

## Supplementary Material for DiffSoup

This supplementary material provides an accompanying video and a detailed document to complement the main paper. The video presents the optimization sequence, novel-view synthesis results with geometry visualizations, and a real-time demonstration in which we interactively render a scene containing *all* models from the *NeRF-Synthetic* [36] and *Shelly* [50] datasets on a mobile device (iPhone 15). This mobile demonstration achieves an average of 51 FPS at  $2K \times 1K$  resolution on camera views computed using Fibonacci spiral sampling over the upper hemisphere (included in our code release). In addition, the following sections provide further details on our experimental setup, method, mathematical derivations, and additional figures and tables.

## A. Experimental Setup

### A.1. Implementation

We implement our method in Python using PyTorch [40] for automatic differentiation. Our differentiable rasterizer is implemented in CUDA and exposed as a custom PyTorch operator via nanobind [21]. Edge splitting routines are likewise implemented in C++ and invoked directly from Python. Our implementation is publicly available at <https://github.com/kenji-tojo/diffsoup>.

### A.2. Hardware

Our experiments were conducted primarily on a workstation equipped with an Intel Core i9-14900K CPU, 64 GB of RAM, and an NVIDIA GeForce RTX 4090 GPU with 24 GB of VRAM. In particular, all results produced by our method were obtained in a single-GPU setting on the RTX 4090. To parallelize the evaluation of baseline methods, we additionally used a separate machine equipped with an NVIDIA RTX A6000 GPU with 48 GB of VRAM. This secondary hardware was used solely to accelerate baseline training; all reported results are from single-GPU runs, and all methods fit within the memory budget of the RTX 4090.

The original MobileNeRF results [7] were obtained using eight NVIDIA V100 GPUs. Reproducing this multi-GPU configuration using a single GPU would not be feasible with our hardware and would also be disproportionate relative to the other baselines. For context, all other baselines—and our method—typically complete training in roughly 30 minutes on a single GPU. In contrast, MobileNeRF’s training is considerably more computationally demanding: keeping all hyperparameters fixed and running on a single GPU results in training times of around 6 hours. Replicating their original eight-GPU setup would multiply the computational cost and exceed our available resources. To enable a fair comparison, we therefore run the MobileNeRF code on the same single GPU used for all other meth-

---

### Algorithm A1: Stochastic opacity masking

---

**Input:** fragments  $\mathcal{F} = \{\mathbf{f}_1, \dots, \mathbf{f}_{|\mathcal{F}|}\}$

**Output:** pixel color  $\hat{\mathbf{C}}$  and color/opacity gradients

#### 1. Forward pass

compute  $f^*$  and  $\hat{\mathbf{C}}$  by sampling thresholds  $\tau$ , and store  $(\mathbf{f}_{f^*}, \hat{\mathbf{C}})$  in image-sized buffers

#### 2. Backward pass

(a) **Color gradient:** differentiate  $\mathcal{L}_1(\hat{\mathbf{C}})$

(b) **Opacity gradient:**

**foreach**  $f' \in \{1, \dots, |\mathcal{F}|\}$  **do**

**if**  $f' = f^*$  **then**  $s \leftarrow 1/\alpha_{f^*}$

**else if**  $D_{f'} < D_{f^*}$  **then**  $s \leftarrow -1/(1 - \alpha_{f'})$

**else**  $s \leftarrow 0$

    add  $\mathcal{L}_1(\hat{\mathbf{C}}) s$  to the opacity gradient of  $f'$

---

ods. This results in an effective batch size one eighth of the original, while the number of iterations and the learning rate remain identical to those reported in the paper.

## B. Method Details

This section provides additional details about our method.

### B.1. Stochastic Opacity Masking Pseudocode

Algorithm A1 summarizes the forward and backward passes of our stochastic opacity masking procedure, as described in the main paper.

### B.2. Hyperparameters

For Adam [27] and VectorAdam [31], we set  $\beta_1 = 0.9$ ,  $\beta_2 = 0.999$ , and  $\epsilon = 1.0 \times 10^{-8}$ . We exponentially anneal the learning rate so that it is reduced by a factor of 100 over the course of training. We use an initial learning rate of  $5.0 \times 10^{-2}$  for color and alpha features, and  $1.0 \times 10^{-2}$  for neural network weights. For vertex positions, we set the initial learning rate to  $1.0 \times 10^{-2}$  for real scenes and  $1.0 \times 10^{-3}$  for synthetic scenes, as geometry in synthetic datasets tends to be more compact.

