SEELE: A Unified Acceleration Framework for Real-Time Gaussian Splatting on Mobile Devices

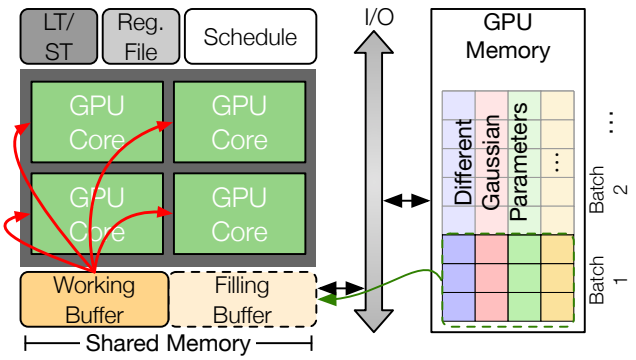

Fig. 1. A toy example of applying double buffering with a single streaming multiprocessor (SM). The shared memory of this SM is split into a working buffer and a filling buffer. The working buffer provides the data for processing the current batch, while the filling buffer loads the data from GPU memory for the next batch.

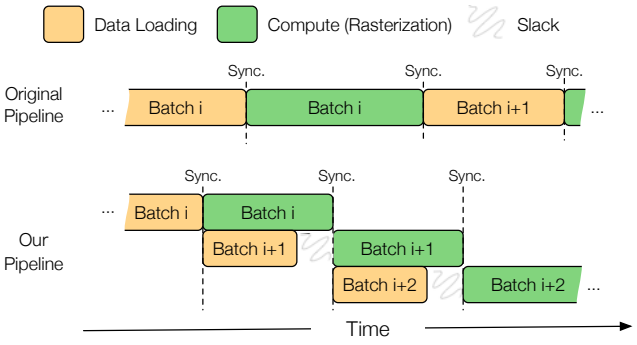

Fig. 2. The comparison between the original pipeline and our fine-grained pipelining. Our pipeline leverages the double buffering in shared memory and avoids the extra synchronization between data loading and computation. In our pipeline, we only need to synchronize between two compute batches.

A . Methodology

In this appendix, we will explain more detailed designs in our framework. Specifically, we introduce our *double buffering* technique during rasterization in Sec. A .1. We then show that the significance of each Gaussian contribution is strongly correlated to the image frequency in Sec. A .2. Lastly, we give an analytic explanation of why our contribution-aware rasterization only introduces minimal quality loss in Sec. A .3.

A .1. Additional GPU Optimization

In our *contribution-aware rasterization*, we further introduce a fine-grained pipelining that overlaps computation with data fetching from GPU memory to shared memory.

Double Buffering. Fig. 1 illustrates a simplified GPU design with our double buffering strategy. Specifically, it shows a streaming multiprocessor (SM), in Nvidia's terminology, which consists of multiple compute cores. Each SM includes several GPU cores and a shared memory, a relatively small on-chip buffer (32-64 KB) for faster data access. Compared to accessing GPU memory, reading data from shared memory is often much faster. Note that, we intentionally ignore other buffers, e.g., L2 caches, in the GPU memory hierarchy to keep the illustration simple.

Our double buffering strategy splits the shared memory into two buffers: a working buffer and a filling buffer. The filling buffer preloads data for the next batch of computations, while the working buffer stores the data required for the current batch. In Fig. 1, the red arrows show that different GPU cores access data from the working buffer for their own computations. Meanwhile, as the green arrow shows, the filling buffer fetches data from GPU memory, preparing for the next batch of computations.

Fine-Grained Pipelining. With our double buffering strategy, the rasterization pipeline achieves a more fine-grained pipelining compared to the conventional rasterization pipeline in canonical Gaussian splatting. Fig. 2 provides a simplified example of rendering frames within a single warp. In the original algorithm, data loading and computing (rasterization) have to execute sequentially due to the warp-level synchronization. The on-chip data need to be synchronized to ensure the rendering's correctness. However, the synchronization after compute batch unnecessary since there is no data dependency between computing the current batch and loading data for the next batch. This unnecessary synchronization is because the entire shared memory is allocated to a single batch of computation, forc-

ing synchronization after each batch.

Thus, our pipeline partitions shared memory, allocating only half for each batch of computation. This allows data to be loaded into the second half while the current batch is being processed, as illustrated in Fig. 2.

Specifically, each warp loads the attributes of two Gaussians, with each thread within a warp loading one byte of the Gaussian attributes. A single GPU block, which consists of 8 warps, effectively loads 16 Gaussian points into shared memory. Here, we intentionally keep the batch size small to achieve finer-grained pipelining. For more details on the implementation, please refer to our code, which will be made available upon acceptance of this work.

A .2. The Correlation between Gaussian Contribution and Pixel Frequency

According to Parseval's theorem, the high-frequency energy at a given point is intrinsically linked to the local gradient. In the context of Gaussian distributions, the samples located in regions with larger gradients correspond to higher-frequency components, while those in smoother regions capture low-frequency information.

