Supplementary Material: Volumetrically Consistent 3D Gaussian Rasterization

The supplementary material is organized as follows. In section Sec. A, we show a qualitative comparison between our volumetric rasterizer and ray marching to visualize the impact of the sorting and non-overlapping assumptions we make in our method (see Sec. 4.1 in the main paper). We derive an expression for the 1D Gaussian distribution along the camera ray in Sec. B (eq.15 in the main paper). In Sec. C we validate that our method outperforms 3DGS at approximating sharp edges and constant texture regions in a multiview setting on the nerf-synthetic dataset (Sec. 4 in the main text). We describe the hyperparameters we use and more details about our modifications to 3DGS's densification strategy in Sec. D (Sec. 5 of the main paper), and report the reconstruction quality metrics for all the scenes individually in Sec. E. We report our average training times in Sec. F (Sec 6.1 of main paper). In Sec. G we compare the reconstruction quality (SSIM and LPIPS) of 3DGS and our method across different memory budgets, referenced in Sec. 6.1 of the main paper. We present the results of an ablation study on our method in Sec. H. We analyze the failure cases from our method compared to 3DGS on the MipNeRF-360 dataset (see Sec. 7 in the main text) and present a quantitative analysis in Sec. I.

A. Comparing volumetric rasterizer with ray marching

Our method enables volumetrically consistent rasterization by analytically computing the volume rendering integral for each primitive. Our approach is accurate under the assumption of correctly sorted and non-overlapping primitives (Sec. 3.2 main paper). In this section, we qualitatively show the effect of this approximation, i.e. to what extent and in which situations it deviates from a reference solution to the volume rendering equation *without approximations*. Consequences of overlapping primitives have no relation to view dependence of color, so we assume view-independent colors in this experiment to simplify the ray-marching process.

We discretize the volume rendering equation using the quadrature rule, similar to eq. 3 in Mildenhall et al. [6] and evaluate it via ray-marching. Given a radiance field parameterized by 3D Gaussians G_i with view independent colors c_i, the density $\sigma(\boldsymbol{x})$ and color $c(\boldsymbol{x})$ at the 3D point $\boldsymbol{x}$ are given as:

$$\sigma(\boldsymbol{x}) = \sum_{i=1}^{N} G_i(\boldsymbol{x}), \quad c(\boldsymbol{x}) = \frac{1}{\sigma(\boldsymbol{x})} \sum_{i=1}^{N} c_i G_i(\boldsymbol{x}). \tag{1}$$

The color and density above are used to evaluate the volume rendering integral at a pixel by ray-marching along the corresponding ray. We compare our method to the ray-marching reference for overlapping and non-overlapping primitives in Fig. 1. Our method assumes non-overlapping primitives while computing the volume rendering integral (eq.10, Sec. 4.1 in the main paper), closely matching the ray-marching reference in fidelity in Fig. 1 (row 1 and row 2). Since our alpha computation assumes non-overlapping primitives, we observe a mismatch between our method and the ray-marching reference when primitives overlap (row 3, Fig. 1). But in practice, the primitives are quite small, and instances of very high overlap are relatively few. In row 4 of Fig. 1, we render 100 Gaussian primitives with random means and covariances and observe that our method matches the ray-marching reference closely in all regions apart from those with overlapping primitives, which have a minor impact on the final rendered image.

