

SA-GS: Scale-Adaptive Gaussian Splatting for Training-Free Anti-Aliasing

Xiaowei Song^{*1,2}, Jv Zheng^{*1,3}, Shiran Yuan^{1,4}, Huan-ang Gao¹, Jingwei Zhao⁵, Xiang He⁵, Weihao Gu⁵, and Hao Zhao^{†1}

¹ Institute for AI Industry Research (AIR), Tsinghua University

² Tongji University

³ Ocean University of China

⁴ Duke Kunshan University

⁵ Haomo.ai

kevin729@tongji.edu.cn, zhengju@stu.ouc.edu.cn

zhaohao@air.tsinghua.edu.cn

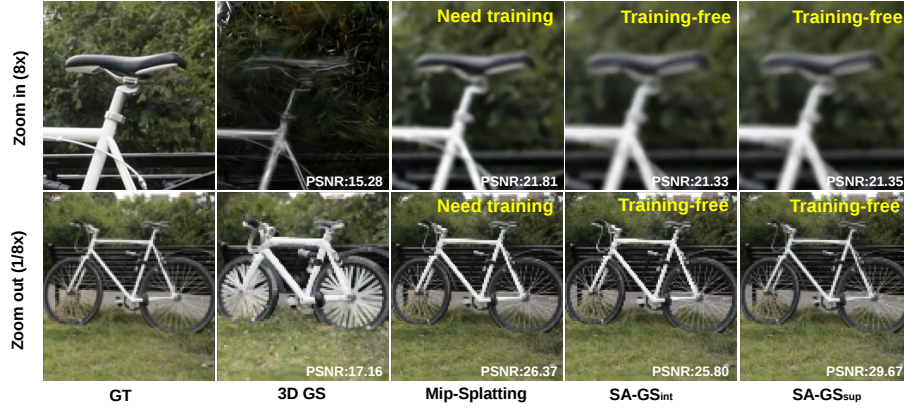


Fig. 1: Under zoom-in, 3D Gaussian Splatting [10] (*3DGS*) exhibits significant erosion artefacts, while under zoom-out, it undergoes dramatic dilation. Mip-Splatting [24] utilizes 3D smoothing and 2D Mip filters to regularize primitives **during training**. In contrast, our method is **training-free** and maintains scale consistency using solely a single **2D scale-adaptive filter**. Scale adaptation allows us to use **super-sampling** (named as *SA-GS_{sup}* later in the paper) and its limiting case **integration** (named as *SA-GS_{int}* later in the paper) to obtain more accurate results when zooming out.

Abstract. In this paper, we present a Scale-adaptive method for Anti-aliasing Gaussian Splatting (SA-GS). While the state-of-the-art method Mip-Splatting needs modifying the training procedure of Gaussian splatting, our method functions at test-time and is training-free. Specifically, SA-GS can be applied to any pretrained Gaussian splatting field as a plugin to significantly improve the field’s anti-aliasing performance. The core technique is to apply 2D scale-adaptive filters to each Gaussian during test time. As pointed out by Mip-Splatting, observing Gaussians at

* Indicates Equal Contribution. † Indicates Corresponding Author.

different frequencies leads to mismatches between the Gaussian scales during training and testing. Mip-Splatting resolves this issue using 3D smoothing and 2D Mip filters, which are unfortunately not aware of testing frequency. In this work, we show that a 2D scale-adaptive filter that is informed of testing frequency can effectively match the Gaussian scale, thus making the Gaussian primitive distribution remain consistent across different testing frequencies. When scale inconsistency is eliminated, sampling rates smaller than the scene frequency result in conventional jaggedness, and we propose to integrate the projected 2D Gaussian within each pixel during testing. This integration is actually a limiting case of super-sampling, which significantly improves anti-aliasing performance over vanilla Gaussian Splatting. Through extensive experiments using various settings and both bounded and unbounded scenes, we show SA-GS performs comparably with or better than Mip-Splatting. Note that super-sampling and integration are only effective when our scale-adaptive filtering is activated. Our codes, data and models are available at <https://github.com/zsy1987/SA-GS>.

Keywords: 3D Vision, Novel View Synthesis, Rasterization, Scale Consistency, Super-Sampling

1 Introduction

Novel View Synthesis (NVS) has played an important role in fields such as visualization [12], simulation [20, 21], automation [26, 27], and VR/AR [16, 19]. The advent of Neural Radiance Fields (NeRFs) [13, 25] has significantly enhanced the quality of view synthesis while bypassing the need of reconstructing geometry, texture, material and lighting (which is typically a very under-determined inverse problem). Recently, another method, 3D Gaussian Splatting (3DGS) [10] has garnered attention from both academia and industry for its high synthesis quality and fast rendering speed. Gaussian primitive-based representation and its corresponding efficient CUDA implementation enable 3DGS [10] to render scenes in real-time with intricate details, greatly accelerating NVS systems intended for tasks such as gaming, simulation and multimedia.

Problem. Unlike implicit representations like NeRFs, 3DGS [10] utilizes Gaussian primitives to represent 3D scenes in an explicit manner. This is achieved by optimizing the position, scale, transparency, rotation and spherical harmonic coefficients of each Gaussian primitive to fit input images, producing a continuous 3D signal with a complex Gaussian mixture distribution. However, as pointed out by a recent study Mip-Splatting [24], there is a trick in 3DGS not mentioned in the paper, which is introduced to ensure numerical stability. Specifically, 2D dilation is added during training to expand the distribution over a planar region to eliminate the case in which the region is smaller than one single pixel (thus causing instability). This operation guarantees steady updating of the Gaussian primitives, but results in inconsistencies in the degree of dilation and the degree of change in Gaussian scale if the intrinsic and extrinsic parameters of the camera are not equal to the training situation, with artefacts illustrated in Fig. 1.

This is due to the fact that 2D dilation is fixed on the pixel space and is not informed of the scale variation of the Gaussian, as illustrated in Fig. 3.

Cause & Solution. Training data is typically produced using consistent camera settings. Therefore, the use of a fixed dilation operation during the training phase does not result in variations in the dilated scale at the same location. In this case, the Gaussian primitives can still learn a reasonable 2D projection distribution. However, during the rendering phase, the Gaussian scene may be observed at various resolutions and distances. This can compromise the otherwise good 2D projection distribution, resulting in a different α -blending process than during the training phase, which ultimately affects the rendering quality. In this paper, we name this phenomenon as **Gaussian scale mismatch**, which is a property specific to 3DGS and absent in NeRFs. We believe that the 2D projection distribution of Gaussian primitives in the rendering phase should be consistent with the training phase. We correct the 2D dilation operation (in 3DGS) and the 3D smoothing+2D Mip filter (in Mip-Splatting) via a 2D scale-adaptive filter to enforce scale consistency at different rendering parameter settings.

Anti-aliasing. When the projected 2D Gaussian distribution remains consistent with the training at different rendering settings, anti-aliasing is simplified to ensure the synergy of the sampling frequency and the scene frequency. As the sampling frequency decreases, the Nyquist sampling theorem [15, 18] may not be satisfied at a certain frequency level, resulting in aliasing effects in the image. Therefore, we introduce conventional anti-aliasing ideas, super-sampling and its limiting case integration, into 3DGS so that the Nyquist sampling theorem is satisfied when zooming out. Notably, super-sampling and integration only make sense after the Gaussian scale mismatch issue is addressed.

Significance. As shown in Fig. 1, our method can well address the artefacts of vanilla 3DGS while being training-free. Under the zoom-in and zoom-out settings, vanilla 3DGS shows severe visual quality degradation because of erosion and dilation. The quality degradation is actually caused by an intertwined effect of Gaussian scale mismatch and aliasing, which is different from the case that, in NeRFs, artefacts under zoom-in and zoom-out are solely caused by aliasing. While Mip-Splatting [24] can address these artefacts, our method is: (1) **More flexible.** Mip-Splatting needs to modify the training procedure of 3DGS, but our method is a training-free plugin; (2) **More elegant.** Mip-Splatting resolves the Gaussian scale mismatch issue with 3D smoothing and 2D Mip filter, but our method exploits a single 2D scale-adaptive filter; (3) **More accurate.** Our method performs comparably with Mip-Splatting but out-performs it under zoom-out because our scale-adaptive formulation can unleash the power of simple and effective strategies super-sampling (and its limiting case integration).

In summary, our contributions are as follows:

1. We introduce a training-free approach that can be directly applied to the inference process of any pretrained 3DGS [10] model to resolve its visual artefacts at drastically changed rendering settings. The method itself is named as scale-adaptive Gaussian splatting (SA-GS) as a whole.

2. Technically, we propose a 2D scale-adaptive filter that keeps the Gaussian projection scale consistent with the training phase scale at different rendering settings. This scale-adaptive filter also allows simple anti-aliasing techniques (super-sampling and its limiting case integration) to work effectively.
3. Extensive qualitative and quantitative experiments were conducted on the Mip-NeRF 360 [2] and Blender [13] datasets. Our method achieves superior or comparable performance compared to the state-of-the-art Gaussian anti-aliasing methods, while being training-free.

2 Related Works

2.1 Anti-aliased Neural Radiance Fields

Aliasing is a common issue in conventional computer graphics that occurs when the rendering frequency drastically changes, resulting in visual artefacts such as jagged edges or moiré patterns. These artefacts are caused by the discrete sampling of continuous physical signals. Neural rendering, which is represented by neural radiance fields (NeRFs) [13] or other techniques [4, 9, 11], is also troubled by aliasing. Anti-aliasing neural radiance fields are an active research direction, with notable milestones like Mip-NeRF [1], Mip-NeRF 360 [2], Zip-NeRF [3] and Tri-MipRF [7]. Mip-NeRF [1] featurizes the 3d frustum with an approximate ellipse such as the view-dependent distance-aware visual effects are captured by a closed-form geometric encoding of the (approximate) ellipse. Mip-NeRF 360 [2] accelerates Mip-NeRF [1] with a density proposal network and addresses unbounded scenes using a heuristic contraction rule. Zip-NeRF [3] designs a spiral sampling pattern, instead of volume encoding, for the anti-aliasing of radiance fields based on hash grids. Tri-MipRF [7] interpolates down-sampled tri-planes (at corresponding scales) to build a mipmap-like representation. Despite various techniques designed for different NeRF backbones (MLP-based, triplane-based or grid-based), most of them are motivated by two old ideas: super-sampling and mipmap. Our method SA-GS is also motivated by super-sampling, but as will be discussed in the next section, in Gaussian splatting fields, other issues need to be addressed before the power of super-sampling can be fully unleashed.

2.2 Gaussian Splatting

3D Gaussian Splatting (3DGS) [10] is a recently proposed neural¹ rendering paradigm that is primitive-based (like Point-NeRF [23] or ADOP [17]), uses spherical harmonics as the color representation (like Plenoxels [5]), and renders at a very fast rate (faster than Instant-NGP [14]). While this new paradigm has recently seen much progress in terms of physical integration (PhysGaussian [22]) and geometrical alignment (SuGaR [6]), anti-aliasing for 3DGS [10] is not yet well addressed. We note that one fundamental challenge is that when 3DGS [10]

¹ This is somewhat inaccurate because many people argue that 3DGS does not involve typical neural networks but only matrix multiplication.

is rendered at different distances (i.e., resolutions), the Gaussian scale might not match the training-time scale. This mismatch issue entangles with aliasing, making the problem extremely complicated. This has been pointed out by a recent study Mip-Splatting [24], showing that the dilation and erosion issues were caused by this mismatch. In this paper, we demonstrate that classical super-sampling (and its limiting case of integration) is effective for 3DGS [10] anti-aliasing, but only under the case of matched Gaussian scales. Unlike the heuristic and problematic Gaussian scale filtering techniques used by 3DGS [10] and Mip-Splatting [24], our adaptive solution well addresses the mismatch issue, thus fully unleashing the power of super-sampling for anti-aliasing.