### B.3. MLP Architecture

For deferred shading, we apply a shared-weight multilayer perceptron (MLP) with two 16-dimensional hidden layers to the per-pixel feature vector. As input to the MLP, we concatenate the 7-dimensional color feature vector with the per-pixel view direction encoded using spherical harmonic bases up to degree 2, resulting in a  $7 + 9 = 16$ -dimensional input vector. All MLP weights are initialized following Xavier initialization [13]. All texture feature vectors are initialized to zero, which results in an initial value of 0.5 after applying the sigmoid.

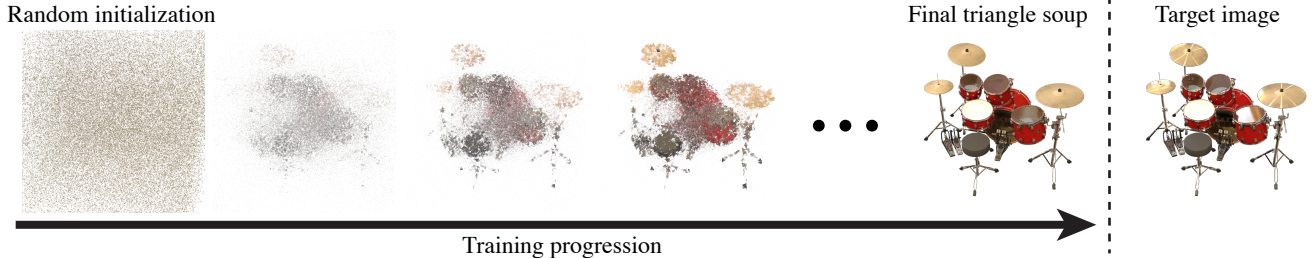


Figure A1. (Left) Starting from a randomly initialized triangle soup, training gradually drives irrelevant triangles to become transparent, and these are regularly removed by our adaptive resampling step. This process yields a clean geometry that captures even thin structures and challenging topology. (Right) The corresponding ground-truth image is shown for reference.

We observe that color vibrancy can be diminished after passing through the MLP. To mitigate this, we interpret the first four channels of the color feature as RGBA and compute the final RGB output as a blend of this RGBA feature and the MLP prediction, yielding a parameter-free, lightweight skip-connection layer [16].

#### B.4. Coarse-to-Fine Optimization

As described in the main paper, we transition from the coarse feature grid to the fine grid at iteration 5000. This is performed by sampling the coarse color feature at each grid coordinate and copying it to the new  $R_{\min}$  level. Because coarse-stage training often produces oversaturated alpha values that hinder later shape refinement, we reinitialize the alpha feature during this transition.

#### B.5. Random Triangle Soup Initialization

For the synthetic data experiments, we additionally report results obtained using a random initialization of the triangle-soup geometry. We generate this initialization by uniformly sampling seed points inside the bounding box used for MobileNeRF [7] and placing a triangle around each point with a random orientation and a radius of 0.01 (the bounding-box scale typically ranges from 2 to 4).

For a randomly initialized triangle soup, we apply a custom adaptive triangle-resampling procedure to avoid unnecessary triangles. Specifically, we first remove triangles that are fully transparent—that is, those that receive no pixel coverage in any training image. We then iteratively split the longest edges until the triangle count reaches the target value, ensuring a consistent number of triangles throughout training. Figure A1 shows the optimization progression from random initialization.

#### B.6. Data Size

Our representation uses 9 floating-point numbers (3 per vertex) to encode the geometry of each triangle. This matches the data footprint of standard 3DGS variants [24], which also require 9 floating-point values (3 for the center, 3 for the rotation, and 3 for the scale) per primitive.

We use a triangle texture resolution of  $R_{\max} = 5$  at inference time, which produces  $(2^4 + 1) \times (2^5 + 1) = 561$  grid points per triangle. At this resolution, all triangle textures can be packed into a 4K grid provided the number of triangles remains below

$$4096^2 / 561 \approx 29,905. \quad (\text{A1})$$

Note that 4K textures are widely supported across rendering environments, including mobile devices.

In the comparison on the *MipNeRF360* [3] dataset, Textured Gaussians (TexGS) [5] employ a  $50 \times 50$  texture grid (2500 texels) per primitive. Because their representation combines RGBA textures with per-primitive spherical harmonic (SH) coefficients, the effective number of texture channels per primitive is  $2500 \times 4 = 10,000$ , whereas our method uses only  $561 \times 8 = 4,488$  channels per triangle (recall that we use a 7-channel color feature plus alpha).

