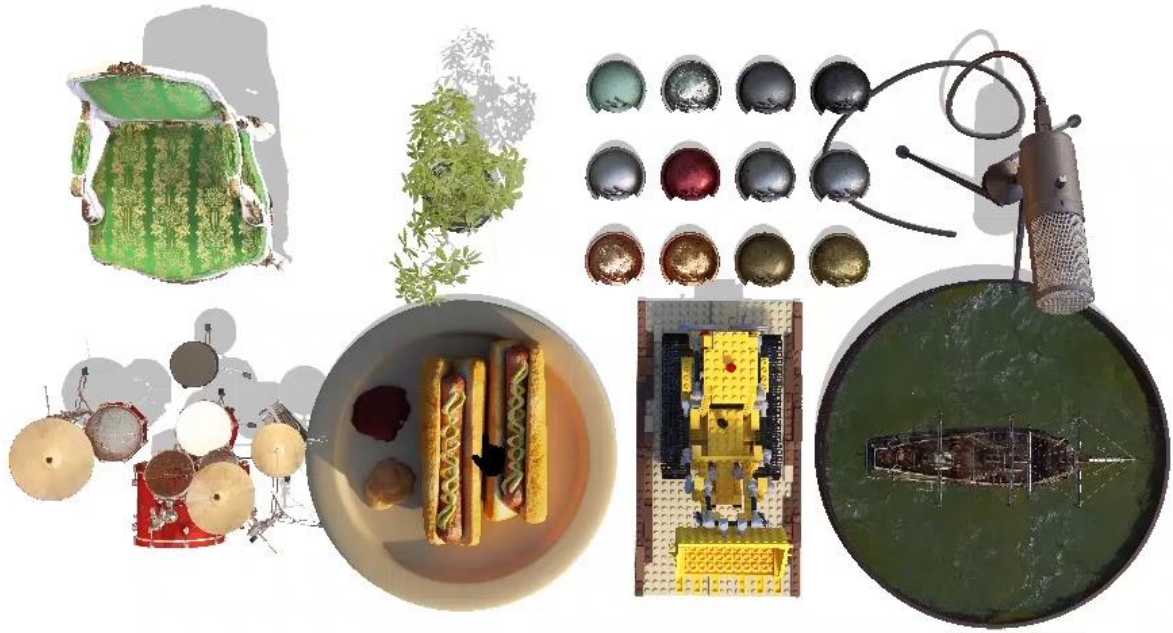
Scene editing



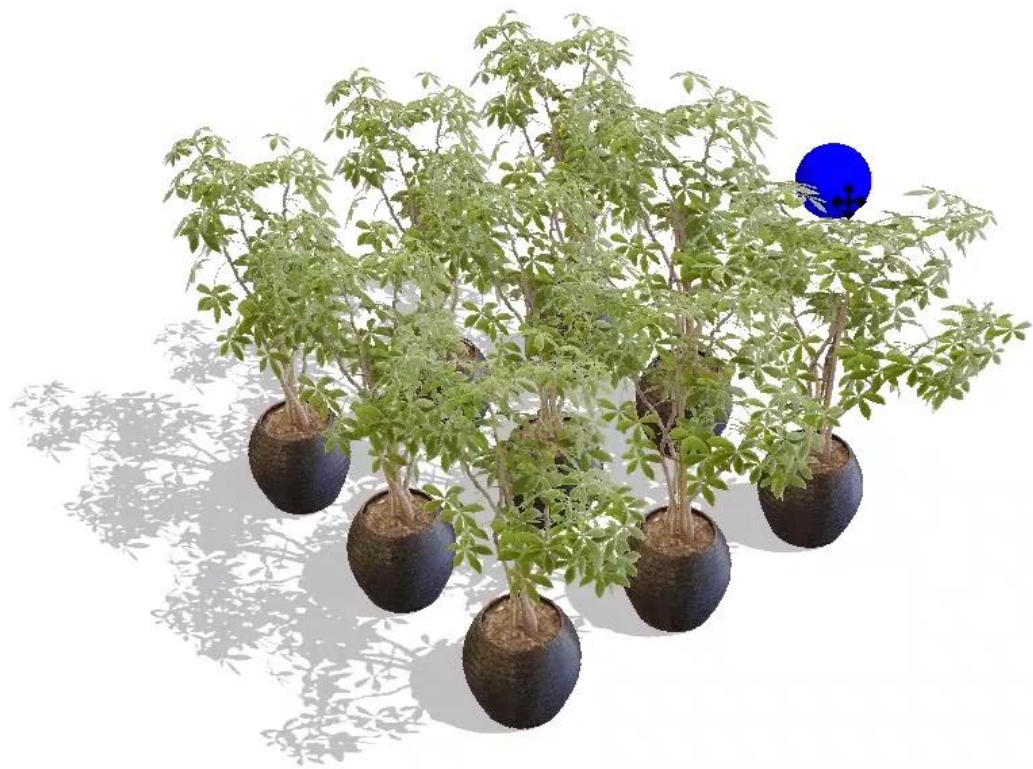
Scene editing











Limitations



Ours, viewing from side



Scene: room, forward-facing

(a) Wrong geometry

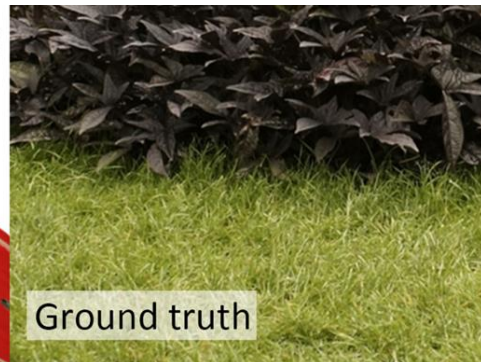


Ours



Scene: drums, synthetic 360°

(b) No semi-transparency



Ours



Scene: flower, unbounded 360°

(c) Fixed mesh resolution

Thank you!

Poster session
THU-AM-009