Here, we give a detailed mathematical explanation. Given the Gaussian α-computation function $G(d)$,

$$G(d) = oe^{-\frac{1}{2}d^T\Sigma^{-1}d},$$

where d denotes the distance between the pixel $\mathbf{p}$ and the Gaussian centroid x, Σ denotes the covariance matrix, o denotes Gaussian opacity.

The gradient of $G(d)$, $||\nabla G_d||$, can be expressed as,

$$||\nabla G_d|| = G(d)\sqrt{d^T\Sigma^{-2}d}. \tag{1}$$

As shown in Eqn. 1, we can show two points. First, $||\nabla G_d||$ is linearly correlated to the value of $G(d)$. Thus, samples with lower $G(d)$ have lower gradients. Second, $G(d)$ follows a Gaussian distribution. Greater d also indicates lower $G(d)$. Both points suggest that the insignificant Gaussians are more associated with low-frequency features. These insignificant Gaussians are intended to be located in smooth regions. Based on this observation, we can basically identify low contribution Gaussians without the need to compute the α value of this Gaussian pixel by pixel.

A .3. Rasterization Error Analysis

Our *contribution-aware rasterization* organizes every $w \times w$ pixels as groups, and skips color blending of the whole group if the leader pixel's sampled α value is lower than α_θ. Given distance d, the blending error, $R = \frac{G(d+\Delta)}{G(d)}$, can be estimated as,

$$R(d,\Delta) = e^{-\Delta^T\Sigma^{-1}d-\frac{1}{2}\Delta^T\Sigma^{-1}\Delta},$$

where $||\Delta|| = w - 1$. For simplicity, we discard $-\frac{1}{2}\Delta^T\Sigma^{-1}\Delta$ and define the bound, $\hat{R}$:

$$\hat{R}(d,\Delta) = e^{-\Delta^T\Sigma^{-1}d},$$

which has the maximum value when Δ and d are in the opposite direct,

$$\hat{R}_{max} = e^{\sqrt{\Delta^T\Sigma^{-1}\Delta}\sqrt{d^T\Sigma^{-1}d}}.$$

According to the inherent feature of the Gaussian representation and the filtering rule, Δ and d have constraints:

$$\Delta^T\Sigma^{-1}\Delta \le \frac{w-1}{\lambda_{min}}, d^T\Sigma^{-1}d = -2ln\frac{\alpha_\theta}{o},$$

where λ_{min} refers to the minimal eigenvalue of the Gaussian, and we would derive,

$$\hat{R}_{max} = e^{-\frac{(w-1)\sqrt{2(lno-ln\alpha_\theta)}}{\lambda_{min}}}.$$

As we set $w = 2$, $\alpha_\theta = \frac{1}{255}$, for Gaussians with $\lambda_{min} > 2$, the blending of pixels with $\alpha < 5.28\alpha_\theta$ will be skipped, which has a negligible impact on the quality of the rendering.

B . Evaluation

B .1. Performance and Quality

Tbl. 1 presents the overall evaluation of SEELE on three widely-used 3DGS pipelines: 3DGS [5], MiniSplatting [4], and LightGaussian [3]. Here, we show the quality metrics on individual scenes. The per-scene results are consistent with the overall results. In terms of the quality metrics, SEELE generally outperforms the corresponding baselines with few exceptions. In terms of the performance metrics, SEELE outperforms the baselines with large margins. Fig. 3 and Fig. 4 show the examples of the qualitative comparison between SEELE and its corresponding baselines. Visually, there is no quality difference between SEELE and the corresponding baselines.

B .2. Ablation Study

We further show the contributions of individual optimizations on the other two algorithms: MiniSplatting [4] and LightGaussian [3]. Four variants are evaluated here:

- +Opti.: this variant only includes the additional code optimizations proposed.
- +Opti.+HP: this variant includes both the code optimizations and *hybrid preprocessing*.
- +Opti.+CR: this variant includes code optimizations and *contribution-aware rasterization*.
- SEELE: this variant is the full-fledged algorithm including all optimizations proposed in this paper.

Table 1. Quantitative evaluation of our method against the recent works: 3DGS [5], MiniSplatting [4], and LightGaussian [3]. The **green bold** results highlight the better results between ours, SEELE, and the corresponding baselines.