### C. Mathematical Derivations

In this section, we present the full mathematical derivations corresponding to the equations introduced in the method section of the main paper.

#### C.1. Equivalence of stochastic opacity masking and radiance field loss

In the main paper, we claim that our stochastic opacity masking technique selects a fragment  $f$  with probability

$$p(f) = \left( \prod_{f' \in \{f'; D_{f'} < D_f\}} (1 - \alpha_{f'}) \right) \alpha_f, \quad (\text{A2})$$

which coincides with the sample weight  $w_f$  used as the blending weight in the radiance field loss [62] and in the standard volume rendering formulation of NeRFs [36]. This equivalence follows directly from the observation that fragment  $f$  is selected as the foremost visible fragment if and only if all fragments in front of it are *not* visible and fragment  $f$  itself is visible. Fragments behind  $f$  do not influ-



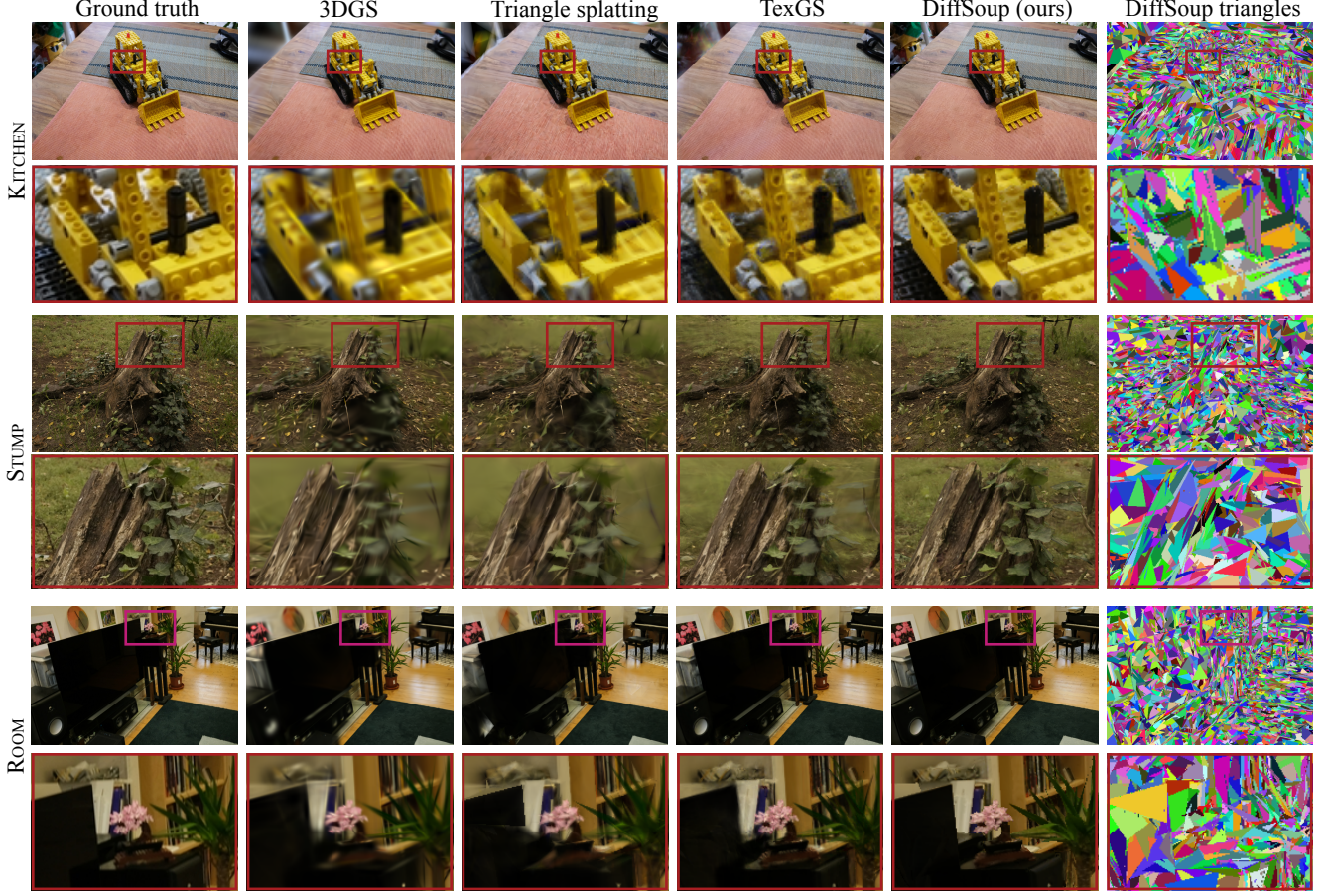


Figure A2. Additional qualitative results on the *MipNeRF360* [3] dataset. (Left to right) Ground-truth test images; views synthesized by 3DGS [24], Triangle Splatting [17], TexGS [5], our method; and a visualization of our reconstructed triangle geometry. Rows correspond to different scenes, and all models use 15K primitives.

