

Trick-GS: A Balanced Bag of Tricks for Efficient Gaussian Splatting

Anil Armagan, Albert Saà-Garriga, Bruno Manganelli, Mateusz Nowak, and M. Kerim Yucel

Samsung R&D Institute UK (SRUK)

Abstract—Gaussian splatting (GS) for 3D reconstruction has become quite popular due to their fast training, inference speeds and high quality reconstruction. However, GS-based reconstructions generally consist of millions of Gaussians, which makes them hard to use on computationally constrained devices such as smartphones. In this paper, we first propose a principled analysis of advances in efficient GS methods. Then, we propose Trick-GS, which is a careful combination of several strategies including (1) progressive training with resolution, noise and Gaussian scales, (2) learning to prune and mask primitives and SH bands by their significance, and (3) accelerated GS training framework. Trick-GS takes a large step towards resource-constrained GS, where faster run-time, smaller and faster-convergence of models is of paramount concern. Our results on three datasets show that Trick-GS achieves up to 2 \times faster training, 40 \times smaller disk size and 2 \times faster rendering speed compared to vanilla GS, while having comparable accuracy.

3D reconstruction, Gaussian splatting, 3DGS.

I. INTRODUCTION

3D reconstruction is a long standing problem in computer vision, with applications in robotics [1], VR [2] and multimedia [3]. It is a notorious problem since it requires lifting 2D images to 3D space in a dense, accurate manner. Gaussian splatting (GS) [4] is the current state-of-the-art approach for 3D reconstruction, owing to its fast rendering, convergence and high accuracy characteristics. However, it tends to have a large disk size due to the usage of millions of Gaussians, take longer to train compared to hash-grid based NeRF methods [5] and can still benefit from improvements across various axes.

In this paper, with the aim of taking a step towards device friendly GS, we perform a principled analysis of various approaches that produce efficient [6], [7] GS pipelines. These include Gaussian pruning [6], progressive training [8], [9], blurring [8] and spherical harmonic masking [10]. After a careful exploration, we then propose **Trick-GS**, a GS method that is created by systematically combining these approaches. *Trick-GS* has comparable accuracy to vanilla GS and other compact GS methods on multiple benchmark datasets, and enjoys up to 2 \times faster training, 40 \times smaller disk size and 2 \times faster rendering speed compared to other methods. Furthermore, *Trick-GS* is flexible, where one can tune the design to prioritize different aspects of the pipeline, such as convergence and rendering speed, accuracy and disk size.

II. RELATED WORK

Storage efficient GS. Gaussian primitives in GS are represented with 59 attributes. Storing all Gaussians, therefore, might lead to large disk size consumption. There are various methods achieving storage efficient GS, via reducing the number of Gaussians by pruning subject to various criteria [11]–[15] and reducing the number of attributes in each Gaussian by masking/compressing spherical harmonic bands or color parameters [6], [11], [14]. Applying Vector Quantization [6], [11], [16] and bit-quantization [10], [16] have also found use. Using anchors to derive the Gaussians [17]–[20], using self-organizing mapping between 2D grids and Gaussian attributes

[13] or adopting representations such as Octree [21] or tri-planes [22] are other example methods for storage efficient GS.

Fast-converging GS. Most GS methods need fast convergence as they require per-scene training. [23] speeds-up training speed by separating 0th and higher SH bands into different tensors and load into rasterizer separately. [24] introduce ways to accelerate atomic operations prevalent in gradient computations.

Fast-rendering GS. Despite the already fast rendering speed of GS, new GS methods have managed to improve the rendering speed further. [25] removes Gaussians via offline clustering. [26] proposes hardware acceleration units to focus on Gaussian shape-aware intersection tests, hierarchical sorting, and sub-tile skipping.

III. METHODOLOGY

A. Preliminaries

3DGS reconstructs a scene by fitting a collection of 3D Gaussian primitives, which can be efficiently rendered in a differentiable volume splatting rasterizer by extending EWA volume splatting [27]. A scene represented with 3DGS typically consists of hundreds of thousands to millions of Gaussians, where each 3D Gaussian primitive consists of 59 parameters. The 2D Gaussian G'_i projected from a 3D Gaussian G_i is defined as

$$G'_i(x) = e^{-\frac{1}{2}(x - \mu'_i)^T \Sigma'_i^{-1} (x - \mu'_i)}, \quad (1)$$

where x denotes the position vector, μ_i represents 3D position, and Σ_i is the 3D covariance matrix which is later parameterized using scaling matrix S and a rotation matrix that is represented in quaternions q . The symbol ' represents 2D reprojection of corresponding parameters. Each Gaussian primitive has an opacity ($\alpha \in [0, 1]$), a diffused color and a set of spherical harmonics (SH) coefficients, typically consisting of 3-bands, to represent view-dependent colors. Color C of each pixel in the image plane is later determined by N Gaussians contributing to that pixel with blending the colors in a sorted order.

$$C = \sum_{i \in N} c_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j), \quad (2)$$

where c_i and α_i are the view-dependent color and opacity of a Gaussian. 3DGS is trained using a reconstruction loss with L1 and SSIM losses, and Gaussians are pruned and densified based on

Copyright 2025 IEEE. Published in ICASSP 2025 – 2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), scheduled for 6-11 April 2025 in Hyderabad, India. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works, must be obtained from the IEEE. Contact: Manager, Copyrights and Permissions / IEEE Service Center / 445 Hoes Lane / P.O. Box 1331 / Piscataway, NJ 08855-1331, USA. Telephone: + Intl. 908-562-3966.

