Speeding Up the Learning of 3D Gaussians with Much Shorter Gaussian Lists

Supplementary Material

8. Sum of Weights Equals Unity

We proof the unity of the sum of weights along each ray. This ensures that we can calculate the entropy using weights as a distribution. In volumetric rendering with N Gaussians and a background, the transmittance satisfies,

$$T_i = \prod_{j=1}^{i-1}(1-\alpha_j), \quad T_1 = 1, \tag{16}$$

and the blending weight is $w_i = T_i\alpha_i$ for $i \in [1, N]$. The background weight is $w_{N+1} = T_{N+1}$. We prove that $\sum_{i=1}^{N+1} w_i = 1$.

Proof. Equivalently, we show that $T_{N+1} = 1 - \sum_{i=1}^{N} w_i$ by mathematical induction.

Base case ($N = 1$):

$$\sum_{i=1}^{1} w_i + T_2 = T_1\alpha_1 + T_1(1-\alpha_1) = T_1 = 1. \tag{17}$$

Inductive step: Assume $\sum_{i=1}^{k} w_i = 1 - T_{k+1}$.

For $N = k + 1$:

$$\sum_{i=1}^{k+1} w_i = \sum_{i=1}^{k} w_i + w_{k+1} = (1 - T_{k+1}) + T_{k+1}\alpha_{k+1} \tag{18}$$

$$= 1 - T_{k+1}(1-\alpha_{k+1}) = 1 - T_{k+2}. \tag{19}$$

Therefore, $\sum_{i=1}^{N+1} w_i = 1$.

9. Gradient of Entropy with respect to Alpha

Setup. For a pixel with N Gaussians, recall that the transmittance and blending weights for the i-th Gaussian are:

$$T_i = \prod_{j=1}^{i-1}(1-\alpha_j), \quad T_1 = 1, \tag{20}$$

$$w_i = T_i\alpha_i, \quad i \in [1, N]. \tag{21}$$

Including the background weight $w_{N+1} = T_{N+1}$, we have $\sum_{i=1}^{N+1} w_i = 1$ (see Sec. 8).

Entropy loss. The entropy for one pixel is

$$H = -\sum_{i=1}^{N+1} w_i \log w_i, \tag{22}$$

with gradient

$$\frac{\partial H}{\partial w_i} = -\log w_i - 1. \tag{23}$$

Chain rule. For each Gaussian, the learnable attributes (position, scale, rotation, opacity) are encapsulated in α_i. We compute the gradient with respect to α_i using the chain rule:

$$\frac{\partial H}{\partial \alpha_i} = \sum_{k=i}^{N+1} \frac{\partial H}{\partial w_k}\frac{\partial w_k}{\partial \alpha_i}. \tag{24}$$

Key components. We derive $\frac{\partial w_k}{\partial \alpha_i}$ for three cases.

Case 1: $k = i$**.**

$$\frac{\partial w_i}{\partial \alpha_i} = \frac{\partial (T_i\alpha_i)}{\partial \alpha_i} = T_i. \tag{25}$$

Case 2: $k > i$**.** Since $w_k = T_k\alpha_k = \prod_{j=1}^{k-1}(1-\alpha_j)\cdot\alpha_k$, we have

$$\frac{\partial w_k}{\partial \alpha_i} = \alpha_k \cdot \frac{\partial}{\partial \alpha_i}\left[\prod_{j=1}^{k-1}(1-\alpha_j)\right] \tag{26}$$

$$= \alpha_k \cdot \prod_{j=1, j\neq i}^{k-1}(1-\alpha_j)\cdot(-1) \tag{27}$$

$$= -\frac{\prod_{j=1}^{k-1}(1-\alpha_j)\cdot\alpha_k}{1-\alpha_i} \tag{28}$$

$$= -\frac{w_k}{1-\alpha_i}. \tag{29}$$

Case 3: $k = N + 1$**.**

$$\frac{\partial w_{N+1}}{\partial \alpha_i} = \frac{\partial T_{N+1}}{\partial \alpha_i} = -\frac{T_{N+1}}{1-\alpha_i} = -\frac{w_{N+1}}{1-\alpha_i}. \tag{30}$$

General formula. Substituting the three cases into the chain rule, we obtain:

$$\frac{\partial H}{\partial \alpha_i} = \frac{\partial H}{\partial w_i}\frac{\partial w_i}{\partial \alpha_i} + \sum_{k=i+1}^{N+1}\frac{\partial H}{\partial w_k}\frac{\partial w_k}{\partial \alpha_i} \tag{31}$$

$$= (-\log w_i - 1)T_i + \sum_{k=i+1}^{N+1}(-\log w_k - 1)\left(-\frac{w_k}{1-\alpha_i}\right) \tag{32}$$

$$= (-\log w_i - 1)T_i + \frac{1}{1-\alpha_i}\sum_{k=i+1}^{N+1}(\log w_k + 1)w_k. \tag{33}$$

Define the intermediate variable:

$$R_i = \sum_{k=i}^{N+1} (\log w_k + 1) w_k. \tag{34}$$

Then the gradient simplifies to:

$$\frac{\partial H}{\partial \alpha_i} = (-\log w_i - 1) T_i + \frac{R_{i+1}}{1 - \alpha_i}. \tag{35}$$

Efficient computation. The intermediate values R_i can be computed efficiently in reverse order:

$$R_{N+1} = (\log w_{N+1} + 1) w_{N+1}, \tag{36}$$

$$R_i = (\log w_i + 1) w_i + R_{i+1}, \quad i = N, \ldots, 1. \tag{37}$$

This backward accumulation requires only $O(N)$ operations and can be parallelized across pixels in CUDA.

10. Gradients with respect to Attributes

Having derived $\frac{\partial H}{\partial \alpha_i}$, we now compute gradients with respect to the learnable Gaussian attributes using the chain rule.

Opacity. Recall that $\alpha_i = \sigma_i \cdot g_i$, where $\sigma_i \in [0, 1]$ is the opacity and g_i is the 2D Gaussian density. The gradient with respect to opacity is:

$$\frac{\partial H}{\partial \sigma_i} = \frac{\partial H}{\partial \alpha_i} \frac{\partial \alpha_i}{\partial \sigma_i} = \frac{\partial H}{\partial \alpha_i} \cdot g_i. \tag{38}$$

Gaussian density. The 2D Gaussian density is given by:

$$g_i = \exp(d_i), \tag{39}$$

where d_i is the exponent term (quadratic form):

$$d_i = -\frac{1}{2} (x - \mu_i')^T (\Sigma_i')^{-1} (x - \mu_i'). \tag{40}$$

Here μ_i' and Σ_i' denote the projected 2D mean and covariance of the i-th Gaussian.