B. 1D Gaussian along camera ray

We outline the proof for eq.15 in the main paper, which describes the 3D Gaussian density of a primitive as a 1D Gaussian along a camera ray. Recall that in the main paper (eq.4, Sec. 3.2), we defined the 3D Gaussians as

$$G_j(\boldsymbol{x}) = \exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu_j})^T {\boldsymbol{\Sigma_j}}^{-1}(\boldsymbol{x}-\boldsymbol{\mu_j})\right\} \tag{2}$$

The 1D distribution $g_j(t)$ along the ray $\boldsymbol{x} = \boldsymbol{o} + t\boldsymbol{d}$ is given by (eq. 15 main paper):

$$g_j(t) = G_j(\gamma_j \boldsymbol{d}) \exp\left\{\frac{-(t-\gamma_j)^2}{2\beta_j^2}\right\}, \tag{3}$$

where $G_j(\gamma_j \boldsymbol{d})$ is the maximum value of the Gaussian along the ray, and the 1D Gaussian has mean γ_j and variance β_j, which are defined as

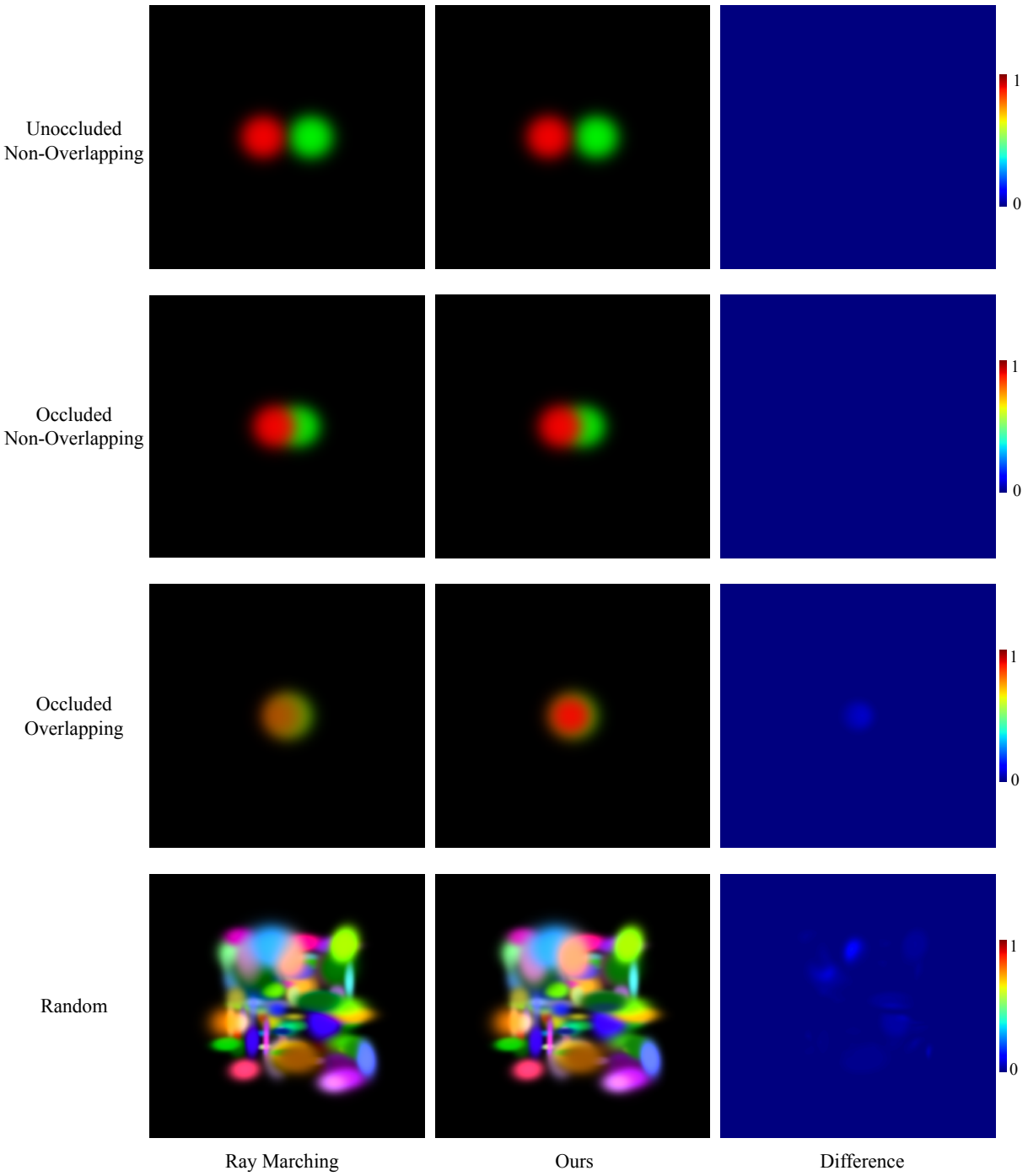

Figure 1. **Comparison with ray marching** We compare our method with ray marching as a reference for some representative configurations of 3D Gaussians. Since our alpha computation is exact for un-occluded primitives, we match the ray marching result with very high fidelity for non-overlapping Gaussians, in both unoccluded (*Row 1*) and occluded (*Row 2*) cases. Note that in row 2, the primitives occlude each other but are placed far apart along the camera axis ensuring no overlap between them. In (*Row 3*), we place the two primitives very close to each other with a high degree of overlap, resulting in a mismatch between ray marching and our method. To evaluate the effect of the non-overlapping assumption on a more complex setting, we construct 100 Gaussians with uniformly sampled positions and scales in (*Row 4*). Since the primitives are typically smaller than the scene's spatial extent, any overlapping mismatches remain localized. Our method matches the reference accurately in most regions of the image, with minimal differences in areas with overlapping primitives.

