A^{2}TG: ADAPTIVE ANISOTROPIC TEXTURED GAUSSIANS FOR EFFICIENT 3D SCENE REPRESENTATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Gaussian Splatting has emerged as a powerful representation for high-quality, real-time 3D scene rendering. While recent works extend Gaussians with learnable textures to enrich visual appearance, existing approaches allocate a fixed square texture per primitive, leading to inefficient memory usage and limited adaptability to scene variability. In this paper, we introduce **adaptive anisotropic textured Gaussians** (A^{2}TG), a novel representation that generalizes textured Gaussians by equipping each primitive with an anisotropic texture. Our method employs a gradient-guided adaptive rule to jointly determine texture resolution and aspect ratio, enabling non-uniform, detail-aware allocation that aligns with the anisotropic nature of Gaussian splats. This design significantly improves texture efficiency, reducing memory consumption while enhancing image quality. Experiments on multiple benchmark datasets demonstrate that A^{2}TG consistently outperforms fixed-texture Gaussian Splatting methods, achieving comparable rendering fidelity with substantially lower memory requirements.

1 INTRODUCTION

Recent advances in Gaussian Splatting have established it as a powerful paradigm for 3D scene representation, offering both high-fidelity reconstructions and real-time rendering performance. In particular, 3D Gaussian Splatting (3DGS) and its 2D variant, 2D Gaussian Splatting (2DGS), have attracted significant attention due to their ability to combine explicit geometric structure with efficient differentiable rasterization, enabling compelling results across a range of applications in view synthesis, reconstruction, and immersive content creation.

Unlike mesh-based representations, which compactly encode repetitive high-frequency appearance via texture maps, Gaussian splatting typically approximates such detail by instantiating many small primitives, leading to artifacts and inefficient memory usage. Chao et al. (2025) show that attaching a learnable, uniform square texture patch to each Gaussian substantially enhances visual richness and realism. However, this design ignores variability in primitive geometry, and opacity. Some Gaussians require high-frequency textures to capture fine detail, whereas others only need to convey coarse appearance with simple texture. Consequently, fixed-size texture allocations waste memory, inflate storage, and poorly adapt to the anisotropic Gaussians.

In this paper, we propose **adaptive anisotropic textured Gaussians** (A^{2}TG), a novel representation designed to address these limitations. Instead of attaching a uniform square texture to each Gaussian, A^{2}TG assigns an anisotropic texture whose resolution and aspect ratio are jointly determined by a gradient-based adaptive texture control strategy. This adaptive anisotropic texture allows Gaussians to have different patterns and details across the scene. Gaussians covering high-frequency or directionally elongated regions are assigned higher-resolution anisotropic textures, while simpler regions receive more compact representations. However, it is challenging to devise a principle to guide per-Gaussian texture resolution and pixel values. Many 2D Gaussians yield little benefit from textures due to small projected footprints, low opacity, and occlusion. We introduce **gradient-guided adaptive control** to iteratively upscale the textures according to the positional gradient and the geometry of the Gaussians. As a result, A^{2}TG improves memory efficiency while also achieving comparable image quality, as texture parameters are more efficiently allocated.

We evaluate A^{2}TG across diverse datasets and benchmark tasks. Our results show that the proposed approach achieves more effective texture utilization, leading to comparable image quality compared to textured Gaussian Splatting methods with fixed square textures, while substantially reducing memory overhead. These findings suggest that our adaptive anisotropic texture allocation is a promising path toward scalable and efficient textured Gaussian representations.

Our main contributions are summarized as follows.

- We introduce A^{2}TG, a generalization of Textured Gaussians that leverages an adaptive anisotropic texture for each Gaussian.
- We propose a gradient-based adaptive texture control strategy for allocating texture resolution and aspect ratio, enabling non-uniform, detail-aware texture mapping.
- We demonstrate that A^{2}TG improves memory efficiency while maintaining comparable visual fidelity across multiple benchmarks, advancing the state-of-the-art in texture-based Gaussian Splatting.

2 Related Work

2.1 Novel View Synthesis

Novel View Synthesis addresses the challenge of generating realistic renderings of 3D scenes from previously unseen viewpoints, given a set of captured images with known camera poses (Chaurasia et al., 2013; Hedman et al., 2018b; Kopanas et al., 2021). Neural Radiance Fields (NeRF) represent scenes using a multi-layer perceptron (MLP) that models both geometry and view-dependent appearance, trained through volume rendering to achieve high-fidelity image synthesis (Barron et al., 2021; Tancik et al., 2020; Mildenhall et al., 2021). On the other hand, 3D Gaussian Splatting (3DGS) (Kerbl et al., 2023) has emerged as a powerful alternative, capable of delivering real-time novel view synthesis with impressive visual quality. Building on its success, extensions of 3DGS have rapidly appeared across diverse applications. More recently, 2D Gaussian Splatting (2DGS) (Huang et al., 2024) builds upon 3DGS by introducing a "flattened" variant of Gaussian primitives that aligns more naturally with object surfaces, allowing more accurate intersection calculation and opens the door to further applications such as mesh extraction.

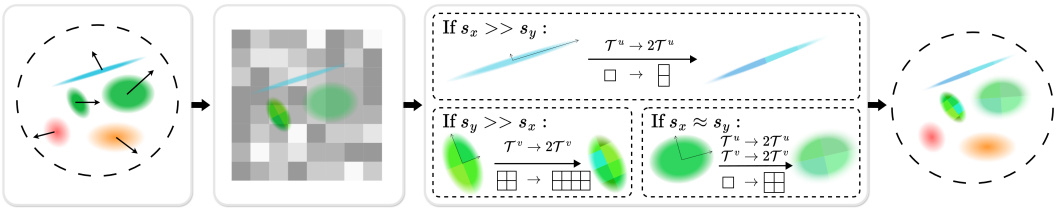
2.2 Textured Gaussian Splatting Representation

The color of each Gaussian primitive is typically parameterized by spherical-harmonic (SH) coefficients, which, given a viewing direction, evaluate to a view-dependent RGB. Consequently, for a fixed view, each primitive yields a single color distributing over its 2D footprint, limiting intra-primitive color detail. Huang & Gong (2024) adds spatially varying appearance to each 3D Gaussian, enabling richer 3DGS representation. Texture-GS (Xu et al., 2024b) maps textures onto 3D Gaussians, but the method is demonstrated primarily on small objects, limiting scalability. Chao et al. (2025) further introduces RGBA texture maps to improve visual quality.

By contrast, 2DGS (Huang et al., 2024) conforms more closely to underlying surfaces and provides a more accurate parameterization for texture mapping, making it better suited for texture integration (Svitov et al., 2024; Xu et al., 2024a; Weiss & Bradley, 2024; Rong et al., 2025; Song et al., 2024). These methods employ predefined RGB or RGBA textures, and (Rong et al., 2025) further adopts world-space texture mapping. However, these methods assign a fixed-size and square texture to every 2D Gaussian. Because Gaussians have varying geometry and opacity, uniform texture sizes are memory-inefficient; tiny or highly transparent Gaussians gain little benefit while inflating storage. Thus, we propose A^{2}TG, which uses adaptive anisotropic textured Gaussians that allocate texture resolution to each primitive according to its footprint and gradient.

