Scaffold-GS: Structured 3D Gaussians for View-Adaptive Rendering

Supplementary Material

6. Overview

This supplementary is organized as follows: (1) In the first section, we elaborate implementation details of our *Scaffold*-GS, including anchor point feature enhancement (Sec.3.2.1), structure of MLPs (Sec.3.2.2) and anchor point refinement strategies (Sec.3.3); (2) The second part describes our dataset preparation steps. We then show additional experimental results and analysis based on our training observations.

7. Implementation details.

Anchor Growing Our anchor growing process is summarized in Algorithm 1:

Algorithm 1 Anchor Growing

$V \leftarrow$ Anchors ▷ Positions, features, neural gaussians
$C \leftarrow []$ ▷ New anchors candidate
$\tau_g, \epsilon_g \leftarrow$ Gradient threshold, Voxel size
$m \leftarrow 1$ ▷ Iteration Count
$M \leftarrow$ Maximum iterations
while m<=M **do**
  $\tau \leftarrow \tau_g \cdot 2^{m-1}$
  $\epsilon \leftarrow 16\epsilon_g \div 4^{m-1}$
  for all neural gaussians g generated from V **do**
    if g's gradient $> \tau$ **then**
      C.append($\lfloor \frac{\mathbf{g.xyz}}{\epsilon} \rceil$, g's anchor feature f])
    end if
  end for
  $C \leftarrow$MergeSamePosition(C) ▷ Feature merging using scatter max pooling
  $C \leftarrow$RandomElimination(C)
  $C \leftarrow$RemoveOccupiedLocations(V, C)
  $V \leftarrow$Concatenate(V, C)
  $C = []$
  m=m+1
end while

Feature Bank. To enhance the view-adaptability, we update the anchor feature through a view-dependent encoding. Following calculating the relative distance δ_{vc} and viewing direction $\vec{\mathbf{d}}_{vc}$ of a camera and an anchor, we predict a weight vector $w \in \mathbb{R}^3$ as follows:

$$(w, w_1, w_2) = \text{Softmax}(F_w(\delta_{vc}, \vec{\mathbf{d}}_{vc})), \tag{13}$$

where F_w is a tiny MLP that serves as a view encoding function. We then encode the view direction information to the anchor feature f_v by compositing a feature bank containing information with different resolutions as follows:

$$\hat{f}_v = w \cdot f_v + w_1 \cdot f_{v_{\downarrow_1}} + w_2 \cdot f_{v_{\downarrow_2}}, \tag{14}$$

In practice, we implement the feature bank via slicing and repeating, as illustrated in Fig. 9. We found this slicing and mixture operation improves *Scaffold*-GS's ability to capture different scene granularity. The distribution of feature bank's weights is illustrated in Fig. 10.

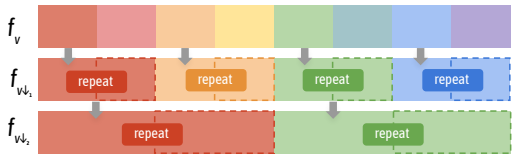

Figure 9. **Generation of Feature Bank.** We expand the anchor feature f into a set of *multi-resolution* features $\{f_v, f_{v_{\downarrow_1}}, f_{v_{\downarrow_2}}\}$ via slicing and repeating. This operation improves *Scaffold*-GS's ability to capture different scene granularity.

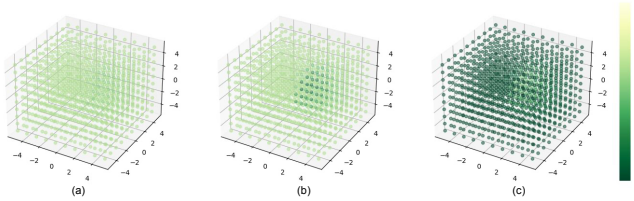

Figure 10. **View-based feature bank's weight distribution.** (a), (b) and (c) denote the predicted weights $\{w_2, w_1, w\}$ for $\{f_{v_{\downarrow_2}}, f_{v_{\downarrow_1}}, f_v\}$ from a group of uniformally distributed viewpoints. Light color denotes larger weights. For this anchor, finer features are more activated at center view positions. The patterns exhibit the ability to capture different scene granularities based on view direction and distance.