3 Method

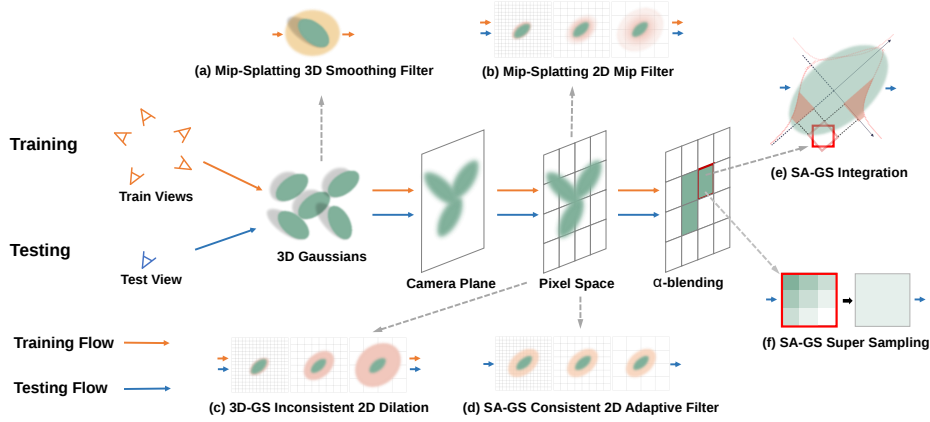


Fig. 2: Paradigm Comparison of Gaussian Rasterization Process. All Gaussian Splatting methods share this framework for training and rendering, but different models use different strategies to process Gaussian primitives. During training, 3DGS [10] uses (c) in pixel space for training stability, but results in scale inconsistencies at different rendering settings; Mip-Splatting utilises (a) to restrict the Gaussian frequency upper bound in 3D space, and (b) to emulate box filtering in pixel space. But Mip-Splatting [24] still suffers from scale inconsistency and needs to modify the training procedure of 3DGS. Our approach is training-free and only operates on the testing flow. We use (d) in pixel space to maintain the scale consistency of the Gaussian primitives, and further enhance the anti-aliasing capability of 3DGS by applying (e) and (f) to the α -blending process. Note that (e) and (f) only make sense with (d) activated.

A **Paradigm comparison** is presented in Fig. 2. Overall, our SA-GS method aims to mitigate the artefacts of 3DGS when rendered at different settings (e.g., $8\times$ zoom-in and $1/8\times$ zoom-out shown in Fig. 1). A notable fact is that in

NeRFs [1–3, 8, 13], artefacts under zoom-in/out are caused by aliasing, while for 3DGS these artefacts are caused by the intertwined effects of Gaussian scale mismatch and aliasing. Thus, anti-aliasing techniques make sense only after the Gaussian scale mismatch issue is addressed. The root of Gaussian scale mismatch in vanilla 3DGS, as pointed out by [24], is not described in the paper of [10]. The root is shown in Fig. 2-(c), which is designed to expand projected 2D Gaussians so that the case of the region is smaller than one single pixel is eliminated. To alleviate the mismatch, Mip-splatting introduces 3D smoothing (Fig. 2-(a)) and 2D Mip filter (Fig. 2-(b)), during training. Unfortunately, these heuristic methods (Fig. 2-(a/b/c)) do not address the mismatch issue in principle, so that conventional anti-aliasing techniques fail to work for 3DGS and Mip-splatting. Our SA-GS features a 2D scale adaptive filter (Fig. 2-(d)) that resolves the mismatch issue in principle and is a training-free plugin. It also unleashes the power of simple anti-aliasing techniques like super-sampling (Fig. 2-(f)) and its limiting case integration (Fig. 2-(e)) to work for 3DGS.

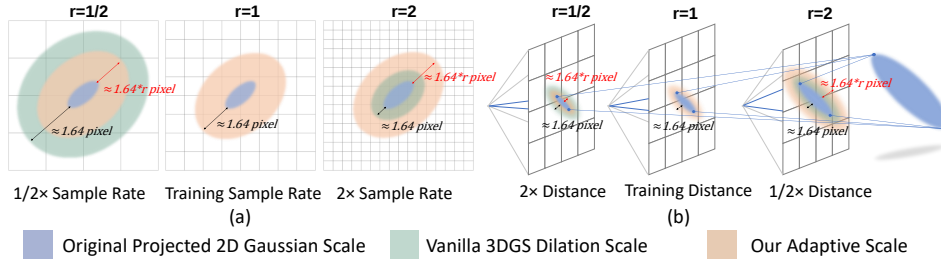


Fig. 3: Scale ambiguity. The heuristic 2D dilation process in vanilla 3DGS code (pointed out by [24]) operates on the pixel space and enlarges the projected 2D Gaussian by a fixed amount (around 1.64 pixel). However, a fixed 2D dilation (1.64 pixel) can result in scale ambiguities when representing the same scene at different rendering settings, as shown by the green expansion area. (a) When the Gaussian scale is held constant and the resolution changes, the dilation scale (green) changes inconsistently. (b) When the Gaussian scale changes and the resolution remains constant, the dilation scale (green) does not change with the Gaussian. Our 2D scale-adaptive filter ensures that the Gaussian scale remains consistent across different rendering settings, as shown by the red expansion area. This keeps the scale consistent with the training setup.

3.1 2D Scale-adaptive Filter

The dilation operation (heuristic and fixed at 1.64 pixel) used by 3DGS [10] during training introduces scale ambiguity to the 3D scene, as shown in Fig. 3. As mentioned above, it is crucial to maintain the scale of the Gaussian in the training setup consistent at different rendering settings.

We propose a 2D scale-adaptive filter that bridges the scale gap between the rendering stage and the training setup, whose effects are shown in Fig. 4. In pixel

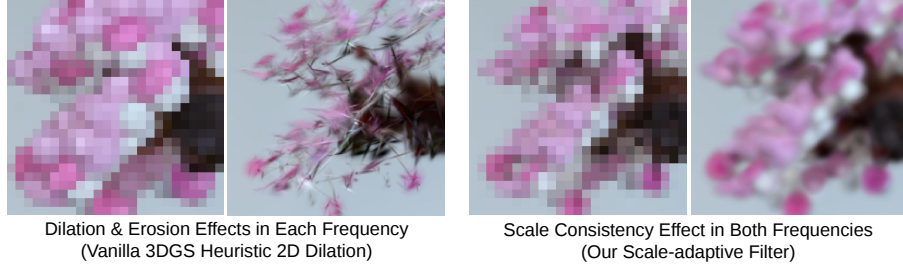


Fig. 4: 3DGS heuristic dilation and our scale-adaptive filter. Our 2D scale-adaptive filter maintains the structure of the scene at any resolution, whereas 3DGS [10] dilation (fixed at 1.64 pixel) leads to erroneous dilation at low frequencies and erosion at high frequencies. (Note *dilation* refers to method and artefacts in different contexts.)

space, a 2D Gaussian primitive can be expressed parametrically by its mean $\mathbf{p}_k$ and covariance Σ_k as follows:

$$\mathcal{G}_k^{2D}(\mathbf{x}) = e^{-\frac{1}{2}(\mathbf{x}-\mathbf{p}_k)^T \Sigma_k^{-1}(\mathbf{x}-\mathbf{p}_k)} \quad (1)$$

Problem. During the training of vanilla 3DGS, a low-pass Gaussian kernel function $\mathcal{G}_l$ is applied for dilation. This is formally expressed as a convolution between two Gaussians and can eventually be written as $\mathcal{M}_k$:

$$\begin{aligned} \mathcal{G}_k^{2D}(x)_{3DGS} &= \sqrt{\frac{|\Sigma_k + \sigma_l \cdot \mathbf{I}|}{|\Sigma_k|}} (\mathcal{G}_k(\mathbf{p}_k, \Sigma_k) * \mathcal{G}_l(\mathbf{p}_k, \sigma_l \cdot \mathbf{I}))(x) \\ &= \mathcal{M}_k(\mathbf{p}_k, \Sigma_k + \sigma_l \cdot \mathbf{I})(x) \end{aligned} \quad (2)$$

Here, σ_l is a fixed hyperparameter that controls the scale of $\mathcal{G}_l$, while $\mathbf{I}$ is 2D unit matrix. Here the problem of 3DGS is that σ_l is fixed as 0.3, which approximately leads to $\sqrt{0.3} \times 3 \approx 1.64$ dilation as shown in Fig. 3.

Solution. During rendering, the scale of the Gaussian primitive in camera space should remain constant, regardless of any changes in rendering frequency. This is achieved by calculating the ratio $r = \frac{\Delta R_p}{\Delta D_c}$. ΔR_p is the resolution ratio between training and rendering, solving the problem described in Fig. 3-(a). ΔD_c is the distance(focal length) ratio between the rendering camera and the closest orientated training camera, solving the problem described in Fig. 3-(b).

$$\begin{aligned} \mathcal{G}_k^{2D}(x, r)_{SA-GS} &= \sqrt{\frac{|\Sigma_k + \sigma_l r^2 \cdot \mathbf{I}|}{|\Sigma_k|}} (\mathcal{G}_k(\mathbf{p}_k, \Sigma_k) * r \mathcal{G}_l(\mathbf{p}_k, \sigma_l \cdot \mathbf{I}))(x) \\ &= \mathcal{M}_k(\mathbf{p}_k, \Sigma_k + \sigma_l r^2 \cdot \mathbf{I})(x) \end{aligned} \quad (3)$$

Via this operation (named as 2D scale-adaptive filter), we can ensure a consistent scale and distribution of 2D Gaussian projections in camera space at different rendering settings, such that we can match the training settings.

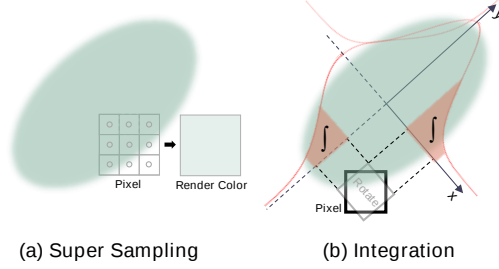


Fig. 5: Super Sampling and Integration applied on a Gaussian primitive. Our super sampling method, denoted as (a), involves dividing each pixel into 9 sub-pixels when traversing the order-sorted Gaussians within a tile. Each sub-pixel independently undergoes α -blending and weights the Gaussian spherical harmonic coefficient according to the sub-pixel sampling locations. (b) is our integration method that diagonalizes the Gaussian covariance matrix by pixel rotation. This decomposes the integration operation into the product of two marginal Gaussian distributions.

3.2 Making Conventional Anti-Aliasing Great Again for Gaussians

Our 2D scale-adaptive filter ensures that the Gaussian distribution remains consistent via matching arbitrary rendering settings with the training setting. Only after this scale adaptation, we can tackle the aliasing issue. Specifically, due to the Nyquist sampling theorem [15], the image will show aliasing effects as the rendering frequency decreases. In the conventional graphics literature, there are two techniques that can be used to deal with this problem of aliasing: super-sampling and pre-filtering. 3DGS [10] cannot leverage these old techniques to deal with anti-aliasing due to the aforementioned issue of Gaussian scale mismatch. Our method maintains consistent Gaussian scale across different resolutions, allowing for effective removal of scene aliasing. We chose to use super-sampling and its limiting case, integration, instead of pre-filtering, because the pre-filtering affects the α -blending procedure of 3DGS which may be a future pursuit.

