DropGaussian: Structural Regularization for Sparse-view Gaussian Splatting

Supplementary Material

1. Introduction

In this supplementary material, we provide more detailed explanations of the proposed method. First, we explain implementation details including the process of Gaussian initialization and training. In Section 3, we provide in-depth analysis of the relationship between the visibility of Gaussians and the magnitude of gradients, highlighting their impact on the optimization process. Finally, the improvements achieved by our method, particularly focusing on reducing floater artifacts and mitigating the overfitting problem, are discussed in Section 4.

2. Implementation Details

2.1. Initialization

Following previous methods [2, 9], we initialize our pipeline with unstructured multi-view images by calibrating them via Structure-from-Motion (SfM) [6, 7]. Specifically, we generate the initial point cloud through dense stereo matching in COLMAP [6] using "patch-match-stereo" and obtain the fused stereo point cloud based on "stereo-fusion". As a next step, we initialize the SH coefficients with the degree of 0. In addition, we set positions of 3D Gaussians based on the fused point cloud and the opacity value to 0.1 while other coefficients (e.g., scale, rotation, etc.) are set to 0. For datasets where point cloud generation was not available (e.g., Blender dataset [5]), we followed the approach described in DNGaussian [3], which initializes our method with 10,000 Gaussians randomly distributed.

2.2. Training

During training, we begin with spherical harmonics (SH) with the degree of 0, representing a basic approximation of lighting effects. We increase the degree by 1 every 1,000 iterations until ultimately reaching a maximum degree of 3. This gradual increment improves the details of the lighting representation as training progresses. The learning rates for different parameters are set as follows: 0.00016 for position, 0.0025 for SH coefficients, 0.05 for opacity, 0.005 for scaling factor, and 0.001 for rotation, respectively. We reset the opacity value of all Gaussians to 0.01 every 3,000 iterations. Algorithm 1 provides a detailed description of the training pipeline of DropGaussian.

Algorithm 1 The training pipeline of DropGaussian

Training view images $I = \{I_i \in \mathbb{R}^{H\times W\times 3}\}_{i=1}^{N}$ and camera poses $P = \{\phi_i \in \mathbb{R}^{3\times 4}\}_{i=1}^{N}$.
Run SfM step with the input images and camera poses to obtain an initial point cloud P, used to define 3D Gaussian functions $G = \{G_i(\mu_i, \sigma_i, c_i, o_i)\}_{i=1}^{K}$.
Initialize SH coefficients to degree 0, opacity o_i to 0.1, scaling factors γ to 0.2.
while until convergence **do**
  Sample an image $I_i \in I$ with camera pose ϕ_i.
  Calculate dropping rate at iteration t: $r_t = \gamma \cdot \frac{t}{t_{\text{total}}}$,
  Generate compensation factors $M_i(r_t)$ for each Gaussian based on r_t:

$$M_i(r_t) = \begin{cases} 0 & \text{if dropped} \\ \frac{1}{1-r_t} & \text{otherwise} \end{cases}$$

  Update opacity $\tilde{o}_i = o_i \cdot M_i(r_t)$ for each Gaussian.
    ▷ $\tilde{o}_i$ is a temporal opacity only used in training.
  Rasterize the RGB image $\hat{I}_i$ with opacity $\tilde{o}_i$.
  $L = \|I_i - \hat{I}_i\|_1 + \lambda\text{D-SSIM}(I_i, \hat{I}_i)$
  if IsRefinementIteration(t) **then**
    for $G_i(\mu_i, \sigma_i, c_i, o_i) \in G$ **do**
      if $o_i < \epsilon$ **then** ▷ Pruning
        RemoveGaussian(G_i)
      end if
      if $\nabla_p L > t_{\text{pos}}$ **then** ▷ Densification
        GaussianDensify(G_i)
      end if
    end for
  end if
  Update Gaussian parameters $G_i(\mu_i, \sigma_i, c_i, o_i)$ with gradient descent.
end while

3. Analysis on Visibility of Gaussian

3.1. Relationship between Gaussian Visibility and Gradient Magnitude

We demonstrate the relationship between α values, representing each Gaussian's contribution to a single pixel, and their corresponding gradient magnitudes in Fig. 1. In the case without using DropGaussian, Gaussians nearby the camera mostly contribute to the pixel value while occluded Gaussians exhibit lower α values and smaller gradient magnitudes (see right-top part in Fig. 1). In contrast, after Gaussians located in the front are dropped, the remaining Gaus-

