SparseGS: Sparse View Synthesis using 3D Gaussian Splatting

Supplementary Material

Supplementary Contents

This supplementary is organized as follows:
- Sec. A presents the gradient derivation for our softmax-scaling depth formulation.
- Sec. B details the view sampling methods used in our UVR module.
- Sec. C provides algorithm details for our Advance Floater Pruning method.

A. Derivation of the Softmax Depth Gradient

In this section, we derive the gradients used for the softmax depth implemented in the rasterizer. We assume we are rendering a single pixel and that the Gaussians G_i along the camera ray are ordered from closest to the camera plane to the farthest. (i.e. α_1 corresponds to the Gaussian closest to the camera, while α_N corresponds to the farthest)
If we let $w_k = T_k\alpha_k$ and $T_k = \prod_{i=1}^{k-1}(1-\alpha_i)$[2], then

$$d_{x,y}^{\text{softmax}} = \log\left(\frac{\sum_{i=1}^{N} w_i e^{\beta w_i} d_i}{\sum_{i=1}^{N} w_i e^{\beta w_i}}\right) \tag{1}$$

The derivative w.r.t the Gaussian's camera depth value is:

$$\frac{\partial d_{x,y}^{\text{softmax}}}{\partial d_k} = \frac{w_k e^{\beta w_k}}{\sum_{i=1}^{N} w_i e^{\beta w_i} d_i} \tag{2}$$

The derivative w.r.t the Gaussian's alpha value is:

$$\frac{\partial d_{x,y}^{\text{softmax}}}{\partial \alpha_k} =$$

$$\frac{(1+\beta w_k)T_k e^{\beta w_k} d_k - \frac{1}{1-\alpha_k}\sum_{j=k+1}^{N}(1+\beta w_j)w_j e^{\beta w_j} d_j}{\sum_{i=1}^{N} w_i e^{\beta w_i} d_i} \tag{3}$$

$$- \frac{(1+\beta w_k)T_k e^{\beta w_k} - \frac{1}{1-\alpha_k}\sum_{j=k+1}^{N}(1+\beta w_j)w_j e^{\beta w_j}}{\sum_{i=1}^{N} w_i e^{\beta w_i}} \tag{4}$$

The sums in the denominators are retained during the forward pass and subsequently transferred to the backward pass. To avoid an $O(n^2)$ blowup, an accumulator strategy is used as in [2].

Let $\mathcal{A}_k^l, \mathcal{A}_k^r$ be the accumulators pertaining to $\frac{\partial d_{x,y}^{\text{softmax}}}{\partial \alpha_k}$ for expressions (3) and (4) respectively.

$$\begin{aligned}
\mathcal{A}_0^l &= \mathcal{A}_0^r = 0 \\
\mathcal{A}_k^l &= \alpha_{k-1}(1+\beta w_{k-1})e^{\beta w_{k-1}} d_{k-1} + (1-\alpha_{k-1})\mathcal{A}_{k-1}^l \\
\mathcal{A}_k^r &= \alpha_{k-1}(1+\beta w_{k-1})e^{\beta w_{k-1}} + (1-\alpha_{k-1})\mathcal{A}_{k-1}^r
\end{aligned}$$

It can be shown that

$$\frac{\partial d_{x,y}^{\text{softmax}}}{\partial \alpha_k} = T_k\left(\frac{(1+\beta w_k)e^{\beta w_k} d_k - \mathcal{A}_k^l}{\sum_{i=1}^{N} w_i e^{\beta w_i} d_i} - \frac{(1+\beta w_k)e^{\beta w_k} - \mathcal{A}_k^r}{\sum_{i=1}^{N} w_i e^{\beta w_i}}\right)$$

B. View Sampling for UVR

As described in the main paper, although the UVR as a whole is designed to regularize 3DGS training from sampled views, each component has a different strategy of camera sampling.

Elliptic Cylinder Sampled Views. To compute the SDS loss, we generate random camera views directed at the scene center using an elliptical cylinder that best fits the positions of the training views. The process begins by transforming the camera positions to align with the world coordinate system (where the positive Z-axis is perpendicular to the ground) using PCA. We then determine the scaling factors for the elliptical axes by taking the 90th percentile of the absolute displacements from the scene center, ensuring the ellipse encompasses most of the input views.

The vertical bounds for the z-coordinate are set at the 10th and 90th percentiles of the input views' z-coordinates. We uniformly randomly select a camera translation vector with x-y coordinates on the ellipse and a z-coordinate within the aforementioned z-bounds. Finally, the camera's rotation matrix is computed by directing its look-at vector towards the scene center and aligning the up vector with the average up vector of the input views.