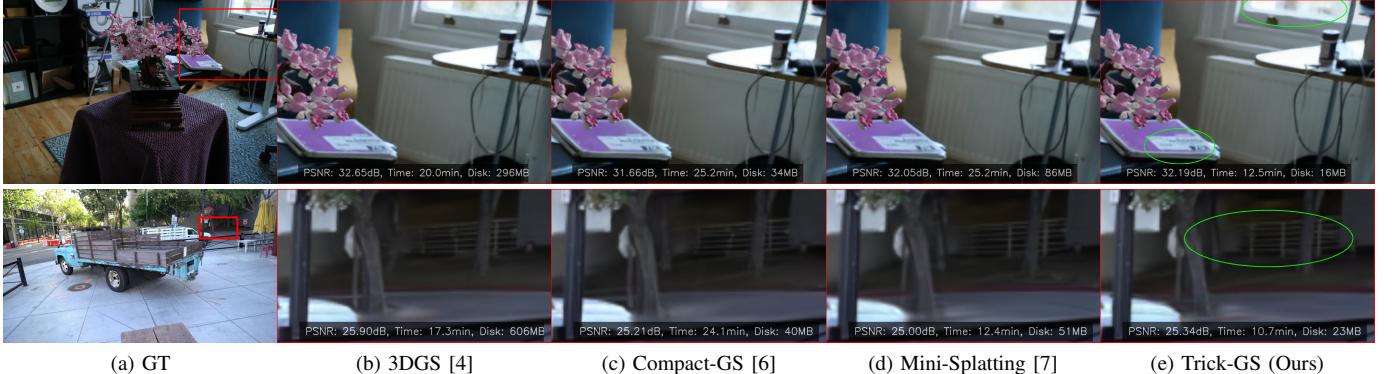


Fig. 1: Qualitative comparison of the methods. Our method can recover more consistent text and background (top), or better structure metallic fence (bottom) while keeping low training time and great compression rates. We show zoomed prediction images (b-e) for the cropped areas with red rectangles as in ground-truth image (a) and green circles for improvements.

their opacities and gradient magnitudes during training. We refer to original paper for more details.

B. Tricks for Learning Efficient 3DGS Representations

In this section we give details of strategies we adopt to obtain a balanced 3DGS representation in terms of disk storage space, training time and rendering time.

1) Pruning with Volume Masking: Gaussians with low scale tend to have minimal impact on the overall quality, therefore we adopt a strategy [13] that learns to mask and remove such Gaussians. In short, N binary masks, $M \in \{0, 1\}^N$, are learned for N Gaussians and applied on its opacity, $\alpha \in [0, 1]^N$, and non-negative scale attributes, $s \in \mathbb{R}_+^{N \times 3}$ by introducing a mask parameter, $m \in \mathbb{R}^N$, for the Gaussian scene representation. The learned masks are then thresholded to generate hard masks M .

$$M_n = \text{sg}(\mathbb{1}(\sigma(m_n) > \epsilon) - \sigma(m_n)) + \sigma(m_n), \quad (3)$$

where ϵ is the masking threshold, sg is the stop gradient operator, and $\mathbb{1}(\cdot)$ and $\sigma(\cdot)$ are indicator and sigmoid function, respectively for the n^{th} Gaussian. In training, these Gaussians are pruned at densification stage using these masks and are also pruned at every k_m iteration after densification stage.

2) Pruning with Significance of Gaussians: In parallel to scale and opacity based pruning, another axis is to find and remove Gaussians with little overall impact. For further pruning, we adopt a strategy [11] where the impact of a Gaussian is determined by considering how often it is hit by a ray. More concretely, the so called significance score is calculated as $\mathbb{1}(G(X_j), r_i)$ where $\mathbb{1}(\cdot)$ is the indicator function, r_i is ray i for every ray in the training set and X_j is the times Gaussian j hit r_i . This score is then harmonized with the Gaussian's volume and opacity as in Eq. 4.

$$\text{GS}_j = \sum_{i=1}^{MHW} \mathbb{1}(G(X_j), r_i) \cdot \alpha_j \cdot \gamma(\Sigma_j), \quad (4)$$

where j is the Gaussian index, i is the pixel, $\gamma(\Sigma_j)$ is the Gaussian's volume, M , H , and W represents the number of training views, image height, and width, respectively. This volume calculation is then refined by normalizing the volume of a Gaussian by the volume of 90% largest of all sorted Gaussians to avoid excessive floating Gaussians. This significance score is used to prune a preset percentile of Gaussians during training. Since it is a costly operation, we apply this pruning k_{sg} times during training with a decay factor considering the percentile removed in the previous round.