Metrics	Method	Mip-NeRF360									Tanks&Temples		Deep Blending	
		Bicycle	Bonsai	Counter	Flowers	Garden	Kitchen	Room	Stump	Treehill	Train	Truck	Playroom	Drjohnson
PSNR↑	3DGS [5]	**25.11**	**32.20**	29.08	21.44	27.38	31.24	31.56	**26.70**	22.46	22.06	25.44	29.92	29.26
	SEELE + 3DGS	25.02	33.15	**29.60**	**21.50**	**27.45**	**31.67**	**32.04**	26.57	**22.49**	**22.39**	**25.65**	**30.11**	**29.48**
	MiniSplatting [4]	24.86	31.86	28.74	21.16	26.18	31.38	31.47	26.78	22.63	21.37	25.00	30.42	29.66
	SEELE + MiniSplatting	**25.21**	**32.86**	**29.31**	**21.38**	**26.96**	**32.11**	**31.69**	**26.97**	**22.79**	**22.02**	**25.46**	**30.37**	**29.67**
	LightGaussian [3]	**25.13**	32.20	28.98	21.41	27.01	31.14	31.87	**26.66**	22.53	22.16	25.47	30.03	**29.44**
	SEELE + LightGaussian	24.94	**32.72**	**29.32**	**21.44**	**27.06**	**31.43**	**32.03**	26.52	**22.54**	**22.30**	**25.52**	**30.03**	29.43
	AdR-Gaussian [3]	**24.58**	31.41	28.73	20.34	26.81	30.91	31.27	**25.84**	22.38	21.78	25.07	**30.01**	29.35
	SEELE + AdR-Gaussian	24.57	**32.18**	**29.23**	**20.41**	**26.96**	**31.38**	**31.68**	25.79	**22.53**	**22.20**	**25.36**	29.68	**29.59**
SSIM↑	3DGS [5]	0.745	0.945	0.915	0.589	0.857	0.931	0.927	**0.768**	0.634	0.817	0.879	0.902	0.902
	SEELE + 3DGS	**0.745**	**0.950**	**0.919**	**0.593**	**0.858**	**0.934**	**0.929**	0.765	**0.634**	**0.821**	**0.882**	**0.903**	**0.903**
	MiniSplatting [4]	0.750	0.948	0.913	0.592	0.827	0.932	0.930	0.787	0.649	0.803	0.872	0.908	0.908
	SEELE + MiniSplatting	**0.763**	**0.953**	**0.920**	**0.607**	**0.842**	**0.937**	**0.932**	**0.795**	**0.655**	**0.815**	**0.878**	**0.908**	**0.908**
	LightGaussian [3]	0.743	0.941	0.907	0.585	0.840	0.926	0.925	**0.769**	0.630	0.806	0.878	0.902	**0.899**
	SEELE + LightGaussian	**0.743**	**0.945**	**0.911**	**0.590**	**0.843**	**0.929**	**0.926**	0.767	**0.633**	**0.811**	**0.880**	**0.902**	0.898
	AdR-Gaussian [3]	0.705	0.937	0.906	0.519	0.830	0.926	0.918	0.729	0.608	0.796	0.874	**0.903**	0.899
	SEELE + AdR-Gaussian	**0.707**	**0.942**	**0.911**	**0.526**	**0.835**	**0.930**	**0.921**	**0.729**	**0.613**	**0.801**	**0.874**	0.898	**0.899**
LPIPS↓	3DGS [5]	0.245	0.181	0.183	0.359	0.124	0.118	0.197	0.245	0.347	0.203	0.148	0.247	0.241
	SEELE + 3DGS	**0.238**	**0.176**	**0.177**	**0.353**	**0.118**	**0.113**	**0.193**	**0.241**	**0.343**	**0.194**	**0.141**	**0.243**	**0.237**
	MiniSplatting [4]	0.256	0.166	0.176	0.353	0.171	0.115	0.184	0.237	0.344	0.223	0.150	0.245	0.240
	SEELE + MiniSplatting	**0.240**	**0.161**	**0.168**	**0.340**	**0.155**	**0.109**	**0.181**	**0.223**	**0.332**	**0.214**	**0.144**	**0.245**	**0.239**
	LightGaussian [3]	0.255	0.192	0.201	0.372	0.148	0.130	0.204	0.247	0.367	0.227	0.150	0.251	0.249
	SEELE + LightGaussian	**0.248**	**0.186**	**0.194**	**0.366**	**0.142**	**0.125**	**0.200**	**0.243**	**0.358**	**0.219**	**0.143**	**0.250**	**0.248**
	AdR-Gaussian [3]	0.312	0.201	0.203	0.438	0.171	0.129	0.217	0.311	0.421	0.238	0.165	**0.257**	0.253
	SEELE + AdR-Gaussian	**0.303**	**0.196**	**0.196**	**0.430**	**0.162**	**0.124**	**0.213**	**0.305**	**0.415**	**0.232**	**0.160**	0.258	**0.251**
FPS↑ on Orin	3DGS [5]	12.18	33.93	24.61	27.84	16.57	19.65	22.15	23.00	22.28	43.67	40.39	28.94	20.41
	SEELE + 3DGS	**40.15**	**98.47**	**67.76**	**71.91**	**46.83**	**54.63**	**86.06**	**45.00**	**68.86**	**139.28**	**118.08**	**93.38**	**80.70**
	MiniSplatting [4]	93.41	59.33	45.65	96.50	91.85	50.32	67.71	95.34	96.43	149.66	137.40	133.58	108.93
	SEELE + MiniSplatting	**176.80**	**116.21**	**89.38**	**177.59**	**162.34**	**88.62**	**121.36**	**161.34**	**184.57**	**264.27**	**271.89**	**233.54**	**175.84**
	LightGaussian [3]	18.71	43.64	35.29	41.92	28.43	29.11	29.01	36.43	31.90	74.57	58.57	51.89	36.96
	SEELE + LightGaussian	**61.43**	**102.47**	**81.44**	**88.64**	**62.33**	**69.09**	**75.58**	**80.80**	**82.33**	**209.73**	**163.67**	**143.35**	**120.88**
	AdR-Gaussian [3]	40.99	89.28	70.49	86.90	48.67	61.60	82.03	71.29	82.15	121.90	107.38	92.33	68.83
	SEELE + AdR-Gaussian	**60.27**	**136.28**	**126.72**	**136.78**	**88.97**	**97.82**	**97.32**	**81.82**	**96.02**	**205.02**	**183.80**	**142.93**	**140.82**