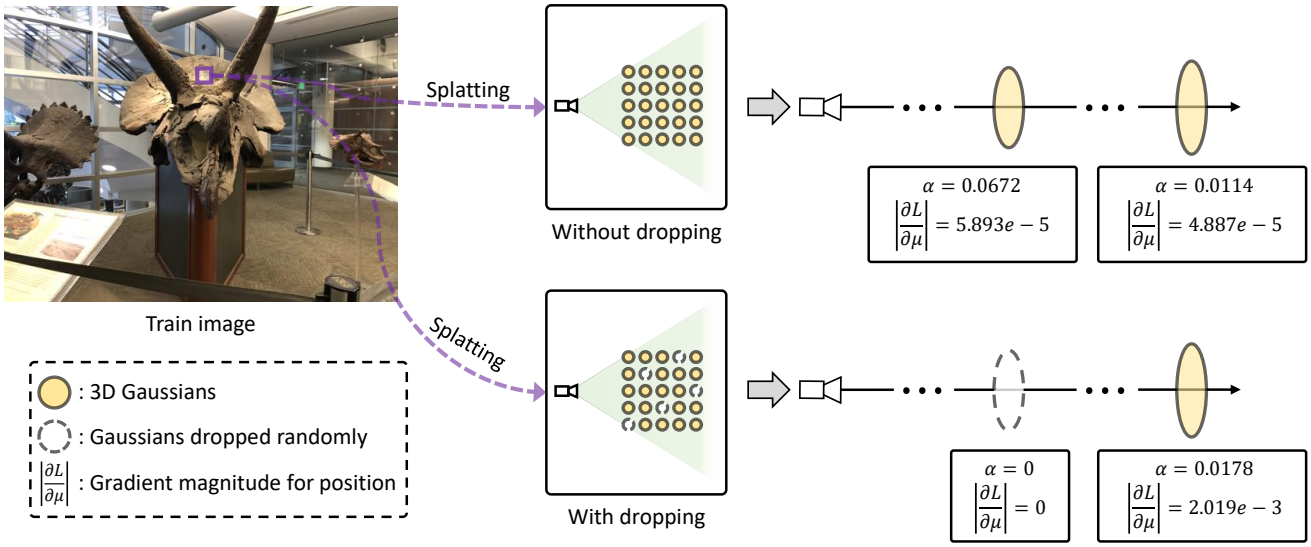

Figure 1. Visualization of the relationship between Gaussian visibility and gradient magnitudes. Without dropping, foreground Gaussians cause occluded Gaussians to have lower α values, leading to smaller gradient magnitude. When occluding Gaussians are dropped, occluded Gaussians show relatively higher α values and larger gradient magnitude.

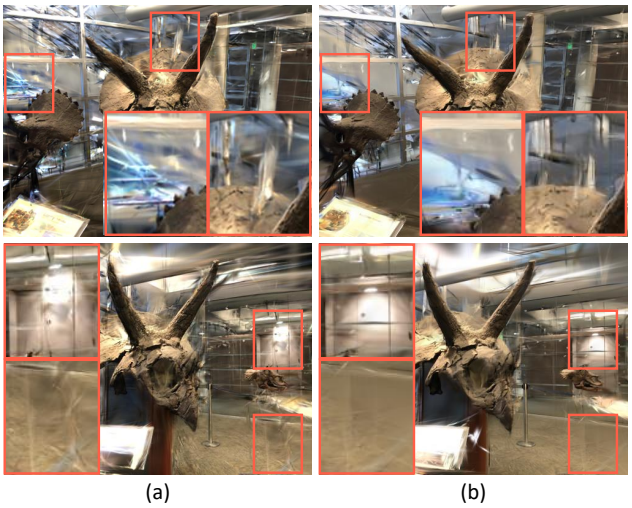

Figure 2. (a) Novel view rendering using the standard 3DGS. (b) Novel view rendering generated by the standard 3DGS implementation combined with DropGaussian.

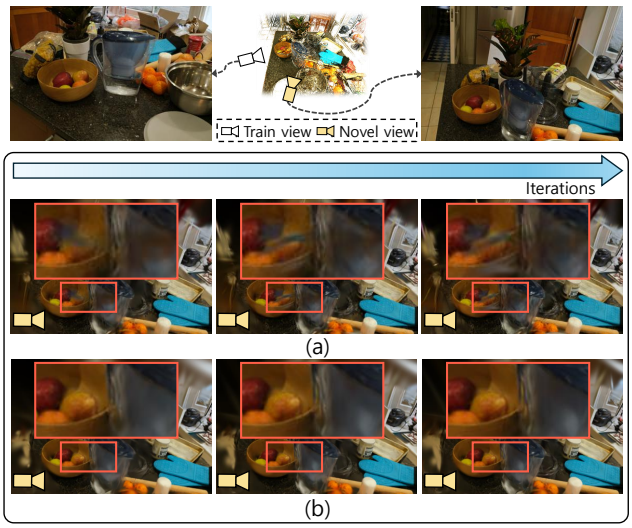

Figure 3. Comparison of novel view renderings across training iterations. (a) Results of standard 3DGS, where floaters are observed to grow. (b) Results of 3DGS with DropGaussian, which effectively suppresses floaters.