Super-sampling Given a pixel P_t , when traversing Gaussian primitives that have been order-sorted within a tile, we compute the distance between the centers of the $S \times S$ sub-pixels and the center of the Gaussian primitive separately, as shown in Fig. 5(a). These sub-pixels have independent α -blending processes and cumulative transparency T_s . The color of a pixel C_t is determined by averaging the color of these sampled sub-pixels:

$$C_t = \frac{1}{S^2} \sum_{i=1}^G \sum_{s=1}^{S^2} \alpha_s^i \times T_s^i \times \mathcal{F}(SH_i) \quad (4)$$

$$T_s^i = \begin{cases} 1, i = 1 \\ \prod_{j=1}^{i-1} (1 - \alpha_s^j), i > 1 \end{cases}$$

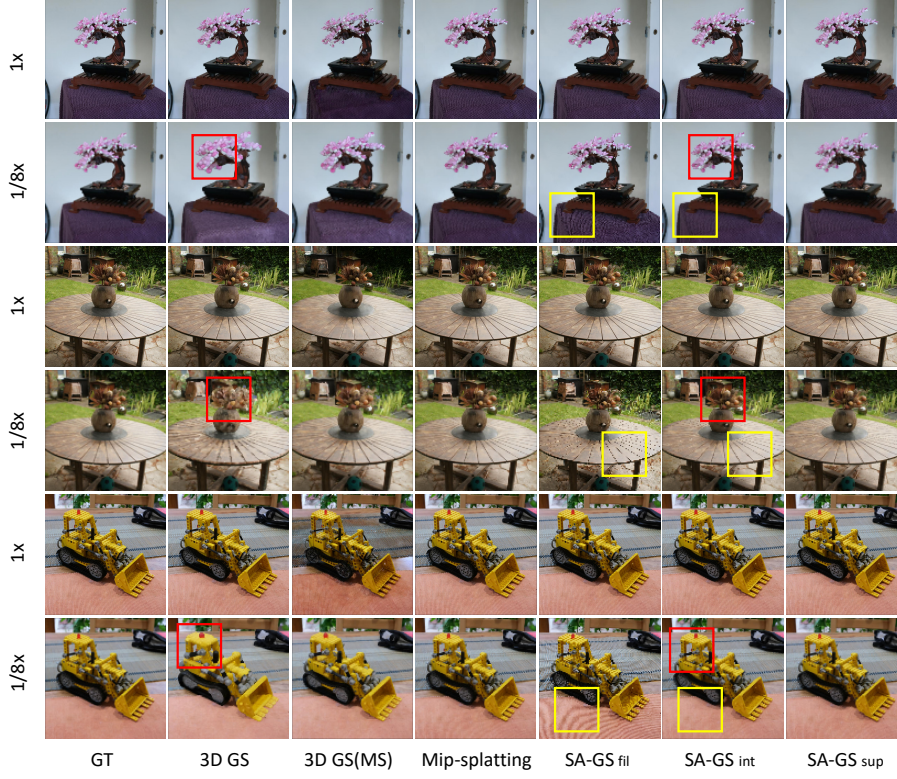


Fig. 6: Single-scale Training and multi-scale testing on the Mip-NeRF 360 Dataset [2] for zoom-out effect. 3DGS [10] has dilation artefacts (red boxes) at low resolutions. Our 2D scale-adaptive filter maintains the Gaussian scale consistency at low resolutions, and the super-sampling and integration methods further remove the aliasing artefacts (yellow boxes), yielding results that surpass Mip-Splatting [24].

where G is the number of Gaussian primitives on the z-buffer, α_s^i is the opacity calculated based on the distance between s_{th} sub-pixel and i_{th} Gaussian, and SH_i is the spherical harmonic coefficient of the i_{th} Gaussian. The function $\mathcal{F}(\cdot)$ is used for the spherical harmonic coefficient to color conversion. For fast convergence, we set $S = 3$ in all experimental settings.

Integration When the super-sampling hyperparameter of S goes to infinity, it becomes integration. Consider a single Gaussian’s projection on the 2D camera plane, a 2D Gaussian whose PDF we can represent as $f(x, y)$, where x and y are coordinates on the camera plane, and we take axes of the 2D Gaussian Projection as the coordinate system axes, thus making the correlation of the projected Gaussians zero. As the correlation is zero we have that $f(x, y)$ can further be factored into the product $g_x(x)g_y(y)$, where $g(t) = \frac{\exp(-\frac{1}{2}(\frac{t}{\sigma})^2)}{\sqrt{2\pi}\sigma}$. Let $\Phi(t) = \int_{-\infty}^t g(t)dt$ (subscripts x or y omitted) be the Gaussian integral. Let the

region inside the pixel be P . Hence when calculating α during the traversal of the Gaussian z-buffer, we need to find the following double integral:

$$\alpha = \iint_P f(x, y) dx dy \quad (5)$$

Axis-aligned Case. When the axes of the pixel are parallel to the 2D Gaussian Projection’s axes, the evaluation of (5) is simple because it can be calculated as the product of two Gaussian single integrals:

$$\begin{aligned} [t] \iint_P f(x, y) dx dy &= \left(\int_{P_x} g_x(x) dx \right) \left(\int_{P_y} g_y(y) dy \right) \\ &= (\Phi_x(P_{x \max}) - \Phi_x(P_{x \min})) (\Phi_y(P_{y \max}) - \Phi_y(P_{y \min})) \quad (6) \end{aligned}$$

where $P_{x \min}$ and $P_{x \max}$ are marginal Gaussian intervals on the x-axis, $P_{y \min}$ and $P_{y \max}$ are the same on the y-axis. However, when the pixel’s sides is not aligned with the axes, x and y will be related. In the evaluation of (5) this is reflected as the inner integral containing variable instead of constant limits, and therefore the factorization carried out in (6) will no longer be applicable.

Pixel Rotation. In our implementation, we solve this problem by *rotating the pixel* such that it aligns with the Gaussian’s axes, as shown in Fig. 5(b). Thus, the integral can be easily computed using this approach as described in (6). However, rotating the pixel causes a deviation between the integral region and the original pixel region. We prove that for any pixel close enough to the center of the Gaussian to be affected during α -blending, there exists a theoretical upper bound for the error. We also verify through numerical experimentation that this is empirically a good approximation. Detailed proofs and experimental results are provided in the supplementary material.

	PSNR $\uparrow$						SSIM $\uparrow$						LPIPS $\downarrow$					
	1 Res.	1/2 Res.	1/4 Res.	1/8 Res.	Avg.		1 Res.	1/2 Res.	1/4 Res.	1/8 Res.	Avg.		1 Res.	1/2 Res.	1/4 Res.	1/8 Res.	Avg.	
3DGS [10]	29.26	26.84	22.16	19.63	24.47		0.877	0.863	0.726	0.612	0.769		0.185	0.148	0.198	0.223	0.189	
3DGS(MS) [10]	20.11	23.50	32.51	23.72	24.96		0.604	0.774	0.956	0.832	0.792		0.389	0.212	0.051	0.118	0.192	
Mip-Splatting [24]	29.26	30.23	30.56	29.61	29.91		0.875	0.909	0.929	0.934	0.911		0.187	0.116	0.080	0.066	0.113	
SA-GS _{fil} (ours)	29.26	29.80	28.29	25.58	28.23		0.877	0.901	0.875	0.809	0.866		0.185	0.123	0.126	0.171	0.151	
SA-GS _{int} (ours)	29.14	30.06	30.13	28.81	29.53		0.873	0.905	0.921	0.919	0.904		0.188	0.118	0.086	0.078	0.118	
SA-GS _{sup} (ours)	29.26	30.45	31.75	32.53	31.00		0.876	0.912	0.938	0.951	0.919		0.186	0.114	0.073	0.053	0.106	

Table 1: Single-scale training and multi-scale testing on the Mip-NeRF 360 Dataset [2]. Except for 3DGS(MS) [10], which is trained on multiple scales, all other methods are trained on the largest scale ($1\times$) and evaluated across four scales ($1\times$, $1/2\times$, $1/4\times$, and $1/8\times$), simulating zoom-out effect. SA-GS_{fil} means we only use the 2D scale-adaptive filter. All our variants significantly surpass 3DGS at low resolutions, and SA-GS_{sup} yields better results than Mip-Splatting [24].

4 Experiments

We first present the implementation details of SA-GS. We then evaluate its performance on the the unbounded Mip-NeRF 360 dataset [2] and the bounded Blender dataset [13]. Finally, we discuss some limitations of our approach.

4.1 Implementation

In our SA-GS framework, an advantage over Mip-splatting is that modifications are exclusively performed during the testing phase of 3DGS [10]. Therefore, in the training phase, we kept all the settings of the original Gaussian model, including training rounds, hyperparameter settings and densification strategy. In our super-sampling method, we synchronize all pixel threads⁸ within the same block before performing the alpha calculation to initialize the cumulative transparency in the shared memory. In our integration method, we project the pixel corner points towards the Gaussian axis to obtain an interval of Gaussian distribution for the marginal distributions. To simulate the pixel rotation, we multiply the pixel region area by a weight of $\frac{1}{\sin\theta + \cos\theta}$ to compensate the error caused by rotation, where θ is the angle between the long Gaussian axis and the x-axis of the pixel plane. Please refer to the supplementary material for details.

4.2 Evaluation on the Mip-NeRF 360 Dataset

Single-scale Training and Multi-scale Testing: We simulated the effect of zoom-out and zoom-in on this dataset and retrained all the baseline models for comparison. The same setup of Mip-Splatting [24] is used. Specifically, for zoom-out, we trained 3DGS [10] using full resolution images and then plug in our method to test on progressively lower resolutions ($1\times$, $1/2\times$, $1/4\times$, $1/8\times$). For zoom-in, we trained 3DGS [10] using $1/8\times$ resolution images and then plug in our method to test on progressively higher resolutions ($1\times$, $2\times$, $4\times$, $8\times$). Table 1 and Table 2 show the quantitative results for these two experiments.

When **zooming out**, our method gives comparable results at the training resolution and better performance at lower resolutions. The (heuristic and fixed) dilation operation of 3DGS [10] leads to severe degradation at low resolutions. Mip-Splatting [24] replaces 3DGS’s dilation operation with 3D smoothing and a 2D Mip filter, but still suffers from scale ambiguity at different resolutions. As depicted in Figure 6, our 2D scale-adaptive filter ensures that the Gaussian distribution is consistent with the training settings. Our integration and super-sampling modules further enhance the anti-aliasing effect. The super-sampling version of our method gives the best results for this setting, which exceeds Mip-Splatting [24] by **1.1dB** in PSNR (see Tab. 1).

When **zooming in**, 3DGS [10] shows severe erosion artefacts at high resolution. Mip-Splatting [24] uses the 3D smoothing filter to reduce the effects of scale ambiguity of 2D filtering. As depicted in Figure 7, our method keeps the

⁸ In 3DGS, each pixel is associated with a thread.

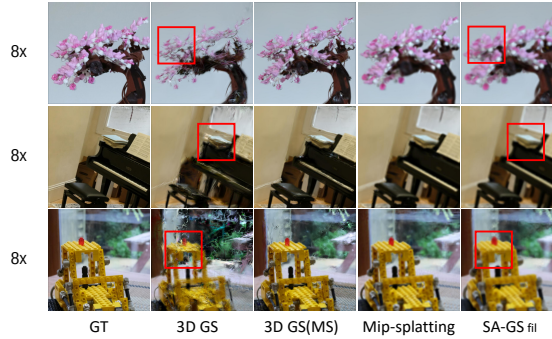


Fig. 7: Single-scale Training and multi-scale testing on the Mip-NeRF 360 Dataset [2] for zoom-in effect. 3DGS [10] sees erosion artefacts (red boxes) at high resolution. By using only 2D scale-adaptive filters, we achieve a stable quality improvement at high resolutions without any re-training.

Gaussian scales consistent and greatly reduces erosion artefacts. Note that integration and super sampling are only designed to address the decrease in sampling frequency (zoom-out). The most significant contribution is made by 2D scale-adaptive filter, which produces results comparable to Mip-Splatting [24].

	PSNR $\uparrow$						SSIM $\uparrow$						LPIPS $\downarrow$					
	1 Res.	2 Res.	4 Res.	8 Res.	Avg.		1 Res.	2 Res.	4 Res.	8 Res.	Avg.		1 Res.	2 Res.	4 Res.	8 Res.	Avg.	
3DGS [10]	33.96	22.47	18.69	17.32	23.11		0.974	0.747	0.514	0.460	0.674		0.028	0.204	0.410	0.504	0.286	
3DGS(MS) [10]	23.72	32.51	23.50	20.11	24.96		0.832	0.956	0.774	0.604	0.792		0.118	0.051	0.212	0.389	0.192	
Mip-Splatting [24]	34.62	28.86	25.99	24.95	28.60		0.977	0.872	0.718	0.641	0.802		0.025	0.154	0.318	0.430	0.232	
SA-GS _{fil} (ours)	33.96	27.89	25.32	24.40	27.89		0.974	0.840	0.677	0.615	0.777		0.028	0.189	0.360	0.465	0.260	

Table 2: Single-scale training and multi-scale testing on the Mip-NeRF 360 Dataset [2]. Except for 3DGS(MS) [10], which is trained on multiple scales, all other re-training methods are trained on the $1/8$ scale ($1\times$) and evaluated across four scales ($1\times$, $2\times$, $4\times$, and $8\times$), simulating zoom-in effect. SA-GS_{fil} means we only use 2D Scale-adaptive Filter. Our method significantly surpasses 3DGS [10] at high resolutions and produce results comparable to Mip-Splatting [24]. Note that the performance of SA-GS_{fil} is achieved without re-training.