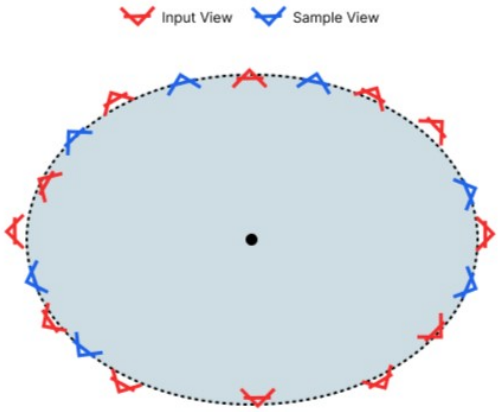

Figure A. **Bird's Eye View of Elliptic Cylinder sampling.** The dashed line is the ellipse best fit to the input views (in red). The blue views are the sampled views on the ellipse which also have a random variation in the z-coordinate.

Warped Views. We cannot use the same sampled viewpoints as in SDS because they are too far to provide large view overlapping. Instead, for each training camera, we apply a warp by a predefined angle, θ, around the average up-vector of all the training cameras. Specifically, the up-vector of each camera is extracted as:

$$Y = \begin{bmatrix} R_{11} \\ R_{21} \\ R_{31} \end{bmatrix} \tag{3}$$

with R denoting the rotation matrix in the camera pose. Next, we compute the normalized average up-vector across all cameras to serve as an estimate of the scene center and orientation. Novel viewpoints are then created by rotating the training cameras around this estimated center axis by various angles using the Rodrigues' rotation formula [3].

$$\bar{Y} = \frac{1}{N} \sum_i Y_i \tag{4}$$

$$\bar{Y} = \frac{\bar{Y}}{||\bar{Y}||} \tag{5}$$

$$P' = P \cos\theta + (P \times \bar{Y}) \sin\theta + \bar{Y}(\bar{Y} \cdot P)(1 - \cos\theta) \tag{6}$$

where P represents the original camera pose, and $\bar{Y}$ is the estimated up-axis at the scene center. We select the angles θ to be -3°,-1.5°, 1.5°, and 3°, resulting in four warped viewpoints for each training camera.

Apply at	Pearson Depth Loss	SDS Loss	Depth Warping Loss	Floater Pruning
Input Views (12/24)	✓			✓
Elliptic Cylinder Sampled Views		✓		
Warped Views			✓	

Table A. The Pearson Depth loss and floater pruning are applied to sparse input views, whereas the SDS and Depth Warping losses are applied to differently sampled views.

C. Advance Floater Pruning Algorithm

In 3DGS, "floaters" refer to relatively low-opacity Gaussians positioned close to the camera center. These floaters do not appear prominently when rendering the alpha-blending depth of a scene because they are "averaged out", but they stand out in the mode-selection depth. Since the Pearson depth loss is a relatively a soft constraint, misalignments between the mode and alpha depth can still occur. This misalignment is largely due to 3DGS halting rasterization once a transmittance threshold is reached. The floater Gaussians near the camera deplete this transmittance budget early, preventing other Gaussians from converging at the correct depth.

To address this, we threshold the areas of misalignment to generate a floater mask F for each input sparse view to identify and delete Gaussians that are incorrectly placed too close to the camera. Specifically, we first compute the relative difference between the mode-selection depth and alpha-blending depth for each training view i as $\Delta_i = \frac{(d^{mode} - d^{alpha})}{d^{alpha}}$. As discussed in the main paper, when visualizing the distributions of Δ_i, images with many floaters have bimodal histograms, as floaters tend to deviate significantly from the true depth. Based on this observation, we apply dip test [1], a measure of uni-modality, on the distribution. We then average the uni-modality score across all training views for a scene, since the presence of floaters is a scene-wide metric, and use the mean score to select a percentile threshold for the relative differences.

The conversion from the dip statistic to a threshold is done using an exponential curve with parameters a and b. We estimate these parameters by manually examining Δ_i and F_i for various scenes from different datasets and real world capture, and our values are $a = 0.97, b = -7.5$. This process was carefully designed to allow our floater pruning method to be adaptive. The full pruning algorithm is as follow:

ALGORITHM 1: Algorithm to prune floaters

Function *prune_floaters*(G, I, a, b)
- **Input :** 3DGS Representation G
- **Input :** Training Cameras I
- **Input :** Curve Parameters a, b
- $\bar{D} \leftarrow 0.0$
- **foreach** $i \in I$ **do**
  - $d^{\text{alpha}} \leftarrow$ alpha_blending_depth(G, i)
  - $d^{\text{mode}} \leftarrow$ mode_selection(G, i)
  - $\Delta_i \leftarrow \frac{d^{\text{mode}} - d^{\text{alpha}}}{d^{\text{alpha}}}$
  - $\bar{D} \leftarrow \bar{D} +$ dip_test(Δ_i)
- **end**
- $\bar{D} \leftarrow \bar{D}/|I|$
- **foreach** $i \in I$ **do**
  - $\tau_i \leftarrow$ percentile$(\Delta_i, ae^{b\bar{D}})$
  - $F_i \leftarrow 1[\Delta_i > \tau]$
  - $g_0, g_1, ..., g_n \leftarrow$ mask_to_Gaussian(F_i)
  - remove_Gaussians$(g_0, g_1, ..., g_n)$
- **end**