FPS↑ on Orin NX	3DGS [5]	1.74	5.03	3.46	4.35	2.53	2.73	3.23	3.59	3.33	6.34	5.75	4.50	3.06
	SEELE + 3DGS	**6.62**	**14.68**	**10.81**	**12.04**	**7.02**	**7.71**	**12.07**	**9.44**	**10.00**	**21.93**	**17.76**	**16.59**	**13.40**
	MiniSplatting [4]	9.32	8.05	6.16	10.01	10.26	6.82	8.88	10.36	10.32	21.68	19.57	18.76	15.28
	SEELE + MiniSplatting	**20.25**	**16.38**	**13.02**	**20.73**	**20.05**	**13.17**	**18.39**	**19.48**	**20.94**	**43.39**	**40.29**	**36.05**	**29.42**
	LightGaussian [3]	3.25	6.60	4.71	5.71	3.93	4.30	3.58	4.91	4.31	10.28	8.04	7.00	4.92
	SEELE + LightGaussian	**10.14**	**18.03**	**13.11**	**15.85**	**10.33**	**11.12**	**13.91**	**12.46**	**13.27**	**31.83**	**24.13**	**22.45**	**18.67**
	AdR-Gaussian [3]	6.21	14.43	11.16	13.49	7.53	9.50	13.15	11.13	13.17	17.33	16.02	15.39	11.92
	SEELE + AdR-Gaussian	**13.26**	**22.72**	**17.10**	**21.17**	**11.90**	**14.31**	**19.31**	**16.94**	**20.46**	**35.42**	**28.91**	**21.62**	**17.89**
FPS↑ on A6000	3DGS [5]	78.47	212.49	147.78	186.60	107.79	123.18	142.96	160.67	148.65	190.77	204.83	193.24	130.62
	SEELE + 3DGS	**217.39**	**499.75**	**401.61**	**398.57**	**233.92**	**289.02**	**449.64**	**336.36**	**353.98**	**573.72**	**504.29**	**556.79**	**480.08**
	MiniSplatting [4]	518.67	348.55	274.80	532.77	508.65	304.32	407.17	546.45	529.94	685.40	324.36	768.64	645.16
	SEELE + MiniSplatting	**809.72**	**558.97**	**464.90**	**798.08**	**720.46**	**478.47**	**640.61**	**725.16**	**824.40**	**1081.08**	**1137.66**	**1124.86**	**952.38**
	LightGaussian [3]	129.25	260.08	193.09	234.69	165.84	179.21	190.95	217.20	168.69	343.64	305.53	285.31	198.85
	SEELE + LightGaussian	**334.78**	**594.53**	**449.03**	**527.70**	**342.35**	**401.61**	**487.57**	**424.27**	**449.64**	**827.81**	**694.44**	**741.29**	**589.28**
	AdR-Gaussian [3]	231.19	537.09	415.47	502.27	280.31	353.58	489.55	414.21	490.33	477.08	436.92	548.20	424.45
	SEELE + AdR-Gaussian	**415.18**	**705.79**	**534.37**	**685.97**	**369.94**	**450.21**	**612.93**	**538.99**	**632.48**	**848.72**	**692.53**	**841.96**	**696.66**
#Inst.(10^6)↓	3DGS [5]	3333.20	1390.33	1948.76	1456.05	2607.93	2446.63	2668.26	1806.44	1860.57	944.96	1123.78	1212.77	2077.46
	SEELE + 3DGS	**1101.41**	**523.39**	**613.71**	**594.67**	**1017.66**	**990.87**	**746.22**	**732.72**	**683.65**	**320.99**	**391.32**	**356.96**	**486.52**
	MiniSplatting [4]	727.18	831.48	913.19	652.21	700.11	1032.53	962.47	679.40	683.06	312.32	347.06	347.31	418.42
	SEELE + MiniSplatting	**399.97**	**459.61**	**482.96**	**359.97**	**392.56**	**565.16**	**492.82**	**403.60**	**371.00**	**169.88**	**174.92**	**205.22**	**243.94**
	LightGaussian [3]	2008.94	1060.64	1291.92	1142.53	1782.13	1630.33	1928.39	1467.85	1488.80	582.87	816.55	851.42	1344.23
	SEELE + LightGaussian	**731.61**	**411.15**	**504.45**	**458.44**	**740.96**	**693.52**	**625.09**	**601.90**	**541.49**	**219.18**	**300.37**	**276.82**	**371.69**
	AdR-Gaussian [3]	1275.77	581.57	667.32	663.19	1209.57	1120.36	866.07	825.80	757.74	402.00	483.97	407.35	562.65
	SEELE + AdR-Gaussian	**399.27**	**241.16**	**271.10**	**258.96**	**436.29**	**413.08**	**339.27**	**285.61**	**300.61**	**135.21**	**174.53**	**153.81**	**202.51**
Mem.↓ (MB)	3DGS [5]	1320.0	295.7	276.2	772.4	1305.1	410.5	352.0	942.9	720.8	257.5	604.3	543.3	736.6
	SEELE + 3DGS	**706.9**	**145.4**	**166.6**	**382.3**	**728.3**	**295.8**	**204.3**	**433.9**	**364.2**	**135.5**	**279.6**	**387.0**	**414.3**
	MiniSplatting [4]	157.6	147.0	130.7	142.4	176.0	142.3	121.5	157.8	136.7	85.0	81.5	117.4	192.3
	SEELE + MiniSplatting	**107.8**	**91.7**	**106.9**	**95.7**	**126.1**	**122.4**	**98.0**	**114.8**	**95.4**	**62.6**	**57.7**	**104.9**	**154.3**
	LightGaussian [3]	106.7	24.1	23.8	64.2	94.4	34.6	28.3	89.4	68.7	23.5	44.2	38.9	66.0
	SEELE + LightGaussian	**65.3**	**15.0**	**19.4**	**37.2**	**66.3**	**30.6**	**21.9**	**55.3**	**41.2**	**14.7**	**26.6**	**33.6**	**49.2**
	AdR-Gaussian [3]	539.8	126.2	130.0	269.9	554.3	180.7	141.3	419.2	236.2	114.5	291.4	254.0	407.0
	SEELE + AdR-Gaussian	**276.1**	**71.8**	**54.6**	**157.8**	**268.5**	**56.5**	**67.2**	**239.2**	**136.7**	**49.5**	**162.9**	**84.7**	**170.8**

