DashGaussian: Optimizing 3D Gaussian Splatting in 200 Seconds

Supplementary Material

In this supplementary material, we include explanations of the implementation details in Sec. 7, the discussion over primitive growth in Sec. 8 and furthermore extensive experiments in Sec. 9, Sec. 10 and Sec. 11.

7. Implementation Details

In this section, we elaborate on the implementation of our ground truth downsampling strategy, the modification to the learning rate, the modulation on the resolution scheduler, and how we equip DashGaussian to the each backbones.

7.1. Ground Truth Downsampling

In the LR optimization stage, DashGaussian downsamples the ground truth to supervise the LR renderings from 3D Gaussians. We choose the official anti-alias downsampling algorithm from Pytorch to downsample the ground truth images, preventing possible misguidance from aliased supervision hindering the optimization.

7.2. Learning Rate

Because the optimization scheme in DashGaussian is specifically tailored, the learning rate for 3DGS optimization should be properly scheduled to allow DashGaussian to perform the best. For all the experiments, we keep all the hyper-parameters unchanged as in the original 3DGS [4] expect the positional learning rate. We hold it constant as the initial value in the LR optimization stage, and only allow it to decay from the iteration $i = \mathrm{argmin}_i \left\{ \lfloor r^{(i)} \rfloor = 1 \right\}$.

7.3. Modulation for Resolution Scheduler

As we pointed out in the paper, DashGaussian allocates the iterations spent on the LR and HR optimization stage, in order to accelerate the optimization process while maintaining the rendering quality of 3DGS. To this end, we further modulate s_r to suppress the LR stage while expanding the HR stage. Specifically, we apply the natural logarithm to modulate the relative multiple for different resolutions, resulting in a fraction function

$$f(\mathbf{F}, \mathbf{F}_{r_m}, \mathbf{F}_{r_i}) = \frac{\ln\left(\mathcal{X}(\mathbf{F})/\mathcal{X}(\mathbf{F}_{r_i})\right)}{\ln\left(\mathcal{X}(\mathbf{F})/\mathcal{X}(\mathbf{F}_{r_m})\right)}, \tag{6}$$

with which we have

$$\begin{aligned} s_{r_i} &= S \cdot (1 - f(\mathbf{F}, \mathbf{F}_{r_m}, \mathbf{F}_{r_i})) \\ &= S \cdot \frac{\ln\left(\mathcal{X}(\mathbf{F}_{r_i})/\mathcal{X}(\mathbf{F}_{r_m})\right)}{\ln\left(\mathcal{X}(\mathbf{F})/\mathcal{X}(\mathbf{F}_{r_m})\right)}. \end{aligned} \tag{7}$$

Eq. (7) is the actual scheduling function we use for DashGaussian and all experiments in the paper.

7.4. Densification of DashGaussian

We explain how a densification step is performed in DashGaussian. Given an optimization step i where a densification operation is performed, we set N_{GS} as the current number of Gaussian primitives, and P_i as the desired primitive number after this densification operation. We first perform a prune operation on the primitives, resulting in N'_{GS} primitives left. Then we perform clone and split operation to those Gaussian primitives that satisfy the densification condition of the backbone while having a top $P_i - N'_{\mathrm{GS}}$ densification score. Each primitive is cloned or split depending on its size, which follows the same routine as in 3DGS [4]. When the densification score differs, slight difference is there to adopt DashGaussian to the backbones.

7.5. Equipping DashGaussian to Backbones

When we enhance a 3DGS backbone with DashGaussian, only few modifications to the codebase are adopted, including a new learning rate schedule (see Sec. 7.2 in supplementary), the rendering resolution schedule, the primitive growth schedule, and a distinct densification operation (see Sec. 7.4). These four modifications from DashGaussian are basically common for different backbones, with slight differences in the densification operation. We introduce these differences in detail below.

3DGS [4] The original 3DGS has a densification score defined as the positional gradient of each primitive. We keep the densification score, and follow Sec. 7.4 to perform densification. The optimizer is Adam [5].

Taming-3DGS [9] At the moment of this paper is written, the codebase of Taming-3DGS has an efficient backward pass implementation with the rest same as 3DGS [4]. So the densification operation is the same as Sec. 7.4. The optimizer is Sparse Adam [9].