2.3 Memory-Efficient Gaussian Splatting Representation

Gaussian primitives offer efficient real-time rendering via learned positions, covariances, color, and opacity; however, 3DGS models are typically storage-intensive due to the large number of primitives and per-primitive attributes. Several 3DGS compression methods address this issue, including



(a) Optimization (b) Gradient-Driven Gaussian Selection (c) Adaptive Texture Upscaling

Figure 1: **Overview of gradient-based adaptive texture control.** Given an initial 2DGS model, (a) our system first optimize the parameters of the 2D Gassians and their textures. (b) Next, we compute the positional gradient of the textured 2D Gassians (as depicted in gray blocks) and select the 2D Gassians that need to increase the resolution of the texture to gain more details. (c) Finally, we adaptively upscale the textures according to the anisotropy of the Gaussians.

vector quantization (Fan et al., 2024; Lee et al., 2024; Niedermayr et al., 2024), and splat-count reduction (Papantonakis et al., 2024; Zhang et al., 2024; Mallick et al., 2024), and context models (Wang et al., 2024) . These approaches can improve memory efficiency, while the results have similar visual fidelity to the original Gaussian models (Bagdasarian et al., 2025). In this work, we target memory-efficient textured Gaussians by adaptively controlling per-Gaussian texture resolution; this direction is orthogonal and complementary to prior compression methods and remains compatible with them.

3 Preliminaries

Textured Gaussian splatting methods naturally build upon **2D Gaussian splatting** (Huang et al., 2024), as the latter provides a convenient framework for defining UV coordinates in the local space of 2D splats, thereby enabling consistent texture sampling.

More concretely, 2D Gaussian splatting represents a 3D scene using a collection of flat 2D Gaussians instead of volumetric 3D Gaussians (Kerbl et al., 2023). Each Gaussian is parameterized as

$$\{\boldsymbol{\mu}_i, \mathbf{s}_i, \mathbf{r}_i, o_i, c_i^{\mathrm{SH}}\},$$

where μ_i denotes its center position, $\mathbf{s}_i$ the 2D scale, $\mathbf{r}_i$ a quaternion rotation, o_i the opacity, and SH_i the spherical harmonics coefficients encoding view-dependent appearance.

For each 2D Gaussian, a local UV coordinate system is defined. The mapping from this local space to the screen space is expressed as

$$\mathbf{x} = \mathbf{WH}(u, v, 1, 1)^\top, \tag{1}$$

where $\mathbf{x} = (x, y, 1, 1)^\top$ is the homogeneous ray passing through pixel (x, y), $(u, v, 1, 1)^\top$ is the corresponding intersection point in the local Gaussian space, $\mathbf{W} \in \mathbb{R}^{4\times4}$ is the world-to-screen transformation matrix, and $\mathbf{H}$ is the local-to-world transformation constructed from the Gaussian's position, rotation, and scale Huang et al. (2024).

To render a set of Gaussians, uv-coordinates of the ray–splat intersection are first computed as $\mathbf{u}(\mathbf{x}) = (u(\mathbf{x}), v(\mathbf{x}))$ by inverting Equation 1. The final color of each pixel is then obtained using front-to-back alpha compositing:

$$\mathbf{c}(\mathbf{x}) = \sum_{i=1} \mathbf{c}_i \, o_i \, \mathcal{G}_i(\mathbf{u}(\mathbf{x})) \prod_{j=1}^{i-1} \big(1 - o_j \, \mathcal{G}_j(\mathbf{u}(\mathbf{x}))\big), \tag{2}$$

where $\mathbf{c}_i$ is the color of the i-th Gaussian and $\mathcal{G}_i$ is its Gaussian kernel evaluated at the uv-coordinates.

4 Methodology

Recent works augment 2D Gaussians with texture maps to improve rendering quality and efficiency. Most assign a fixed, square texture to every Gaussian, regardless of its scale, opacity, or visibility.

This creates redundancy and unnecessary memory usage. We instead assign each 2D Gaussian an individual texture resolution and shape, thereby allocating parameters where they matter most.

Our framework starts with training the given scene by integrating a Markov Chain Monte Carlo (MCMC) densification scheme (Kheradmand et al., 2024) to the 2DGS pipeline (Huang et al., 2024) for the first $30,000$ iterations. With MCMC densification, we can easily control the total number of the Gaussians in the final results. During this pretraining stage, each Gaussian's texture is fixed to 1×1 with texture color $T^{\mathrm{RGB}} = 0$ and texture alpha $T^{A} = 0$ (i.e., without textures). Next, we fix the total number of the 2D Gaussians, set the texture alpha $T^{A} = 1$, and train the 2DGS model with the gradient-based adaptive texture control for another $30,000$ iterations. At each iteration, we first update the parameters of 2D Gaussian and texture (i.e., T^{RGB} and T^{A}), followed by applying *gradient-based adaptive texture control*, which consists of *gradient-driven Gaussian selection* and *adaptive texture upscaling* as illustrated in Figure 1. This procedure adjusts the resolution and aspect ratio of each texture to match the anisotropic footprint of its Gaussian, upscaling where gradients are large. We detail the gradient-based adaptive texture control in the following subsection.

4.1 Anisotropic Textured Gaussians

We first augment each 2D Gaussian a RGB texture T_i^{RGB} and an alpha texture T_i^{A}, then we utilize the local space of the 2D Gaussians for texture mapping with RGBA textures, as the prior works of textured Gaussians (Xu et al., 2024a; Svitov et al., 2024; Chao et al., 2025). After getting the uv-coordinate by taking the inverse of Equation 1, we rescale the range of u and v from $[-1, 1]$ to $[0, \mathcal{T}_i^u]$ and $[0, \mathcal{T}_i^v]$, where $\mathcal{T}_i^u$ and $\mathcal{T}_i^v$ are the texture width and height of the i-th 2D Gaussian, so every texture may have different width and height.

The color contribution of the i-th 2D Gaussian to pixel $\mathbf{x}$ is now calculated by combining the RGB color from the spherical harmonic $\mathbf{c}_{\mathrm{SH}_i}$ and the texture color as

$$c_i(\mathbf{x}) = c_i^{\mathrm{SH}} + T_i^{\mathrm{RGB}}\left(\mathbf{u}(\mathbf{x})\right), \tag{3}$$

where $\mathbf{u}$ maps the pixel $\mathbf{x}$ to the uv-coordinate. The alpha value of the i-th Gaussian is calculated by

$$\alpha_i(\mathbf{x}) = o_i \cdot \mathcal{G}(\mathbf{u}(\mathbf{x})) \cdot T_i^{\mathrm{A}}\left(\mathbf{u}(\mathbf{x})\right), \tag{4}$$

where $\mathcal{G}$ and o_i are the Gaussian distribution function and the opacity of the i-th 2D Gaussian. For querying the texture values $T_i^{\mathrm{RGB}}\left(\mathbf{u}(\mathbf{x})\right)$ and $\cdot T_i^{\mathrm{A}}\left(\mathbf{u}(\mathbf{x})\right)$ from uv coordinates $\mathbf{u}(\mathbf{x})$, we use bilinear interpolation, following (Chao et al., 2025). Finally, the resulting pixel value at pixel $\mathbf{x}$ is calculated on the sorted 2D Gaussians from front to back as

$$\mathbf{c}(\mathbf{x}) = \sum_{i=1} c_i(\mathbf{x})\alpha_i(\mathbf{x}) \prod_{j=1}^{i-1}(1 - \alpha_j(\mathbf{x})). \tag{5}$$

We next utilize the pixel and alpha values to compute the gradient and guide the upscaling of the textures.