4.3 Evaluation on the Blender Dataset

Multi-scale Training and Multi-scale Testing: Following Mip-splatting, all baseline models were trained using multi-scale data from the *train + test* section of the dataset and evaluated with multiscale data from the *val* section. We follow the image sampling ratio in Mip-Splatting [24] to train the 3DGS [10]. Our quantitative evaluation is shown in Table 3. Our approach yields comparable results with Mip-Splatting [24], and we use vanilla 3DGS [10] on multi-scale training only and do not need to modify the training procedure. Meanwhile, our approach significantly outperforms 3DGS [10], demonstrating stable performance at different resolutions. 3DGS [10] performance degrades as resolution decreases, even in the case that it is trained on multi-scale.

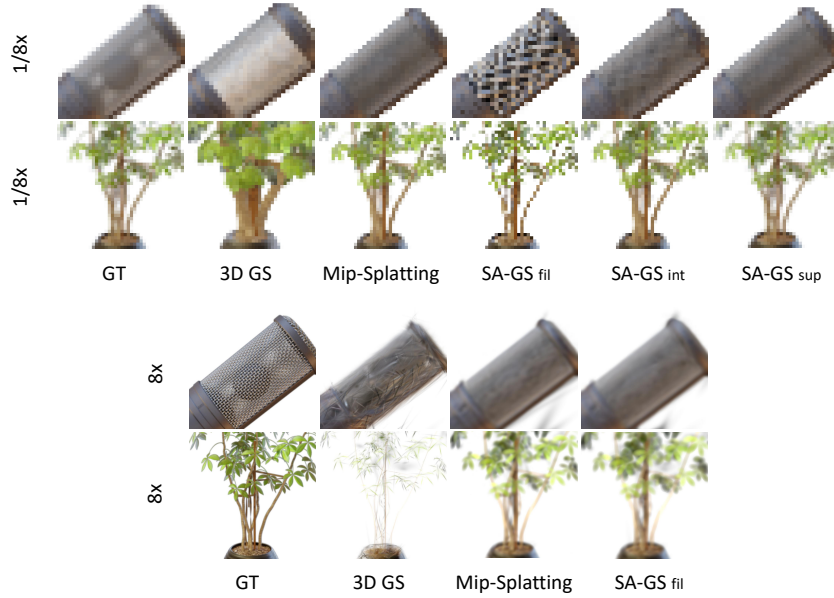


Fig. 8: Single-scale Training and multi-scale testing on the Blender [13] Dataset for zoom-in and zoom out effect. Our 2D scale-adaptive filter maintains the consistency of the 2D Gaussian projection when zooming out. Moreover, it alleviates erosion artefacts and does not modify the training procedure when zooming in. We use super-sampling and integration methods to further address aliasing.

	PSNR $\uparrow$					SSIM $\uparrow$					LPIPS $\downarrow$				
	1 Res.	$1/2$ Res.	$1/4$ Res.	$1/8$ Res.	Avg.	1 Res.	$1/2$ Res.	$1/4$ Res.	$1/8$ Res.	Avg.	1 Res.	$1/2$ Res.	$1/4$ Res.	$1/8$ Res.	Avg.
3DGS [10]	31.51	32.66	31.21	28.25	30.91	0.962	0.972	0.968	0.945	0.962	0.050	0.031	0.030	0.045	0.039
Mip-Splatting [24]	32.81	34.49	35.45	35.50	34.56	0.967	0.977	0.983	0.988	0.979	0.035	0.019	0.013	0.010	0.019
<i>SA-GS_{int}</i> (ours)	30.84	32.71	34.26	32.80	32.65	0.956	0.969	0.978	0.979	0.971	0.055	0.031	0.021	0.019	0.032
<i>SA-GS_{sup}</i> (ours)	30.80	32.67	35.06	35.77	33.58	0.956	0.969	0.980	0.985	0.973	0.056	0.032	0.020	0.014	0.031

Table 3: Multi-scale Training and Multi-scale Testing on the Blender dataset [13]. Our training-free approach yields comparable results compared to Mip-Splatting [24]. Meanwhile, our approach significantly outperforms 3DGS [10] and demonstrates stable performance at different resolutions.

	PSNR $\uparrow$					SSIM $\uparrow$					LPIPS $\downarrow$				
	1 Res.	$1/2$ Res.	$1/4$ Res.	$1/8$ Res.	Avg.	1 Res.	$1/2$ Res.	$1/4$ Res.	$1/8$ Res.	Avg.	1 Res.	$1/2$ Res.	$1/4$ Res.	$1/8$ Res.	Avg.
3DGS	35.10	27.91	22.42	18.76	26.05	0.974	0.949	0.862	0.736	0.880	0.029	0.033	0.069	0.133	0.066
3DGS(MS)	31.51	32.66	31.21	28.25	30.91	0.962	0.972	0.968	0.945	0.962	0.050	0.031	0.030	0.045	0.039
Mip-Splatting	34.59	35.11	31.98	28.14	32.45	0.973	0.979	0.975	0.952	0.970	0.032	0.019	0.019	0.029	0.025
<i>SA-GS_{fil}</i> (ours)	34.60	34.33	31.02	27.59	31.89	0.973	0.977	0.968	0.947	0.966	0.031	0.022	0.036	0.067	0.039
<i>SA-GS_{int}</i> (ours)	34.35	34.39	30.99	26.89	31.65	0.972	0.978	0.971	0.940	0.965	0.032	0.020	0.023	0.039	0.029
<i>SA-GS_{sup}</i> (ours)	34.49	36.58	37.50	35.64	36.06	0.972	0.980	0.985	0.985	0.981	0.032	0.018	0.014	0.013	0.019

Table 4: Single-scale training and Multi-scale testing on the Blender dataset [13] for zoom-out effect. We use the same experiment protocol and model naming with the Mip-NeRF 360 [2] experiment (of Table 1). Our method outperforms 3DGS [10], while *SA-GS_{sup}* significantly surpasses all previous works.

Single-scale Training and Multi-scale Testing: We maintain a experiment protocol consistent with the Mip-NeRF 360 [2] experiment mentioned above (Section 4.2) to evaluate the zoom-out and zoom-in effects. We also keep the same data split as in the multi-scale training experiment described above. Table 4 and Table 5 show the quantitative results for zoom-out and zoom-in effects. The qualitative results are shown in Fig. 8.

For zoom-out, our method achieves performance close to 3DGS [10] at training resolution and a steady increase in performance at lower resolutions. The super-sampling version of our method $SA-GS_{sup}$ significantly outperforms Mip-Splatting [24] to achieve performance with a gain of **3.61dB** in PSNR in this setting (see Tab. 4). **For zoom-in,** Our 2D scale-adaptive filter achieves comparable results to Mip-Splatting [24], and our method is training-free.

	PSNR $\uparrow$					SSIM $\uparrow$					LPIPS $\downarrow$				
	1 Res.	2 Res.	4 Res.	8 Res.	Avg.	1 Res.	2 Res.	4 Res.	8 Res.	Avg.	1 Res.	2 Res.	4 Res.	8 Res.	Avg.
3DGS [10]	36.97	24.33	21.01	19.63	25.44	0.988	0.886	0.820	0.821	0.879	0.013	0.065	0.130	0.159	0.092
3DGS(MS) [10]	28.25	31.21	32.66	31.51	30.91	0.945	0.968	0.972	0.962	0.962	0.045	0.030	0.031	0.050	0.039
Mip-Splatting [24]	36.50	30.72	27.81	26.51	30.39	0.986	0.959	0.920	0.893	0.939	0.015	0.048	0.099	0.130	0.073
$SA-GS_{fil}$ (ours)	35.74	30.38	27.63	26.36	30.03	0.984	0.953	0.912	0.885	0.933	0.016	0.059	0.111	0.141	0.082

Table 5: Single-scale training and Multi-scale testing on the Blender dataset [13] for zoom-in effect. We use the experiment and model naming with the Mip-NeRF 360 [2] experiment (of Table 2). Our methods yields comparable results with Mip-Splatting [24]. $SA-GS_{fil}$ achieves this performance while being training-free.

5 Conclusion

We present SA-GS, a training-free framework that can seamlessly integrate with 3DGS [10] to enhance its anti-aliasing ability at arbitrary rendering frequencies. Specifically, we propose a 2D scale-adaptive filter, which maintains the 2D Gaussian projection scale’s consistency under different rendering settings. In addition, we employ conventional anti-aliasing techniques, super-sampling, and integration to significantly reduce image aliasing at lower sampling rates. SA-GS demonstrates superior or comparable performance to the state-of-the-art, as extensive validation is performed on both bounded and unbounded scenarios.

Limitations. Our method has no computational burden when zooming in, but when zooming out, the application of integration and super-sampling methods increases the rendering time. Due to shared memory, the elapsed time for super-sampling is comparable to that of integration, making it 15%~20% slower than the vanilla 3DGS [10]. However, integration can still be optimized (approximation calculations or table lookups), leading to further speedups. Overall, our approach receives a significant anti-aliasing performance boost with minimal trade-offs.

References

1. Barron, J.T., Mildenhall, B., Tancik, M., Hedman, P., Martin-Brualla, R., Srinivasan, P.P.: 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) [4](#), [6](#)
2. Barron, J.T., Mildenhall, B., Verbin, D., Srinivasan, P.P., Hedman, P.: 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) [4](#), [6](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#)
3. Barron, J.T., Mildenhall, B., Verbin, D., Srinivasan, P.P., Hedman, P.: Zip-nerf: Anti-aliased grid-based neural radiance fields. arXiv preprint arXiv:2304.06706 (2023) [4](#), [6](#)
4. Chen, W., Ling, H., Gao, J., Smith, E., Lehtinen, J., Jacobson, A., Fidler, S.: Learning to predict 3d objects with an interpolation-based differentiable renderer. Advances in neural information processing systems **32** (2019) [4](#)
5. Fridovich-Keil, S., Yu, A., Tancik, M., Chen, Q., Recht, B., Kanazawa, A.: Plenoxels: Radiance fields without neural networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 5501–5510 (2022) [4](#)
6. Guédon, A., Lepetit, V.: Sugar: Surface-aligned gaussian splatting for efficient 3d mesh reconstruction and high-quality mesh rendering. arXiv preprint arXiv:2311.12775 (2023) [4](#)
7. Hu, W., Wang, Y., Ma, L., Yang, B., Gao, L., Liu, X., Ma, Y.: Tri-miprf: Tri-mip representation for efficient anti-aliasing neural radiance fields. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 19774–19783 (2023) [4](#)
8. Hu, W., Wang, Y., Ma, L., Yang, B., Gao, L., Liu, X., Ma, Y.: Tri-miprf: Tri-mip representation for efficient anti-aliasing neural radiance fields (2023) [6](#)
9. Kato, H., Ushiku, Y., Harada, T.: Neural 3d mesh renderer. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 3907–3916 (2018) [4](#)
10. Kerbl, B., Kopanas, G., Leimkühler, T., Drettakis, G.: 3d gaussian splatting for real-time radiance field rendering (2023) [1](#), [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#)
11. Liu, S., Li, T., Chen, W., Li, H.: Soft rasterizer: A differentiable renderer for image-based 3d reasoning. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 7708–7717 (2019) [4](#)
12. Mildenhall, B., Srinivasan, P.P., Ortiz-Cayon, R., Kalantari, N.K., Ramamoorthi, R., Ng, R., Kar, A.: Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. ACM Transactions on Graphics (TOG) **38**(4), 1–14 (2019) [2](#)
13. Mildenhall, B., Srinivasan, P.P., Tancik, M., Barron, J.T., Ramamoorthi, R., Ng, R.: Nerf: Representing scenes as neural radiance fields for view synthesis. Communications of the ACM **65**(1), 99–106 (2021) [2](#), [4](#), [6](#), [11](#), [13](#), [14](#)
14. Müller, T., Evans, A., Schied, C., Keller, A.: Instant neural graphics primitives with a multiresolution hash encoding. ACM Transactions on Graphics (ToG) **41**(4), 1–15 (2022) [4](#)
15. Nyquist, H.: Certain topics in telegraph transmission theory. Transactions of the American Institute of Electrical Engineers **47**(2), 617–644 (1928) [3](#), [8](#)
16. Peng, Z., Hu, W., Shi, Y., Zhu, X., Zhang, X., Zhao, H., He, J., Liu, H., Fan, Z.: Synctalk: The devil is in the synchronization for talking head synthesis. arXiv preprint arXiv:2311.17590 (2023) [2](#)