Revising-3DGS* [2] Because Revising-3DGS is not open-sourced, we reproduce it on top of the codebase of Taming-3DGS [9]. Notice that, Revising-3DGS demands appointing the primitive number in the final 3DGS model. To this end, we follow [2] to set P_{fin} as a constant, with our momentum-based primitive budgeting disabled. Revising-3DGS has its densification score defined as rendering error, thus the densification operation follows Sec. 7.4 the same. The optimizer is Sparse Adam [9].

Mip-Splatting [10] Mip-Splatting defines its densification scores as two different scalars, where these two scores are used together to specify the primitives to be densified. For the primitive selection in densification, we merge the

Table 5. Scene-wise quantitative results over Mip-NeRF 360 dataset [1].

Method	bicycle					garden					stump				
	N_{GS} ↓	PSNR↑	SSIM↑	LPIPS↓	Time↓	N_{GS} ↓	PSNR↑	SSIM↑	LPIPS↓	Time↓	N_{GS} ↓	PSNR↑	SSIM↑	LPIPS↓	Time↓
3DGS [4]	5.74M	25.61	0.778	0.203	24.82	5.10M	27.80	**0.874**	**0.102**	23.99	4.64M	26.91	0.784	0.207	19.87
+Ours	**5.38M**	**25.70**	**0.785**	**0.202**	**14.86**	**3.42M**	**27.93**	0.873	0.110	**11.72**	**3.80M**	**27.32**	**0.797**	**0.201**	**9.64**
Mip-Splatting [10]	7.82M	**25.95**	**0.803**	**0.162**	38.81	5.60M	27.90	**0.884**	**0.089**	32.78	5.66M	27.15	**0.800**	**0.181**	28.57
+Ours	**5.42M**	25.85	0.796	0.176	**17.67**	**3.87M**	**28.18**	0.882	0.096	**14.89**	**4.58M**	**27.20**	0.800	0.189	**11.39**
Revising-3DGS* [2]	3.20M	25.58	0.776	0.205	5.44	2.65M	27.75	0.873	0.106	6.78	3.03M	27.04	0.797	0.191	5.43
+Ours	3.20M	**25.66**	**0.791**	**0.191**	**4.19**	2.65M	**27.89**	**0.878**	**0.101**	**4.77**	3.03M	**27.42**	**0.812**	**0.182**	**3.25**
Taming-3DGS [9]	**3.85M**	25.58	0.766	0.226	6.95	3.07M	27.63	**0.869**	**0.110**	6.89	3.73M	26.66	0.775	0.220	5.26
+Ours	3.87M	**25.81**	**0.782**	**0.210**	**4.38**	**2.46M**	**27.91**	0.868	0.119	**3.80**	**3.14M**	**27.34**	**0.792**	**0.211**	**2.71**

Method	flowers					treehill					room				
	N_{GS} ↓	PSNR↑	SSIM↑	LPIPS↓	Time↓	N_{GS} ↓	PSNR↑	SSIM↑	LPIPS↓	Time↓	N_{GS} ↓	PSNR↑	SSIM↑	LPIPS↓	Time↓
3DGS [4]	3.36M	21.86	0.622	0.328	16.50	**3.62M**	22.85	0.652	0.317	17.05	1.51M	**31.72**	**0.927**	**0.191**	15.60
+Ours	**3.22M**	**22.17**	**0.631**	**0.318**	**10.78**	3.63M	**23.14**	**0.659**	**0.309**	**12.13**	**1.29M**	31.58	0.923	0.203	**8.06**
Mip-Splatting [10]	4.32M	22.06	**0.656**	**0.266**	23.80	5.01M	22.61	0.655	**0.269**	26.12	2.10M	31.94	**0.933**	**0.175**	21.07
+Ours	**3.88M**	**22.13**	0.648	0.282	**13.76**	**4.60M**	**23.15**	**0.662**	0.285	**15.47**	**1.59M**	**32.08**	0.931	0.183	**9.90**
Revising-3DGS* [2]	2.05M	21.56	0.625	0.286	5.46	2.17M	22.65	0.647	0.307	4.80	1.00M	30.81	0.922	**0.192**	4.92
+Ours	2.05M	**22.05**	**0.648**	**0.279**	**3.88**	2.17M	**22.80**	**0.656**	**0.300**	**3.30**	1.00M	**31.97**	**0.929**	0.194	**2.53**
Taming-3DGS [9]	2.38M	21.83	0.615	0.336	4.72	**2.52M**	22.92	0.649	0.330	4.81	1.11M	31.51	**0.925**	**0.198**	4.32
+Ours	**2.34M**	**22.31**	**0.633**	**0.320**	**3.30**	2.88M	**23.33**	**0.658**	**0.315**	**3.60**	**0.90M**	**31.85**	0.922	0.205	**2.38**