4.2 Gradient-Driven Gaussian Selection

It's difficult to design a principle to determine the needs of resolution and pixel values of textures for each 2D Gaussian. First, some 2D Gaussians gain little from textures because their projected footprint covers few pixels in the training views or their opacity is very small. Second, occlusion by foreground splats limits the utility of textures even when a Gaussian's opacity is large. Finally, in regions with homogeneous appearance (e.g., uniformly colored walls), textures provide minimal benefit regardless of Gaussian size or opacity. Therefore, we introduce the gradient-driven Gaussian selection which is similar to the densification control from the 3DGS.

Given a rendered view of the 2DGS and its corresponding training view, let $\mathcal{L}_j$ be the L_1 and SSIM loss between the views at j-th pixel and $\boldsymbol{\mu}_i = (\mu_{i,x}, \mu_{i,y}, \mu_{i,z})^\top$ be the position of the i-th Gaussian. The positional gradient from the j-th pixel coordinate $\mathbf{x}_j$ with respect to the i-th Gaussian is calculated as $\nabla_{\mu_i}\mathcal{L} = \left(\frac{\partial \mathcal{L}_j}{\partial \mu_{i,x}}, \frac{\partial \mathcal{L}_j}{\partial \mu_{i,y}}, \frac{\partial \mathcal{L}_j}{\partial \mu_{i,z}}\right)^\top$. We then derive the $\frac{\partial \mathcal{L}_j}{\partial \mu_{i,x}}$ with the pixel value and

the alpha value of the ith Gaussian as

$$\frac{\partial \mathcal{L}_j}{\partial \mu_{i,x}} = \sum_{k=1}^{3} \frac{\partial \mathcal{L}_j}{\partial \mathbf{c}^k(\mathbf{x}_j)} \cdot \frac{\partial \mathbf{c}^k(\mathbf{x}_j)}{\partial \alpha_i(\mathbf{x}_j)} \cdot \frac{\partial \alpha_i(\mathbf{x}_j)}{\partial \mu_{i,x}}. \tag{6}$$

We distinguish pixel colors and Gaussian colors by using superscripts and subscripts, respectively: $\mathbf{c}^k(\mathbf{x}_j)$ denotes the k-th channel of the pixel color at pixel $\mathbf{x}_j$, where as c_i denotes the color of the ith Gaussian. The first term corresponds to our idea of using pixel differences to guide texture upscaling, which eliminates the possibility of introducing additional parameters on already well-reconstructed regions of the scene. The last two terms involved in the positional gradient calculation contain the information of occlusion from other Gaussians, the opacity, and the pixel contribution count. They can be computed as

$$\frac{\partial \mathbf{c}^k(\mathbf{x}_j)}{\partial \alpha_i(\mathbf{x}_j)} = \prod_{l=1}^{i-1}(1-\alpha_l(\mathbf{x}_j))c_i(\mathbf{x}_j) + \sum_{p=i+1}^{N} c_p(\mathbf{x}_j)\frac{\partial w_p(\mathbf{x}_j)}{\partial \alpha_i(\mathbf{x}_j)}, \tag{7}$$

where $\frac{\partial w_p(\mathbf{x}_j)}{\partial \alpha_i(\mathbf{x}_j)} = -\alpha_p(\mathbf{x}_j)\prod_{\substack{l=1 \\ l\neq i}}^{p-1}(1-\alpha_l(\mathbf{x}_j))$, and

$$\frac{\partial \alpha_i(\mathbf{x}_j)}{\partial \mu_{i,x}} = o_i \cdot \frac{\partial \mathcal{G}_i(\mathbf{u}(\mathbf{x}_j))}{\partial \mu_{i,x}} \cdot T_i^{\mathrm{A}}\left(\mathbf{u}(\mathbf{x}_j)\right). \tag{8}$$

Similarly, we can derive the equations for $\frac{\partial \mathcal{L}_j}{\partial \mu_{i,y}}$ and $\frac{\partial \mathcal{L}_j}{\partial \mu_{i,z}}$. For i-th the Gaussian, we then accumulate the absolute value of its positional gradient over the covered pixels as $\nabla_{\mu_i}\mathcal{L}$ following AbsGS(Ye et al., 2024). If the magnitude of the accumulated gradient exceeds a threshold (i.e., $\|\nabla_{\mu_i}\mathcal{L}\|_2 > k_{\mathrm{G}}$), it indicates the presence of high-frequency content on the Gaussian. We then select the Gaussian as the candidate for adaptive texture upscaling.

4.3 Adaptive Texture Upscaling

After selecting candidate 2D Gaussians, we adaptively increase their texture resolution based on anisotropy. Specifically, we compute the ratio of the two semi-axes, s_x and s_y, of each 2D Gaussian and use it to guide to determine the anisotropy. The texture resolutions $\mathcal{T}^u$ and $\mathcal{T}^v$ are then updated according to the following rules:

- If $\mathrm{s}_x/\mathrm{s}_y > k_{\mathrm{A}}$ and $s_y < k_{\mathrm{S}}$, then we double $\mathcal{T}^u$.
- If $\mathrm{s}_y/\mathrm{s}_x > k_{\mathrm{A}}$ and $s_x < k_{\mathrm{S}}$, then we double $\mathcal{T}^v$.
- Otherwise, we double the resolution of both $\mathcal{T}^u$ and $\mathcal{T}^v$.

Here, k_{A} and k_{S} are predefined thresholds. This adaptive strategy allocates resolution more effectively: textures are refined preferentially along the dimension most sensitive to gradient variations, while still allowing isotropic refinement when Gaussians are approximately square or both axes require higher detail.

When upscaling, the new texture color T^{RGB} and texture alpha T^A are initialized from the nearest pixel of the previous texture. Both T^{RGB} and T^A, together with the Gaussian parameters, are then optimized in subsequent iterations. The adaptive texture upscaling is applied every 500 iterations, allowing gradients to accumulate before resolution adjustments.

5 Experiments

5.1 Experimental Setup

Datasets and metrics. We conducted experiments on the Mip-NeRF 360 dataset (Barron et al., 2022)(7 scenes), the Tanks and Temples dataset (Knapitsch et al., 2017)(2 scenes), and the Deep Blending dataset (Hedman et al., 2018a)(2 scenes). The evaluation metrics include PSNR, SSIM, LPIPS, the number of Gaussians, and the memory size of trainable parameters for each algorithm. The testing set was constructed by selecting every eighth image, while the remaining images were used for training.