The gradient with respect to d_i is:

$$\frac{\partial H}{\partial d_i} = \frac{\partial H}{\partial \alpha_i} \frac{\partial \alpha_i}{\partial g_i} \frac{\partial g_i}{\partial d_i} \tag{41}$$

$$= \frac{\partial H}{\partial \alpha_i} \cdot \sigma_i \cdot g_i \tag{42}$$

$$= \sigma_i g_i \cdot \frac{\partial H}{\partial \alpha_i}. \tag{43}$$

Inverse covariance matrix. The gradient with respect to the inverse covariance matrix is:

$$\frac{\partial H}{\partial (\Sigma_i')^{-1}} = \frac{\partial H}{\partial d_i} \frac{\partial d_i}{\partial (\Sigma_i')^{-1}} \tag{44}$$

$$= \frac{\partial H}{\partial d_i} \cdot \left(-\frac{1}{2} (x - \mu_i')(x - \mu_i')^T \right). \tag{45}$$

Mean. The gradient with respect to the 2D mean is:

$$\frac{\partial H}{\partial \mu_i'} = \frac{\partial H}{\partial d_i} \frac{\partial d_i}{\partial \mu_i'} \tag{46}$$

$$= \frac{\partial H}{\partial d_i} \cdot (\Sigma_i')^{-1} (x - \mu_i'). \tag{47}$$

11. Additional Results

The training efficiency across iterations, as illustrated in Fig. 13, demonstrates that our method achieves the fastest convergence among all compared approaches. Fig. 14 presents the PSNR progression throughout the training. Both DashGaussian and our method employ resolution scheduling, which results in higher PSNR values at lower resolutions compared to other methods. Fig. 15 confirms the effectiveness of both proposed modules (scale reset and entropy regularization) across all evaluated scenes. The notation convention is detailed in Sec. 5.4.

We also report the average testing FPS on Mip-NeRF 360 in Tab. 9, measured at the same rendering resolution for all methods, where our method achieves the highest rendering throughput among all compared approaches.

Table 9. Average testing FPS on the Mip-NeRF 360 dataset.

Method	FPS↑
3DGS	140.91
Taming	219.60
LiteGS	233.76
Ours	**343.92**

Qualitative comparisons of the rendered results are presented in Fig. 16, Fig. 17, Fig. 18, and Fig. 19. Additional qualitative comparisons of rendering with our method and DashGaussian at specific training times are provided in Fig. 20 of the supplementary material.

Heatmaps measuring the per-tile Gaussian list lengths in color are presented in Fig. 21. Please also see our video for additional visualizations of Gaussian list lengths from multiple testing perspectives and how they evolve during training. The ablation study examining the effect of individual modules is detailed in Fig. 22. Per-pixel entropy heatmap comparison between LiteGS and our method across scenes in the Mip-NeRF 360 dataset is shown in Fig. 23. Per-scene distributions of Gaussian length, scale, and opacity are presented in Fig. 24, demonstrating consistent improvements across all scenes. Gaussians are visualized in Fig. 25, showing our method produces smaller Gaussians than the original 3DGS.

12. Code

Code is available at:
https://github.com/MachinePerceptionLab/ShorterSplatting.

13. Video

We provide a video to show more visualizations and details.

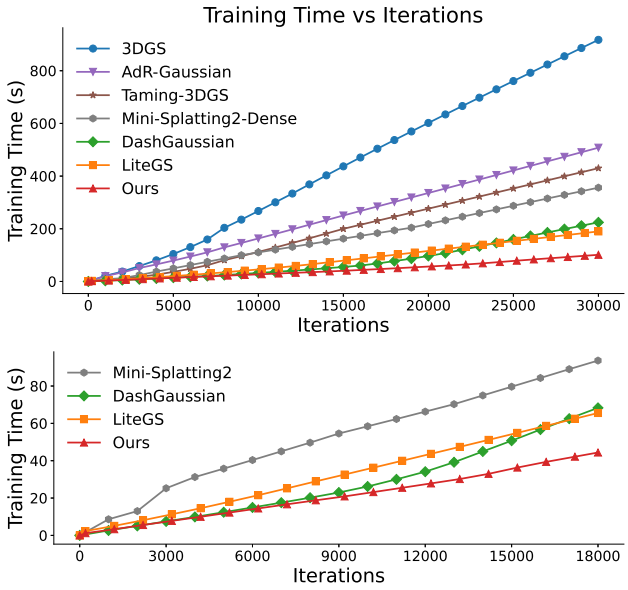


Figure 13. Training time versus iterations for full and compact models, corresponding to Fig. 9.

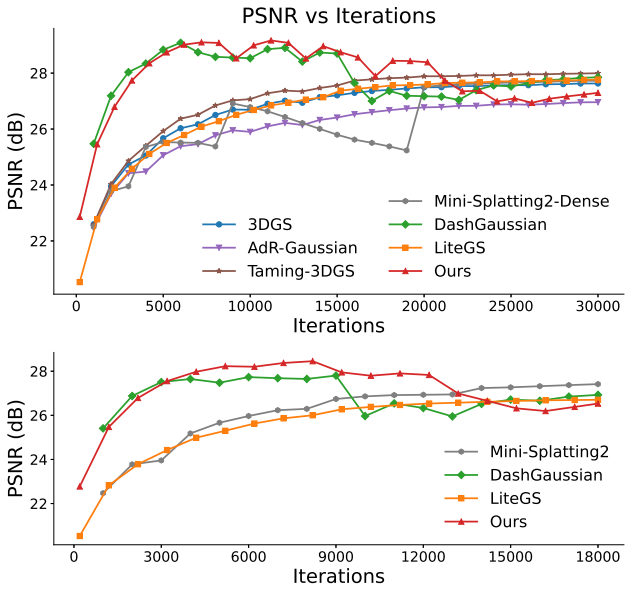


Figure 14. PSNR convergence across iterations for full and compact models, corresponding to Fig. 10.