3) Spherical Harmonic (SH) Masking.: SHs are used to represent view dependent color for a Gaussian, however, one can appreciate that not all Gaussians will have the same levels of varying colors depending on the scene, which provides a further pruning opportunity. We adopt a strategy [10] where SH bands are pruned based on a mask learned during training, and unnecessary bands are removed after the training is complete. Specifically, each Gaussian learns a mask per SH band. SH masks are calculated as in Eq. 5 and SH values for the i^{th} Gaussian for the corresponding SH band, l , is set to zero if its hard mask, $M_{sh_i}^l$, value is zero and unchanged otherwise.

$$M_{sh_i}^l = \text{sg}(\mathbb{1}(\sigma(m_{sh_i}^l) > \epsilon_{sh}) - \sigma(m_{sh_i}^l)) + \sigma(m_{sh_i}^l), \quad (5)$$

where $m_i^l \in (0, 1)$, $M_{sh_i}^l \in \{0, 1\}$. Finally, each masked view-dependent color is defined as $\hat{\mathbf{c}}_i^l = M_{sh_i}^l \mathbf{c}_i^l$ where $\mathbf{c}_i^l \in \mathbb{R}^{(2l+1) \times 3}$. Masking loss for each degree of SH is weighted by its number of coefficients, since the number of SH coefficients vary per SH band.

4) Progressive Training.: Progressive training of Gaussians refers to starting from a coarser, less detailed image representation and gradually changing the representation back to the original image. This approach has been proven to work as a regularization scheme [8], [9] for densification and pruning of Gaussians as the initialization of the Gaussians from an *SfM* point-cloud can be sub-optimal. We use three different strategies in this aspect.

a) Progressive training by blurring.: Gaussian blurring is used to change the level of details in an image [8]. Kernel size is progressively lowered at every k_b iteration based on a decay factor. This strategy helps to remove floating artifacts from the sub-optimal initialization of Gaussians and serves as a regularization to converge to a better local minima. It also significantly impacts the training time since a coarser scene representation requires less number of Gaussians to represent the scene.

b) Progressive training by resolution.: Another strategy to mitigate the over-fitting on training data is to start with smaller images and progressively increase the image resolution during training to help learning a broader global information [8], [9]. This approach specifically helps to learn finer grained details for pixels behind the foreground objects. Resolution of an image during training calculated as in Eq. 6.

$$\text{res}(i) = \min([\text{res}_s + (\text{res}_e - \text{res}_s) \cdot i/\tau_{\text{res}}], \text{res}_e), \quad (6)$$

where $\text{res}(i)$ is the image resolution at the i^{th} training iteration, res_s is the starting image resolution, res_e is the full training image resolution, and τ_{res} is the threshold iteration.

Dataset	Mip-NeRF 360										Tanks&Temples						
	PSNR ↑ SSIM ↑ LPIPS ↓	Storage ↓	Time ↓	FPS ↑	Max-#GS ↓ ×1000	#GS ↓ ×1000	PSNR ↑ SSIM ↑ LPIPS ↓	Storage ↓	Time ↓	FPS ↑	Max-#GS ↓ ×1000	#GS ↓ ×1000					
*3DGS [4]	27.21	0.815	0.214	734	41.52	134	-	-	23.14	0.841	0.183	411	26.90	154	-	-	
*INGP-base [5]	25.30	0.671	0.371	13	5.62	12	-	-	21.72	0.723	0.330	13	5.43	17	-	-	
*INGP-big [5]	25.59	0.699	0.331	48	7.50	9	-	-	21.92	0.745	0.305	48	6.98	14	-	-	
*Mini-Splatting [7]	27.34	0.822	0.217	-	-	-	490	23.18	0.835	0.202	-	-	-	-	200		
*Compact-GS [6]	27.08	0.798	0.247	49	33.10	128	-	-	23.32	0.831	0.201	39.4	18.30	185	-	-	
3DGS [4]	27.56	0.818	0.202	770	23.83	142	3265	3255	23.67	0.845	0.178	431	14.69	172	1824	1824	
Mini-Splatting [7]	27.25	0.820	0.219	118	21.92	364	4235	496	23.23	0.833	0.202	48	12.34	452	4291	203	
Compact-GS [6]	26.95	0.797	0.245	47	41.74	128	2567	1406	23.29	0.830	0.202	38	21.15	192	1467	834	
Trick-GS-small	26.714	0.780	0.285	19	12.42	256	1033	392	23.21	0.818	0.232	11	8.79	342	570	220	
Trick-GS	27.16	0.802	0.245	39	15.41	222	1369	830	23.48	0.830	0.209	20	10.56	298	696	443	

TABLE I: Quantitative evaluation on MipNeRF 360 and Tanks&Temples datasets. Results with marked ** method names are taken from the corresponding papers. Results between the double horizontal lines are from retraining the models on our system. We color the results with 1st, 2nd and 3rd rankings in the order of solid to transparent colored cells for each column. Trick-GS can reconstruct scenes with much lower training time and disk space requirements while not sacrificing on the accuracy.

c) **Progressive training by scales of Gaussians.**: Another strategy is to focus on low-frequency details during the early stages of the training by controlling the low-pass filter in the rasterization stage. Some Gaussians might become smaller than a single pixel if each 3D Gaussian is projected as Eq. 1, which results in artifacts. Therefore, the covariance Σ'_i is replaced with $\Sigma'_i + sI$ by adding a small value to the diagonal element [4] where $s = 0.3$ and I is an identity matrix.