Method	Mip-NeRF 360					Tanks and Temples					DeepBlending				
	PSNR ↑	SSIM ↑	LPIPS ↓	#GS	Mem	PSNR ↑	SSIM ↑	LPIPS ↓	#GS	Mem	PSNR ↑	SSIM ↑	LPIPS ↓	#GS	Mem
2DGS*	26.72	0.787	0.308	259k	60MB	22.32	0.791	0.275	216k	50MB	28.89	0.889	0.302	259k	60MB
2DGS*-MCMC	27.47	0.806	0.228	259k	60mb	22.55	0.803	0.198	216k	50MB	29.14	0.893	0.218	259k	60MB
Super Gaussians	25.29	0.791	0.300	202k	60MB	21.46	0.783	0.273	169k	50MB	29.01	0.895	0.292	203k	60MB
BBSplat	23.07	0.614	0.462	14k	60MB	16.21	0.522	0.579	12k	50MB	26.19	0.839	0.385	14k	60MB
Textured Gaussians*	27.02	0.789	0.261	123k	60MB	22.61	0.784	0.239	102k	50MB	29.13	0.891	0.232	123k	60MB
A^{2}TG (Ours)	27.64	0.809	0.230	207k	**57MB**	22.68	0.798	0.210	170k	**50MB**	29.34	0.893	0.221	200k	**55MB**

Table 1: **Quantitative comparison under a fixed memory budget.** We report PSNR ↑, SSIM ↑, LPIPS ↓, number of Gaussians (#GS), and memory (Mem, MB). The top three results are highlighted in red, orange, and yellow, and the least memory sizes are in **bold**.

Method	Mip-NeRF 360				Tanks & Temples				DeepBlending			
	PSNR ↑	SSIM ↑	LPIPS ↓	Mem(MB,+%)	PSNR ↑	SSIM ↑	LPIPS ↓	Mem(MB,+%)	PSNR ↑	SSIM ↑	LPIPS ↓	Mem(MB,+%)
#GS = 500k												
2DGS*	27.77	0.811	0.270	116 (0%)	23.08	0.824	0.220	116 (0%)	29.45	0.894	0.279	116 (0%)
2DGS*-MCMC	27.91	0.822	0.197	116 (0%)	22.85	0.815	0.167	116 (0%)	29.37	0.892	0.204	116 (0%)
Super Gaussians	28.01	0.831	0.245	148 (28%)	23.31	0.835	0.206	148 (28%)	29.49	0.903	0.266	148 (28%)
Textured Gaussians*	28.32	0.831	0.186	244 (110%)	23.53	0.822	0.164	244 (110%)	29.76	0.897	0.193	244 (110%)
A^{2}TG (Ours)	28.19	0.829	0.188	142 (**22**%)	23.16	0.817	0.165	144 (**24**%)	29.45	0.892	0.201	135 (**16**%)
#GS=100k												
2DGS*	–	–	–	–	19.17	0.734	0.324	23 (0%)	28.32	0.879	0.324	23 (0%)
2DGS*-MCMC	26.35	0.765	0.296	23 (0%)	21.96	0.772	0.246	23 (0%)	28.60	0.880	0.255	23 (0%)
Super Gaussians	23.12	0.737	0.366	30 (28%)	20.11	0.737	0.327	30 (28%)	26.61	0.875	0.331	30 (28%)
BBSplat	27.73	0.828	0.216	432 (1764%)	23.40	0.842	0.164	432 (1764%)	28.78	0.894	0.268	432 (1764%)
Textured Gaussians*	26.67	0.777	0.278	49 (110%)	22.50	0.779	0.242	49 (110%)	29.16	0.886	0.241	49 (110%)
A^{2}TG (Ours)	26.61	0.773	0.285	29 (**26**%)	22.21	0.775	0.243	29 (**26**%)	28.94	0.883	0.248	28 (**22**%)

Table 2: **Quantitative comparison under a fixed number of Gaussians.** We report PSNR ↑, SSIM ↑, LPIPS ↓, #GS, and memory (MB). The parameters increase for texture parameters relative to the size of 2DGS is added as percentage after memory (MB). The top three results are highlighted in red, orange, and yellow, and the least parameter increases are in **bold**.

Baselines. We compare A^{2}TG against 2DGS (Huang et al., 2024), and textured Gaussian methods including Textured Gaussians (Chao et al., 2025), BBSplat (Svitov et al., 2024), and SuperGaussians (Xu et al., 2024a).

To ensure a fair comparison, we disable the depth-distortion and normal-consistency losses in 2DGS, as these terms primarily improve mesh quality at the cost of slightly reduced image fidelity. Furthermore, since both Textured Gaussians and A^{2}TG are built on 2DGS with an MCMC-based density control, we report results for two 2DGS variants: the original 2DGS with the adaptive density control of Kerbl et al. (2023) and 2DGS equipped with the same MCMC strategy(Kheradmand et al., 2024)

We use an unofficial implementation of Textured Gaussians that employs 2DGS for rasterization instead of 3DGS.

We denote the modified baselines as 2DGS*, 2DGS*-MCMC, and Textured Gaussians*, respectively.

Implementation details. Textured Gaussians* also adopt a two-stage training process, beginning with an MCMC pretraining. For fairness, all experiments with Textured Gaussians* and our A^{2}TG model share the same first-stage results when trained with the same number of Gaussians. In the second stage, Textured Gaussians use a fixed texture resolution of 4×4, while our A^{2}TG method applies adaptive texture upscaling at iterations 500 and 1000. As a result, final textures in Textured Gaussians remain 4×4, whereas those in A^{2}TG vary within $\{1, 2, 4\} \times \{1, 2, 4\}$, including both square and anisotropic shapes. The parameters of A^{2}TG are set as $k_{\mathrm{A}} = 4.0$, $k_{\mathrm{S}} = 0.01$, $k_{\mathrm{G}} = 0.00002$.

5.2 Comparisons

We employ two different comparison settings for comprehensive evaluation on the performance of our method, one with fixed memory budget, and the other with fixed number of Gaussians. Metrics

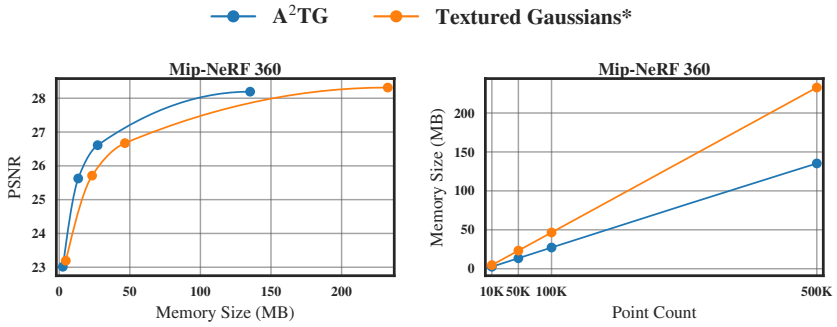


Figure 2: **Comparison of A²TG and Textured Gaussians* on Mip-NeRF 360 datasets.** Left: PSNR versus memory size (MB). Right: memory size (MB) versus point count. A²TG achieves higher reconstruction quality under the same memory budget and requires less memory for the same number of Gaussians.

including PSNR, SSIM, LPIPS on all the datasets are reported separately. Note that BBSplat is designed for a lower number of Gaussians because of their extra parameters; thus, they were not evaluated under the setting of 500k Gaussians.