end

D. More Qualitative Comparisons with Few-shot Methods [4, 5]

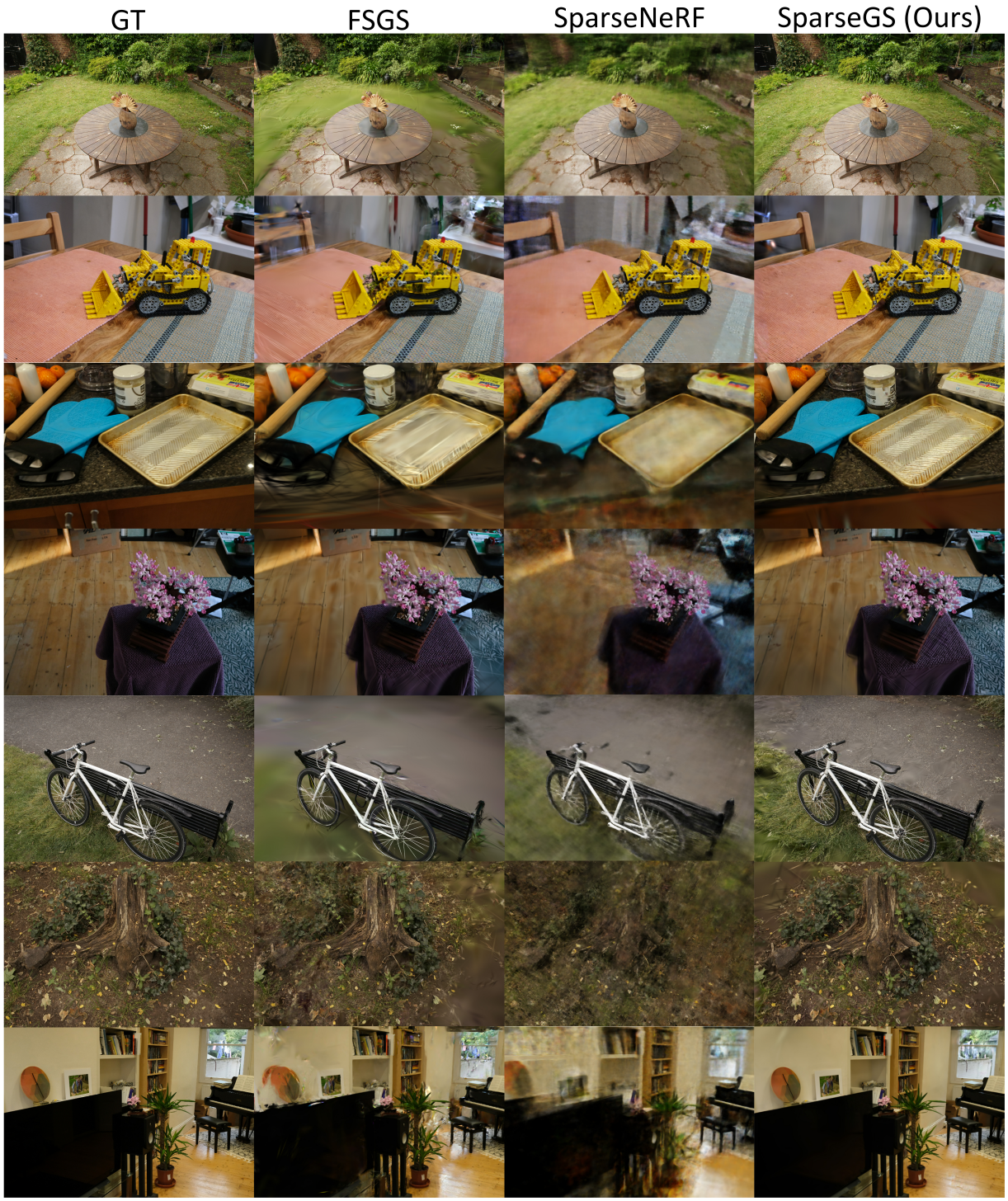

Figure B. **Qualitative comparisons with FSGS and SparseNeRF on the MipNeRF360 dataset.** All experiments above are trained with 12 input views. SparseNeRF often over-smooths regions that lack coverage from the training views. In contrast, our method preserves sharp details and more accurately fills in these missing regions. FSGS excels in maintaining foreground structure and details but can lack background coherence and produce floaters.

References

[1] J. A. Hartigan and P. M. Hartigan. The Dip Test of Unimodality. *The Annals of Statistics*, 13(1):70 – 84, 1985. 3

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

[3] O. Rodrigues. Lois du mouvement des systèmes de points matériels. *Journal de l'Ecole Royale Polytechnique*, 19:319–429, 1840. 2

[4] Guangcong Wang, Zhaoxi Chen, Chen Change Loy, and Ziwei Liu. Sparsenerf: Distilling depth ranking for few-shot novel view synthesis. *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023. 4

[5] Zehao Zhu, Zhiwen Fan, Yifan Jiang, and Zhangyang Wang. Fsgs: Real-time few-shot view synthesis using gaussian splatting, 2023. 4