ence its visibility and are therefore marginalized out of the probability computation, yielding the expression in (A2).

## C.2. Application of the likelihood-ratio identity

The main paper derives the unbiased estimator using the likelihood-ratio identity [14, 53]. Although this identity is classical and not part of our technical contribution, we briefly verify that it applies to our gradient expression as well, to keep the presentation self-contained.

We first express the expected value in summation form:

$$E_{p(f)}[\mathcal{L}(\hat{\mathcal{C}})] = \sum_{f \in \mathcal{F}} \mathcal{L}(\hat{\mathcal{C}}) p(f). \quad (\text{A3})$$

By differentiating the expression with respect to  $\alpha_{f'}$ , we

obtain

$$\begin{aligned} \frac{\partial}{\partial \alpha_{f'}} E_{p(f)}[\mathcal{L}(\hat{\mathcal{C}})] &= \sum_{f \in \mathcal{F}} \frac{\partial \mathcal{L}(\hat{\mathcal{C}})}{\partial \alpha_{f'}} p(f) \\ &+ \sum_{f \in \mathcal{F}} \mathcal{L}(\hat{\mathcal{C}}) \frac{\partial p(f)}{\partial \alpha_{f'}}, \end{aligned} \quad (\text{A4})$$

where the first term is simply the expected value

$$\sum_{f \in \mathcal{F}} \frac{\partial \mathcal{L}(\hat{\mathcal{C}})}{\partial \alpha_{f'}} p(f) = E_{p(f)} \left[ \frac{\partial \mathcal{L}(\hat{\mathcal{C}})}{\partial \alpha_{f'}} \right], \quad (\text{A5})$$

by the definition of expectation. The second term can also

be written as an expectation by observing that

$$\begin{aligned} \sum_{f \in \mathcal{F}} \mathcal{L}(\hat{c}) \frac{\partial p(f)}{\partial \alpha_{f'}} &= \sum_{f \in \mathcal{F}} \mathcal{L}(\hat{c}) \left( \frac{1}{p(f)} \frac{\partial p(f)}{\partial \alpha_{f'}} \right) p(f) \\ &= \sum_{f \in \mathcal{F}} \mathcal{L}(\hat{c}) \frac{\partial \log p(f)}{\partial \alpha_{f'}} p(f) \\ &= E_{p(f)} \left[ \mathcal{L}(\hat{c}) \frac{\partial \log p(f)}{\partial \alpha_{f'}} \right]. \end{aligned} \quad (\text{A6})$$

Combining the two parts yields the gradient estimator introduced in the main paper.

### C.3. Gradient of the score term

Based on the probability  $p(f)$ , the main paper derives the score term as

$$\frac{\partial \log p(f)}{\partial \alpha_{f'}} = \begin{cases} 1/\alpha_f, & f' = f, \\ -1/(1 - \alpha_{f'}), & D_{f'} < D_f, \end{cases} \quad (\text{A7})$$

which is used for computing the loss gradient. This expression follows directly from applying the chain rule,

$$\frac{\partial \log p(f)}{\partial \alpha_{f'}} = \frac{1}{p(f)} \frac{\partial p(f)}{\partial \alpha_{f'}}, \quad (\text{A8})$$

where we observe that

$$\frac{\partial p(f)}{\partial \alpha_{f'}} = \begin{cases} p(f)/\alpha_f, & f' = f, \\ -p(f)/(1 - \alpha_{f'}), & D_{f'} < D_f, \end{cases} \quad (\text{A9})$$

which is obtained by differentiating the product expression of  $p(f)$  with respect to  $\alpha_{f'}$ .

## D. Additional Figures

Figure A2 shows additional qualitative comparison results on the *MipNeRF360* [3] dataset. Overall, our opaque textured-triangle representation produces the sharpest visuals under a tight primitive budget, and in particular yields noticeably sharper results than TexGS [5], which uses textured translucent primitives. Figure A3 visualizes the texture memory used in our method.