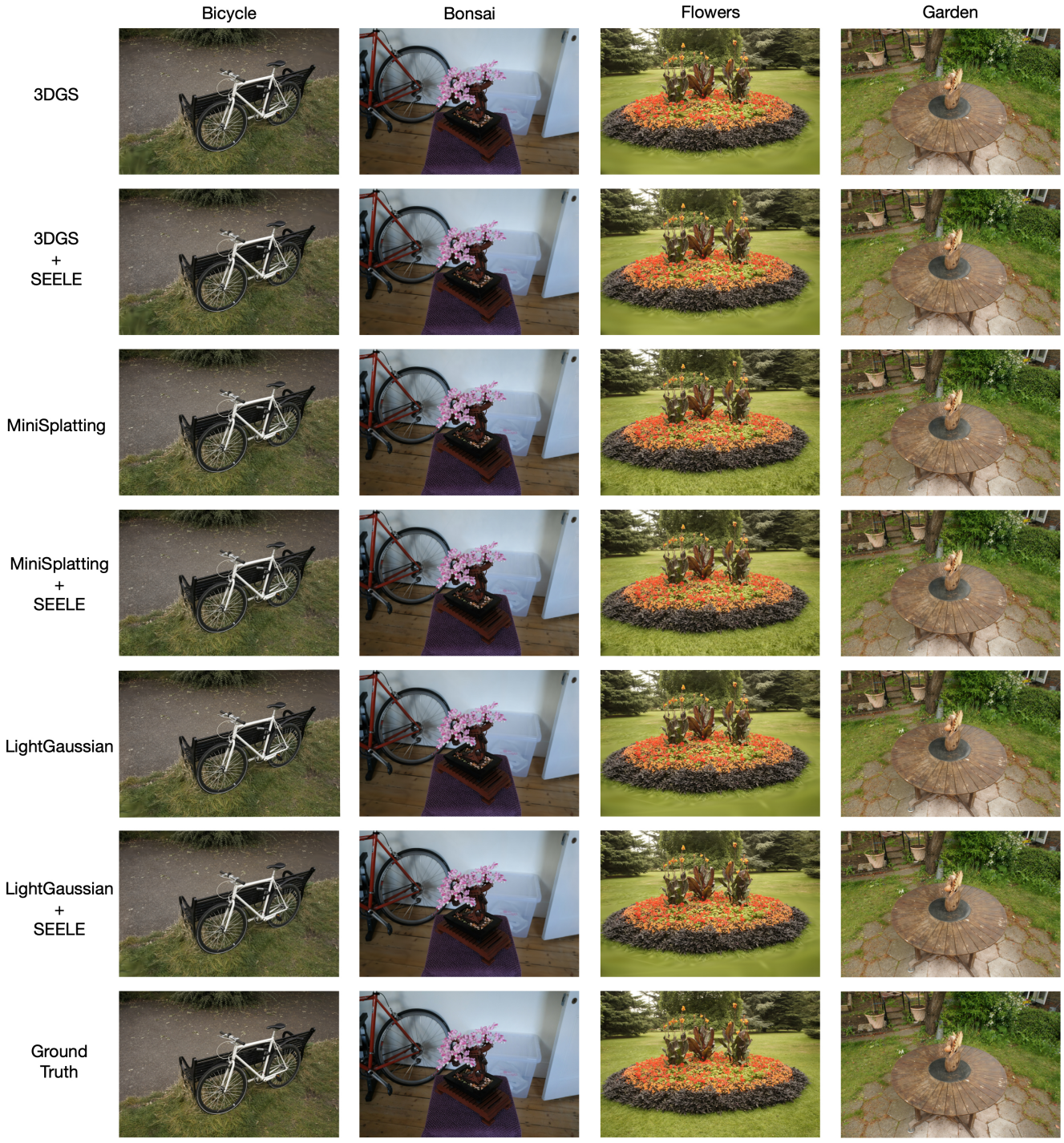

Fig. 3. Example figures of the overall comparison between SEELE and its corresponding baselines.