Method	kitchen					counter					bonsai				
	N_{GS} ↓	PSNR↑	SSIM↑	LPIPS↓	Time↓	N_{GS} ↓	PSNR↑	SSIM↑	LPIPS↓	Time↓	N_{GS} ↓	PSNR↑	SSIM↑	LPIPS↓	Time↓
3DGS [4]	1.74M	31.41	**0.933**	**0.113**	18.28	1.17M	**29.12**	**0.915**	**0.178**	15.08	1.24M	**32.19**	**0.947**	**0.173**	13.56
+Ours	**1.39M**	**31.58**	0.926	0.128	**9.68**	**1.01M**	28.90	0.910	0.188	**7.31**	**1.06M**	32.01	0.944	0.177	**7.28**
Mip-Splatting [10]	2.13M	**31.86**	**0.936**	**0.106**	23.57	1.47M	**29.33**	**0.920**	**0.165**	19.72	1.61M	**32.41**	**0.951**	**0.157**	18.01
+Ours	**1.71M**	31.76	0.930	0.119	**12.01**	**1.28M**	29.22	0.915	0.176	**9.14**	**1.35M**	32.32	0.946	0.164	**9.21**
Revising-3DGS* [2]	1.43M	**31.93**	**0.934**	**0.114**	7.08	0.89M	**29.23**	**0.916**	**0.175**	6.51	1.01M	**32.26**	**0.947**	**0.166**	5.11
+Ours	1.43M	31.78	0.930	0.123	**3.66**	0.89M	29.03	0.912	0.185	**3.01**	1.01M	32.03	0.945	0.169	**2.58**
Taming-3DGS [9]	1.68M	31.09	**0.930**	**0.117**	7.71	1.02M	29.10	**0.914**	**0.180**	4.92	1.16M	**32.20**	0.943	**0.175**	4.00
+Ours	**1.21M**	**31.52**	0.926	0.127	**3.89**	**0.78M**	**29.10**	0.910	0.188	**2.63**	**0.80M**	32.10	**0.943**	0.175	**2.38**

primitives selected with these two densification scores as a whole set and randomly select $P_i - N'_{GS}$ primitives from them for densification. The optimizer is Adam [5].

8. Primitive Growth of DashGaussian

8.1. Why Fewer Primitives in DashGaussian?

As we discussed in Sec. 7.4 of the supplementary, the densified primitives are selected by the vanilla densification score together with a top-k selection to control the growth of primitives. In the early LR optimization stage, the top-k selection prevents the situation where a large amount of Gaussians pass the threshold resulting in an explosion of primitive growth. In the late HR optimization stage, the optimized 3DGS is relatively stable and there are less than $P_i - N'_{GS}$ primitives passing over the densification threshold. The threshold-based densification and the top-k selection constrain each other, resulting in less densified primitives in the final optimized 3DGS than P_{fin}.

8.2. Constant Primitive Number

Intuitively, P_{fin}, as the primitive number upper-bound, is likely to influence the size of the optimized 3DGS. When P_{fin} is extremely large, the primitive number N_{GS} in the optimized 3DGS is likely to explode, resulting in out-of-memory (OOM) issue. Nonetheless, thanks to the double-condition densification scheme as discussed in Sec. 8.1, we can prevent OOM when P_{fin} is arbitrarily large. Concretely, as Tab. 9 in the supplementary shows, when we remove the primitive scheduler, which is equivalent to setting P_{fin} as infinity, N_{GS} still approximates to the primitive number of the backbone. This indicates the primitive growth of DashGaussian is stable and safe. Also, one can simply remove the threshold condition and only keep the top-k selection in densification, such that N_{GS} will exactly equal to P_{fin} after the optimization.

Table 6. Scene-wise quantitative results over Deep Blending dataset [3].