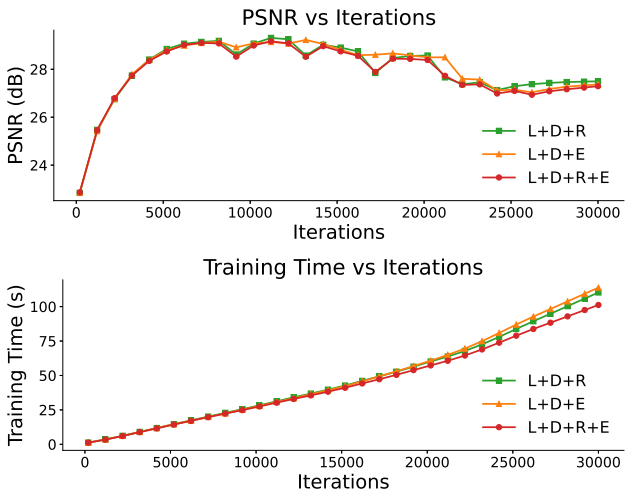
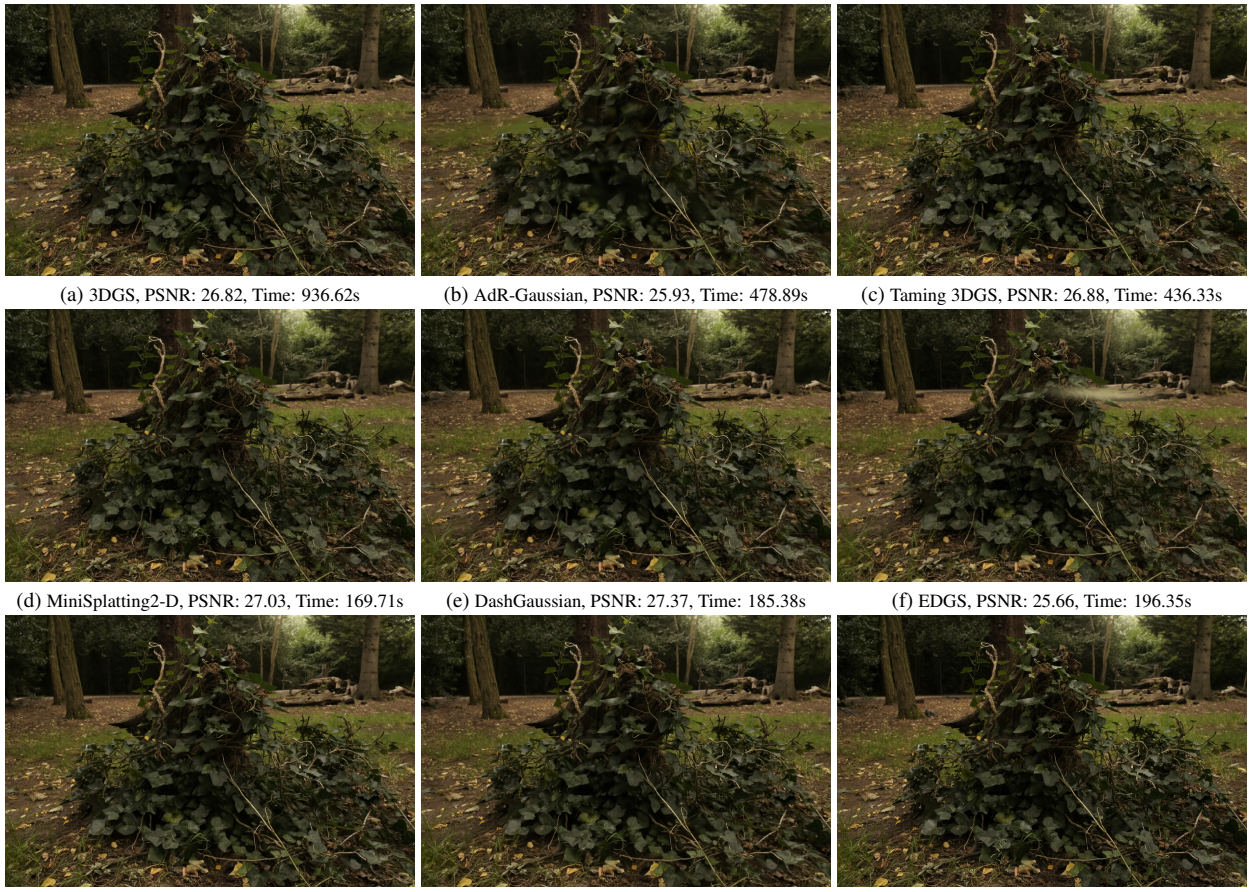


Figure 15. PSNR and time curves across iterations showing effects of individual modules. See Sec. 5.4 for more details.