[28] progressively changes s for each Gaussian during the optimization to ensure the minimum area that each Gaussian covers in the screen space. Using a larger s in the beginning of the optimization enables Gaussians receive gradients from a wider area and therefore the coarse structure of the scene is learned efficiently. The value s ensures the projected Gaussians area to be larger than $9\pi s$, please refer to [28] for a proof, and s is defined as $s = HW/9\pi N$, where N , H and W indicate the number of Gaussians, image height and width, respectively. This strategy is lower bounded with a default value of $s = 0.3$ as the number of Gaussians increases. We start the optimization with the 20% of the 3D points from the *SfM* initialization that has the lowest error to help for a more efficient training.

5) **Accelerated training for 3DGS**: We adopt a strategy [23] that is more focused on the training time efficiency. We follow [23] on separating higher SH bands from the 0th band within the rasterization, thus lower the number of updates for the higher SH bands than the diffused colors. SH bands (45 dims) cover a large proportion of the these updates, where they are only used to represent view-dependent color variations. [23] modifies the rasterizer to split SH bands from the diffused color. SH bands are updated every 16 iterations where as diffused color is updated at every step. To make the training faster we further modify the *SSIM* loss calculation [23] with optimized CUDA kernels. *SSIM* is configured to use 11×11 Gaussian kernel convolutions by standard, where as an optimized version is obtained by replacing the larger 2D kernel with two smaller 1D Gaussian kernels. *SSIM* metric is calculated with a fused kernel from the output of convolutions. Applying less number of updates for higher SH bands and optimizing the *SSIM* loss calculation have a negligible impact on the accuracy, while it significantly helps to speed-up the training time as shown by our experiments.

Finally, we can define our loss as

$$\mathcal{L} = (1 - \lambda_{SSIM})\mathcal{L}_1 + \lambda_{SSIM}\mathcal{L}_{D-SSIM} + \lambda_m\mathcal{L}_m + \lambda_{sh}\mathcal{L}_{sh} \quad (7)$$

since only Gaussian and SH masking are learnable. Please see the corresponding papers [11], [13] for the definitions of L_m and L_{sh} .

Dataset	Deep Blending												
	Method	PSNR ↑ SSIM ↑ LPIPS ↓	Storage ↓	Time ↓	FPS ↑	Max-#GS ↓ ×1000	#GS ↓ ×1000	PSNR ↑ SSIM ↑ LPIPS ↓	Storage ↓	Time ↓	FPS ↑	Max-#GS ↓ ×1000	#GS ↓ ×1000
*3DGS [4]	29.41	0.903	0.243	676	36.03	137	-	-	-	-	-	-	-
INGP-base [5]	23.62	0.797	0.423	13	6.52	3	-	-	-	-	-	-	-
*INGP-big [5]	24.96	0.817	0.390	48	8.00	3	-	-	-	-	-	-	-
*Mini-Splatting [7]	29.98	0.908	0.253	-	-	-	-	-	-	-	-	350	
*Compact-GS [6]	29.79	0.901	0.258	43	27.55	181	-	-	-	-	-	-	-
3DGS [4]	29.46	0.899	0.266	664	25.29	121	2808	2808	-	-	-	-	-
Mini-Splatting [7]	29.78	0.906	0.251	80	16.53	367	4521	339	-	-	-	-	-
Compact-GS [6]	29.78	0.901	0.258	41	37.59	182	2273	1043	-	-	-	-	-
Trick-GS-small	29.63	0.899	0.266	15	11.67	287	1014	363	-	-	-	-	-
Trick-GS	29.46	0.899	0.260	25	13.11	260	1308	639	-	-	-	-	-

TABLE II: Quantitative evaluation on Deep Blending dataset. Results with marked * method names are taken from the corresponding papers. Results below the double horizontal line are trained and evaluated on our system.

IV. EXPERIMENTAL RESULTS

A. Experimental Setup

We follow the setup of [4] on real-world scenes. 15 scenes from bounded and unbounded indoor/outdoor scenarios; nine from Mip-NeRF 360 [29], two (*truck* and *train*) from Tanks&Temples [30] and two (*DrJohnson* and *Playroom*) from Deep Blending [31] datasets are used. *SfM* points and camera poses are used as provided by the authors [4] and every 8th image in each dataset is used for testing.