17. Rückert, D., Franke, L., Stamminger, M.: Adop: Approximate differentiable one-pixel point rendering. *ACM Transactions on Graphics (ToG)* **41**(4), 1–14 (2022) [4](#)
18. Sainz, M., Pajarola, R.: Point-based rendering techniques. *Computers & Graphics* **28**(6), 869–879 (2004) [3](#)
19. Straub, J., Whelan, T., Ma, L., Chen, Y., Wijmans, E., Green, S., Engel, J.J., Mur-Artal, R., Ren, C., Verma, S., Clarkson, A., Yan, M., Budge, B., Yan, Y., Pan, X., Yon, J., Zou, Y., Leon, K., Carter, N., Briales, J., Gillingham, T., Mueggler, E., Pesqueira, L., Savva, M., Batra, D., Strasdat, H.M., Nardi, R.D., Goesele, M., Lovegrove, S., Newcombe, R.: The Replica dataset: A digital replica of indoor spaces. *arXiv preprint arXiv:1906.05797* (2019) [2](#)
20. Wei, Y., Wang, Z., Lu, Y., Xu, C., Liu, C., Zhao, H., Chen, S., Wang, Y.: Editable scene simulation for autonomous driving via collaborative llm-agents. *arXiv preprint arXiv:2402.05746* (2024) [2](#)
21. Wu, Z., Liu, T., Luo, L., Zhong, Z., Chen, J., Xiao, H., Hou, C., Lou, H., Chen, Y., Yang, R., et al.: Mars: An instance-aware, modular and realistic simulator for autonomous driving. In: *CAAI International Conference on Artificial Intelligence*. pp. 3–15. Springer (2023) [2](#)
22. Xie, T., Zong, Z., Qiu, Y., Li, X., Feng, Y., Yang, Y., Jiang, C.: Physgaussian: Physics-integrated 3d gaussians for generative dynamics. *arXiv preprint arXiv:2311.12198* (2023) [4](#)
23. Xu, Q., Xu, Z., Philip, J., Bi, S., Shu, Z., Sunkavalli, K., Neumann, U.: Pointnerf: Point-based neural radiance fields. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* pp. 5428–5438 (2022), <https://api.semanticscholar.org/CorpusID:246210101> [4](#)
24. Yu, Z., Chen, A., Huang, B., Sattler, T., Geiger, A.: Mip-splatting: Alias-free 3d gaussian splatting. *arXiv preprint arXiv:2311.16493* (2023) [1](#), [2](#), [3](#), [5](#), [6](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#)
25. Yuan, S., Zhao, H.: Slimmerf: Slimmable radiance fields. *arXiv preprint arXiv:2312.10034* (2023) [2](#)
26. Zhou, Q., Li, W., Jiang, L., Wang, G., Zhou, G., Zhang, S., Zhao, H.: Pad: A dataset and benchmark for pose-agnostic anomaly detection. *Advances in Neural Information Processing Systems* **36** (2024) [2](#)
27. Zhu, Z., Chen, Y., Wu, Z., Hou, C., Shi, Y., Li, C., Li, P., Zhao, H., Zhou, G.: Latitude: Robotic global localization with truncated dynamic low-pass filter in city-scale nerf. In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. pp. 8326–8332. IEEE (2023) [2](#)

SA-GS: Scale-Adaptive Gaussian Splatting for Training-Free Anti-Aliasing

Supplementary Material

In this supplementary material, we first present the implementation details of the integration in A. Next, we proof the theoretical upper bound for the rotational error in B.1 and present the results of numerical experiments in B.2. Additionally, we present ablation studies of SA-GS in C. Finally, we report additional quantitative and qualitative results in D.

A Implementation Details of Integration

As stated in the main text, we simplify the integral operation by *rotating the pixel*. In the concrete implementation, the rotation is simulated by projection. Denote the two normalised eigenvectors of the 2D Gaussian distribution as $\mathbf{v}_{long}$ and $\mathbf{v}_{short}$, corresponding to larger and smaller eigenvalues respectively. We project the four corner points of the pixel in the direction of $\mathbf{v}_{long}$ and $\mathbf{v}_{short}$ using dot product:

$$\begin{aligned}x_{min} &= \min(P_{lu,lb,ru,rb} \cdot \mathbf{v}_{long}) \\x_{max} &= \max(P_{lu,lb,ru,rb} \cdot \mathbf{v}_{long}) \\y_{min} &= \min(P_{lu,lb,ru,rb} \cdot \mathbf{v}_{short}) \\y_{max} &= \max(P_{lu,lb,ru,rb} \cdot \mathbf{v}_{short})\end{aligned}\tag{1}$$

where $P_{lu,lb,ru,rb}$ are the coordinates of the four corner points of the pixel, specifically the left upper, left lower, right upper and right lower. $x_{max} - x_{min}$ or $y_{max} - y_{min}$ is equivalent to the side length of the rotated pixel.

However, when rotating pixels in the above manner, their edge lengths always increase, resulting in an area that is larger than the correct range. On the other hand, restricting the area of the rotated pixel to be inside the original pixel causes a loss of integral area. To balance this issue, we scale the pixel area before projection to ensure that the rotated pixel area is equal to the original pixel area. Specifically, we multiply the original pixel edge lengths by the $\frac{1}{\sin\theta + \cos\theta}$, as illustrated in Fig. 1.

B Analysis of Rotational Errors

Although rotating the pixel simplifies the integral calculation, it inevitably introduces an error, even if the area of the pixel after rotation is equal to the original. We prove that for any pixel close enough to the center of the Gaussian to be affected during α -blending, there exists a theoretical upper bound for the error. Additionally, We verify through numerical experimentation that this is empirically a good approximation.

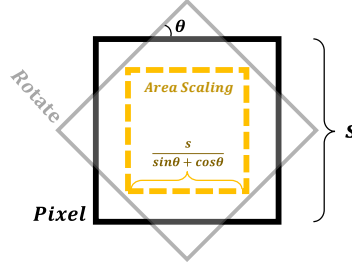


Fig. 1: Area scaling when rotating pixels. In integration method, the pixel area is scaled before projection to ensure that the projected (rotated) pixel area is equal to the original pixel area. θ is the rotation angle of the pixel.

B.1 Theoretical Upper Bound

Normalization and Rotation Let the center of the pixel be at coordinates (x_c, y_c) , and the side length of the pixel be l . Let the pixel have a counterclockwise tilt of θ with respect to the standard x-axis. The first problem we need to solve would be to eliminate the tilt in order to simplify the double integral. In order to do so, we would need to apply a rotation transformation to the Bivariate Gaussian Distribution without changing its main form. Hence we first normalize it into a Bivariate Normal Distribution via scaling.

We construct $x^*, y^* = \frac{x}{\sigma_x}, \frac{y}{\sigma_y}$ such that we have the Normal Distribution $g_x^*(x^*) = \frac{\exp(-\frac{1}{2}x^{*2})}{\sqrt{2\pi}} = \sigma_x g_x(x)$ (g_y^* is analogous). Hence we have:

$$f^*(x^*, y^*) = \sigma_x \sigma_y g_x(x) g_y(y) = \sigma_x \sigma_y f(x, y) \quad (2)$$

The rotational symmetry of the Bivariate Normal Distribution now allows us to rotate the pixel clockwise by θ with respect to the origin without changing the integral (this does not affect the integral). Now the new pixel region, P^* , is a parallelogram with top and bottom edges parallel to the x-axis. Let the four corners (labeled in order analogously to quadrants) be labeled respectively $P_1(x_1, y_1)$, $P_2(x_2, y_1)$, $P_3(x_3, y_3)$, $P_4(x_4, y_3)$, and the slope of the edges P_1P_4 and P_2P_3 be k . WLOG let $y_1 \geq 0$ (otherwise conduct a reflection across the x-axis).

Double Integration In order to find bounds for $\iint_{P^*} f^*(x^*, y^*) dx^* dy^*$, if we keep the current boundary for P^* , we will need to integrate Φg , which would be unrepresentable. In order to find the theoretical range of the error, we instead try to fix the sliced Gaussian distribution by fixing y^* . Combining with (2), we readily have the following bounds:

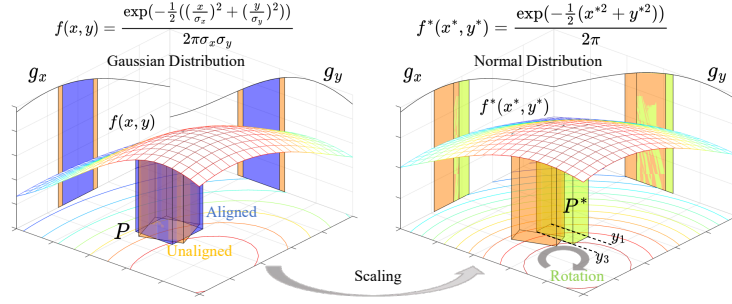


Fig. 2: Visual demonstration of our theoretical analysis. We scale the general Gaussian distribution to a standard normal distribution to estimate an upper bound on the error between the rotated pixel and the original pixel.

$$\iint_P f(x, y) dx dy > \frac{1}{\sigma_x \sigma_y} \int_{y_3}^{y_1} \int_{\frac{y^* - y_1}{k} + x_2}^{\frac{y - y_1}{k} + x_1} f^*(x, \max\{0, y_3\}) dx dy \quad (3)$$

$$\iint_P f(x, y) dx dy < \frac{1}{\sigma_x \sigma_y} \int_{y_3}^{y_1} \int_{\frac{y^* - y_1}{k} + x_2}^{\frac{y - y_1}{k} + x_1} f^*(x, \max\{y_1, |y_3|\}) dx dy \quad (4)$$

Fig. 2 presents a visualisation of the theoretical analysis. Obviously, when the coordinates on the pixel are bounded by a constant times the respective standard deviations, coordinates of P_i will also be bounded by the constant. Further, k is only affected by θ , σ_x , and σ_y . Hence the original double integral is bounded in our approximation, and so is its error.

B.2 Numerical Experiments

We verify the above theoretical analysis by numerical experiments. Adopting the notation established in B.1, we denote (x_c, y_c) as the coordinates of the pixel centroid, l as the pixel side length, θ as the angle defining the counterclockwise rotation of the pixel, and σ_x and σ_y as the standard deviations corresponding to the directions of the two principal eigenvectors of the Gaussian distribution, respectively.

We adopted the framework of parametric sensitivity analysis for our experimental setup, by fixing certain parameters while sampling within reasonable ranges for the others. This approach aims to quantify the differences between the original pixel integrals and their counterparts after rotation. Specifically, we set $l = 1$ and $x_c = 0$, allowing θ and y_c to uniformly sample six values from their respective intervals $[0, \frac{\pi}{4}]$ and $[0.05, 0.25]$, thereby generating 36 sub-tables, as depicted in Fig. 3. In each sub-table, the parameters σ_x and σ_y delineate the horizontal and vertical axes, correspondingly, with both parameters uniformly

sampling 30 values from the interval $[0.15, 3.77]$. This interval encompasses the core portion of the Gaussian distribution represented in the tile.

The final numerical experiment give an average relative error of **0.51%**. Since most of the errors are 0 or close to 0, for ease of visualisation, we convert all the errors to the (0,1) range and widen the differences in the region close to 0 by $y = \frac{1}{1+e^{-800x}}$, as shown in Fig. 3. It can be seen that the error increases as the anisotropy of the Gaussian distribution becomes more pronounced, and that the error range increases as θ increases. However, the overall error values calculated are small, confirming that our method is a good estimation.

C Ablation

In this section, we evaluate the effectiveness of 2D scale-adaptive filter(C.1) and anti-aliasing methods(C.2). Additionally, we present corresponding qualitative and quantitative results.