sians gain relatively higher α values and larger gradient magnitudes, thereby increasing their influence during optimization (see right-bottom part in Fig. 1). This highlights the importance of managing the visibility of Gaussians to effectively consider the gradient of Gaussians even far from the camera, which is helpful to alleviate the overfitting problem in sparse-view conditions.

Furthermore, the gradient associated with the Gaussian attribute μ, which determines its position, also plays a crucial role in the densification process. In the sparse-view setting, the densification process often concentrates only on specific regions, such as those closer to the camera, leading to the overfitting problem as well as the generation of floaters. By modulating gradients with DropGaussian as shown above, this imbalance can be mitigated, enabling a more adaptive allocation of Gaussians during the densification process. An example of the corresponding effect is shown in Fig. 2.

3.2. Alleviation of Floater Artifacts

During the training process, the growth of floaters poses a significant challenge in accurately rendering a given

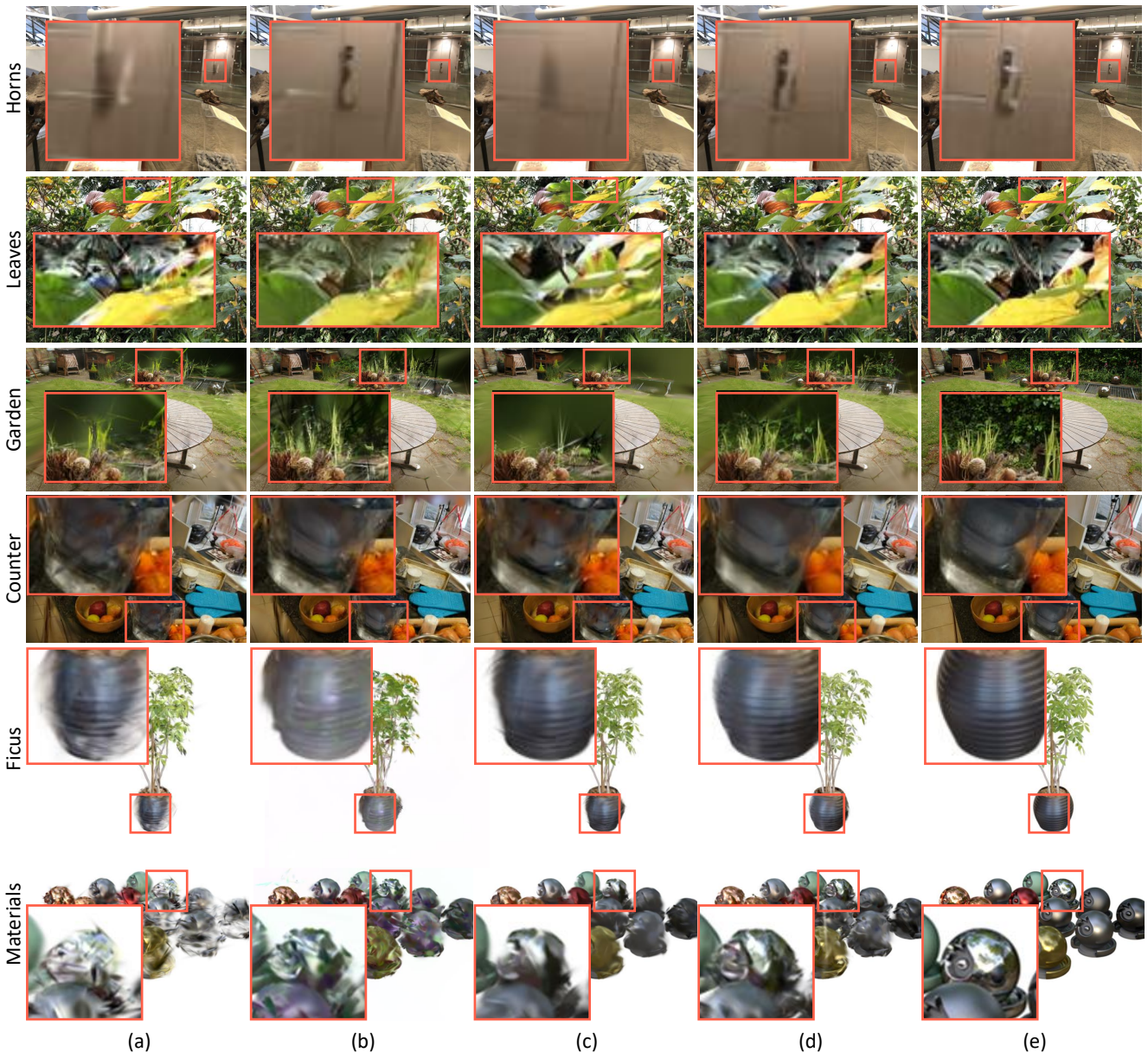

Figure 4. Novel view rendering results for LLFF [4] (first–second rows), Mip-NeRF360 [1] (third–fourth rows), and Blender [5] (fifth–sixth rows). From left to right: (a) 3DGS, (b) FSGS [9], (c) CoR-GS [8], (d) Ours, and (e) GT image.

scene. Floaters are unintended density artifacts that appear in front of the camera, degrading the visual quality. As shown in Fig. 3 (a), floaters become more prominent over time with higher α value (i.e., large gradients). This leads to repeated densification near the camera, further exacerbating the generation of floaters. By applying DropGaussian, the dominant effect of such Gaussians in the process of the gradient update is reduced, leading to the update of all the Gaussians in a more balanced way. Consequently, floaters are significantly suppressed, resulting in cleaner and more precise renderings (see (b) of Fig. 3).

4. Discussion on Improvements

4.1. Qualitative Results

As shown in Fig. 4, our method demonstrates notable improvements compared to existing approaches. Specifically, complex textures are reliably rendered by the proposed method whereas other approaches often yield distortions of the underlying structure in a given scene (see the second and third rows of Fig. 4). Moreover, while other approaches often render details at inaccurate positions in novel views, the proposed method accurately places them (see the