MLPs as feature decoders. The core MLPs include the opacity MLP F_α, the color MLP F_c and the covariance MLP F_s and F_q. All of these F_* are implemented in a LINEAR $\rightarrow$ RELU $\rightarrow$ LINEAR style with the hidden dimension of 32, as illustrated in Fig. 11. Each branch's output is activated with a head layer.

- For *opacity*, the output is activated by Tanh, where value 0 serves as a natural threshold for selecting valid samples and the final valid values can cover the full range of [0,1).
- For *color*, we activate the output with Sigmoid function:

$$\{c_0, ..., c_{k-1}\} = \text{Sigmoid}(F_c), \tag{15}$$

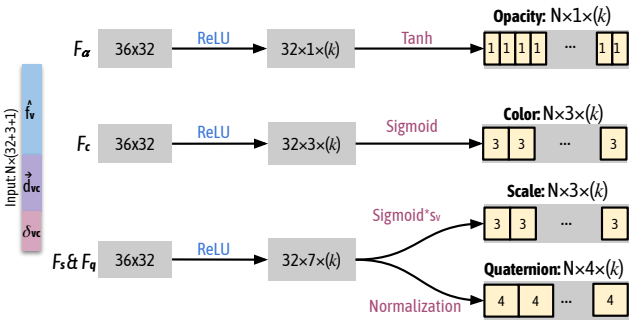

Figure 11. **MLP Structures.** For each anchor point, we use small MLPs (F_α, F_c, F_s, F_q) to predict attributes (opacity, color, scale and quaternion) of k neural Gaussians. The input to MLPs are anchor feature $\hat{f}_v$, relative viewing direction $\vec{\mathbf{d}}_{vc}$ and distance δ_{vc} between the camera and anchor point.

which constrains the color into a range of (0,1).

- For *rotation*, we follow 3D-GS [22] and activate it with a normalization to obtain a valid quaternion.
- For *scaling*, we adjust the base scaling s_v of each anchor with the MLP output as follows:

$$\{s_0, ..., s_{k-1}\} = \text{Sigmoid}(F_s) \cdot s_v, \tag{16}$$

Voxel Size. The voxel size ϵ sets the finest anchor resolution. We employ two strategies: 1) Use the median of the nearest-neighbor distances among all initial points: ϵ is adapted to point cloud density, yielding denser anchors with enhanced rendering quality but might introduce more computational overhead; 2) Set ϵ manually to either 0.005 or 0.01: this is effective in most scenarios but might lead to missing details in texture-less regions. We found these two strategies adequately accommodate various scene complexities in our experiments.

Anchor Refinement. As briefly discussed in the main paper, the voxelization process suggests that our method may behave sensitive to initial SfM results. We illustrate the effect of the anchor refinement process in Fig. 12, where new anchors enhance scene details and fill gaps in large texture-less regions and less observed areas.

8. Experiments and Results

Additional Data Preprocessing. We used COLMAP [39] to estimate camera poses and generate SfM points for VR-NeRF [51] and BungeeNeRF [49] datasets. Both two datasets are challenging in terms of varying levels of details presented in the captures. The VR-NeRF dataset was tested using its eye-level subset with 3 cameras. For all other datasets, we adhered to the original 3D-GS [22] method, sourcing them from public resources.