$$\gamma_j = \frac{(\boldsymbol{\mu_j} - \boldsymbol{o})^T \Sigma_j^{-1} \boldsymbol{d}}{\boldsymbol{d}^T \Sigma_j^{-1} \boldsymbol{d}}, \quad \beta_j = \frac{1}{\sqrt{\boldsymbol{d}^T \Sigma_j^{-1} \boldsymbol{d}}}. \tag{4}$$

To derive this result, we substitute $\boldsymbol{x} = \boldsymbol{o} + t\boldsymbol{d}$ in Eq. (3):

$$g_j(t) = G_j(\boldsymbol{o} + t\boldsymbol{d}) = \exp\left\{-\frac{1}{2}\Delta\right\}, \tag{5}$$

where the argument of the exponent,

$$\Delta = \Big(t\boldsymbol{d} - (\boldsymbol{\mu_j} - \boldsymbol{o})\Big)^T \boldsymbol{\Sigma_j}^{-1} \Big(t\boldsymbol{d} - (\boldsymbol{\mu_j} - \boldsymbol{o})\Big). \tag{6}$$

The expression for Δ can be further expanded as

$$\Delta = t^2 \boldsymbol{d}^T \boldsymbol{\Sigma_j}^{-1} \boldsymbol{d} - 2t(\boldsymbol{\mu_j} - \boldsymbol{o})^T \boldsymbol{\Sigma_j}^{-1} \boldsymbol{d} \tag{7}$$

$$+ (\boldsymbol{\mu_j} - \boldsymbol{o})^T \boldsymbol{\Sigma_j}^{-1} (\boldsymbol{\mu_j} - \boldsymbol{o}) \tag{8}$$

$$\text{Setting } K = \frac{(\boldsymbol{\mu_j} - \boldsymbol{o})^T \boldsymbol{\Sigma_j}^{-1} (\boldsymbol{\mu_j} - \boldsymbol{o})}{\boldsymbol{d}^T \boldsymbol{\Sigma_j}^{-1} \boldsymbol{d}} - \gamma_j^2, \tag{9}$$

simplifies the expression for Δ as follows -

$$\Delta = \frac{1}{\beta_j^2}\Big(t^2 - 2t\gamma_j + \gamma_j^2 + K\Big) \tag{10}$$

$$\implies \Delta = \frac{1}{\beta_j^2}(t - \gamma_j)^2 + K \tag{11}$$

$$\implies g_j(t) = \exp\left\{\frac{-(t-\gamma_j)^2}{2\beta_j^2}\right\} \exp\left\{-\frac{K}{2}\right\}. \tag{12}$$

$$\tag{13}$$

In the above equations, K is independent of t. Let $t = t_{\max}$ maximize $g_j(t)$. Above, we showed that

$$g_j(t) = G_j(\boldsymbol{o} + t\boldsymbol{d}) = \exp\left\{\frac{-(t-\gamma_j)^2}{2\beta_j^2}\right\} \exp\left\{-\frac{K}{2}\right\}.$$