Figure 5. Novel view renderings on the LLFF dataset [4] using DropGaussian. While DropGaussian effectively mitigates overfitting, some renderings may exhibit slight blurry results due to the random dropping of Gaussians.

fifth row of Fig. 4). In addition, the proposed method accurately renders small-scale details whereas other approaches often yield somewhat blurry outputs in novel views (see the sixth row of Fig. 4).

4.2. Limitations

Although DropGaussian effectively mitigates the overfitting problem, its stochastic process may occasionally remove Gaussians that are crucial for reconstructing fine details of the scene. Consequently, some novel view renderings on the LLFF dataset [4] generate slightly blurry results, as shown in Fig. 5. This occurs because the random removal of foreground Gaussians can sometimes eliminate Gaussians that are critical for capturing fine details, which are essential for high-quality texture reconstruction.

References

[1] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, pages 5470–5479, 2022. 3

[2] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 42(4):139–1, 2023. 1

[3] Jiahe Li, Jiawei Zhang, Xiao Bai, Jin Zheng, Xin Ning, Jun Zhou, and Lin Gu. Dngaussian: Optimizing sparse-view 3d gaussian radiance fields with global-local depth normalization. In *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, pages 20775–20785, 2024. 1

[4] Ben Mildenhall, Pratul P Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. *ACM Trans. Graph.*, 38(4):1–14, 2019. 3, 4

[5] 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 *Proc. Eur. Conf. Comput. Vis.*, pages 405–421, 2020. 1, 3

[6] Johannes L Schonberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, pages 4104–4113, 2016. 1

[7] Johannes L Schönberger, Enliang Zheng, Jan-Michael Frahm, and Marc Pollefeys. Pixelwise view selection for unstructured multi-view stereo. In *Proc. Eur. Conf. Comput. Vis.*, pages 501–518, 2016. 1

[8] Jiawei Zhang, Jiahe Li, Xiaohan Yu, Lei Huang, Lin Gu, Jin Zheng, and Xiao Bai. Cor-gs: sparse-view 3d gaussian splatting via co-regularization. In *Proc. Eur. Conf. Comput. Vis.*, pages 335–352, 2024. 3

[9] Zehao Zhu, Zhiwen Fan, Yifan Jiang, and Zhangyang Wang. Fsgs: Real-time few-shot view synthesis using gaussian splatting. In *Proc. Eur. Conf. Comput. Vis.*, pages 145–163, 2024. 1, 3