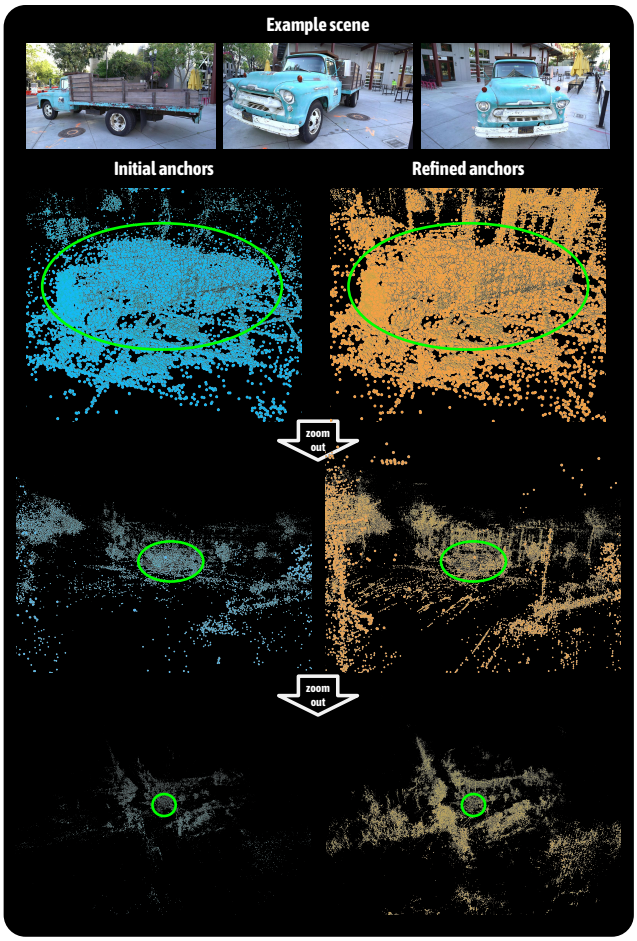

Figure 12. **Anchor Refinement.** We visualize the initial and refined anchor points on the truck scene [23]. The truck is highlighted by the circle. Note that the refined points effectively covers surrounding regions and fine-scale structures, leaning to more complete and detailed scene renderings.

Selection Process by Opacity. We examine the decoded opacity from neural Gaussians and our opacity-based selection process (Fig. 2(b)) from two aspects. First, without the anchor point refinement module, we filter neural Gaussian using their decoded opacity values to extract geometry from a random point cloud. Fig. 15 demonstrates that the remained neural Gaussians effectively reconstruct the coarse structure of the bulldozer model from random points, highlighting its capability for implicit geometry modeling under mainly rendering-based supervision. We found this is conceptually similar to the proposal network utilized in [4], serving as the geometry proxy estimator for efficient sampling. Second, we apply different k values in our method. Fig. 14 shows that regardless of the k value, the final number of activated neural Gaussians converges to a similar amount through the training, indicating *Scaffold*-GS's preference to select a collection of non-redundant Gaussians

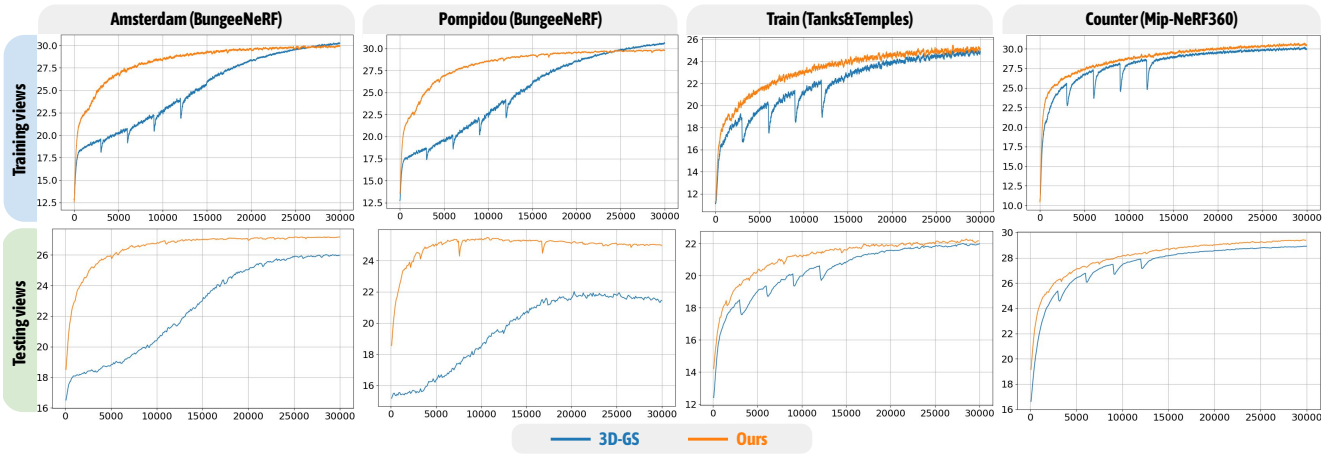

Figure 13. **PSNR curve of *Scaffold*-GS and 3D-GS [22] across diverse datasets [4, 17, 49].** We illustrate the variations in PSNR during the training process for both training and testing views. The orange curve represents *Scaffold*-GS, while the blue curve corresponds to 3D-GS. Our method not only achieves rapid convergence but also exhibits superior performance, marked by a significant rise in training PSNR and consistently higher testing PSNR, in contrast to 3D-GS.