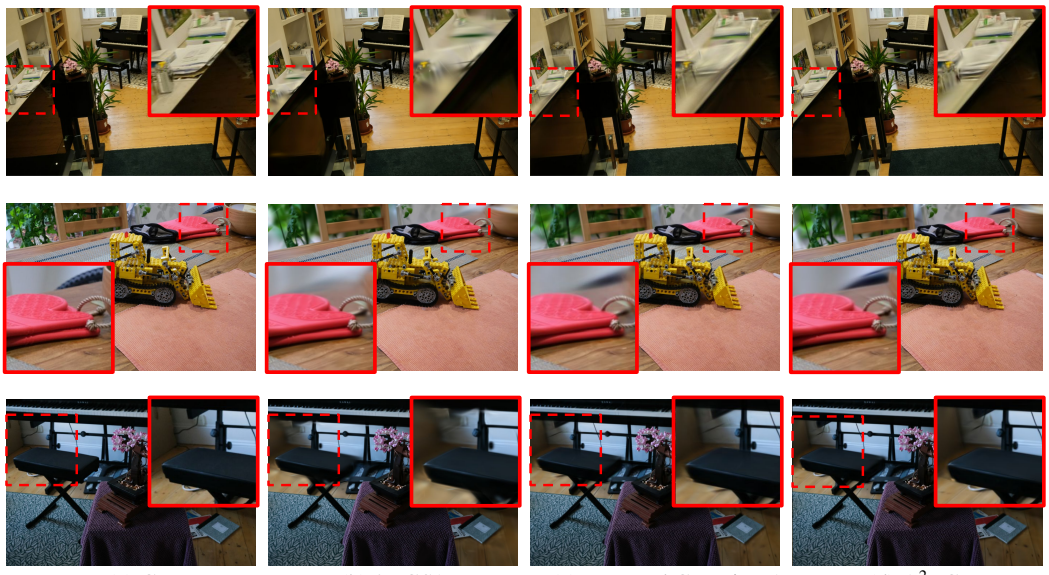
Comparison under a fixed memory budget. We report the visual quality metrics, number of Gaussians, and memory consumption of trainable parameters of all the baselines and our A²TG method in Table 1. Our method has achieved the best performance across various textured-based 2DGS works under the same memory constraint demonstrating the efficiency of our method. Since our adaptive texture control is based on gradients threshold selection, we can't achieve the exact same memory consumption as other methods, but our A²TG method still achieve higher overall visual quality with less memory expecially for PSNR. Other texture-based methods suffer from excessive texture parameters, thus result in worse visual quality under a fixed memory constraint.

For visual comparison, we present the testing-view results of A²TG alongside 2DGS* and Textured Gaussians* in Figure 3. Owing to the flexibility of textured representations, both Textured Gaussians* and A²TG achieve notably higher visual quality than 2DGS*, despite the latter using a larger number of Gaussians. Moreover, under equal memory budgets, A²TG produces higher-fidelity renderings than Textured Gaussians*, primarily because it can represent the scene with a larger set of Gaussians.

Comparison under a fixed number of Gaussians. We report the visual quality metrics and the memory consumption in Table 2. The percentage increase in memory is measured relative to 2DGS* with the number of Gaussians fixed to 500k and 100k. All methods outperform 2DGS due to the textures. BBSplat and Textured Gaussians* typically have slightly better visual quality than our method but require substantially more memory because of the inefficient ways to use the texture parameters. In particular, Textured Gaussians* takes up to more then 4 times memory than A²TG with around 0.4 increase in terms of PSNR. We also report the full experiments in Appendix A.

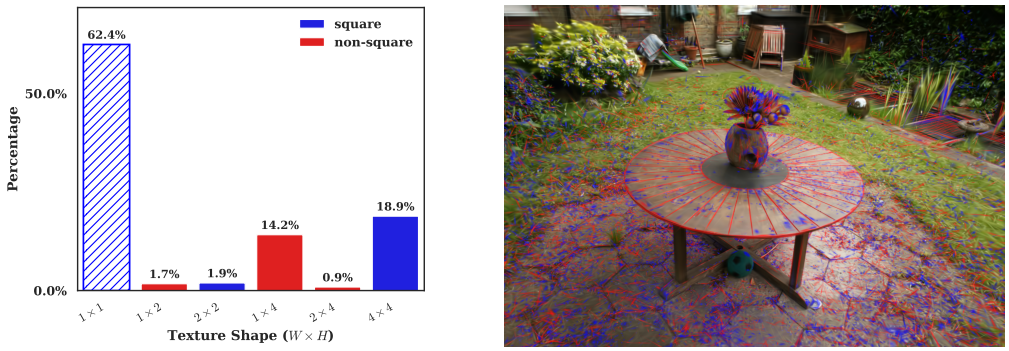
To better illustrate the efficiency of our method, we compare our A²TG and Textured Gaussians for different number of Gaussians, and plot the relation between memory and PSNR, as well as the relation between point count and memory as shown in Figure 2. Overall, our method not only has a better reconstruction quality under same memory constraint, but also with less memory usage under the same number of Gaussians.

Distribution of upscaled textures. To illustrate A²TG's efficiency on a complex scene, we color-code Gaussians: blue for 2×2 or 4×4 textures, red for non-square textures, and the original color for 1×1 (no upscaling). Figure 4 (right) shows the Garden scene from the Mip-NeRF 360 dataset, while Figure 4 (left) plots the percentage distribution of all the texture resolutions. Notably, 62.4% of Gaussians retain 1×1 textures under our gradient-driven gaussian selection, indicating that texture resolution is allocated sparingly where additional detail is unnecessary. Red highlights concentrate along sharp edges (e.g., gaps between table planks), showing that A²TG detects thin anisotropic Gaussians and assigns non-square textures accordingly.



(a) GT (b) 2DGS* (c) Textured Gaussians* (d) A^{2}TG

Figure 3: **Qualitative comparisons.** We show the qualitative comparisons of 2DGS* and Textured Gaussians* with A^{2}TG under fixed memory constraint from MipNeRF360 datasets. With textures, both Textured Gaussians* and A^{2}TG reconstruct fine scene details, whereas A^{2}TG uses less memory.



(a) Texture Resolution Percentage (b) Texture Resolution Distribution

Figure 4: **The percentage and distribution.** This figure shows the percentage and distribution of texture resolution produced by the adaptive texture upscaling on the scene Garden from Mip-Nerf360 dataset. Gaussians highlighted in blue have square texture of 2×2 and 4×4, and those highlighted in red have non-square texture resolution.