Tbl. 2 presents the results of our ablation study on MiniSplatting [4] and LightGaussian [3]. In terms of the rendering quality, the additional code optimizations do not change the accuracy. HP and CR improve the rendering quality modestly.

Performance-wise, all optimizations we proposed achieve consistent speedup across all models and all datasets. Overall, HP can achieve over 1.5× speedup. The speedup of CR varies from 1.2 to 1.5. An interesting thing about the overall instruction counts, #Inst., is that SEELE

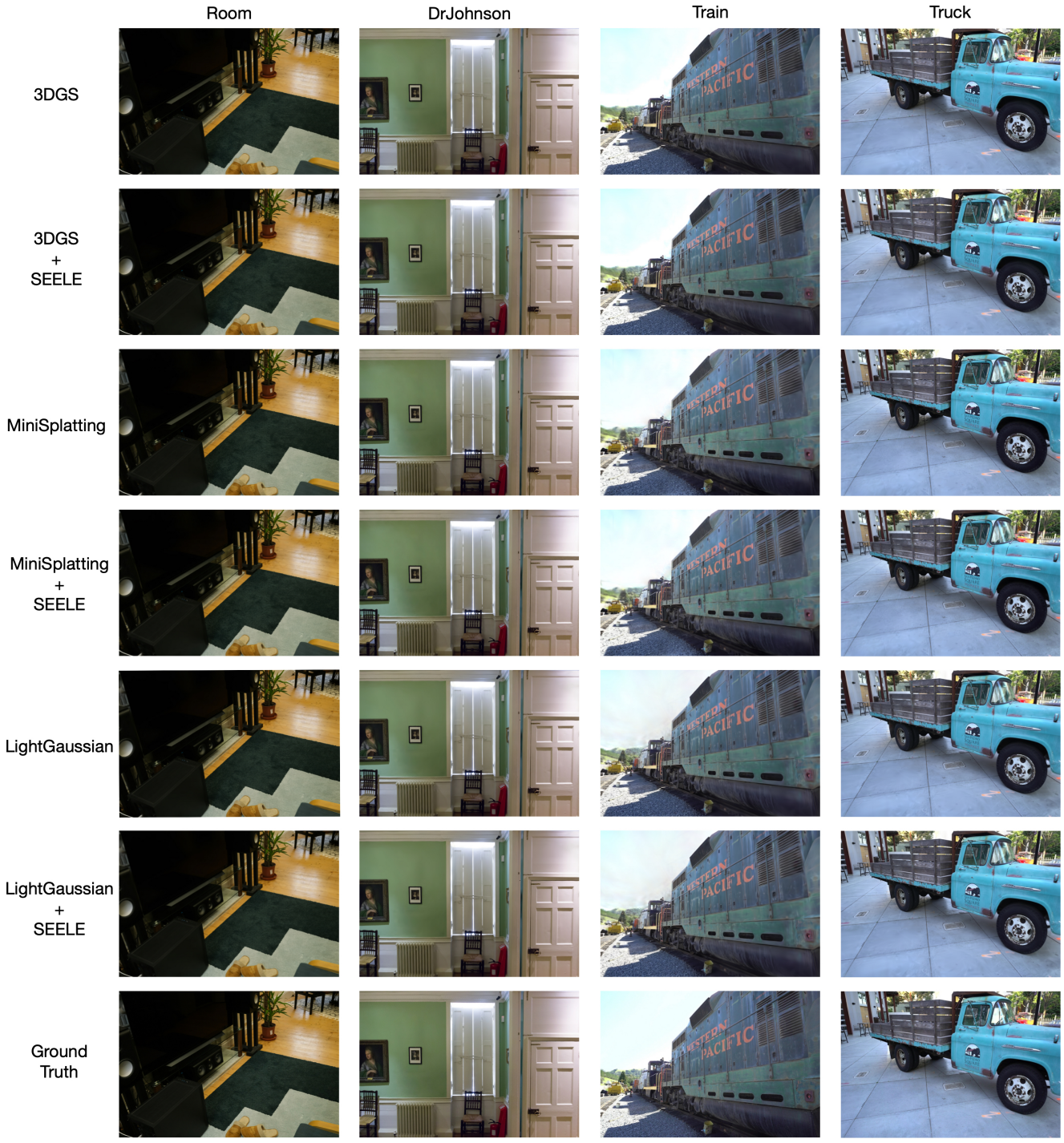

Fig. 4. More example figures of the overall comparison between SEELE and its corresponding baselines.

sometimes has higher instruction counts than +Opti.+HP, while SEELE is consistently much faster than +Opti.+HP. The reason is that SEELE may increase the overall instruction count but improves the overall speedup due to the reductions on warp divergence, i.e., more instructions are executed in parallel.