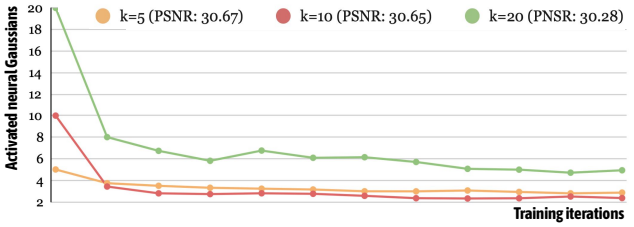

Figure 14. **Learning with different k values.** Despite varying initial k values under different hyper-parameter settings, they converge to activate a similar number of neural Gaussians with comparable rendering fidelity.

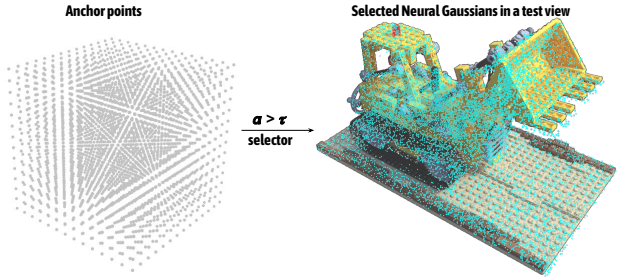

Figure 15. **Geometry culling via selector.** (Left) Anchor points from randomly initialized points; (Right) Activated neural Gaussians derived from each anchor under the current view. In synthetic Blender scenes, with all 3D Gaussians visible in the viewing frustum, our opacity filtering functions similar to a geometry proxy estimator, excluding unoccupied regions before rasterization.

that are sufficient to represent the scene.

Per-scene Results. Here we list the error metrics used in our evaluation in Sec.4 across all considered methods and scenes, as shown in Tab. 6-17.

Table 6. **SSIM scores for Mip-NeRF360 [4] scenes.**

Method | Scenes	bicycle	flowers	garden	stump	treehill	room	counter	kitchen	bonsai
3D-GS [22]	**0.771**	**0.605**	**0.868**	**0.775**	0.638	0.914	0.905	0.922	0.938
Mip-NeRF360 [4]	0.685	0.583	0.813	0.744	0.632	0.913	0.894	0.920	0.941
iNGP [31]	0.491	0.450	0.649	0.574	0.518	0.855	0.798	0.818	0.890
Plenoxels [13]	0.496	0.431	0.606	0.523	0.509	0.842	0.759	0.648	0.814
Ours	0.739	0.577	0.851	0.764	**0.641**	**0.929**	**0.917**	**0.932**	**0.949**

Table 7. **PSNR scores for Mip-NeRF360 [4] scenes.**

Method | Scenes	bicycle	flowers	garden	stump	treehill	room	counter	kitchen	bonsai
3D-GS [22]	**25.25**	21.52	**27.41**	26.55	22.49	30.63	28.70	30.32	31.98
Mip-NeRF360 [4]	24.37	**21.73**	26.98	26.40	22.87	31.63	29.55	**32.23**	**33.46**
iNGP [31]	22.19	20.35	24.60	23.63	22.36	29.27	26.44	28.55	30.34
Plenoxels [13]	21.91	20.10	23.49	20.66	22.25	27.59	23.62	23.42	24.67
Ours	25.02	21.25	27.32	**26.65**	**23.21**	**31.90**	**29.61**	31.73	32.78

Table 8. **LPIPS scores for Mip-NeRF360 [4] scenes.**

Method | Scenes	bicycle	flowers	garden	stump	treehill	room	counter	kitchen	bonsai
3D-GS [22]	**0.205**	**0.336**	**0.103**	0.210	**0.317**	0.220	**0.204**	0.129	0.205
Mip-NeRF360 [4]	0.301	0.344	0.170	0.261	0.339	0.211	0.204	0.127	**0.176**
iNGP [31]	0.487	0.481	0.312	0.450	0.489	0.301	0.342	0.254	0.227
Plenoxels [13]	0.506	0.521	0.386	0.503	0.540	0.419	0.441	0.447	0.398
Ours	0.266	0.372	0.134	0.260	0.347	**0.191**	**0.184**	**0.117**	0.178