Texture Decomposition. To better understand the role of adaptive textures in A^{2}TG, we perform a qualitative ablation by selectively disabling components of the appearance model. Figure 5 shows three versions of each scene: (1) the full A^{2}TG rendering, (2) a version without textures, and (3) a version without base color. For the "no textures" variant, we set all RGB textures T_i^{RGB} to zero and all alpha textures T_i^{A} to one, removing texture-driven modulation. For the "no base color" variant, we set the spherical harmonics color $\mathbf{c}_{\mathrm{SH}_i}$ to zero, leaving appearance determined solely by the learned textures.

As shown in Figure 5, removing textures leads to a clear loss of high-frequency detail—e.g., leaf boundaries, grass structure, and fine variations in carpets and fabrics—indicating that adaptive textures capture view-consistent residual appearance beyond what low-order spherical harmonics can express. Since every Gaussian contains at least 1×1 texture, removing textures effect all regions.

Conversely, removing the SH base color produces overly dark renderings with missing global shading. Although textures recover much of the fine detail, the absence of SH color removes low-

Figure 5: Qualitative visualization of what the adaptive textures learn. **Left:** full rendering from A^{2}TG. **Middle:** rendering without textures (RGB textures set to zero, alpha textures set to one). **Right:** rendering without SH base color. The comparison, visualized on two scenes shows that textures capture high-frequency residual appearance such as foliage structure and fabric detail, while SH color provides smooth, low-frequency shading. Together, they produce the final photorealistic result.

Method	PSNR	SSIM	LPIPS	Mem
#GS = 500k				
w/o Upscaling	26.71	0.843	0.189	116.00
w/o Anisotropy	26.93	0.846	0.185	147.88
Ours	26.93	0.846	0.185	140.01
#GS = 100k				
w/o Upscaling	25.64	0.806	0.266	23.20
w/o Anisotropy	25.92	0.811	0.259	30.23
Ours	25.92	0.810	0.259	28.43

Method	PSNR	SSIM	LPIPS	Mem
#GS = 50k				
w/o Upscaling	24.85	0.776	0.319	11.60
w/o Anisotropy	25.10	0.782	0.311	15.06
Ours	25.12	0.782	0.311	14.07
#GS = 10k				
w/o Upscaling	22.40	0.688	0.482	2.31
w/o Anisotropy	22.66	0.696	0.469	2.85
Ours	22.65	0.696	0.469	2.65

Table 3: **Ablation study with varying numbers of Gaussians.** We report PSNR (↑), SSIM (↑), LPIPS (↓), and memory usage in MB (↓), averaged across Mip-NeRF 360, Tanks&Temples, and DeepBlending. The top three results in each column are highlighted in red, orange, and yellow.

frequency illumination and structural coherence, showing that the SH term provides a smooth, lighting-aware foundation.

Overall, these ablations highlight the complementary roles of SH color and adaptive textures: SH models coarse shading and global appearance, while textures encode local, high-frequency details. This further supports our choice of allocating texture resolution adaptively so that texture capacity is focused where fine detail is needed.

5.3 Ablation Studies

We examine the impact of resolution scaling and anisotropy on our adaptive texture upscaling. Ablations are run on three datasets with the number of Gaussians fixed at 10k, 50k, 100k, and 500k, comparing A^{2}TG to two variants: (i) without texture upscaling (w/o Upscaling) and (ii) without anisotropic texture upscaling (w/o Anisotropy). Table 3 reports average memory usage and reconstruction quality (PSNR, SSIM, LPIPS). Removing scaling yields the lowest memory but the worst

quality. Disabling anisotropy produces quality close to the full method but uses more memory because square textures cannot match anisotropic Gaussian as efficiently.

6 Conclusion

Textured Gaussians enhance visual quality by augmenting each Gaussians with a RGB texture map for spatial varying color and a alpha texture for spatial varying opacity, but these methods typically introduce too much parameters and causes memory inefficiency. To address this, we propose adaptive anisotropic textured Gaussians (A^2TG) that allow rectangle texture map whose aspect ratios and resolutions are aligned to the geometry of the Gaussians. This reduces memory storage while preserving fine details. We also introduce a gradient-based adaptive texture control strategy to efficiently determine the texture shapes along with parameter update for joint optimization. Our method has achieved best visual quality among state-of-the-art methods under the same memory constraint, demonstrating efficiency of over method.

Limitation and future work. We demonstrate that A^2TG produces efficient textured Gaussians with better visual quality and memory efficiency. Currently, we only upscale texture resolution of Gaussians. To further improve the efficiency of texture usage, we can scale down the texture resolution when necessary. Moreover, compressing textures and controlling the shape and size of Gaussians can also improve memory efficiency. For future works, combining adaptive textures with flexible primitives beyond 2D/3D Gaussians such as Deformable Radial Kernal Splatting(Huang et al., 2025) might further enable better texture utilization since those primitives has flexible boundary shapes and sharpness. In addition, we also plan to develop dynamic GS models, such as 4DGS, using A^2TG.

References

Milena T Bagdasarian, Paul Knoll, Y Li, Florian Barthel, Anna Hilsmann, Peter Eisert, and Wieland Morgenstern. 3DGS. zip: A survey on 3d gaussian splatting compression methods. In *Computer Graphics Forum*, volume 44, pp. e70078. Wiley Online Library, 2025.

Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P Srinivasan. Mip-NeRF: A multiscale representation for anti-aliasing neural radiance fields. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 5855–5864, 2021.

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*, pp. 5470–5479, 2022.

Brian Chao, Hung-Yu Tseng, Lorenzo Porzi, Chen Gao, Tuotuo Li, Qinbo Li, Ayush Saraf, Jia-Bin Huang, Johannes Kopf, Gordon Wetzstein, et al. Textured gaussians for enhanced 3d scene appearance modeling. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 8964–8974, 2025.

Gaurav Chaurasia, Sylvain Duchene, Olga Sorkine-Hornung, and George Drettakis. Depth synthesis and local warps for plausible image-based navigation. *ACM transactions on graphics (TOG)*, 32 (3):1–12, 2013.

Zhiwen Fan, Kevin Wang, Kairun Wen, Zehao Zhu, Dejia Xu, Zhangyang Wang, et al. Lightgaussian: Unbounded 3d gaussian compression with 15x reduction and 200+ fps. *Advances in neural information processing systems*, 37:140138–140158, 2024.

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, 2018a.

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 (ToG)*, 37(6):1–15, 2018b.

Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger, and Shenghua Gao. 2D gaussian splatting for geometrically accurate radiance fields. In *ACM SIGGRAPH 2024 conference papers*, pp. 1–11, 2024.

Yi-Hua Huang, Ming-Xian Lin, Yang-Tian Sun, Ziyi Yang, Xiaoyang Lyu, Yan-Pei Cao, and Xiaojuan Qi. Deformable radial kernel splatting. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 21513–21523, 2025.