C.1 Effectiveness of the 2D Scale-adaptive Filter

To evaluate the effectiveness of the 2D Scale-adaptive filter, we perform ablation studies with single-scale training and multi-scale testing(zoom-out and zoom-in) on both the Mip-NeRF 360 dataset and the Blender dataset. The quantitative results are presented in Table 1, Table 2, Table 3, and Table 4.

Due to the scale consistency across rendering settings brought about by the 2D scale-adaptive filter, we get a very noticeable performance improvement over 3DGS in both zoom-out and zoom-in scenarios. 3DGS expands or shrinks at different rendering frequencies, thus exacerbating the aliasing effect, as illustrated in Fig. 4 and Fig. 5.

C.2 Effectiveness of the Anti-aliasing Methods

To evaluate the effectiveness of the anti-aliasing methods(integration and super-sampling), we perform ablation studies with single-scale training and multi-scale testing for zoom-out effect on both the Mip-NeRF 360 dataset and the Blender dataset. The quantitative results are presented in Table 1 and Table 3. Note that the integration and super-sampling methods are intended solely for decreasing rendering frequency. Therefore, we do not focus on analysing their performances in the zoom-in case. Table 2 and Table 4 demonstrate that they perform comparably with 3DGS.

The integration and super-sampling methods are ineffective when the 2D scale-adaptive filter fails due to scale inconsistency in the 3DGS. However, when the 2D scale-adaptive filter is operational, these methods can further enhance the anti-aliasing ability of the scene, as illustrated in Fig. 4. In summary, we conclude that 3DGS does not provide a more robust representation of the scene using conventional anti-aliasing methods, but our 2D scale-adaptive filter completely removes this limitation.

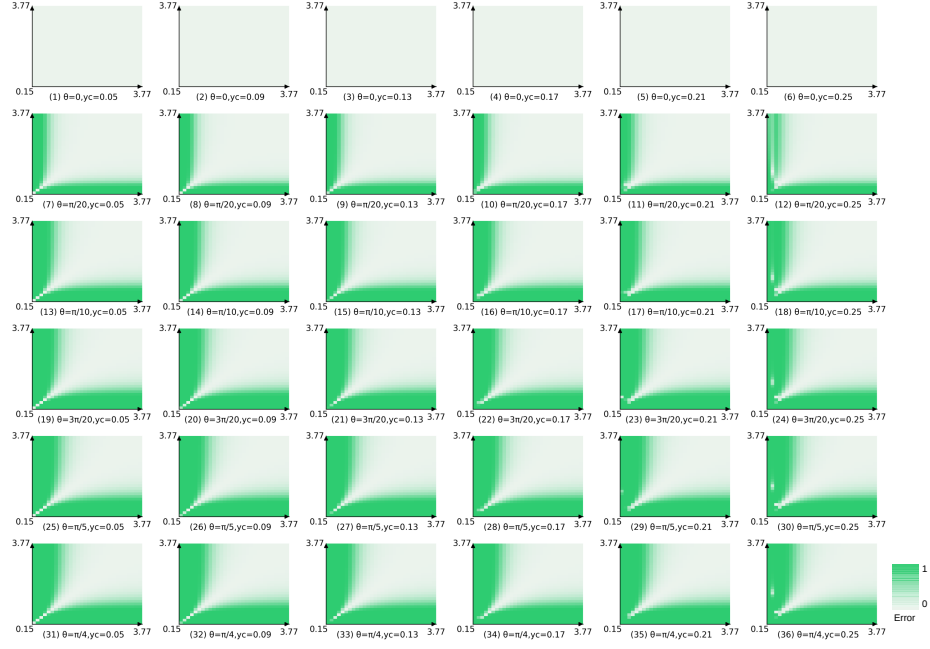


Fig. 3: Numerical Experimental Results of Integration Error. we convert all the errors to the (0,1) range after transformation and widened the differences in the region close to 0. The average relative error is **0.51%**, verifying that our method is a good estimation.

	PSNR $\uparrow$						SSIM $\uparrow$						LPIPS $\downarrow$					
	1 Res.	1/2 Res.	1/4 Res.	1/8 Res.	Avg.		1 Res.	1/2 Res.	1/4 Res.	1/8 Res.	Avg.		1 Res.	1/2 Res.	1/4 Res.	1/8 Res.	Avg.	
3DGS	29.26	26.84	22.16	19.63	24.47	0.877	0.863	0.726	0.612	0.769	0.885	0.185	0.148	0.198	0.223	0.189		
3DGS+Integration	29.14	26.44	22.22	19.32	24.28	0.873	0.850	0.726	0.588	0.759	0.188	0.158	0.197	0.238	0.196			
3DGS+Super-sampling	29.26	26.88	22.79	19.91	24.71	0.876	0.861	0.754	0.633	0.781	0.186	0.153	0.188	0.223	0.188			
3DGS+Adaptive Filter	29.26	29.80	28.26	25.58	28.23	0.877	0.901	0.875	0.809	0.866	0.185	0.123	0.126	0.171	0.151			
Full Method(SA-GS _{int})	29.14	30.06	30.13	28.81	29.53	0.873	0.905	0.921	0.919	0.904	0.188	0.118	0.086	0.078	0.118			
Full Method(SA-GS _{sup})	29.26	30.45	31.75	32.53	31.00	0.876	0.912	0.938	0.951	0.919	0.186	0.114	0.073	0.053	0.106			

Table 1: Ablation studies for zoom-out effect on the Mip-NeRF 360 Dataset. All methods are trained on the largest scale ($1\times$) and evaluated across four scales ($1\times$, $1/2\times$, $1/4\times$, and $1/8\times$). Our 2D scale-adaptive filter removes 3DGS bloat at low rendering frequencies. Additionally, our integration and super-sampling methods further enhance anti-aliasing ability, as illustrated in Fig. 4. It is important to note that the integration and super-sampling methods are only effective when the 2D scale-adaptive filter is active.

	PSNR $\uparrow$						SSIM $\uparrow$						LPIPS $\downarrow$					
	1 Res.	2 Res.	4 Res.	8 Res.	Avg.		1 Res.	2 Res.	4 Res.	8 Res.	Avg.		1 Res.	2 Res.	4 Res.	8 Res.	Avg.	
3DGS	33.96	22.47	18.69	17.32	23.11	0.974	0.747	0.514	0.460	0.674	0.028	0.204	0.410	0.504	0.286			
3DGS+Integration	32.57	22.67	18.74	17.30	22.82	0.962	0.754	0.520	0.463	0.675	0.040	0.204	0.407	0.502	0.288			
3DGS+Super-sampling	33.05	22.65	18.77	17.36	22.96	0.966	0.753	0.520	0.464	0.676	0.038	0.205	0.407	0.501	0.288			
3DGS+Adaptive Filter	33.96	27.89	25.32	24.40	27.89	0.974	0.840	0.677	0.615	0.777	0.028	0.189	0.360	0.465	0.260			

Table 2: Ablation studies for zoom-in effect on the Mip-NeRF 360 Dataset. All methods are trained on the $1/8$ scale ($1\times$) and evaluated across four scales ($1\times$, $2\times$, $4\times$, and $8\times$). Our 2D scale-adaptive filter eliminates erosion artefacts at high rendering frequencies, as illustrated in Fig. 5. Integration and super-sampling methods are not designed for this case, which are comparable to 3DGS.

22 Supplementary Material

	PSNR $\uparrow$						SSIM $\uparrow$						LPIPS $\downarrow$					
	1 Res.	$1/2$ Res.	$1/4$ Res.	$1/8$ Res.	Avg.		1 Res.	$1/2$ Res.	$1/4$ Res.	$1/8$ Res.	Avg.		1 Res.	$1/2$ Res.	$1/4$ Res.	$1/8$ Res.	Avg.	
3DGS	35.10	27.91	22.42	18.76	26.05		0.974	0.949	0.862	0.736	0.880		0.029	0.033	0.069	0.133	0.066	
3DGS+Integration	34.49	28.06	22.78	19.12	26.11		0.972	0.948	0.867	0.745	0.883		0.032	0.036	0.072	0.135	0.069	
3DGS+Super-sampling	34.27	27.10	21.93	18.39	25.42		0.972	0.939	0.848	0.719	0.869		0.032	0.038	0.078	0.146	0.074	
3DGS+Adaptive Filter	34.60	34.33	31.02	27.59	31.89		0.973	0.977	0.968	0.947	0.966	0.031	0.022	0.036	0.067	0.039		
Full Method(<i>SA-GS_{int}</i>)	34.35	34.39	30.99	26.89	31.65		0.972	0.978	0.971	0.940	0.965	0.032	0.020	0.023	0.039	0.029		
Full Method(<i>SA-GS_{sup}</i>)	34.49	36.58	37.50	35.64	36.06		0.972	0.980	0.985	0.985	0.981	0.032	0.018	0.014	0.013	0.019		

Table 3: Ablation studies for zoom-out effect on the Blender Dataset. All methods are trained on the largest scale ($1\times$) and evaluated across four scales ($1\times$, $1/2\times$, $1/4\times$, and $1/8\times$). Our 2D scale-adaptive filter removes 3DGS bloat at low rendering frequencies. Additionally, our integration and super-sampling methods further enhance anti-aliasing ability, as illustrated in Fig. 4. It is important to note that the integration and super-sampling methods are only effective when the 2D scale-adaptive filter is active.

	PSNR $\uparrow$						SSIM $\uparrow$						LPIPS $\downarrow$					
	1 Res.	2 Res.	4 Res.	8 Res.	Avg.		1 Res.	2 Res.	4 Res.	8 Res.	Avg.		1 Res.	2 Res.	4 Res.	8 Res.	Avg.	
3DGS	36.97	24.33	21.01	19.63	25.44		0.988	0.886	0.820	0.821	0.879		0.013	0.065	0.130	0.159	0.092	
3DGS+Integration	34.12	24.48	21.09	19.67	24.84		0.980	0.886	0.819	0.820	0.876		0.025	0.072	0.133	0.161	0.098	
3DGS+Super-sampling	34.01	24.39	21.02	19.63	24.76		0.980	0.884	0.818	0.820	0.876		0.022	0.069	0.132	0.161	0.096	
3DGS+Adaptive Filter	35.74	30.38	27.63	26.36	30.03		0.984	0.953	0.912	0.885	0.933		0.016	0.059	0.111	0.141	0.082	

Table 4: Ablation studies for zoom-in effect on the Blender Dataset. All methods are trained on the $1/8$ scale ($1\times$) and evaluated across four scales ($1\times$, $2\times$, $4\times$, and $8\times$). Our 2D scale-adaptive filter eliminates erosion artefacts at high rendering frequencies, as illustrated in Fig. 5. Integration and super-sampling methods are not designed for this case, which are comparable to 3DGS.

D Additional Results

In this section, we provide more qualitative and quantitative results on the Mip-NeRF 360 dataset(D.1) and the Blender dataset(D.2).

D.1 Mip-NeRF 360 Dataset

We further evaluate the effect of our method on zoom-out and zoom-in settings for each scene of this dataset. The quantitative results with per-scene metrics can be found in Table 5 and Table 6. Qualitative comparison with state-of-the-art methods are provided in Fig. 6 and Fig. 7. Our method achieves superior or comparable performance compared to the state-of-the-art, while being training-free.

D.2 Blender Dataset

We further evaluate the effect of our method on zoom-out and zoom-in settings for each scene of this dataset. The quantitative results with per-scene metrics can be found in Table 7 and Table 8. Qualitative comparison with state-of-the-art methods are provided in Fig. 8 and Fig. 9. Our method achieves superior or comparable performance compared to the state-of-the-art, while being training-free.