Table 9. **Storage size (MB) for Mip-NeRF360 [4] scenes.**

Method | Scenes	bicycle	flowers	garden	stump	treehill	room	counter	kitchen	bonsai
3D-GS [22]	1291	804	1268	1034	813	327	261	414	281
Ours	**275**	**217**	**213**	**230**	**209**	**86**	**83**	**101**	**126**

Table 10. **SSIM scores for Tanks&Temples [23] and Deep Blending [18] scenes.**

Method | Scenes	Truck	Train	Dr Johnson	Playroom
3D-GS [22]	0.879	0.802	0.899	0.906
Mip-NeRF360 [4]	0.857	0.660	0.901	0.900
iNPG [31]	0.779	0.666	0.839	0.754
Plenoxels [13]	0.774	0.663	0.787	0.802
Ours	**0.886**	**0.821**	**0.907**	**0.913**

Training Process Analysis. Figure 13 illustrates the variations in PSNR during the training process for both train-

Table 11. **PSNR scores for Tanks&Temples [23] and Deep Blending [18] scenes.**

Method | Scenes	Truck	Train	Dr Johnson	Playroom
3D-GS [22]	25.19	21.10	28.77	30.04
Mip-NeRF360 [4]	24.91	19.52	29.14	29.66
iNPG [31]	23.26	20.17	27.75	19.48
Plenoxels [13]	23.22	18.93	23.14	22.98
Ours	**25.89**	**22.20**	**29.79**	**31.07**

Table 12. **LPIPS scores for Tanks&Temples [23] and Deep Blending [18] scenes.**

Method | Scenes	Truck	Train	Dr Johnson	Playroom
3D-GS [22]	0.148	0.218	0.244	**0.241**
Mip-NeRF360 [4]	0.159	0.354	**0.237**	0.252
iNPG [31]	0.274	0.386	0.381	0.465
Plenoxels [13]	0.335	0.422	0.521	0.499
Ours	**0.141**	**0.204**	0.250	0.250

Table 13. **Storage size (MB) for Tanks&Temples [23] and Deep Blending [18] scenes.**

Method | Scenes	Truck	Train	Dr Johnson	Playroom
3D-GS [22]	578	240	715	515
Ours	**94**	**58**	**61**	**50**

Table 14. **PSNR scores for Synthetic Blender [30] scenes.**

Method | Scenes	Mic	Chair	Ship	Materials	Lego	Drums	Ficus	Hotdog
3D-GS [22]	35.36	**35.83**	30.80	30.00	**35.78**	26.15	34.87	37.72
Ours	**37.25**	35.28	**31.17**	**30.65**	35.69	**26.44**	**35.21**	**37.73**

Table 15. **Storage size (MB) for Synthetic Blender [30] scenes.**

Method | Scenes	Mic	Chair	Ship	Materials	Lego	Drums	Ficus	Hotdog
3D-GS [22]	50	116	63	35	78	93	59	44
Ours	**12**	**13**	**16**	**18**	**13**	**35**	**11**	**8**

Table 16. **PSNR scores for BungeeNeRF [49] and VR-NeRF [51] scenes.**

Method | Scenes	Amsterdam	Bilbao	Pompidou	Quebec	Rome	Hollywood	Apartment	Kitchen
3D-GS [22]	25.74	26.35	21.20	28.79	23.54	23.25	28.48	29.40
Ours	**27.10**	**27.66**	**25.34**	**30.51**	**26.50**	**24.97**	**28.87**	**29.61**

Table 17. **Storage size (MB) for BungeeNeRF [49] and VR-NeRF [51] scenes.**

Method | Scenes	Amsterdam	Bilbao	Pompidou	Quebec	Rome	Hollywood	Apartment	Kitchen
3D-GS [22]	1453	1337	2129	1438	1626	1642	202	323
Ours	**243**	**197**	**230**	**166**	**200**	**182**	**48**	**90**

ing and testing views. Our method demonstrates quicker convergence, enhanced robustness, and better generalization compared to 3D-GS, as evidenced by the rapid increase in training PSNR and higher testing PSNR. Specifically, for the Amsterdam and Pompidou scenes in BungeeNeRF, we trained them with images at *three coarser scales* and evaluated them at a *novel finer scale*. The fact that 3D-GS achieved higher training PSNR but lower testing PSNR indicates its tendency to overfit at training scales.