Method	drjohnson					playroom				
	N_{GS} ↓	PSNR↑	SSIM↑	LPIPS↓	Time↓	N_{GS} ↓	PSNR↑	SSIM↑	LPIPS↓	Time↓
3DGS [4]	3.31M	**29.09**	**0.901**	**0.244**	19.47	2.32M	**29.90**	**0.907**	**0.244**	17.27
+Ours	**2.47M**	28.75	0.895	0.262	**8.68**	**1.89M**	29.84	0.906	0.249	**7.64**
Mip-Splatting [10]	4.13M	28.66	0.898	**0.243**	26.80	2.83M	29.85	0.906	**0.235**	20.33
+Ours	**3.04M**	**28.80**	0.899	0.255	**10.73**	**2.06M**	**30.29**	**0.906**	0.241	**8.86**
Revising-3DGS* [2]	2.52M	29.12	0.902	**0.248**	4.69	1.51M	29.96	0.909	**0.248**	3.43
+Ours	2.52M	**29.35**	**0.903**	0.252	**2.47**	1.51M	**30.18**	**0.911**	0.250	**1.94**
Taming-3DGS [9]	2.93M	29.24	0.905	**0.241**	5.38	1.72M	30.14	**0.908**	0.249	3.65
+Ours	**2.53M**	**29.56**	**0.905**	0.247	**2.49**	**1.36M**	**30.49**	0.908	**0.249**	**1.91**

Table 7. Scene-wise quantitative results over Tanks&Temples dataset [6].

Method	train					truck				
	N_{GS} ↓	PSNR↑	SSIM↑	LPIPS↓	Time↓	N_{GS} ↓	PSNR↑	SSIM↑	LPIPS↓	Time↓
3DGS [4]	1.08M	21.78	**0.812**	**0.207**	8.46	2.58M	25.45	**0.882**	**0.146**	12.72
+Ours	**1.01M**	**22.01**	0.806	0.223	**5.91**	**1.85M**	**25.66**	0.882	0.156	**6.38**
Mip-Splatting [10]	1.47M	21.76	**0.825**	**0.190**	11.31	3.24M	25.73	**0.892**	**0.123**	17.15
+Ours	**1.18M**	**22.01**	0.820	0.207	**7.05**	**2.07M**	**25.91**	0.892	0.137	**8.07**
Revising-3DGS* [2]	0.96M	22.10	**0.823**	**0.211**	4.18	1.68M	**25.65**	**0.889**	**0.126**	4.81
+Ours	0.96M	**22.13**	0.817	0.223	**2.88**	1.68M	25.56	0.888	0.129	**3.43**
Taming-3DGS [9]	1.07M	21.96	0.816	**0.204**	3.64	1.96M	25.28	0.882	**0.145**	4.40
+Ours	**1.00M**	**22.22**	**0.817**	0.208	**2.75**	**1.39M**	**25.71**	**0.884**	0.153	**2.48**

Table 8. Quantitative results of the ablation study over γ on Mip-NeRF 360 dataset [1], with Taming-3DGS [9] as the backbone of DashGaussian. '1^-' indicates where γ infinitely approaches 1 from below, equivalent to removing the primitive scheduler.

γ	N_{GS} ↓	PSNR↑	SSIM↑	LPIPS↓	Time↓
Taming-3DGS [9]	2.02M	27.61	0.821	0.210	5.51
1^-	2.46M	**27.99**	**0.829**	**0.196**	4.43
0.99	2.17M	27.98	0.828	0.204	3.49
0.98 (Ours)	2.04M	27.92	0.826	0.208	3.23
0.95	1.52M	27.81	0.820	0.218	2.77
0.90	**1.09M**	27.67	0.809	0.237	**2.41**

Table 9. Quantitative results of the ablation study over a on Mip-NeRF 360 dataset [1], with Taming-3DGS [9] as the backbone of DashGaussian. $a = 1$ equals removing the resolution scheduler, which is the same as '+Prim-Sche.' in Tab. 3 of the paper.

a	N_{GS} ↓	PSNR↑	SSIM↑	LPIPS↓	Time↓
Taming-3DGS [9]	2.02M	27.61	0.821	0.210	5.51
1	2.23M	27.85	0.824	0.208	4.60
2	2.05M	**27.97**	**0.827**	**0.204**	4.62
4 (Ours)	2.04M	27.92	0.826	0.208	3.23
6	2.00M	27.93	0.825	0.211	2.88
10	**1.95M**	27.91	0.824	0.213	**2.79**

9. Scene-wise Qualitative Results

As complementary to the Tab. 2 in the paper, we report scene-wise quantitative results on all three datasets in Tab. 5, Tab. 6 and Tab. 7, respectively.

10. Ablation on Hyper-parameters

In this section, we discuss how the hyper-parameters influence the performance of DashGaussian.

Momentum-based Primitive Budgeting. The two hyper-parameters in Eq. 5 of the paper, γ and η, directly affect P_{fin} and thus N_{GS}. Before the ablation experiment, we first analyze the inner connection between γ, η, and P_{fin} to show how we select them.