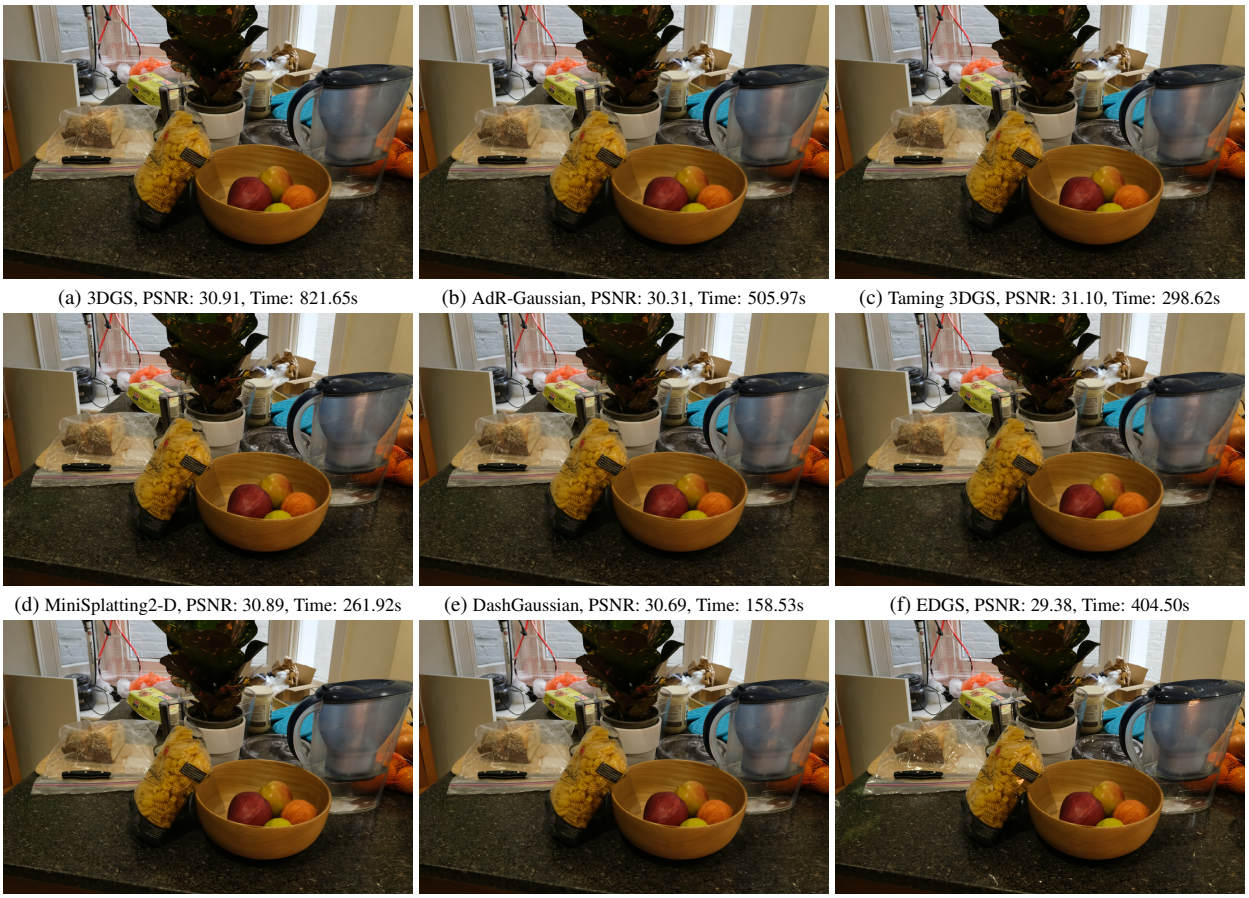


(a) 3DGS, PSNR: 26.82, Time: 936.62s (b) AdR-Gaussian, PSNR: 25.93, Time: 478.89s (c) Taming 3DGS, PSNR: 26.88, Time: 436.33s

(d) MiniSplatting2-D, PSNR: 27.03, Time: 169.71s (e) DashGaussian, PSNR: 27.37, Time: 185.38s (f) EDGS, PSNR: 25.66, Time: 196.35s

(g) LiteGS, PSNR: 27.53, Time: 176.10s (h) Ours, PSNR: 27.33, Time: 87.62s (i) Ground Truth

Figure 16. Qualitative comparison of rendered results on scene Stump. Our method achieves comparable rendering quality (PSNR: 27.33) while being the fastest, with training time of only 87.62 seconds.

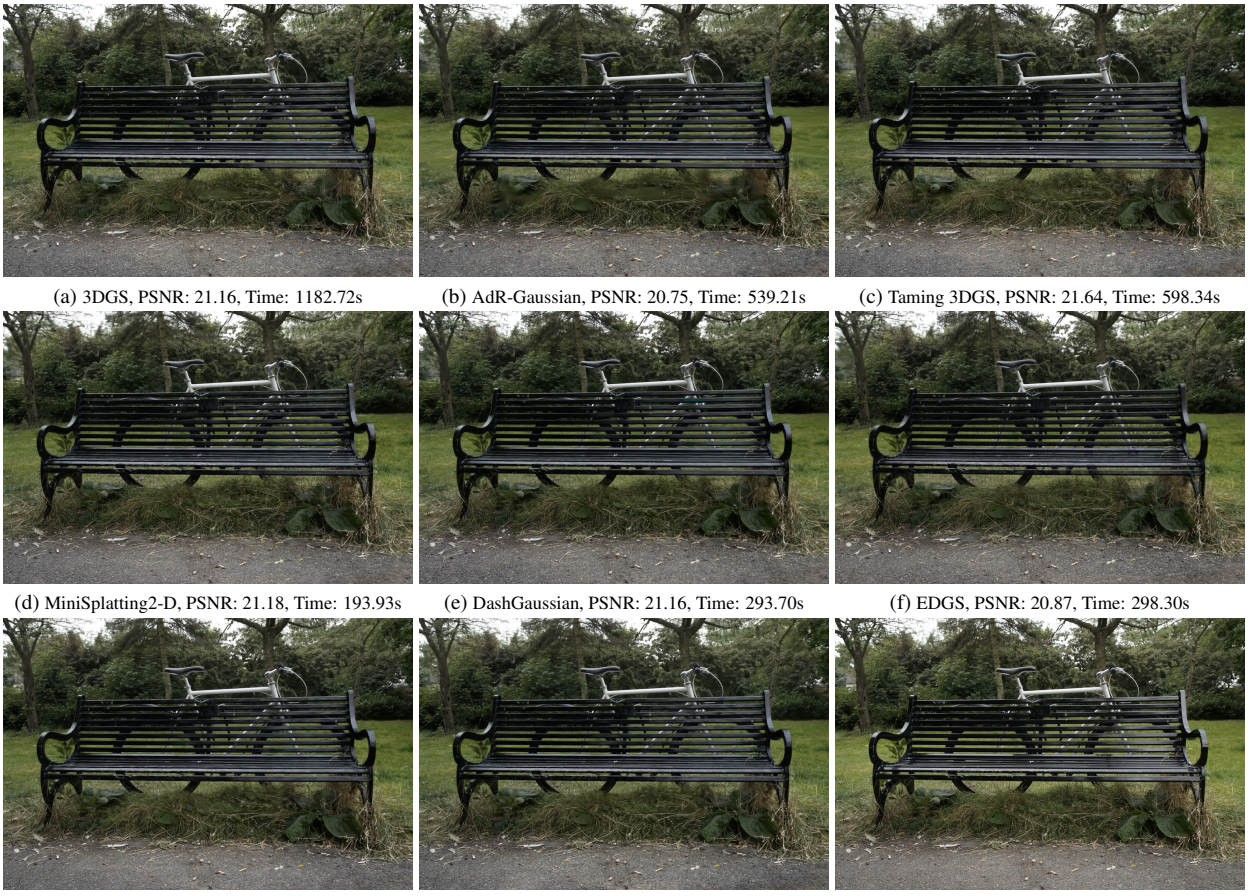


(a) 3DGS, PSNR: 30.91, Time: 821.65s (b) AdR-Gaussian, PSNR: 30.31, Time: 505.97s (c) Taming 3DGS, PSNR: 31.10, Time: 298.62s

(d) MiniSplatting2-D, PSNR: 30.89, Time: 261.92s (e) DashGaussian, PSNR: 30.69, Time: 158.53s (f) EDGS, PSNR: 29.38, Time: 404.50s

(g) LiteGS, PSNR: 30.72, Time: 179.39s (h) Ours, PSNR: 30.38, Time: 86.66s (i) Ground Truth

Figure 17. Qualitative comparison of rendered results on scene Counter. Our method achieves comparable rendering quality (PSNR: 30.38) while being the fastest, with training time of only 86.66 seconds.