Zhentao Huang and Minglun Gong. Textured-GS: Gaussian splatting with spatially defined color and opacity. *arXiv preprint arXiv:2407.09733*, 2024.

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.

Shakiba Kheradmand, Daniel Rebain, Gopal Sharma, Weiwei Sun, Yang-Che Tseng, Hossam Isack, Abhishek Kar, Andrea Tagliasacchi, and Kwang Moo Yi. 3d gaussian splatting as markov chain monte carlo. *Advances in Neural Information Processing Systems*, 37:80965–80986, 2024.

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.

Georgios Kopanas, Julien Philip, Thomas Leimkühler, and George Drettakis. Point-based neural rendering with per-view optimization. In *Computer Graphics Forum*, volume 40, pp. 29–43. Wiley Online Library, 2021.

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*, pp. 21719–21728, 2024.

Saswat Subhajyoti Mallick, Rahul Goel, Bernhard Kerbl, Markus Steinberger, Francisco Vicente Carrasco, and Fernando De La Torre. Taming 3DGS: High-quality radiance fields with limited resources. In *SIGGRAPH Asia 2024 Conference Papers*, pp. 1–11, 2024.

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. *Communications of the ACM*, 65(1):99–106, 2021.

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*, pp. 10349–10358, 2024.

Panagiotis Papantonakis, 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*, 7(1):1–17, 2024.

Victor Rong, Jingxiang Chen, Sherwin Bahmani, Kiriakos N Kutulakos, and David B Lindell. GStex: Per-primitive texturing of 2d gaussian splatting for decoupled appearance and geometry modeling. In *2025 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 3508–3518. IEEE, 2025.

Yunzhou Song, Heguang Lin, Jiahui Lei, Lingjie Liu, and Kostas Daniilidis. Hdgs: Textured 2d gaussian splatting for enhanced scene rendering. *arXiv preprint arXiv:2412.01823*, 2024.

David Svitov, Pietro Morerio, Lourdes Agapito, and Alessio Del Bue. BillBoard Splatting (BBSplat): Learnable textured primitives for novel view synthesis. *arXiv preprint arXiv:2411.08508*, 2024.

Matthew Tancik, Pratul Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in neural information processing systems*, 33:7537–7547, 2020.

Yufei Wang, Zhihao Li, Lanqing Guo, Wenhan Yang, Alex Kot, and Bihan Wen. ContextGS: Compact 3d gaussian splatting with anchor level context model. *Advances in neural information processing systems*, 37:51532–51551, 2024.

Sebastian Weiss and Derek Bradley. Gaussian billboards: Expressive 2d gaussian splatting with textures. *arXiv preprint arXiv:2412.12734*, 2024.

Rui Xu, Wenyue Chen, Jiepeng Wang, Yuan Liu, Peng Wang, Lin Gao, Shiqing Xin, Taku Komura, Xin Li, and Wenping Wang. SuperGaussians: Enhancing gaussian splatting using primitives with spatially varying colors. *arXiv preprint arXiv:2411.18966*, 2024a.

Tian-Xing Xu, Wenbo Hu, Yu-Kun Lai, Ying Shan, and Song-Hai Zhang. Texture-GS: Disentangling the geometry and texture for 3d gaussian splatting editing. In *European Conference on Computer Vision*, pp. 37–53. Springer, 2024b.

Zongxin Ye, Wenyu Li, Sidun Liu, Peng Qiao, and Yong Dou. AbsGS: Recovering fine details in 3d gaussian splatting. In *Proceedings of the 32nd ACM International Conference on Multimedia*, pp. 1053–1061, 2024.

Zhaoliang Zhang, Tianchen Song, Yongjae Lee, Li Yang, Cheng Peng, Rama Chellappa, and Deliang Fan. LP-3DGS: Learning to prune 3d gaussian splatting. *Advances in Neural Information Processing Systems*, 37:122434–122457, 2024.

A Appendix

A.1 Additional experiments comparing our method with baselines.

Table 4 reports visual-quality metrics and memory for all methods at #GS = 500k, 100k, 50k, and 10k on Mip-NeRF 360, Tanks&Temples, and DeepBlending. Memory percentages are measured relative to the corresponding 2DGS baseline within each #GS block. Methods that leverage textures consistently outperform plain 2DGS variants. *Textured Gaussians** and *BBSplat* often obtain the best PSNR/SSIM/LPIPS, but at substantial cost: *Textured Gaussians** incurs about +110% memory across #GS and datasets, while *BBSplat* reaches ∼+1760% at #GS=100k. In contrast, A^2TG achieves comparable visual quality with markedly lower overhead—typically +16–24% at 500k, +19–26% at 100k, +19–24% at 50k, and +14–19% at 10k—demonstrating that adaptive, anisotropic textures can deliver near state-of-the-art fidelity with far more efficient parameter usage.

DeepBlending at 1M Gaussians. For completeness, we additionally evaluate all methods on the DeepBlending dataset at a higher point budget of #GS = 1M. Since this setting is not used for the other datasets, we report these results separately in Table 5. The ranking trends remain consistent with the main table: *Textured Gaussians** achieves the best PSNR and LPIPS, and A^2TG continues to deliver competitive reconstruction quality while maintaining substantially lower memory usage than other textured approaches.

A.2 Additional Details on Computational Efficiency

3D Gaussian Splatting (Kerbl et al., 2023) is known for its real-time rendering capability. Our method builds upon 2D Gaussian Splatting (Huang et al., 2024) by augmenting each Gaussian with a non-uniform, variable-sized texture. During rendering, after computing the UV coordinate of the intersection between a pixel and a Gaussian along the camera ray, we fetch texture values using bilinear interpolation.

To efficiently support non-uniform texture resolutions ranging from $\{1, 2, 4\} \times \{1, 2, 4\}$, we pack all per-Gaussian textures into a single large texture-atlas array located in GPU global memory. Each Gaussian stores additional metadata specifying its texture dimensions and its offset within the atlas. In our custom CUDA rasterization kernel, this structure results in up to 16 global-memory load