Similar to [4], models are trained for 30K iterations and PSNR, SSIM and LPIPS [32] are used for evaluation. We report training time using *torch.cuda.Event* function including the time spent for the densification step, and FPS run-time over 50 runs using *torch.utils.benchmark*.

Note that Trick-GS does not change the structure of 3DGS for efficiency, such as anchor points or explicit representations. Instead our method benefit from existing techniques mostly applied as a training strategy. We apply a simple post-processing by storing all Gaussian parameters with 16-bit precision using half-tensors after the training as we found the accuracy drop is insignificant. To make a fair comparison, we evaluate all methods on the same system using an NVIDIA RTX 3090.

B. Implementation Details

We downsample images by 8× and gradually increase it with a logarithmic decay back to the original resolution until the iteration 19500. Similarly, we apply Gaussian blurring to the training images starting with a square kernel of 9×9 and $\sigma = 2.4$ and gradually reduce it for every $k_b = 100$ iterations until the iteration 19500.

Trick	MipNeRF 360 - Bicycle														
	BL	DS	SG	GM	AT	SHM	DE	SC	PSNR↑	SSIM↑	LPIPS↓	Storage↓ (MB)	Time↓ (min)	FPS↑	Max-#GS↓
✗ ✗ ✗ ✗ ✗ ✗ ✗ ✗ ✗	25.18	0.76	0.21	1409	33.36	70	5958	5958							
✓ ✗ ✗ ✗ ✗ ✗ ✗ ✗ ✗	24.96	0.73	0.28	382	24.46	85	3398	3398							
✗ ✗ ✗ ✗ ✗ ✗ ✗ ✗ ✗	25.23	0.77	0.21	596	29.47	62	5294	5294							
✗ ✗ ✗ ✗ ✗ ✗ ✗ ✗ ✗	25.15	0.76	0.22	233	25.54	132	4789	2067							
✗ ✗ ✗ ✗ ✗ ✗ ✗ ✗ ✗	24.80	0.74	0.25	273	23.90	103	3462	2431							
✗ ✗ ✗ ✗ ✗ ✗ ✗ ✗ ✗	25.15	0.76	0.21	677	29.7	86	6018	6018							
✓ ✗ ✗ ✗ ✗ ✗ ✗ ✗ ✗	24.89	0.73	0.28	338	18.60	102	3002	3002							
✓ ✗ ✗ ✗ ✗ ✗ ✗ ✗ ✗	24.84	0.72	0.29	128	14.91	173	2638	1138							
✓ ✗ ✗ ✗ ✗ ✗ ✗ ✗ ✗	24.62	0.70	0.31	75	11.78	232	1838	670							
✓ ✗ ✗ ✗ ✗ ✗ ✗ ✗ ✗	24.72	0.71	0.31	35	17.98	223	1931	725							
✓ ✗ ✗ ✗ ✗ ✗ ✗ ✗ ✗	24.75	0.71	0.31	35	15.88	262	1931	725							
✓ ✗ ✗ ✗ ✗ ✗ ✗ ✗ ✗	24.85	0.72	0.29	61	18.08	223	1920	1356							
✓ ✗ ✗ ✗ ✗ ✗ ✗ ✗ ✗	25.07	0.75	0.25	73	20.80	212	2374	1574							

TABLE III: Ablation study on tricks adopted by our approach using ‘bicycle’ scene. Our tricks are abbreviated as BL: progressive Gaussian blurring, DS: progressive downsampling, SG: significance pruning, GM: Gaussian masking, SHM: SH masking, AT: accelerated training, DE: late densification, SC: progressive scaling. Our full model *Trick-GS* uses all the tricks while *Trick-GS-small* uses all but DE and SC.

ABE split and progressive scale strategies [28] are used for 10K iterations. Later standard densification [4] is enabled at every 100 iteration until iteration 15K.

Some of our strategies are pruning Gaussians at lower image scales even after the densification stops. Therefore, we enable the densification back between iterations 20K to 20.5K for every 100 iterations to help the model to recover the loss that is from pruning of false-positive Gaussians. Learning rates 0.5 and 0.05 are used for learning the Gaussian and SH masks. The loss weights and the mask thresholds $\lambda_m = \lambda_{sh} = \epsilon_m = 0.05$ and $\epsilon_{sh} = 0.1$ are chosen. Gaussian masks are used to prune Gaussians at every densification step and for every 500 iteration after the densification stops.

Significance score based Gaussian pruning is applied 6 times until iteration 22K with the first pruning rate set to 60%, where a pruning decay factor of 0.7 is used for the next iterations to reduce the number of pruned gaussians gradually in the later iterations.

Different from the literature, we lower the blurring kernel size before we densify Gaussians. This helps to introduce an extra signal on the gradient of Gaussians that are more erroneous when the noise level is reduced. When downsampling and blurring is activated together, the kernel size is lowered by the scale of downsampling at the current stage, since the initial hyperparameter choice is based on original image resolutions.