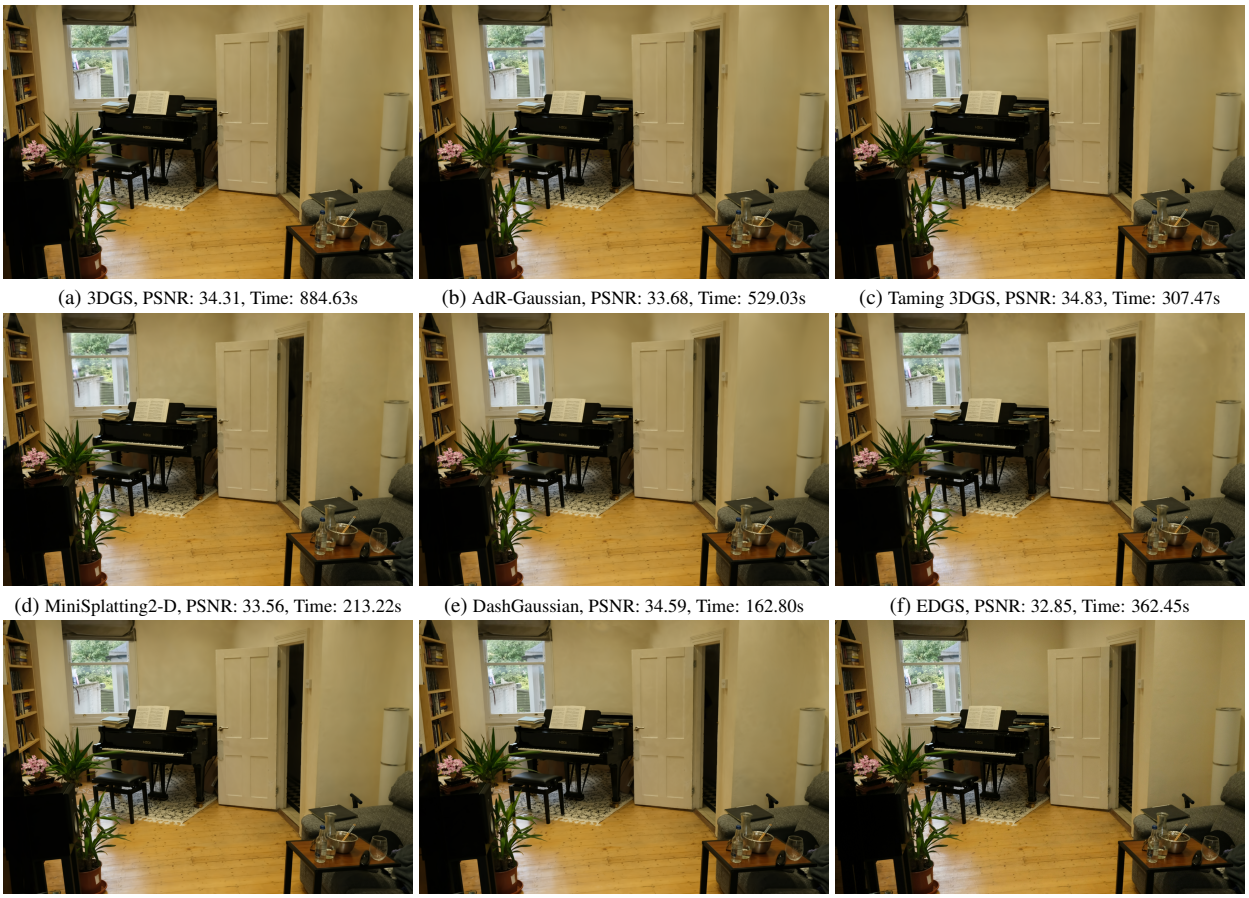


(a) 3DGS, PSNR: 21.16, Time: 1182.72s (b) AdR-Gaussian, PSNR: 20.75, Time: 539.21s (c) Taming 3DGS, PSNR: 21.64, Time: 598.34s

(d) MiniSplatting2-D, PSNR: 21.18, Time: 193.93s (e) DashGaussian, PSNR: 21.16, Time: 293.70s (f) EDGS, PSNR: 20.87, Time: 298.30s

(g) LiteGS, PSNR: 21.27, Time: 238.38s (h) Ours, PSNR: 21.29, Time: 115.37s (i) Ground Truth

Figure 18. Qualitative comparison of rendered results on scene Bicycle. Our method achieves comparable rendering quality (PSNR: 21.29) while being the fastest, with training time of only 115.37 seconds.



(a) 3DGS, PSNR: 34.31, Time: 884.63s (b) AdR-Gaussian, PSNR: 33.68, Time: 529.03s (c) Taming 3DGS, PSNR: 34.83, Time: 307.47s

(d) MiniSplatting2-D, PSNR: 33.56, Time: 213.22s (e) DashGaussian, PSNR: 34.59, Time: 162.80s (f) EDGS, PSNR: 32.85, Time: 362.45s

(g) LiteGS, PSNR: 34.79, Time: 174.86s (h) Ours, PSNR: 34.23, Time: 79.42s (i) Ground Truth

Figure 19. Qualitative comparison of rendered results on scene Room. Our method achieves comparable rendering quality (PSNR: 34.23) while being the fastest, with training time of only 79.42 seconds.