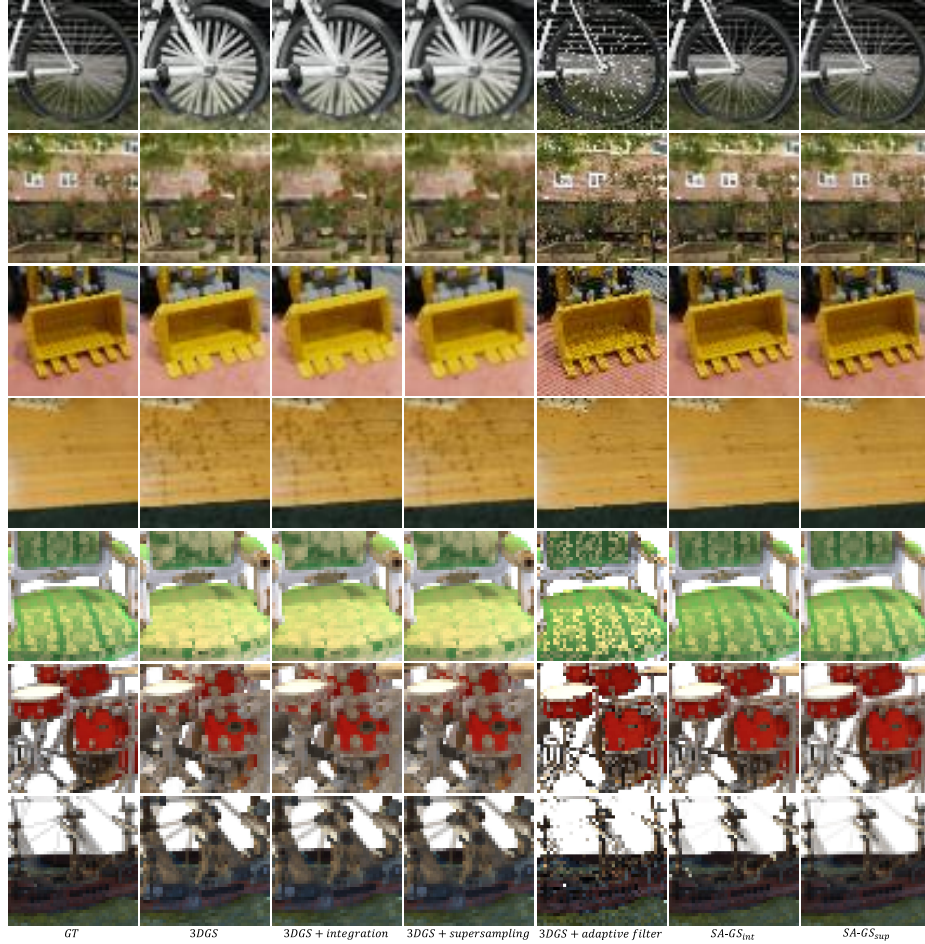


Fig. 4: Single-scale Training and Multi-scale Testing for Zoom-out Effect. All methods are trained on full resolution ($1\times$) and evaluated on smallest resolution ($1/8\times$) to mimic zoom-out case. 3DGS suffers from bloat or erosion artefacts at different rendering frequencies, which can exacerbate the aliasing effect. Our 2D scale-adaptive filter maintains the scale consistency of the Gaussian across different rendering settings. Additionally, our integration and super-sampling methods further enhance the anti-aliasing ability of Gaussian scenes. It is important to note that the integration and super-sampling methods are only effective in combination with the 2D scale-adaptive filter.

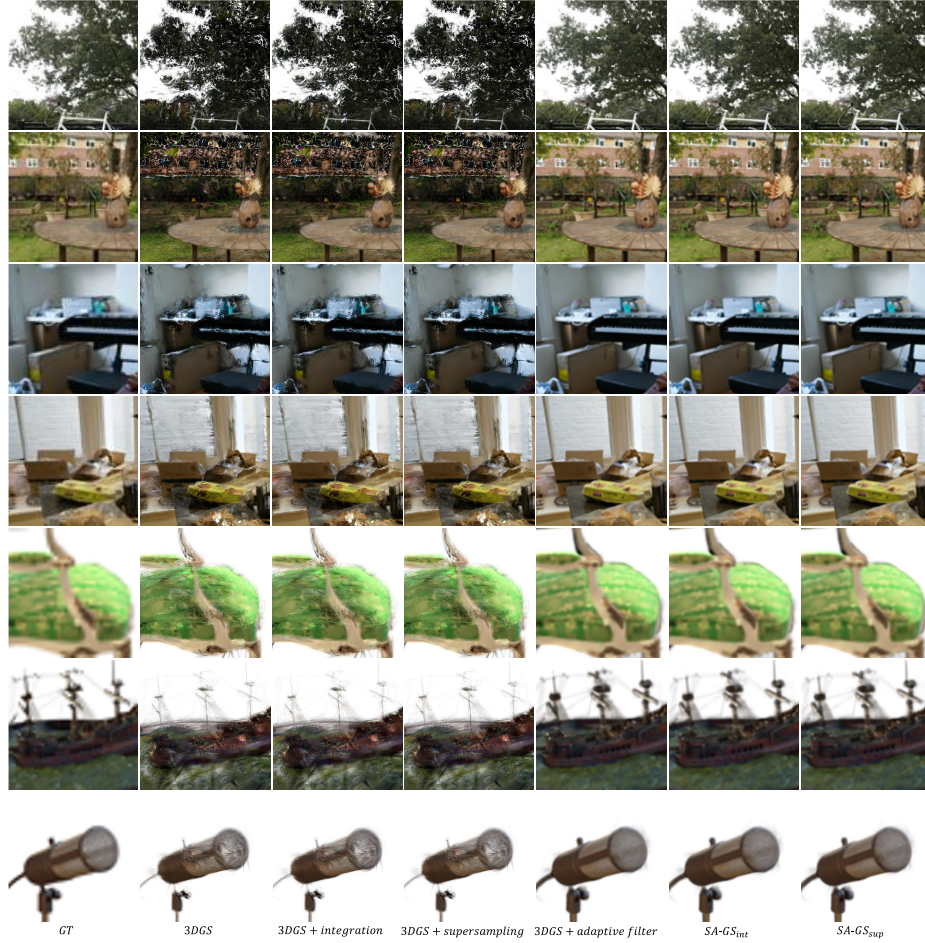


Fig. 5: Single-scale Training and Multi-scale Testing for Zoom-in Effect. All method are trained on smallest resolution($1/8\times$) and evaluated on full resolution($1\times$) to mimic zoom-in case. 3DGS suffers from bloat or erosion artefacts at different rendering frequencies, which can exacerbate the aliasing effect. Our 2D scale-adaptive filter maintains the scale consistency of the Gaussian across different rendering settings. Note that the integration and super-sampling methods are only designed for the zoom-out case, so in the zoom-in case they maintain comparable effects to the addition of a 2D scale-adaptive filter.

	PSNR $\uparrow$									
	<i>bonsai</i>	<i>bicycle</i>	<i>counter</i>	<i>garden</i>	<i>kitchen</i>	<i>room</i>	<i>stump</i>	<i>flowers</i>	<i>treehill</i>	<i>Avg.</i>
3DGS	27.06	21.16	26.13	23.33	26.69	29.15	25.00	20.38	21.37	24.47
3DGS(MS)	22.27	26.82	26.66	21.52	24.62	25.64	30.17	24.55	22.38	24.96
Mip-Splatting	26.44	32.96	30.42	25.63	30.54	32.88	34.01	30.86	25.47	29.91
<i>SA-GS_{fil}(ours)</i>	31.74	24.79	29.56	27.28	30.74	33.37	28.98	23.62	23.98	28.23
<i>SA-GS_{int}(ours)</i>	32.63	26.21	29.41	30.22	32.51	33.85	30.58	25.25	25.14	29.53
<i>SA-GS_{sup}(ours)</i>	34.13	27.74	31.25	31.97	34.19	35.63	31.89	26.14	26.01	31.00

	SSIM $\uparrow$									
	<i>bonsai</i>	<i>bicycle</i>	<i>counter</i>	<i>garden</i>	<i>kitchen</i>	<i>room</i>	<i>stump</i>	<i>flowers</i>	<i>treehill</i>	<i>Avg.</i>
3DGS	0.872	0.687	0.847	0.715	0.835	0.887	0.744	0.650	0.687	0.769
3DGS(MS)	0.718	0.873	0.867	0.686	0.764	0.859	0.914	0.737	0.709	0.792
Mip-Splatting	0.876	0.962	0.942	0.820	0.934	0.959	0.960	0.916	0.836	0.912
<i>SA-GS_{fil}(ours)</i>	0.940	0.813	0.921	0.835	0.900	0.946	0.874	0.774	0.788	0.866
<i>SA-GS_{int}(ours)</i>	0.960	0.872	0.906	0.928	0.957	0.959	0.911	0.814	0.832	0.904
<i>SA-GS_{sup}(ours)</i>	0.966	0.888	0.947	0.943	0.965	0.966	0.923	0.827	0.845	0.919

	LPIPS $\downarrow$									
	<i>bonsai</i>	<i>bicycle</i>	<i>counter</i>	<i>garden</i>	<i>kitchen</i>	<i>room</i>	<i>stump</i>	<i>flowers</i>	<i>treehill</i>	<i>Avg.</i>
3DGS	0.140	0.239	0.143	0.169	0.123	0.134	0.218	0.269	0.264	0.189
3DGS(MS)	0.228	0.154	0.148	0.251	0.183	0.140	0.131	0.234	0.261	0.192
Mip-Splatting	0.136	0.086	0.094	0.183	0.061	0.060	0.088	0.118	0.185	0.112
<i>SA-GS_{fil}(ours)</i>	0.109	0.183	0.120	0.135	0.108	0.112	0.154	0.215	0.225	0.151
<i>SA-GS_{int}(ours)</i>	0.088	0.139	0.119	0.066	0.062	0.090	0.122	0.185	0.188	0.118
<i>SA-GS_{sup}(ours)</i>	0.084	0.125	0.091	0.054	0.055	0.083	0.110	0.177	0.178	0.106

Table 5: Single-scale Training and Multi-scale testing on the Mip-NeRF 360 Dataset. For each scene, we report the arithmetic mean of each metric averaged over the 4 scales ($1\times$, $1/2\times$, $1/4\times$, $1/8\times$) used in the dataset. (MS) means multi-scale training.

	PSNR $\uparrow$									
	<i>bonsai</i>	<i>bicycle</i>	<i>counter</i>	<i>garden</i>	<i>kitchen</i>	<i>room</i>	<i>stump</i>	<i>flowers</i>	<i>treehill</i>	<i>Avg.</i>
3DGS	24.52	20.88	24.64	22.57	24.31	27.85	22.45	19.87	20.89	23.11
3DGS(MS)	22.27	26.82	26.66	21.52	24.62	25.64	30.17	24.55	22.38	24.96
Mip-Splatting	25.85	30.67	29.43	25.17	28.37	30.22	33.10	28.95	25.69	28.60
<i>SA-GS_{fil}(ours)</i>	30.12	25.07	28.80	27.83	29.29	32.16	28.23	24.42	25.12	27.89

	SSIM $\uparrow$									
	<i>bonsai</i>	<i>bicycle</i>	<i>counter</i>	<i>garden</i>	<i>kitchen</i>	<i>room</i>	<i>stump</i>	<i>flowers</i>	<i>treehill</i>	<i>Avg.</i>
3DGS	0.769	0.593	0.771	0.616	0.720	0.850	0.582	0.559	0.603	0.674
3DGS(MS)	0.718	0.873	0.867	0.686	0.764	0.859	0.914	0.737	0.709	0.792
Mip-Splatting	0.733	0.898	0.877	0.704	0.747	0.828	0.921	0.779	0.728	0.802
<i>SA-GS_{fil}(ours)</i>	0.878	0.705	0.862	0.716	0.790	0.910	0.748	0.673	0.707	0.777

	LPIPS $\downarrow$									
	<i>bonsai</i>	<i>bicycle</i>	<i>counter</i>	<i>garden</i>	<i>kitchen</i>	<i>room</i>	<i>stump</i>	<i>flowers</i>	<i>treehill</i>	<i>Avg.</i>
3DGS	0.250	0.316	0.238	0.292	0.266	0.207	0.326	0.336	0.345	0.286
3DGS(MS)	0.228	0.154	0.148	0.251	0.183	0.140	0.131	0.234	0.261	0.192
Mip-Splatting	0.275	0.170	0.188	0.290	0.248	0.198	0.165	0.249	0.302	0.232
<i>SA-GS_{fil}(ours)</i>	0.196	0.305	0.210	0.281	0.237	0.191	0.281	0.318	0.325	0.260

Table 6: Single-scale Training and Multi-scale testing on the Mip-NeRF 360 Dataset. For each scene, we report the arithmetic mean of each metric averaged over the 4 scales ($1\times$, $2\times$, $4\times$, $8\times$) used in the dataset. (MS) means multi-scale training.