C. Performance Evaluation

Our most efficient model *Trick-GS-small* improves over the vanilla 3DGS [4] by compressing the model size drastically, 23×, improving the training time and FPS by 1.7× and 2×, respectively on three datasets. However, this results in slight loss of accuracy, and therefore we use late densification and progressive scale-based training with our most accurate model *Trick-GS*, which is still more efficient than others while not sacrificing on the accuracy.

Trick-GS improves PSNR by 0.2dB on average while losing 50% on the storage space and 15% on the training time compared to *Trick-GS-small*. The reduction on the efficiency with *Trick-GS* is because of the use of progressive scale-based training and late densification that compensates for the loss from pruning of false positive Gaussians.

We tested an existing post-processing step [13] and further reduce the model size as low as 6MB and 12MB respectively for *Trick-GS-small* and *Trick-GS* over MipNeRF 360 dataset. The training time is not heavily impacted by the post-processing but the accuracy drop on *PSNR* metric is 0.33dB which is undesirable for our method. Therefore, we leave the post-processing as a future work.

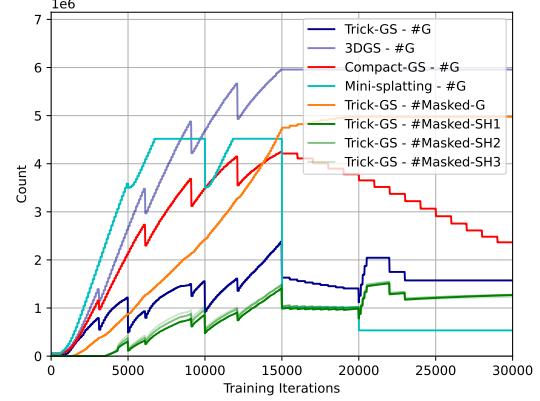


Fig. 2: Number of Gaussians (#G) during training (on MipNeRF 360 - bicycle scene) for all methods, number of masked Gaussians (#Masked-G) and number of Gaussians with a masked SH band for our method. Our method performs a balanced reconstruction in terms of training efficiency by not letting the number of Gaussians increase drastically as other methods during training, which is a desirable property for end devices with low memory.

We compare *Trick-GS* with two recent GS methods, Compact-GS [6] and Mini-Splatting [7]. Tab. I and Tab. II show the comparison of our approach over three datasets. Thanks to our trick choices, *Trick-GS* learns models as low as 10MB for some outdoor scenes while keeping the training time around 10mins for most scenes.

Learning to mask SH bands helps our approach to lower the storage space requirements. *Trick-GS* lowers the storage requirements for each scene over three datasets even though it might result in more Gaussians than Mini-Splatting for some scenes. Our method improves some accuracy metrics over 3DGS while the accuracy changes are negligible. Advantage of our method is the requirement of 1.7× less storage and 4.5× less Gaussians compared to Mini-Splatting and 2× less Gaussians compared to Compact-GS, which is important for

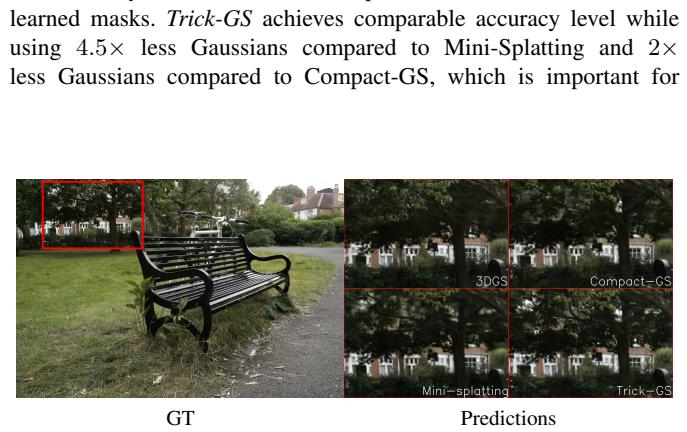


Fig. 3: Impact of progressive training strategies on challenging background reconstructions. We empirically found that progressive training strategies as downsampling, adding Gaussian noise and changing the scale of learned Gaussians have a significant impact on the background objects with holes such as tree branches.

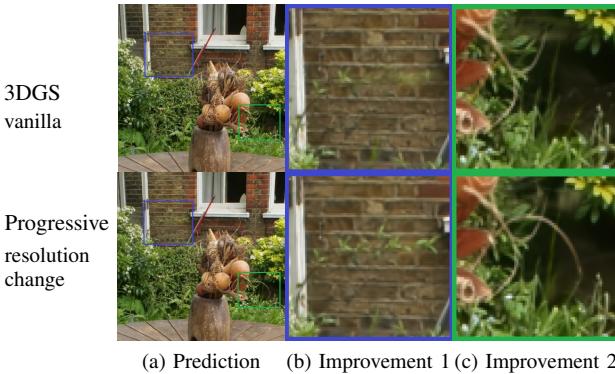


Fig. 4: Visual results (a) from vanilla 3DGS (first row) and a model trained with progressive resolution based strategy (second row) starting with scale 0.125. We use ‘garden’ from MipNeRF360 dataset and zoom into the improvements (b) & (c) for clarity.

the maximum GPU consumption on end devices. Fig. 3 shows the impact of our progressive training strategies. *Trick-GS* obtains structurally more consistent reconstructions of tree branches thanks to the progressive training.