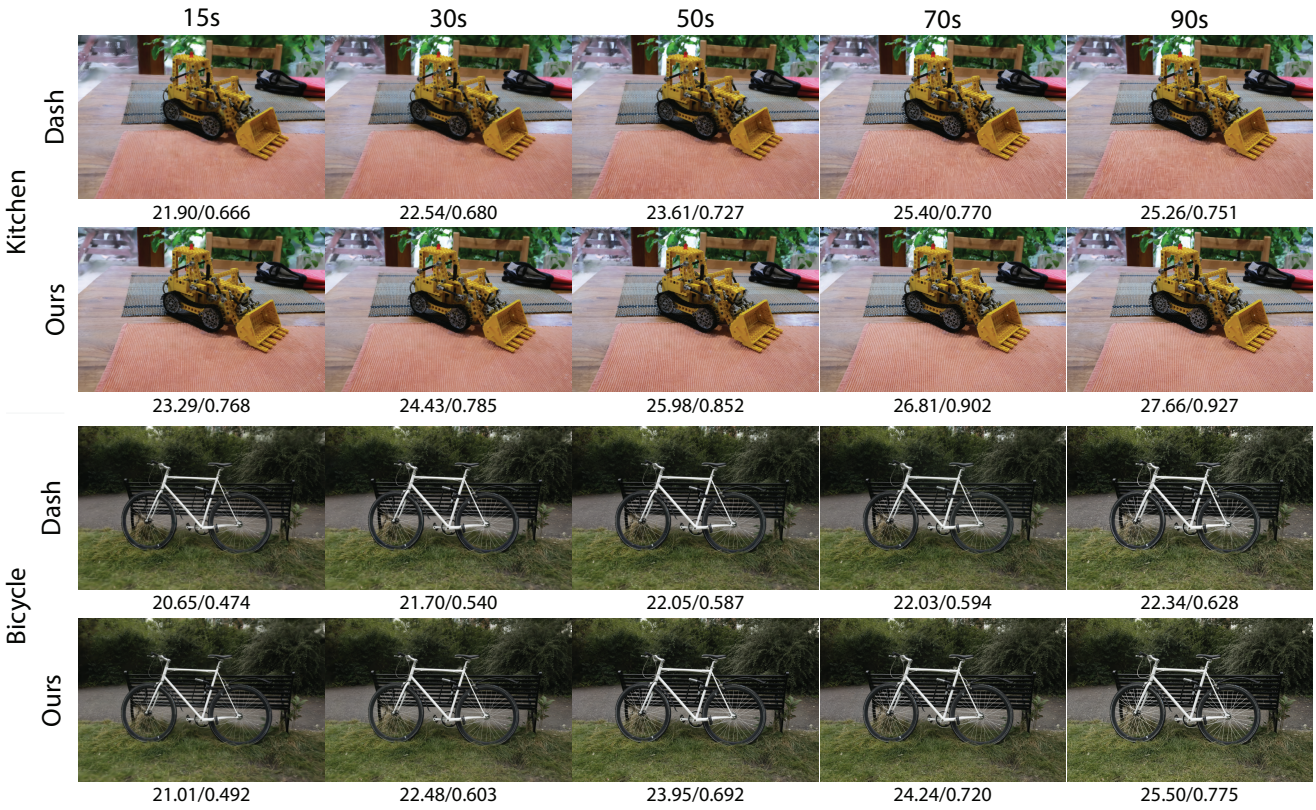


Figure 20. Qualitative comparison of rendering quality across two scenes from Mip-NeRF 360, comparing our method with DashGaussian. Each row shows results at training times of 15s, 30s, 50s, 70s, and 90s. PSNR and SSIM metrics are displayed in each image.

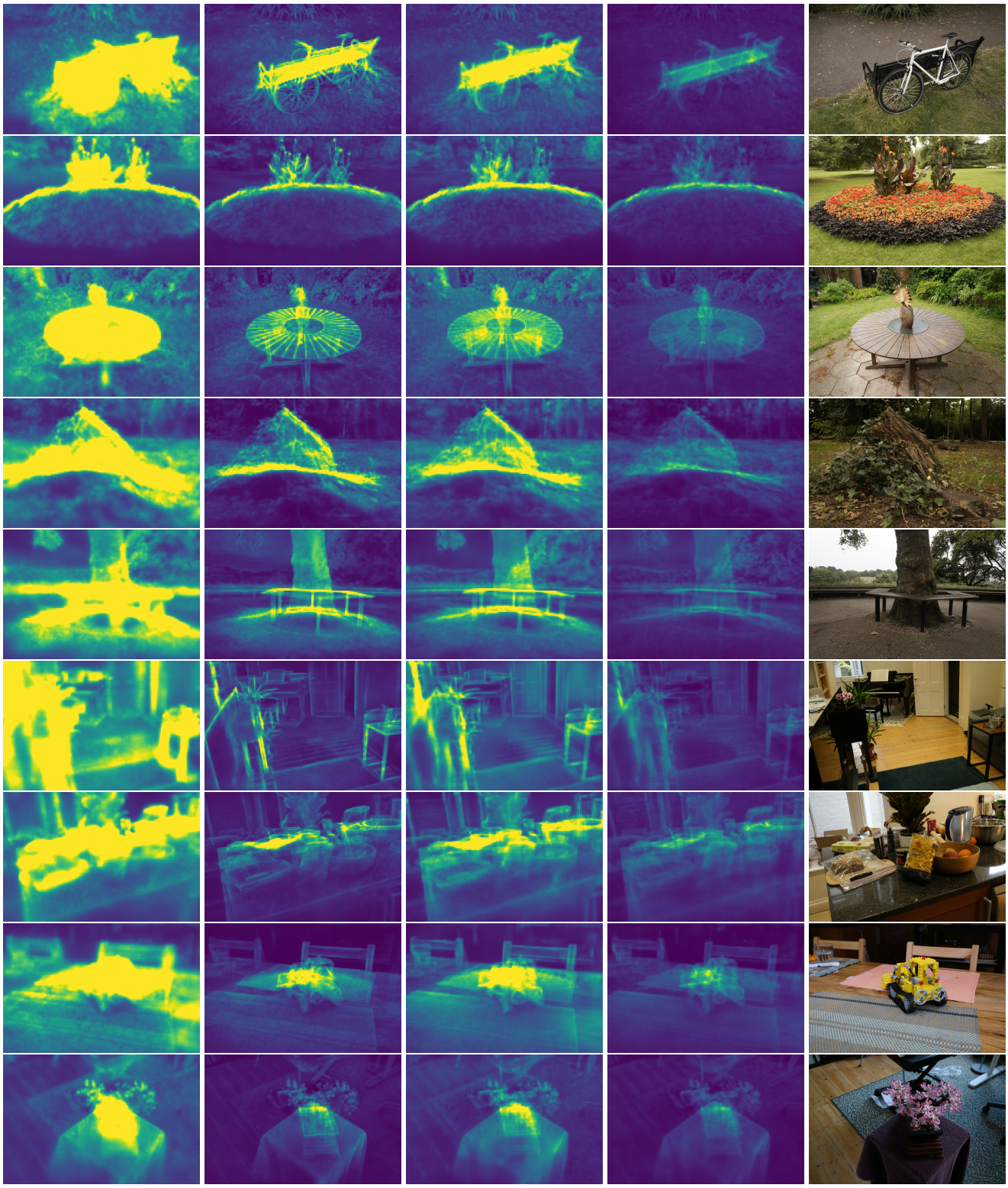


Figure 21. Gaussian count heatmap comparison across scenes in Mip-NeRF 360 dataset, corresponding to Fig. 1. Each row shows ground truth and heatmaps for 3DGS, Taming-3DGS, LiteGS, and our method. Colors represent per-tile Gaussian counts, where purple indicates low counts and yellow indicates high counts. Our method consistently achieves the lowest Gaussian counts (darker colors).