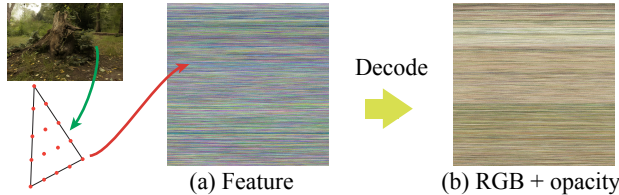


Figure A3. We represent RGB color and opacity within each triangle by linearly interpolating values stored at subdivision points. (a) Each subdivision point stores eight 32-bit floating-point values: a 7-dimensional color feature vector and one opacity value. (b) The color feature vector is decoded into RGB using a small neural network, while the final channel is used directly as opacity.

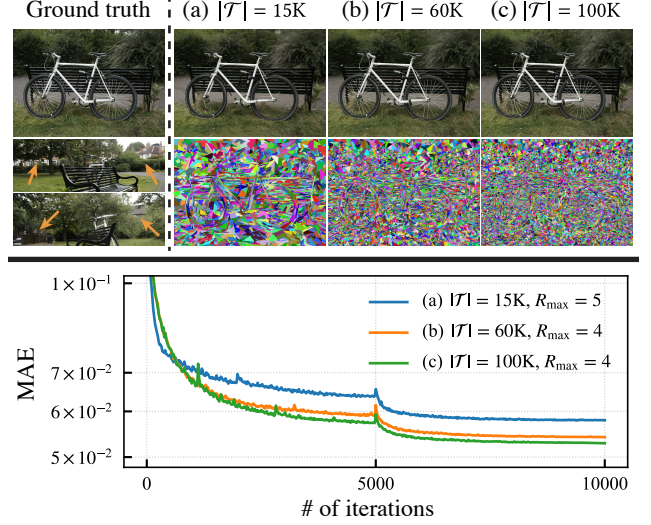


Figure A4. (Top row, left to right) Ground-truth images for the BICYCLE scene from the *MipNeRF360* [3] dataset. This scene exhibits an extremely large spatial extent, as indicated by the distant objects highlighted with orange arrows. (a) Triangle-soup rendering and reconstructed geometry using our method with  $|\mathcal{T}| = 15\text{K}$  triangles and a texture resolution of  $R_{\max} = 5$ . (b) Rendering and geometry using  $|\mathcal{T}| = 60\text{K}$  triangles and  $R_{\max} = 4$ . (c) Rendering and geometry using  $|\mathcal{T}| = 100\text{K}$  triangles and  $R_{\max} = 4$ . (Bottom row) Test error measured as mean absolute error (MAE) over training iterations for the three configurations, demonstrating convergence behavior and clarifying the representational capacity.

**Challenging case** Outdoor scenes are generally more challenging to represent with a small number of triangles than indoor or synthetic scenes, primarily due to their extremely large spatial extent. A particularly notable example in our experiments is the BICYCLE scene from the *MipNeRF360* [3] dataset (Figure A4). In this scene, important visual details—such as the spokes of the bicycle wheel—occupy only a tiny region within a very large overall scene volume.

Our adaptive resampling method encourages roughly uniform pixel coverage per triangle to make the most effective use of the available texture resolution. In such extreme cases, however, geometric structures that fall below a certain relative world-space scale can be missed. This effect can be seen in Figure A4 (a), where large objects are reconstructed clearly under a tight triangle budget, but the thin wheel spokes are lost.

A potential remedy is to decrease the average screen-space footprint of each triangle by increasing the number of triangles. Figure A4 (b) illustrates this approach: we halve the screen-space edge-split threshold, increase the target triangle count to four times that of (a), and reduce the texture resolution by one subdivision level. This adjustment yields noticeably better reconstruction of fine ge-

Table A1. Additional quantitative comparisons with MobileNeRF [7] on synthetic scenes from the *NeRF-Synthetic* [36] and *Shelly* [50] datasets. In addition to the PSNR values reported in the main paper, we report SSIM (top table) and LPIPS (bottom table) scores for each scene. The face count for MobileNeRF is averaged across scenes, whereas all other methods use the exact face count specified. See the main paper for further discussion.