Since $\exp\left\{\frac{-(t-\gamma_j)^2}{2\beta_j^2}\right\}$ attains its maximum value of 1 when $t = \gamma_j$, $t_{\max} = \gamma_j$. This can also be verified by setting $\frac{d\Delta}{dt} = 0$ and solving for t. This gives us

$$g_j(t_{\max}) = G_j(\boldsymbol{o} + t_{\max}\boldsymbol{d}) = \exp\left\{-\frac{K}{2}\right\}. \tag{14}$$

We denote $G_j(\boldsymbol{o} + t_{\max}\boldsymbol{d})$ as $G_j(\gamma_j \boldsymbol{d})$ in the main paper (eq. 15). This concludes the proof of Eq. (3) (eq. 15 in main paper).

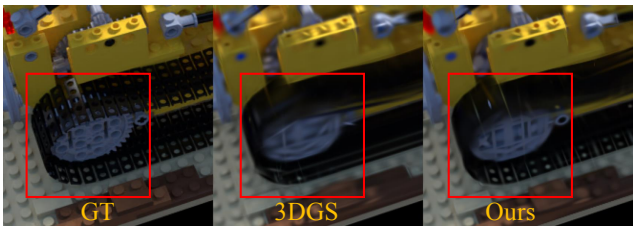

Figure 2. **NerfSynthetic qualitative result.** Our approach renders sharper details for *piecewise opaque regions* (inset) compared to 3DGS (zoom in).

C. Comparing our method and 3DGS on nerf-synthetic

Recall that in Fig.2 and Fig.3 in the main text, we fit a single image with a fixed number of primitives and observe that our method is better than 3DGS at approximating sharp edges and constant regions. Here, we validate this observation in a multi-view setting on the nerf-synthetic (NS) dataset. We optimize our method and 3DGS with the *same initialization, and no densification* to isolate the expressiveness of our method from the optimization dynamics. For all metrics (PSNR, SSIM, LPIPS) we (29.81, 0.941, 0.074) outperform 3DGS (29.56, 0.936, 0.082) averaged over all 8 scenes in NS. Our method renders piece-wise constant textures (common in nerf-synthetic) better than 3DGS. See Fig. 2 for a qualitative comparison.

D. Implementation details and hyperparameters

Implementation details: Unlike 3DGS which uses 2D screen space positional gradients, we use the world space 3D positional gradients to compute the gradient norm for densification thresholding, similar to [5, 7]. We use separate positional gradient thresholds for the splitting (10^{-8}) and cloning (5×10^{-5}) operations. We require much lower gradient threshold values since our method produces more opaque primitives than 3DGS which reduces the positional gradient value. We densify and prune every 200 iterations. For pruning, we prune based on the value of θ, the parameter used to parameterize the density of the primitives κ (see eq. 20 in the main paper). We remove primitives for which θ is less than a minimum opacity threshold. Similar to [5], we apply softplus activation ($\beta = 10$) to the spherical harmonic coefficients.