D. Ablation

We evaluate the contribution of tricks in Tab. III on *MipNeRF360 - bicycle* scene. Our tricks mutually benefits from each other to enable on-device learning. While Gaussian blurring helps to prune almost half of the Gaussians compared to 3DGS with a negligible accuracy loss, downsampling the image resolution helps to focus on the details by the progressive training and hence their mixture model lowers the training time and the Gaussian count by half.

Significancy score based pruning strategy improves the storage space the most among other tricks while masking Gaussians strategy results in lower number of Gaussians at its peak and at the end of learning. Enabling progressive Gaussian scale based training also helps to improve the accuracy thanks to having higher number of Gaussians with the introduced split strategy.

We further evaluate, Fig. 5, the impact of changing image resolutions progressively and show that the improvement on the training time is much more than the loss on PSNR metric when when the training is started with a much smaller image resolution than the

original resolution. Fig. 4 shows a visual example of changing the resolution progressively helping to recover tiny structures in the scene.

V. CONCLUSION

We have proposed a mixture of strategies adopted from the literature to obtain compact 3DGS representations. We have carefully designed and chosen strategies from the literature and showed competitive experimental results. Our approach reduces the training time for 3DGS by $1.7\times$, the storage requirement by $23\times$, increases the FPS by $2\times$ while keeping the quality competitive with the baselines. The advantage of our method being easily tunable w.r.t. the application/device needs and it further can be improved with a post-processing stage from the literature e.g. codebook learning, Huffman encoding. We believe a dynamic and compact learning system is needed based on device requirements and therefore leave automatizing such systems for future work.

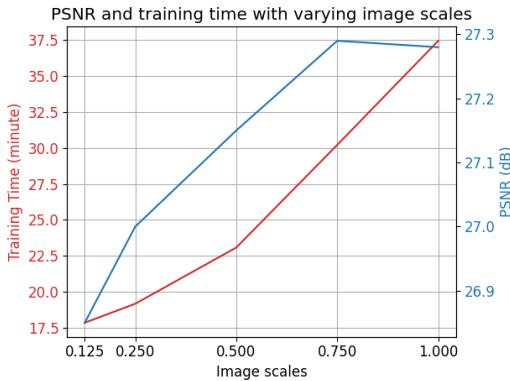


Fig. 5: PSNR and training time evaluations w.r.t. the lowest scale used to start a progressive resolution-based training.