Method	<i>NeRF-Synthetic</i> [36] (SSIM $\uparrow$ )					<i>Shelly</i> [50] (SSIM $\uparrow$ )			
	SHIP	CHAIR	MIC	DRUMS	# Faces $\downarrow$	KHADY	KITTEN	PUG	# Faces $\downarrow$
MobileNeRF [7]	0.817	0.958	0.961	0.910	159K	0.810	0.931	0.857	275K
MobileNeRF + QEM [12]	0.554	0.793	0.857	0.739	15K	0.686	0.847	0.729	15K
Ours w/ QEM init.	<b>0.848</b>	<b>0.975</b>	<b>0.975</b>	<b>0.931</b>	15K	<b>0.838</b>	<b>0.951</b>	<b>0.903</b>	15K
Ours w/ random init.	0.828	0.972	0.972	0.927	15K	0.838	0.950	0.899	15K

Method	<i>NeRF-Synthetic</i> [36] (LPIPS $\downarrow$ )					<i>Shelly</i> [50] (LPIPS $\downarrow$ )			
	SHIP	CHAIR	MIC	DRUMS	# Faces $\downarrow$	KHADY	KITTEN	PUG	# Faces $\downarrow$
MobileNeRF [7]	0.173	0.042	0.073	0.094	159K	0.205	0.099	0.192	275K
MobileNeRF + QEM [12]	0.453	0.227	0.235	0.314	15K	0.306	0.170	0.294	15K
Ours w/ QEM init.	<b>0.119</b>	<b>0.024</b>	<b>0.037</b>	<b>0.066</b>	15K	<b>0.153</b>	<b>0.074</b>	<b>0.116</b>	15K
Ours w/ random init.	0.147	0.027	0.043	0.071	15K	0.154	0.077	0.120	15K

Table A2. Quantitative results for the ablation study on the *Mip-NeRF360* [3] dataset. From top to bottom: (1) our method with opacity fixed to 1, (2) a variant that uses only the highest-resolution feature grid with  $R_{\max} = 5$  (i.e., without multi-resolution features and without the coarse-to-fine strategy), (3) a variant that uses multi-resolution features but does not use the coarse-to-fine strategy, and (4) our full method combining both multi-resolution features and the coarse-to-fine strategy. PSNR, SSIM, and LPIPS scores are reported as scene-level averages. The number of triangles is fixed at  $|\mathcal{T}| = 15\text{K}$  for all methods.

Method	PSNR $\uparrow$	SSIM $\uparrow$	LPIPS $\downarrow$
w/o opacity	24.65	0.744	0.205
$R_{\min} = R_{\max} = 5$	23.54	0.719	0.226
$R_{\min} = 2, R_{\max} = 5$	24.48	0.747	<b>0.203</b>
<b>Ours full</b>	<b>24.76</b>	<b>0.748</b>	0.204

ometry while keeping the overall texture memory roughly unchanged. If the budget allows, further fidelity can be achieved by reducing the edge-split threshold to one third of (a) and increasing the triangle budget to 100K, as shown in Figure A4 (c).

While uniformly increasing the number of triangles improves representational capacity, as seen in the convergence plot of Figure A4, it is still highly desirable to remain within the original tight triangle budget. We believe this is achievable through more sophisticated strategies for allocating triangle density and texture resolution—beyond our current uniform, image-space-driven resampling—which could further improve the reconstruction of challenging outdoor scenes under very limited primitive budgets (e.g., the 15K budget used in our experiments). Designing such spatially

adaptive edge-splitting and texture-resolution schemes remains an important direction for future work.

## E. Additional Quantitative Results

This section presents additional quantitative results for experiments presented in the main paper. Table A1 reports SSIM and LPIPS scores for the comparison against MobileNeRF [7] on synthetic datasets, supplementing the PSNR results presented in the main paper. In terms of these metrics, the two variants of our method consistently achieve the best and second-best performance.

Table A2 reports additional quantitative scores for the ablation study conducted in the main paper. The inability to represent sub-triangle geometry when binary opacity is not used is especially evident in the degraded SSIM and LPIPS scores. Interestingly, the variant that uses the multi-resolution feature grid throughout optimization (i.e., without the coarse-to-fine strategy) provides a competitive alternative, whereas the variant that relies solely on the highest-resolution grid consistently performs the worst. This highlights the effectiveness of our multi-resolution feature-grid design. Our full method additionally incorporates the coarse-to-fine strategy and consistently achieves the best or second-best results across all metrics. We observe that coarse-to-fine training often improves triangle coverage and geometric fitting, as reflected in higher PSNR scores compared to the multi-resolution-only variant.