B .3. Sensitivity Study

Number of Clusters. Fig. 5 shows example results from our sensitivity study on rendering quality and performance

Table 2. The ablation study of MiniSplatting and LightGaussian across all three datasets. +Opti. refers to the code optimization, +HP represents the *hybrid preprocessing*, +CR represents the *contribution-aware rasterization*, and SEELE is our full-fledged algorithm. The **green bold** results highlight the best results.

Dataset	Mip-NeRF360						Tanks&Temples						Deep Blending					
Metrics	Quality			Efficiency			Quality			Efficiency			Quality			Efficiency		
	PSNR↑	SSIM↑	LPIPS↓	FPS↑	#Inst.(10^6)↓	Mem.(MB)↓	PSNR↑	SSIM↑	LPIPS↓	FPS↑	#Inst.(10^6)↓	Mem.(MB)↓	PSNR↑	SSIM↑	LPIPS↓	FPS↑	#Inst.(10^6)↓	Mem.(MB)↓
MiniSplatting	27.23	0.814	0.222	71.31	797.96	145.7	23.18	0.837	0.187	143.27	329.69	83.3	30.04	0.908	0.243	120.01	382.86	154.9
MiniSplatting+Opti.	27.24	0.814	0.223	83.16	669.35	145.7	23.19	0.837	0.188	162.29	277.99	83.3	30.03	0.908	0.243	146.24	302.23	154.9
MiniSplatting+Opti.+HP	27.70	0.823	0.213	102.95	**421.39**	106.5	23.75	0.846	0.181	210.31	**169.92**	60.1	30.02	0.908	0.242	155.67	**197.19**	129.6
MiniSplatting+Opti.+CR	27.22	0.814	0.222	100.14	608.65	145.7	23.18	0.837	0.186	192.47	257.53	83.3	30.04	0.908	0.243	169.15	295.35	154.9
SEELE + MiniSplatting	**27.70**	**0.822**	**0.212**	**131.62**	436.41	**106.5**	**23.74**	**0.846**	**0.179**	**268.03**	172.40	**60.1**	**30.02**	**0.908**	**0.242**	**200.62**	224.58	**129.6**
LightGaussian	27.44	0.807	0.235	30.89	1533.50	59.4	23.82	0.842	0.189	65.60	699.71	33.8	29.74	0.901	0.250	43.17	1097.82	52.4
LightGaussian+Opti.	27.42	0.807	0.236	34.48	1365.02	59.4	23.83	0.841	0.190	72.46	620.48	33.8	29.72	0.900	0.251	48.41	980.20	52.4
LightGaussian+Opti.+HP	27.64	0.811	0.230	70.55	600.20	39.1	24.04	0.847	0.182	172.00	261.86	20.7	29.74	0.901	0.250	126.65	**306.88**	41.4
LightGaussian+Opti.+CR	27.43	0.807	0.235	40.15	1127.65	59.4	23.81	0.842	0.188	85.36	525.97	33.8	29.74	0.901	0.250	58.59	783.34	52.4
SEELE + LightGaussian	**27.56**	**0.810**	**0.229**	**76.36**	**589.85**	**39.1**	**23.91**	**0.846**	**0.181**	**183.86**	**259.78**	**20.7**	**29.73**	**0.900**	**0.249**	**131.16**	324.25	**41.4**