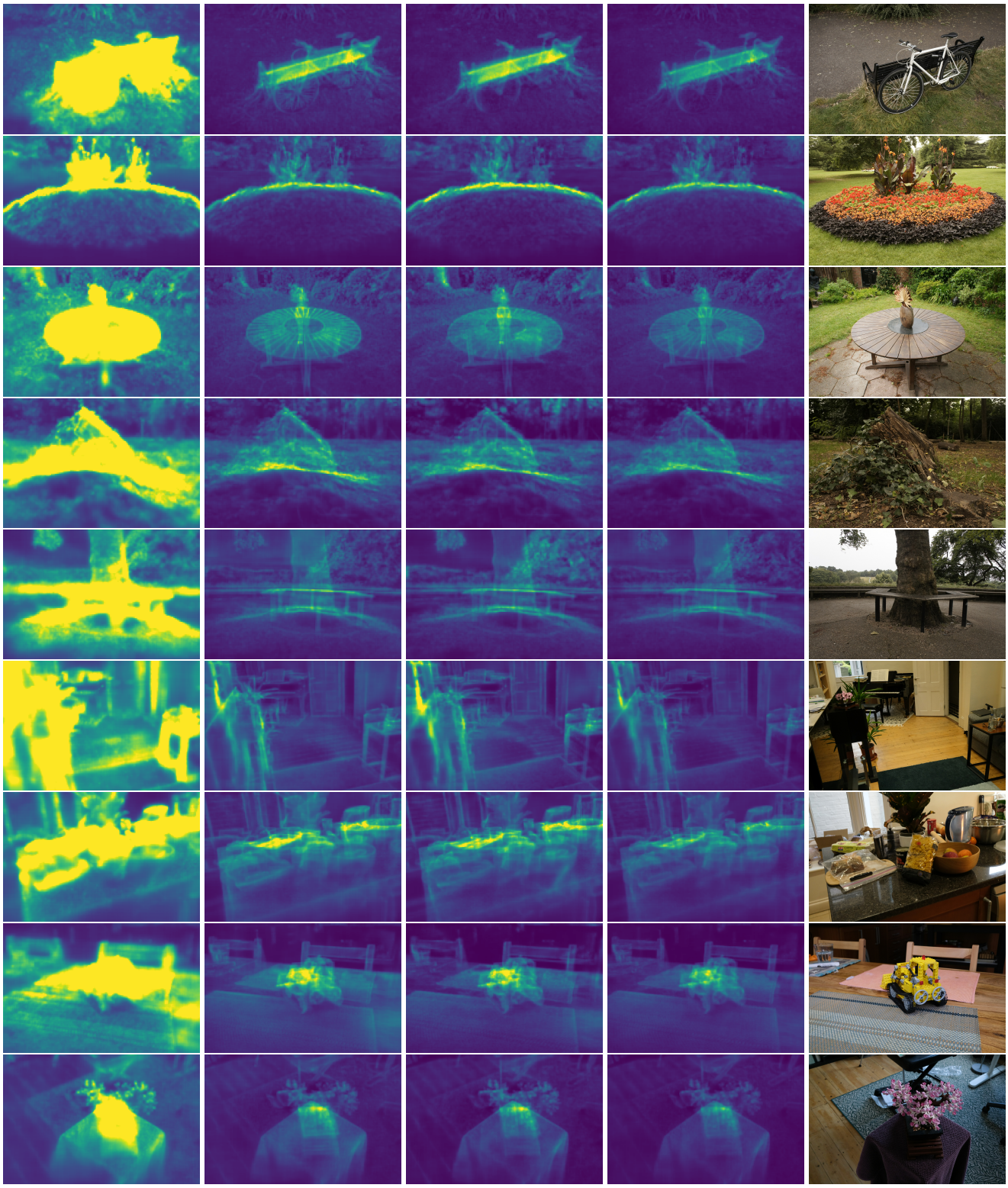


Figure 22. Gaussian count heatmap comparison across scenes in Mip-NeRF 360 dataset for ablation study, corresponding to Fig. 12. Each row shows ground truth and heatmaps for 3DGS, 3DGS+Scale Reset, 3DGS+Entropy Constraint, and 3DGS+Reset+Entropy. Colors represent per-tile Gaussian counts, where purple indicates low counts and yellow indicates high counts.