## F. Additional Comparisons and Ablations

This section presents additional experiments and analyses that complement the main paper, including a comparison



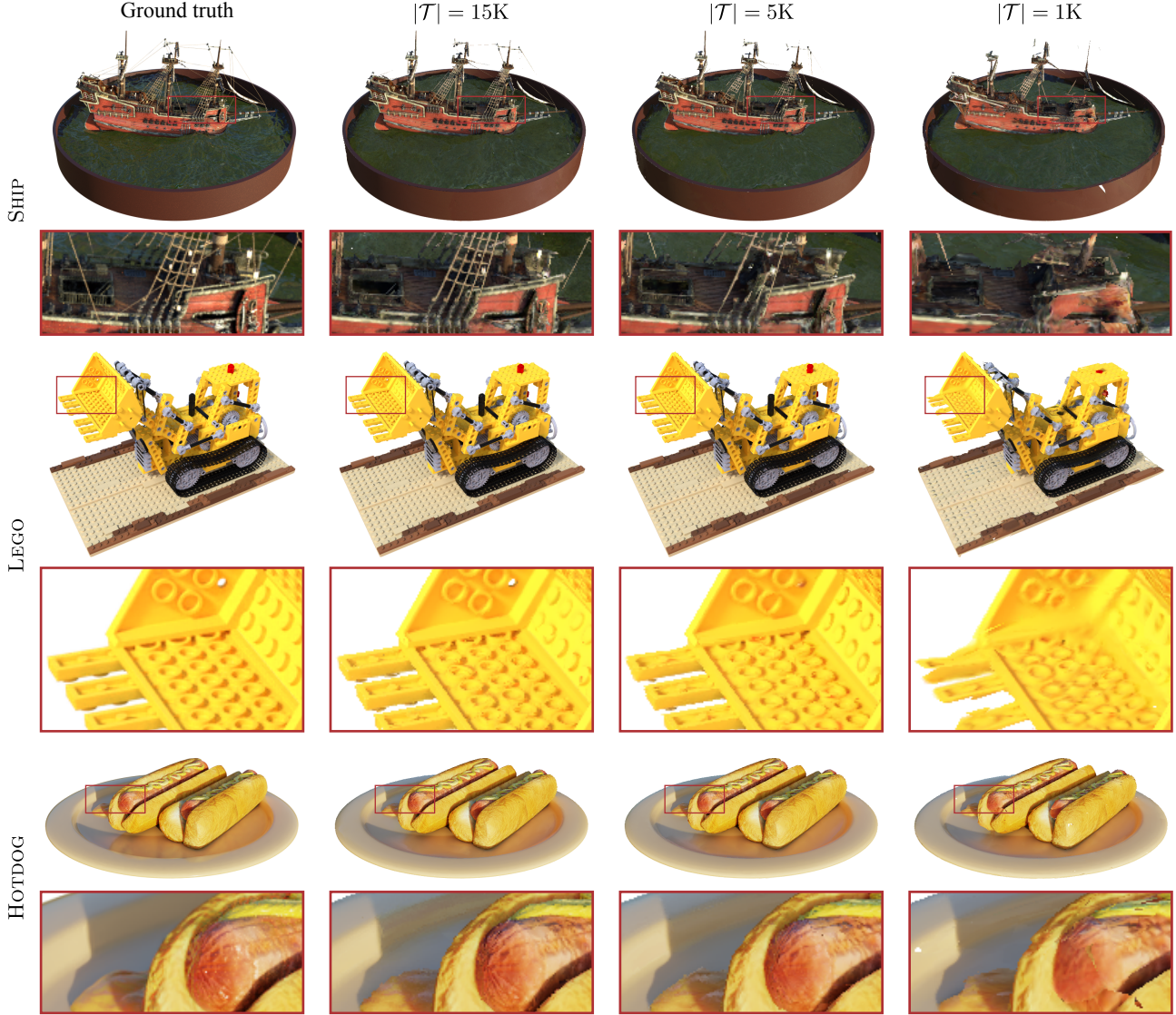


Figure A5. Effect of varying the triangle budget on reconstruction quality for *NeRF-Synthetic* [36] scenes. Each row shows (left to right) a ground-truth view followed by our results with  $|\mathcal{T}| = 15\text{K}$ ,  $5\text{K}$ , and  $1\text{K}$  triangles, respectively.

with 3DGS-MCMC [25], storage costs, training time, and a primitive-budget sweep.