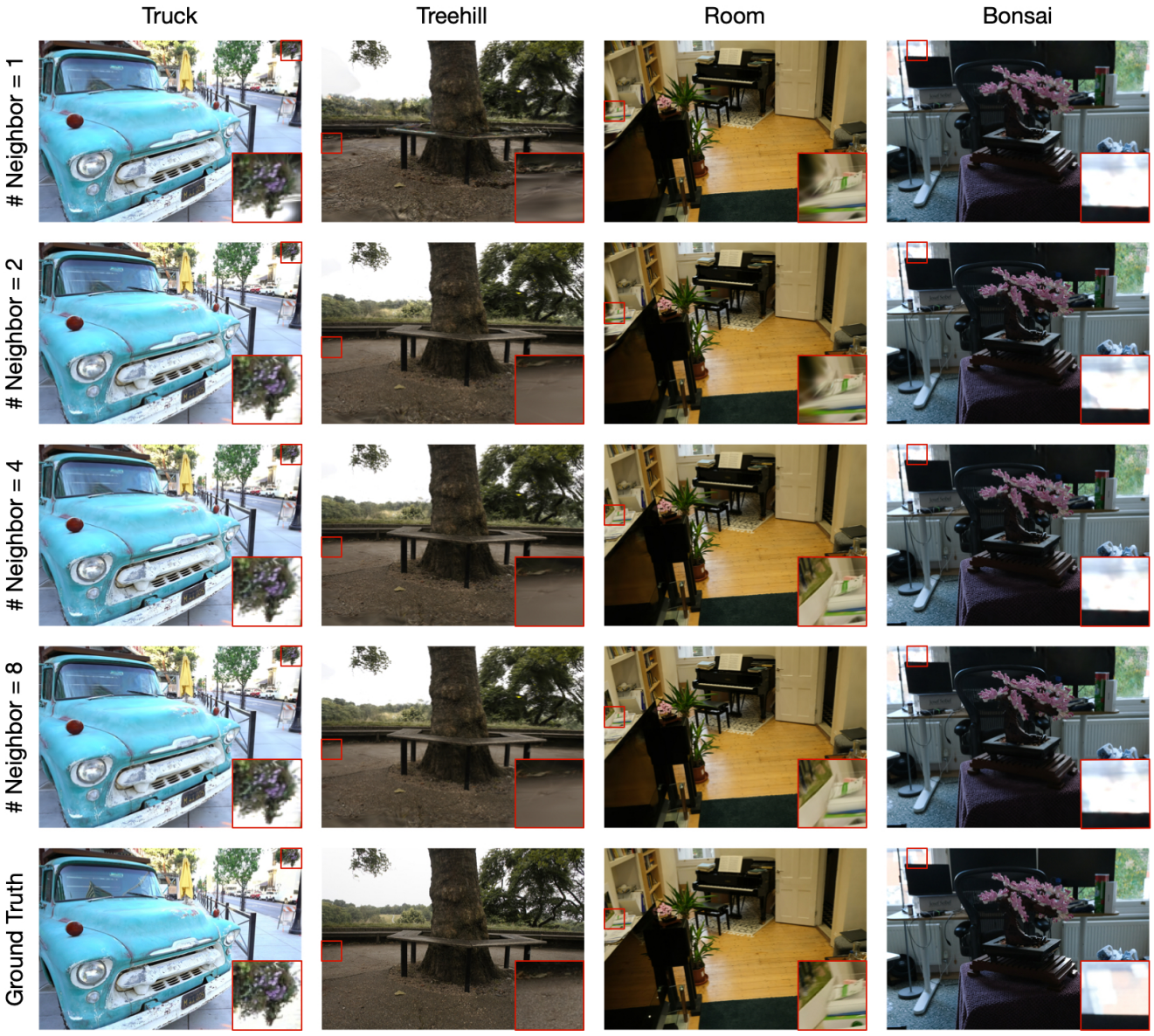

Fig. 5. Example results from our sensitivity study on rendering quality and performance to the number of clusters and cluster neighbors. All results are generated by partitioning the entire scene with 24 clusters while varying the number of neighboring clusters. We find that most artifacts come from uncovered regions, primarily due to an insufficient number of neighboring clusters. See red bounding boxes.

to the number of clusters and cluster neighbors. All results are generated by partitioning the entire scene with 24 clusters while varying the number of neighboring clusters. We find that most artifacts come from uncovered regions, primarily due to an insufficient number of neighboring clusters. Please see details highlighted in red bounding boxes. The lack of neighboring clusters results in black ghosting shadows, as shown in Fig. 5.

It is worth noting that we adopt a universal configuration for all scenes. One potential improvement that could further improve rendering quality is to apply customized settings for each scene, which we leave for future work.

Pixel Group. Fig. 6 presents the qualitative results analyzing the sensitivity of both rendering quality and performance to pixel group size. One potential limitation of this technique is that as the window size increases, artifacts such as aliasing may appear in the rendered images. For instance, when the window size reaches 4×4, noticeable aliasing effects can be observed, as shown on the right side of Fig. 6.

C . Limitations and Discussion

This paper presents SEELE, a unified framework that achieves consistent speedup across GPU platforms. Nevertheless, we acknowledge several limitations of this work:

- *Scene-Specific Configuration in Hybrid Preprocessing*: In this work, we adopt a uniform configuration for clustering in hybrid preprocessing. We believe that scene-specific configurations could further enhance rendering performance and quality. However, manually tuning the number of clusters and neighboring clusters is cumbersome with a human in the loop. An interesting future work would be designing an automatic algorithm to optimize these parameters for each scene.
- *Aliasing in Contribution-Aware Rasterization*: We observe that increasing the window size to 4×4 introduces aliasing artifacts in high-frequency regions when applying *contribution-aware rasterization*. While our current approach balances speedup and quality, further investigation is needed to develop techniques that mitigate aliasing while maintaining efficiency.

Meanwhile, we acknowledge several works that leverage GPU fixed-function units to accelerate alpha blending using GPU RT Core [6], GPU raster [2], or WebGL [1]. Our approach is orthogonal to these methods, and we are excited to explore how our techniques can be integrated with these optimizations to further enhance rendering performance.

Fig. 6. Example results from the sensitivity study of rendering quality and performance to the pixel group size.

References

[1] Webgl 3d gaussian splat viewer. GitHub, 2023. 7

[2] Fast gaussian rasterization. GitHub, 2024. 7

[3] Zhiwen Fan, Kevin Wang, Kairun Wen, Zehao Zhu, Dejia Xu, and Zhangyang Wang. Lightgaussian: Unbounded 3d gaussian compression with 15x reduction and 200+ fps. *arXiv preprint arXiv:2311.17245*, 2023. 2, 3, 4

[4] Guangchi Fang and Bing Wang. Mini-splatting: Representing scenes with a constrained number of gaussians. *arXiv preprint arXiv:2403.14166*, 2024. 2, 3, 4

[5] 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. 2, 3

[6] 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. *ACM Transactions on Graphics (TOG)*, 43(6):1–19, 2024. 7