Hyperparameters: The learning rates for density and color features are set to 0.03 and 0.003 respectively. Initial and final position learning rates are set to 0.00016 and 0.00001. In the 3DGS codebase, the scene extent (or camera extent) is defined as the maximum distance between any

Scene	PSNR	SSIM	LPIPS	#Points
Bicycle	25.04	0.7594	0.2155	3.7M
Bonsai	32.17	0.9498	0.1704	1.5M
Counter	28.92	0.9206	0.1699	1.6M
Garden	27.29	0.858	0.122	3.18M
Flowers	21.41	0.615	0.306	3.8M
Stump	26.70	0.778	0.219	5.3M
Treehill	21.94	0.627	0.329	3.6M
Room	32.00	0.9358	0.1845	1.6M
Kitchen	31.51	0.935	0.113	2.1M

Table 1. Metrics on MipNeRF360 scenes for our method.

Scene	PSNR	SSIM	LPIPS	#Points
Train	22.16	0.825	0.195	1.91M
Truck	25.31	0.8838	0.139	1.54M

Table 2. Metrics on Tanks&Temples scenes for our method.

Scene	PSNR	SSIM	LPIPS	#Points
Playroom	30.23	0.909	0.250	2.2M
DrJohnson	29.22	0.906	0.244	4.7M

Table 3. Metrics on DeepBlending scenes for our method.

training camera and the average camera centroid. The scene extent controls the splitting, cloning, and pruning thresholds in the densification process. In our method, we scale the scene extent by a factor of $2, 5$, and 1.5 for the Mip-NeRF360, Tanks&Temples, and DeepBlending datasets respectively. We also adjust the (hardcoded) pruning hyperparameter to prune points with large scales to 0.01 from the default value 0.1.

E. Per scene metrics

We report metrics from our method for each scene individually in Mip-NeRF360, Tanks&Temples, and DeepBlending datasets in Tab. 1, Tab. 2 and Tab. 3. Note that in the main paper (Tab. 1), we report metrics averaged over all scenes for the three datasets -Mip-NeRF360, Tanks&Temples, and DeepBlending. For per-scene metrics on some of the other baselines, please see [3, 4].

F. Average training times

We use a Nvidia 3090 Ti for training both our method and 3DGS. Our training times are similar to 3DGS, with a slight slowdown that arises from the extra computations needed for our alpha computation compared to 3DGS. 3DGS is able to re-use the splatted 2D Covariance matrix for computing alpha for each pixel. Our method requires computing $t_{\max}$ and other intermediaries for each pixel separately which leads to a slight slowdown. We compute inverses of both the 2D and 3D covariances, as opposed to 3DGS which does the inverse computation only once for the 2D covariance in the vertex shader phase. The vertex shader in our slang.D implementation uses atomic add operations to write the computed inverse covariance matrices to global memory. Our method requires more atomic adds per primitive to store 3D covariance matrices, leading to slower performance compared to 3DGS, which requires fewer atomic adds to store 2D covariance matrices.

For MipNeRF-360 scenes, on average 3DGS and our method generate 2.62M and 2.93M points respectively, and the training times for 30000 iterations are 51.5 min and 61 min respectively. Similarly for Tanks&Temples, on average 3DGS and our method generate 1.79M and 1.72M points respectively, and the training times for 30000 iterations are 20.5 min and 33.2 min respectively.

ZipNerf training: We trained and evaluated ZipNeRF [2] on a Nvidia A-40 GPU, using their official code release [1]. In the official code release for ZipNeRF, the indoor and outdoor scenes in the Mip-NeRF360 dataset are evaluated at 2x and 4x downsampled resolutions respectively. We trained and evaluated ZipNeRF at full resolution on the Tanks&Temples and DeepBlending datasets, but for the MipNeRF-360 dataset, we used 2x downsampled images (for both indoor and outdoor scenes) to fit them in the GPU memory. ZipNeRF took approximately 12-13 hours per scene to train on our GPU. For the Tanks&Temples dataset, we re-ran COLMAP before training Zip-NeRF as highlighted in issue 7 in the official Zip-NeRF code release [1].