### F.1. Comparison with 3DGS-MCMC

3DGS-MCMC [25] provides explicit control over the primitive count while relying on the same primitives as 3DGS, making it a relevant additional baseline. We evaluate it under the same 15K budget on the *MipNeRF360* [3] dataset. Table A3 reports quality metrics, and Table A4 reports rendering speed. Although 3DGS-MCMC outperforms vanilla 3DGS in reconstruction quality, it remains slower than our method and lower in quality than the textured approaches (TexGS and ours) in this low-budget regime.

### F.2. Storage Cost

Table A5 reports the inference-time GPU memory footprint and storage cost of the models used in our experiments. Among the textured methods (i.e., TexGS, MobileNeRF, and ours), our approach achieves the lowest memory and storage cost. Both MobileNeRF and our method leverage 8-bit RGBA textures, which reduce memory usage and enable efficient storage and data transmission via PNG compression. We were unable to load the scene shown in our supplementary video, comprising 14 models, on a mobile device using MobileNeRF; after loading 6 models, the total memory footprint reached 1.58 GB, exceeding the browser memory limit. In contrast, our full scene requires only 916 MB.



Table A3. Reconstruction quality comparison with 3DGS-MCMC [25] on the *MipNeRF360* [3] dataset. All methods use 15K primitives.

Method	PSNR $\uparrow$	SSIM $\uparrow$	LPIPS $\downarrow$
3DGS [24]	23.72	0.664	0.420
3DGS-MCMC [25]	24.22	0.692	0.389
TexGS [5]	<b>24.80</b>	0.697	0.270
<b>Ours</b>	24.76	<b>0.748</b>	<b>0.204</b>

Table A4. Rendering speed comparison with 3DGS-MCMC [25] on the *MipNeRF360* [3] dataset. All methods use 15K primitives.

Method	FPS $\uparrow$ across resolutions		
	Full	1/2	1/4
3DGS [24]	115	482	1.32K
3DGS-MCMC [25]	214	598	943
TexGS [5]	16.8	49.1	94.8
<b>Ours (CUDA)</b>	<b>1.96K</b>	<b>6.11K</b>	<b>13.7K</b>

Table A5. Average GPU memory footprint and storage cost of the models used in our experiments. The average primitive count is also reported.

Method	Memory $\downarrow$	Storage $\downarrow$	# Prims
3DGS [24]	<b>3.38MB</b>	<b>3.38MB</b>	15K
TexGS [5]	576MB	576MB	15K
MobileNeRF [7]	329MB	118MB	209K
<b>Ours</b>	65.9MB	44.3MB	15K

Table A6. Reconstruction quality (PSNR) across triangle budgets on *NeRF-Synthetic* [36] scenes. Quality degrades gracefully from  $|\mathcal{T}|=15\text{K}$  to 5K, with a more noticeable drop at 1K.

# Triangles	PSNR $\uparrow$				
	SHIP	MIC	LEGO	HOTDOG	FICUS
$ \mathcal{T}  = 15\text{K}$	26.68	30.18	29.78	33.53	27.51
$ \mathcal{T}  = 5\text{K}$	26.12	30.03	28.21	33.45	27.39
$ \mathcal{T}  = 1\text{K}$	24.34	28.26	24.36	31.40	24.27

### F.3. Training Time

For the GARDEN scene at the 1/4 resolution reported in Table 1 of the main paper, 3DGS completes training in 4.5 minutes. Our method requires a longer training time of 13.5 minutes, reflecting the use of our prototype software rasterizer during training. Despite this, our training is still significantly faster than TexGS, which also relies on textures and requires 46 minutes. Importantly, our stochastic opacity masking introduces only a small overhead: ablating it reduces the training time to 11.5 minutes.

### F.4. Triangle Budget Analysis

Increasing the number of primitives improves reconstruction fidelity, especially for outdoor scenes with large spatial extent (see Figure A4). For per-object scenes in the *NeRF-Synthetic* [36] dataset, the triangle count can be further reduced with only minor quality degradation. We evaluate this on a subset of scenes using the same QEM initialization as in our MobileNeRF comparison; Table A6 and Figure A5 report quantitative and qualitative results, respectively. For example, on SHIP, lowering the budget from  $|\mathcal{T}| = 15\text{K}$  to 5K decreases PSNR by only 0.56 dB, with the main visible artifact being a slight loss of thin structures. For scenes with an overall compact shape, such as HOTDOG, the drop is even smaller and barely perceptible. In contrast, reducing to 1K triangles leads to noticeable degradation across all scenes, both qualitatively and quantitatively. Automatically selecting the optimal budget given a user-specified error tolerance is an interesting direction for future work.