REFERENCES

- [1] Michał Adamkiewicz, Timothy Chen, Adam Caccavale, Rachel Gardner, Preston Culbertson, Jeannette Bohg, and Mac Schwager, “Vision-only robot navigation in a neural radiance world,” *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4606–4613, 2022.
- [2] Sixu Li, Chaojian Li, Wenbo Zhu, Boyang Yu, Yang Zhao, Cheng Wan, Haoran You, Huihong Shi, and Yingyan Lin, “Instant-3d: Instant neural radiance field training towards on-device ar/vr 3d reconstruction,” in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–13.
- [3] Evangelos Skartados, Mehmet Kerim Yucel, Bruno Manganelli, Anastasios Drosou, and Albert Saà-Garriga, “Finding waldo: Towards efficient exploration of nerf scene spaces,” in *Proceedings of the 15th ACM Multimedia Systems Conference*, 2024, pp. 155–165.
- [4] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis, “3d gaussian splatting for real-time radiance field rendering,” *ACM Transactions on Graphics*, vol. 42, no. 4, July 2023.
- [5] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller, “Instant neural graphics primitives with a multiresolution hash encoding,” *ACM Transactions on Graphics*, vol. 41, no. 4, pp. 1–15, 2022.
- [6] Joo Chan Lee, Daniel Rho, Xiangyu Sun, Jong Hwan Ko, and Eunbyung Park, “Compact 3d gaussian representation for radiance field,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 21719–21728.
- [7] Guangchi Fang and Bing Wang, “Mini-splatting: Representing scenes with a constrained number of gaussians,” in *In Proceedings of European Conference on Computer Vision*, 2024.
- [8] Sharath Girish, Kamal Gupta, and Abhinav Shrivastava, “EAGLES: Efficient accelerated 3d gaussians with lightweight encodings,” in *In Proceedings of European Conference on Computer Vision*, 2024.
- [9] Ziyi Yang, Xinyu Gao, Yangtian Sun, Yihua Huang, Xiaoyang Lyu, Wen Zhou, Shaohui Jiao, Xiaojuan Qi, and Xiaogang Jin, “Spec-gaussian: Anisotropic view-dependent appearance for 3d gaussian splatting,” *arXiv preprint arXiv:2402.15870*, 2024.
- [10] Henan Wang, Hanxin Zhu, Tianyu He, Runsen Feng, Jiajun Deng, Jiang Bian, and Zhibo Chen, “End-to-end rate-distortion optimized 3d gaussian representation,” in *In Proceedings of European Conference on Computer Vision*, 2024.
- [11] 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*, 2024.
- [12] Alex Hanson, Allen Tu, Vasu Singla, Mayuka Jayawardhana, Matthias Zwicker, and Tom Goldstein, “Pup 3d-gs: Principled uncertainty pruning for 3d gaussian splatting,” *arXiv preprint arXiv:2406.10219*, 2024.
- [13] Wieland Morgenstern, Florian Barthel, Anna Hilsmann, and Peter Eisert, “Compact 3d scene representation via self-organizing gaussian grids,” in *In Proceedings of European Conference on Computer Vision*, 2024.
- [14] Panagiotis Papantoniakis, Georgios Kopanas, Bernhard Kerbl, Alexandre Lanvin, and George Drettakis, “Reducing the memory footprint of 3d gaussian splatting,” *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, vol. 7, no. 1, pp. 1–17, May 2024.
- [15] Muhammad Salman Ali, Maryam Qamar, Sung-Ho Bae, and Enzo Tartaglione, “Trimming the fat: Efficient compression of 3d gaussian splats through pruning,” *arXiv preprint arXiv:2406.18214*, 2024.
- [16] Simon Niedermayr, Josef Stumpfegger, and Rüdiger Westermann, “Compressed 3D gaussian splatting for accelerated novel view synthesis,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 10349–10358.
- [17] Tao Lu, Mulin Yu, Lining Xu, Yuanbo Xiangli, Limin Wang, Dahua Lin, and Bo Dai, “Scaffold-gs: Structured 3d gaussians for view-adaptive rendering,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 20654–20664.
- [18] Yihang Chen, Qianyi Wu, Jianfei Cai, Mehrtash Harandi, and Weiyao Lin, “Hac: Hash-grid assisted context for 3d gaussian splatting compression,” in *In Proceedings of European Conference on Computer Vision*, 2024.
- [19] Yufei Wang, Zhihao Li, Lanqing Guo, Wenhan Yang, Alex C Kot, and Bihan Wen, “Contextgs: Compact 3d gaussian splatting with anchor level context model,” *arXiv preprint arXiv:2405.20721*, 2024.
- [20] Xiangrui Liu, Xinju Wu, Pingping Zhang, Shiqi Wang, Zhu Li, and Sam Kwong, “CompGS: Efficient 3d scene representation via compressed gaussian splatting,” in *ACM Multimedia 2024*, 2024.
- [21] Kerui Ren, Lihan Jiang, Tao Lu, Mulin Yu, Lining Xu, Zhangkai Ni, and Bo Dai, “Octree-gs: Towards consistent real-time rendering with lod-structured 3d gaussians,” *arXiv preprint arXiv:2403.17898*, 2024.
- [22] Minye Wu and Tinne Tuytelaars, “Implicit gaussian splatting with efficient multi-level tri-plane representation,” *arXiv preprint arXiv:2408.10041*, 2024.
- [23] 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,” *arXiv preprint arXiv:2406.15643*, 2024.
- [24] Sankeerth Durvasula, Adrian Zhao, Fan Chen, Ruofan Liang, Pawan Kumar Sanjaya, and Nandita Vijaykumar, “Distwar: Fast differentiable rendering on raster-based rendering pipelines,” *arXiv preprint arXiv:2401.05345*, 2023.
- [25] Joongho Jo, Hyeongwon Kim, and Jongsun Park, “Identifying unnecessary 3d gaussians using clustering for fast rendering of 3d gaussian splatting,” *arXiv preprint arXiv:2402.13827*, 2024.
- [26] Junseo Lee, Seokwon Lee, Jungi Lee, Junyoung Park, and Jaewoong Sim, “Gscore: Efficient radiance field rendering via architectural support for 3d gaussian splatting,” in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2024, p. 497–511, Association for Computing Machinery.
- [27] Matthias Zwicker, Hanspeter Pfister, Jeroen Van Baar, and Markus Gross, “Ewa volume splatting,” in *Proceedings Visualization*. IEEE, 2001, pp. 29–538.
- [28] Jaewoo Jung, Jisang Han, Honggyu An, Jiwon Kang, Seonghoon Park, and Seungryong Kim, “Relaxing accurate initialization constraint for 3d gaussian splatting,” *arXiv preprint arXiv:2403.09413*, 2024.
- [29] 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*, 2022, pp. 5470–5479.
- [30] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun, “Tanks and temples: Benchmarking large-scale scene reconstruction,” *ACM Transactions on Graphics*, vol. 36, no. 4, 2017.
- [31] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow, “Deep blending for free-viewpoint image-based rendering,” *ACM Transactions on Graphics*, vol. 37, no. 6, pp. 1–15, 2018.
- [32] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang, “The unreasonable effectiveness of deep features as a perceptual metric,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 586–595.