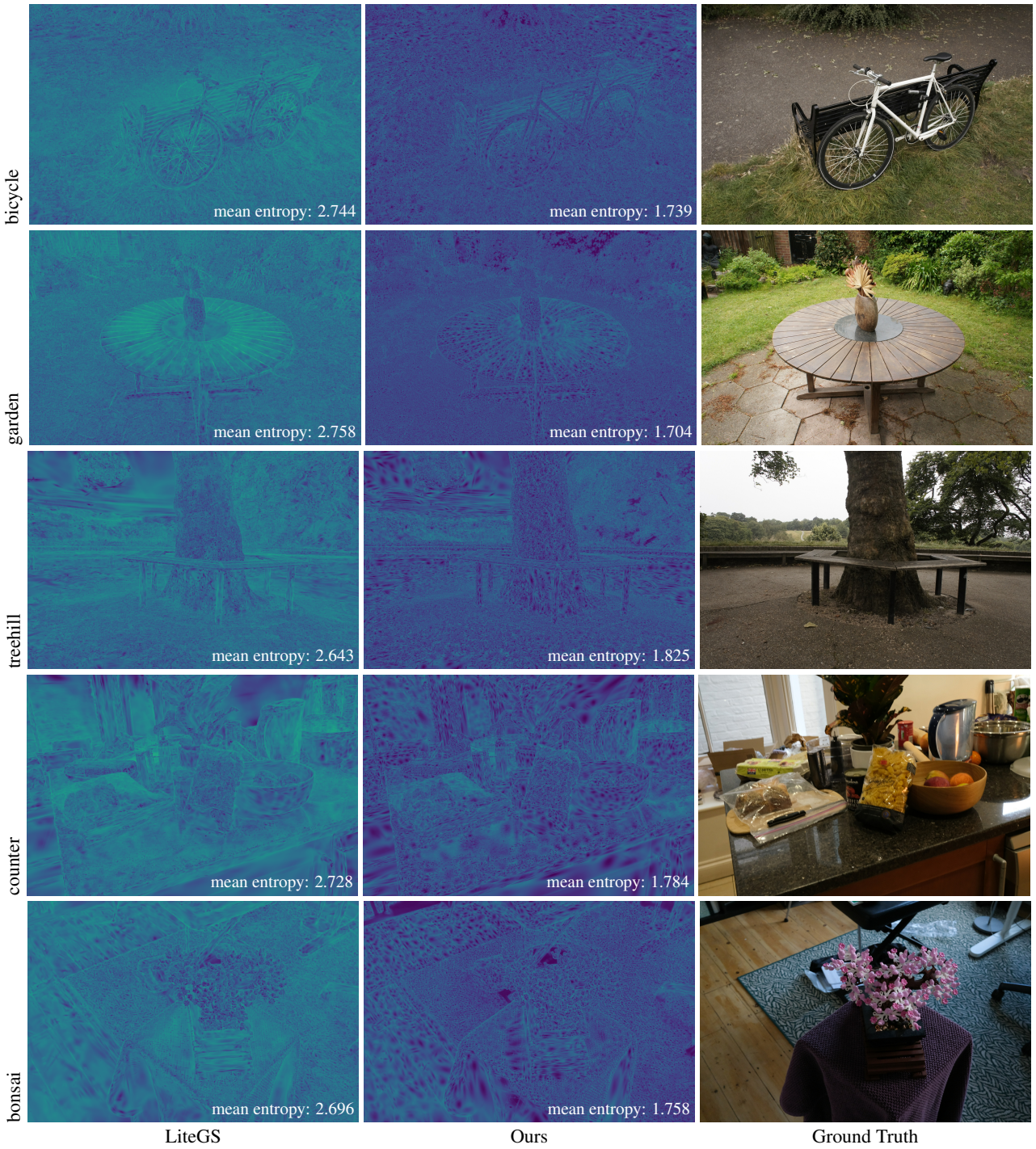


Figure 23. Per-pixel entropy heatmap comparison between LiteGS and our method across scenes in the Mip-NeRF 360 dataset. Darker colors correspond to lower entropy.

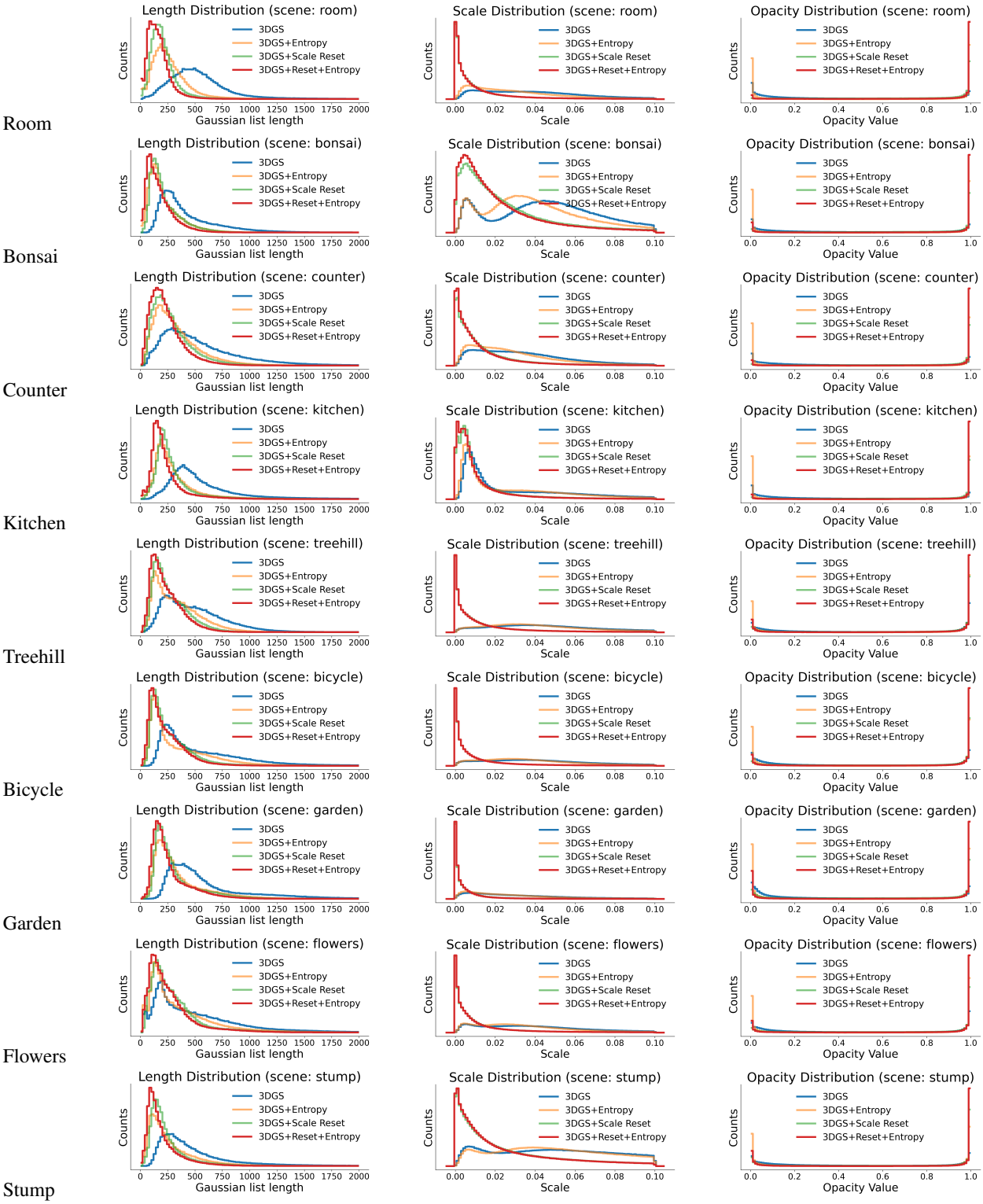


Figure 24. Distribution comparison per scene across all methods on length, scale, and opacity. Results labeled as "3DGS" are obtained using LiteGS. See Sec. 5.4 for more details.

Figure 25. Visualization of Gaussians on selected scenes from the Mip-NeRF 360 dataset. Compared to 3DGS, our method produces smaller and more compact Gaussians.