Consider when Eq. 5 of the paper is stable and P_{fin} converges to a constant, we can rewrite Eq. 5 of the paper, resulting in

$$P_{\text{fin}} = \frac{\eta}{1-\gamma} P_{\text{add}}^{(i)}, \tag{8}$$

which reveals the inner connection between $P_{\text{fin}}, \gamma, \eta$ and $P_{\text{add}}^{(i)}$. With the above equation, one can select γ and η based on the typical $P_{\text{add}}^{(i)}$ to expect a maximum P_{fin}. However, as introduced in Sec. 8.2 of the supplementary, N_{GS} is stable with arbitrarily large P_{fin}. So selecting γ and η is to make P_{fin} approximate to N_{GS}, which effectively functions

Table 10. The report of large-scale reconstruction results on MatrixCity dataset. Time is reported in minutes.

Method	$N_{GS}\downarrow$	PSNR↑	SSIM↑	LPIPS↓	Time↓
LargeBase	18.28M	26.51	0.858	0.186	44.22
+Ours	**18.08M**	**26.59**	**0.865**	**0.182**	**25.82**

the primitive scheduler to accelerate the optimization while preventing under reconstruction with an over small P_{fin}.

Since changing γ and η is essentially equivalent to changing the one parameter $\eta/(1-\gamma)$ in Eq. (8), we only perform ablation on γ. The results are reported in Tab. 9. As analyzed in Eq. (8), a larger γ indicates a larger P_{fin}, and thus a bigger N_{GS}, higher rendering quality, and more optimization time. To notice, even when $\gamma = 1^-$ where P_{fin} is infinite (implemented as removing the primitive scheduler), the value of N_{GS} is stable and significant acceleration brought by our method is still observed.

Initial Resolution. The hyper-parameter a in Sec. 4.2 of the paper decides the initial rendering resolution at the beginning of optimization, where a larger a indicates a smaller initial rendering resolution. We report ablation on a in Tab. 8. Results show that the rendering quality is basically insensitive to a, while the optimization time significantly reduces with the increase of a.

11. Evaluation on Large-scale Reconstruction

We conduct evaluation for DashGaussian on large-scale reconstruction with the MatrixCity [7] dataset, where the backbone is Taming-3DGS [9] and the divide-and-conquer strategy follows VastGaussian [8] to split the scene into 9 blocks, denoted as LargeBase in Tab. 10. The average training time on 9 blocks is reported (rendered under 1080P). While slightly improving the reconstruction quality, DashGaussian helps the backbone to reduce the training time by 41.6% on each block. This experiment again strongly supports our claim that DashGaussian can serve as a generalized plugin to accelerate the optimization of different 3DGS backbones and can well scale up from room-scale to large-scale 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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5470–5479, 2022. 2, 3

[2] Samuel Rota Bulò, Lorenzo Porzi, and Peter Kontschieder. Revising densification in gaussian splatting, 2024. 1, 2, 3

[3] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. Deep blending for free-viewpoint image-based rendering. 37(6):257:1–257:15, 2018. 3

[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), 2023. 1, 2, 3

[5] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, San Diega, CA, USA, 2015. 1, 2

[6] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics*, 36(4), 2017. 3

[7] Yixuan Li, Lihan Jiang, Linning Xu, Yuanbo Xiangli, Zhenzhi Wang, Dahua Lin, and Bo Dai. Matrixcity: A large-scale city dataset for city-scale neural rendering and beyond. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3205–3215, 2023. 4

[8] Jiaqi Lin, Zhihao Li, Xiao Tang, Jianzhuang Liu, Shiyong Liu, Jiayue Liu, Yangdi Lu, Xiaofei Wu, Songcen Xu, Youliang Yan, and Wenming Yang. Vastgaussian: Vast 3d gaussians for large scene reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5166–5175, 2024. 4

[9] Saswat Subhajyoti Mallick, Rahul Goel, Bernhard Kerbl, Francisco Vicente Carrasco, Markus Steinberger, and Fernando De La Torre. Taming 3dgs: High-quality radiance fields with limited resources, 2024. 1, 2, 3, 4

[10] Zehao Yu, Anpei Chen, Binbin Huang, Torsten Sattler, and Andreas Geiger. Mip-splatting: Alias-free 3d gaussian splatting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 19447–19456, 2024. 1, 2, 3