Method	Mip-NeRF 360				Tanks & Temples				DeepBlending			
	PSNR ↑	SSIM ↑	LPIPS ↓	Mem (MB, +%)	PSNR ↑	SSIM ↑	LPIPS ↓	Mem (MB, +%)	PSNR ↑	SSIM ↑	LPIPS ↓	Mem (MB, +%)
#GS = 500k (baseline: 2DGS = 116.00MB for all datasets)												
2DGS	27.38	0.805	0.283	116.00 (0%)	22.91	0.819	0.234	116.00 (0%)	29.24	0.894	0.286	116.00 (0%)
2DGS*	27.77	0.811	0.270	116.00 (0%)	23.08	0.824	0.220	116.00 (0%)	29.45	0.894	0.279	116.00 (0%)
Super Gaussians	28.01	0.831	0.245	147.82 (27%)	23.31	0.835	0.206	147.69 (27%)	29.49	0.903	0.266	147.83 (27%)
2DGS*-MCMC	27.91	0.822	0.197	116.00 (0%)	22.85	0.815	0.167	116.00 (0%)	29.37	0.892	0.204	116.00 (0%)
A^2TG (Ours)	28.19	0.829	0.188	141.79 (22%)	23.16	0.817	0.165	143.64 (24%)	29.45	0.892	0.201	134.72 (16%)
Textured Gaussians*	28.32	0.831	0.186	244.00 (110%)	23.53	0.822	0.164	244.00 (110%)	29.76	0.897	0.193	244.00 (110%)
#GS = 100k (baseline: 2DGS ≈ 23.2MB; 23.16MB for T&T)												
2DGS	–	–	–	–	18.82	0.725	0.340	23.16 (0%)	27.80	0.875	0.336	23.20 (0%)
2DGS*	–	–	–	–	19.17	0.734	0.324	23.16 (0%)	28.32	0.879	0.324	23.20 (0%)
Super Gaussians	23.12	0.737	0.366	29.57 (28%)	20.11	0.737	0.327	29.57 (28%)	26.61	0.875	0.331	29.58 (28%)
BBSplat	27.73	0.828	0.216	432.40 (1764%)	23.40	0.842	0.164	432.40 (1767%)	28.78	0.894	0.268	432.40 (1764%)
2DGS*-MCMC	26.35	0.765	0.296	23.20 (0%)	21.96	0.772	0.246	23.20 (0%)	28.60	0.880	0.255	23.20 (0%)
A^2TG (Ours)	26.61	0.773	0.285	28.54 (23%)	22.21	0.775	0.243	29.21 (26%)	28.94	0.883	0.248	27.58 (19%)
Textured Gaussians*	26.67	0.777	0.278	48.80 (110%)	22.50	0.779	0.242	48.80 (110%)	29.16	0.886	0.241	48.80 (110%)
#GS = 50k (baseline: 2DGS = 11.60MB for all datasets)												
2DGS*-MCMC	25.34	0.725	0.361	11.60 (0%)	21.25	0.737	0.302	11.60 (0%)	27.97	0.867	0.294	11.60 (0%)
A^2TG (Ours)	25.63	0.733	0.350	14.09 (21%)	21.44	0.743	0.298	14.43 (24%)	28.30	0.871	0.285	13.80 (19%)
Textured Gaussians*	25.71	0.739	0.339	24.40 (110%)	21.77	0.747	0.292	24.40 (110%)	28.40	0.873	0.278	24.40 (110%)
#GS = 10k (baseline: 2DGS = 2.32MB for all datasets)												
2DGS*-MCMC	22.77	0.616	0.538	2.32 (0%)	19.40	0.631	0.470	2.32 (0%)	25.02	0.817	0.438	2.32 (0%)
A^2TG (Ours)	23.02	0.625	0.524	2.66 (15%)	19.67	0.640	0.460	2.77 (19%)	25.26	0.823	0.424	2.65 (14%)
Textured Gaussians*	23.19	0.631	0.507	4.88 (110%)	19.96	0.646	0.446	4.88 (110%)	25.58	0.826	0.411	4.88 (110%)

Table 4: Appendix: Full multi-dataset evaluation. Memory percentages are relative to **2DGS** at the same #GS.

Method	PSNR ↑	SSIM ↑	LPIPS ↓	Mem (MB)
2DGS*	29.58	0.898	0.190	232
Super Gaussians	29.38	0.905	0.253	295
Textured Gaussians*	29.66	0.896	0.183	488
A2TG (Ours)	29.55	0.894	0.190	266

Table 5: DeepBlending evaluation at #GS = 1M.

operations per texture lookup, since bilinear interpolation may require up to four texel samples for each of the RGBA channels.

Although these texture fetches introduce additional overhead during both training and inference, the system still maintains real-time rendering speeds above 30 FPS. While we did not explicitly optimize the rasterizer for maximum throughput, further improvements are likely possible through vectorized texture loads or tile-level shared-memory prefetching.

Table 6 reports the inference FPS, per-frame inference time, and total training time for 2DGS (Huang et al., 2024), Textured Gaussians* (Chao et al., 2025), and A^2TG on the DeepBlending dataset (Hedman et al., 2018a). Both A^2TG and Textured Gaussians* incur additional computation compared to 2DGS due to texture sampling, which explains the decrease in inference FPS. Importantly, A^2TG is consistently faster than Textured Gaussians* during both inference and training. The key reason is that A^2TG allocates texture parameters adaptively rather than assigning a fixed-size texture to every Gaussian. This reduces the total number of texel reads and gradient updates, resulting in lower memory traffic and fewer texture operations. Consequently, A^2TG achieves higher inference FPS and shorter training time than Textured Gaussians* under the same number of Gaussians.

A.3 Additional Experiments on Texture Map Resolution

To further analyze the effect of texture map resolution, we conducted additional comparisons among Textured Gaussians*(Chao et al., 2025) and our proposed A^2TG. For Textured Gaussians*, we evaluate two texture resolutions: 2×2 and 4×4, and for A^2TG, evaluate two maximum texture resolutions: Max 2×2 and Max 4×4.

Method	2DGS	A^{2}TG	Textured Gaussians*
Number of Gaussians	500k	500k	500k
Model size (MB)	116	140	244
Inference FPS	250	140	119
Inference time per frame (s)	0.004	0.007	0.008
Training time (min)	7.18	19.7	21.6

Table 6: Computational Efficiency: Comparison of the inference speed under the same number of Gaussians on the DeepBlending dataset. The render speed of A^{2}TG and Textured Gaussians is slower than 2DGS because of the extra texture sampling during rendering, while A^{2}TG has a slightly faster render speed than Textured Gaussians due to fewer texture parameters.

Table 7 reports the results on the Mip-NeRF360 dataset with the number of Gaussians fixed at 50k. As shown, A^{2}TG achieves lower memory usage than Textured Gaussians* and delivers similar visual quality across PSNR, SSIM, and LPIPS. under both 2×2 and 4×4texture resolution. These results demonstrate the effectiveness of adaptive texture allocation, which balances texture resolution with memory efficiency more effectively than fixed-resolution textures.

Method	PSNR ↑	SSIM ↑	LPIPS ↓	Mem (MB)	#GS
Textured Gaussians* (2×2)	25.59	0.734	0.347	14.80	50k
Textured Gaussians* (4×4)	25.77	0.741	0.338	24.40	50k
A^{2}TG (Max 2×2)	25.62	0.735	0.348	12.83	50k
A^{2}TG (Max 4×4)	25.63	0.733	0.350	14.09	50k

Table 7: Comparison of different texture resolutions (W$\times$ H) on the Mip-NeRF360 dataset with 50k Gaussians. A^{2}TG achieves superior memory efficiency and improved or comparable rendering quality relative to Textured Gaussians*.

A.4 Declaration of LLM usage

We used large language models (LLMs) solely for polishing the text in this paper. Specifically, LLMs assisted in refining grammar, improving readability, and adjusting tone for academic writing. All research ideas, methodology, analyses, results, and terminology definitions presented in this work are original contributions from the authors and were not generated by LLMs.