G. Reconstruction quality for different memory budgets

In Fig. 3 we compare the performance of our method against 3DGS across different memory budgets, i.e. across a different number of maximum primitives that each method is allowed to generate. We stop densification and pruning once the optimization reaches the maximum number of allowed primitives. For each memory budget, we average the results over 4 scenes, KITCHEN, STUMP, TRAIN, COUNTER. For the same memory budget, our method consistently outperforms 3DGS on both SSIM and LPIPS. This holds across a range of memory budgets as shown in Fig. 3. Recall that our method represents opaque textures better than 3DGS for the same number of primitives. We demonstrated this through the toy examples in Fig. 2 and Fig. 3 in the main paper. Since LPIPS and SSIM measure perceptual similarity and edge quality, this also translates to quantitative improvements as demonstrated in Fig. 3.

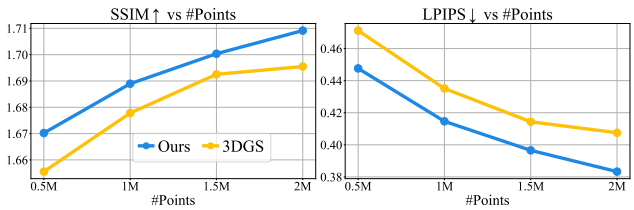

Figure 3. **Varying number of primitives**: Each data point is computed by averaging the test dataset metrics over the scenes KITCHEN, STUMP, TRAIN AND COUNTER

	PSNR ↑	SSIM ↑	LPIPS ↓	Points
3DGS	21.38	0.588	0.360	3.4M
3DGS + Our HyperParams	20.66	0.571	0.329	7.4M
Ours w/o Reparam	21.23	0.600	0.309	7M
Ours w EVER Reparam	20.04	0.519	0.396	9.4M
Ours	21.41	0.615	0.306	3.8M

Table 4. **Ablations**. We show ablations on the FLOWERS scene in MipNeRF-360 dataset.

H. Ablation study

We conduct an ablation study to assess the impact of hyperparameters and density parameterization on our method, with results shown in Tab. 4. We compare our method with 3DGS (row 1), and for fairness, we also run 3DGS with the hyperparameters used by our method. Using 3DGS with our hyper-parameters (row 2) improves LPIPS, but worsens PSNR and SSIM while generating almost 2x more primitives compared to the base 3DGS configuration (row 1). Our method (row 4) fares relatively better on all metrics.

We also run our method without density reparameterization (row 3); we ensure positive density by applying the softplus activation. We observe improvement in LPIPS and SSIM compared to 3DGS but at almost double the number of primitives. We also experiment with the parameterization proposed in Mai et al. [5] (row 4), which results in poorer SSIM and LPIPS while producing too many points compared to our method (row 5).

I. MipNeRF-360 dataset failure case analysis

We analyze failure cases for the 3 metrics. We define a failure case as being worse than 3DGS by a threshold, separately chosen for each metric (1dB PSNR, 0.01 SSIM, 0.005 LPIPS). Averaged on all test images in the dataset, our failure rate is 4.64%, 3.38%, and 2.11% for PSNR, SSIM and LPIPS respectively. TREEHILL shows the highest failure rate among all scenes, with LPIPS and SSIM differences of 0.017 and 0.027, respectively (example in Fig. 4). For PSNR, we observe an outlier failure case from KITCHEN, 5.6 dB worse than 3DGS (Fig. 4). However, averaged over all 34 KITCHEN test images, our method (31.51 dB) performs only slightly worse than 3DGS (31.67 dB) in PSNR.

Figure 4. **MipNerf-360 dataset worst failure cases.** Top row (KITCHEN): massive floater (inset) causes PSNR drop. Bottom row (TREEHILL): failure due to insufficient densification and possibly inaccurate camera poses. 3DGS prefers a blurry solution, ours has sharp opaque artifacts.

See Tab. 5 for per-scene quantitative analysis on all three metrics for Mip-NeRF360 dataset.

Dataset	Worst Gap LPIPS	Better / Comparable %	Failure %	Total Images
flowers	-0.022	100.0	0.000	21
room	0.005	97.4	2.632	38
bonsai	0.004	100.0	0.000	36
bicycle	-0.008	100.0	0.000	24
treehill	0.017	88.2	11.765	17
counter	-0.006	100.0	0.000	29
stump	0.004	100.0	0.000	15
garden	0.007	91.3	8.696	23
kitchen	0.004	100.0	0.000	34
Dataset	Worst Gap SSIM	Better / Comparable %	Failure %	Total Images
flowers	0.005	100.0	0.000	21
room	-0.004	100.0	0.000	38
bonsai	-0.006	100.0	0.000	36
bicycle	-0.002	100.0	0.000	24
treehill	-0.035	58.8	41.176	17
counter	-0.001	100.0	0.000	29
stump	-0.013	93.3	6.667	15
garden	-0.005	100.0	0.000	23
kitchen	-0.007	100.0	0.000	34
Dataset	Worst Gap PSNR	Better / Comparable %	Failure %	Total Images
flowers	-1.204	95.2	4.762	21
room	-1.152	97.4	2.632	38
bonsai	-2.328	94.4	5.556	36
bicycle	-0.414	100.0	0.000	24
treehill	-1.174	88.2	11.765	17
counter	-2.163	93.1	6.897	29
stump	-0.636	100.0	0.000	15
garden	-0.536	100.0	0.000	23
kitchen	-5.764	91.2	8.824	34

Table 5. **Mip-NeRF-360 dataset quantitative analysis** Worst Gap is the largest negative difference between our result and 3DGS amongst test views for a scene, indicating the frame where our method performed the worst relative to 3DGS. For LPIPS (*top table*), a lower worst gap is better, whereas for SSIM (*middle table*) and PSNR (*bottom table*), a higher worst gap is preferable. As measured by LPIPS (*top table*) and SSIM (*middle table*), our approach is better or comparable (LPIPS within 0.005, SSIM within 0.01) to 3DGS for most scenes in Mip-NeRF360 except TREEHILL. As measured by PSNR (*bottom table*), we observe that our approach suffers the most compared to 3DGS for KITCHEN, COUNTER, and BONSAI. A few anomalies cause a high worst gap, though most views are better or comparable to 3DGS. For all other Mip-NeRF360 dataset scenes, we are comparable (PSNR within 1dB) to 3DGS.

References

[1] Jonathan T. Barron, Keunhong Park, Ben Mildenhall, John Flynn, Dor Verbin, Pratul Srinivasan, Peter Hedman, Philipp Henzler, and Ricardo Martin-Brualla. CamP Zip-NeRF: A Code Release for CamP and Zip-NeRF. 4

[2] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Zip-NeRF: Anti-Aliased Grid-Based Neural Radiance Fields. *ICCV*, 2023. 4

[3] Abdullah Hamdi, Luke Melas-Kyriazi, Jinjie Mai, Guocheng Qian, Ruoshi Liu, Carl Vondrick, Bernard Ghanem, and Andrea Vedaldi. GES: Generalized Exponential Splatting for Efficient Radiance Field Rendering. *CVPR*, 2024. 4

[4] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4):1–14, 2023. 4

[5] Alexander Mai, Peter Hedman, George Kopanas, Dor Verbin, David Futschik, Qiangeng Xu, Falko Kuester, Jon Barron, and Yinda Zhang. Ever: Exact volumetric ellipsoid rendering for real-time view synthesis. *arXiv preprint arXiv:2410.01804*, 2024. 3, 5

[6] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020. 1

[7] Nicolas Moenne-Loccoz, Ashkan Mirzaei, Or Perel, Riccardo de Lutio, Janick Martinez Esturo, Gavriel State, Sanja Fidler, Nicholas Sharp, and Zan Gojcic. 3d gaussian ray tracing: Fast tracing of particle scenes. *arXiv:2407.07090*, 2024. 3