26 **Supplementary Material**

	PSNR $\uparrow$								
	<i>chair</i>	<i>drums</i>	<i>figus</i>	<i>hotdog</i>	<i>lego</i>	<i>materials</i>	<i>mic</i>	<i>ship</i>	<i>Avg.</i>
3DGS	25.04	22.09	26.64	28.43	26.12	26.55	28.01	25.51	26.05
3DGS(MS)	30.47	26.16	29.96	34.92	30.91	30.41	32.97	31.45	30.91
Mip-Splatting	32.57	26.74	32.92	34.44	33.39	31.53	35.69	32.36	32.45
<i>SA-GS_{fil}(ours)</i>	32.13	26.71	32.48	35.66	33.05	30.90	33.78	30.39	31.89
<i>SA-GS_{int}(ours)</i>	31.38	26.23	32.32	33.73	32.33	31.00	34.84	31.41	31.65
<i>SA-GS_{sup}(ours)</i>	37.89	28.87	36.34	39.40	38.59	33.01	39.64	34.70	36.06

	SSIM $\uparrow$								
	<i>chair</i>	<i>drums</i>	<i>figus</i>	<i>hotdog</i>	<i>lego</i>	<i>materials</i>	<i>mic</i>	<i>ship</i>	<i>Avg.</i>
3DGS	0.893	0.837	0.913	0.914	0.865	0.898	0.905	0.818	0.880
3DGS(MS)	0.975	0.943	0.965	0.983	0.967	0.962	0.974	0.923	0.962
Mip-Splatting	0.984	0.950	0.983	0.981	0.979	0.973	0.981	0.925	0.970
<i>SA-GS_{fil}(ours)</i>	0.977	0.953	0.981	0.985	0.975	0.972	0.981	0.904	0.966
<i>SA-GS_{int}(ours)</i>	0.978	0.943	0.981	0.978	0.974	0.970	0.977	0.919	0.965
<i>SA-GS_{sup}(ours)</i>	0.994	0.968	0.991	0.992	0.993	0.980	0.994	0.935	0.981

	LPIPS $\downarrow$								
	<i>chair</i>	<i>drums</i>	<i>figus</i>	<i>hotdog</i>	<i>lego</i>	<i>materials</i>	<i>mic</i>	<i>ship</i>	<i>Avg.</i>
3DGS	0.047	0.090	0.059	0.040	0.073	0.051	0.044	0.124	0.066
3DGS(MS)	0.022	0.057	0.033	0.019	0.034	0.036	0.027	0.083	0.039
Mip-Splatting	0.011	0.042	0.013	0.014	0.016	0.020	0.011	0.072	0.025
<i>SA-GS_{fil}(ours)</i>	0.029	0.051	0.020	0.026	0.032	0.027	0.031	0.095	0.039
<i>SA-GS_{int}(ours)</i>	0.016	0.046	0.014	0.018	0.021	0.021	0.015	0.077	0.029
<i>SA-GS_{sup}(ours)</i>	0.006	0.035	0.008	0.010	0.007	0.017	0.006	0.066	0.019

Table 7: Single-scale Training and Multi-scale testing on the Blender Dataset. For each scene, we report the arithmetic mean of each metric averaged over the 4 scales ($1\times$, $1/2\times$, $1/4\times$, $1/8\times$) used in the dataset. (MS) means multi-scale training.

	PSNR $\uparrow$								
	<i>chair</i>	<i>drums</i>	<i>figus</i>	<i>hotdog</i>	<i>lego</i>	<i>materials</i>	<i>mic</i>	<i>ship</i>	<i>Avg.</i>
3DGS	24.26	22.14	23.72	27.87	23.99	25.45	29.22	26.86	25.44
3DGS(MS)	30.47	26.16	29.96	34.92	30.91	30.41	32.97	31.45	30.91
Mip-Splatting	30.11	25.76	28.51	34.65	29.54	30.05	33.79	30.66	30.39
<i>SA-GS_{fil}(ours)</i>	30.42	25.29	27.54	33.89	29.44	29.68	33.87	30.11	30.03

	SSIM $\uparrow$								
	<i>chair</i>	<i>drums</i>	<i>figus</i>	<i>hotdog</i>	<i>lego</i>	<i>materials</i>	<i>mic</i>	<i>ship</i>	<i>Avg.</i>
3DGS	0.891	0.857	0.903	0.915	0.834	0.883	0.921	0.823	0.879
3DGS(MS)	0.975	0.943	0.965	0.983	0.967	0.962	0.974	0.923	0.962
Mip-Splatting	0.951	0.921	0.947	0.971	0.930	0.951	0.965	0.880	0.939
<i>SA-GS_{fil}(ours)</i>	0.949	0.912	0.933	0.967	0.925	0.946	0.965	0.871	0.933

	LPIPS $\downarrow$								
	<i>chair</i>	<i>drums</i>	<i>figus</i>	<i>hotdog</i>	<i>lego</i>	<i>materials</i>	<i>mic</i>	<i>ship</i>	<i>Avg.</i>
3DGS	0.071	0.106	0.068	0.073	0.122	0.084	0.063	0.146	0.092
3DGS(MS)	0.022	0.057	0.033	0.019	0.034	0.036	0.027	0.083	0.039
Mip-Splatting	0.052	0.091	0.059	0.048	0.088	0.060	0.049	0.138	0.073
<i>SA-GS_{fil}(ours)</i>	0.057	0.102	0.074	0.055	0.091	0.068	0.052	0.154	0.082

Table 8: Single-scale Training and Multi-scale testing on the Blender Dataset. For each scene, we report the arithmetic mean of each metric averaged over the 4 scales ($1\times$, $2\times$, $4\times$, $8\times$) used in the dataset. (MS) means multi-scale training.

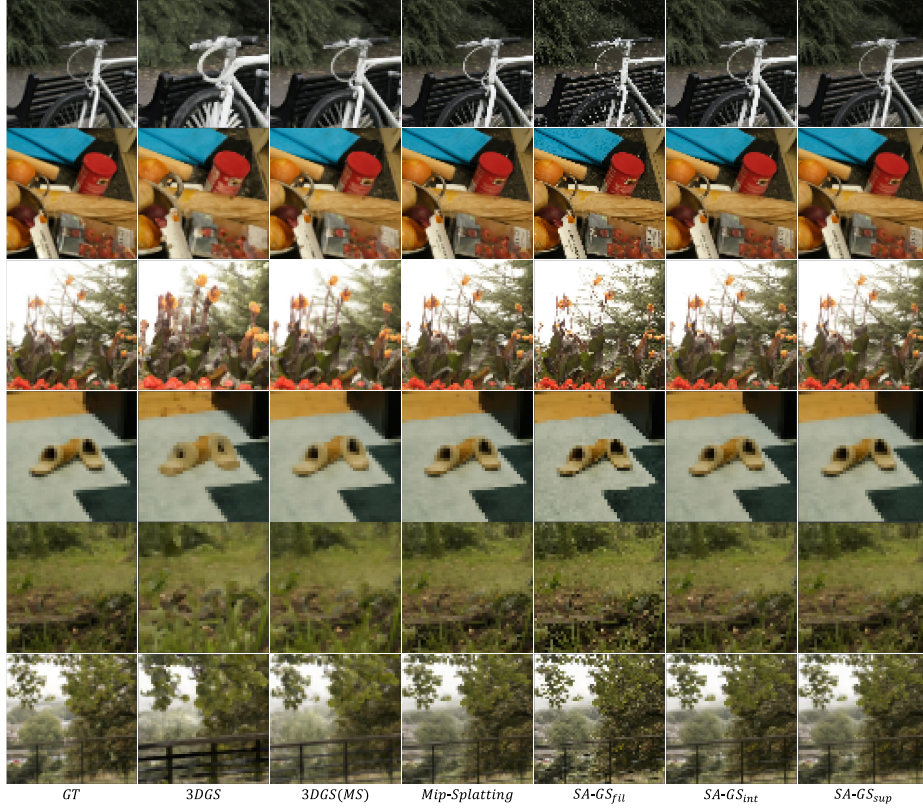


Fig. 6: Single-scale Training and Multi-scale Testing on the Mip-NeRF 360 Dataset. All method are trained on full resolution($1\times$) and evaluated on smallest resolution($1/8\times$) to mimic zoom-out case. Our 2D scale-adaptive filter maintains the consistency of the Gaussian across different rendering settings. $SA-GS_{int}$ achieves results comparable to Mip-Splatting, while $SA-GS_{sup}$ surpasses Mip-Splatting, resulting in optimal performance for this setting.

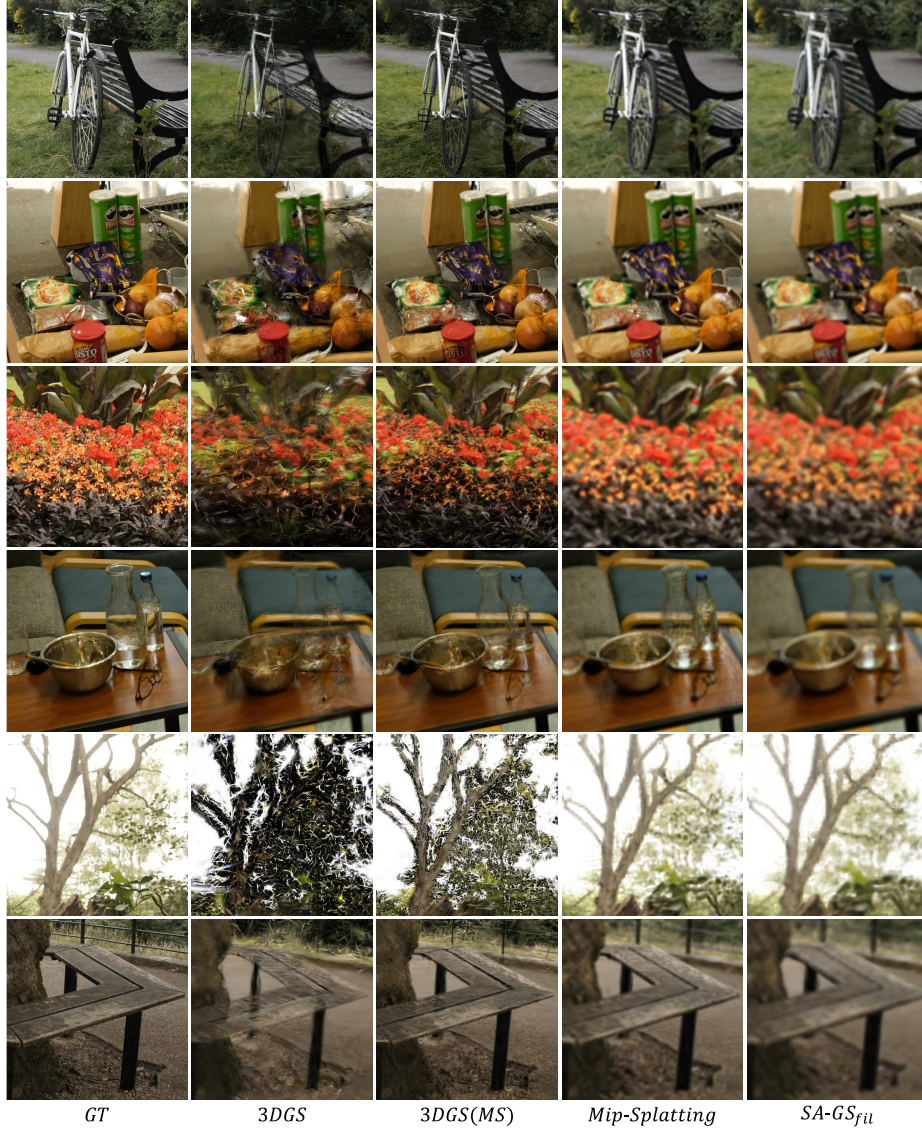


Fig. 7: Single-scale Training and Multi-scale Testing on the Mip-NeRF 360 Dataset. All method are trained on smallest resolution($1/8\times$) and evaluated on full resolution($1\times$) to mimic zoom-in case. $SA-GS_{fil}$ achieves performance comparable to Mip-Splatting. However, Our 2D scale-adaptive filter is training-free and does not add any computational burden.

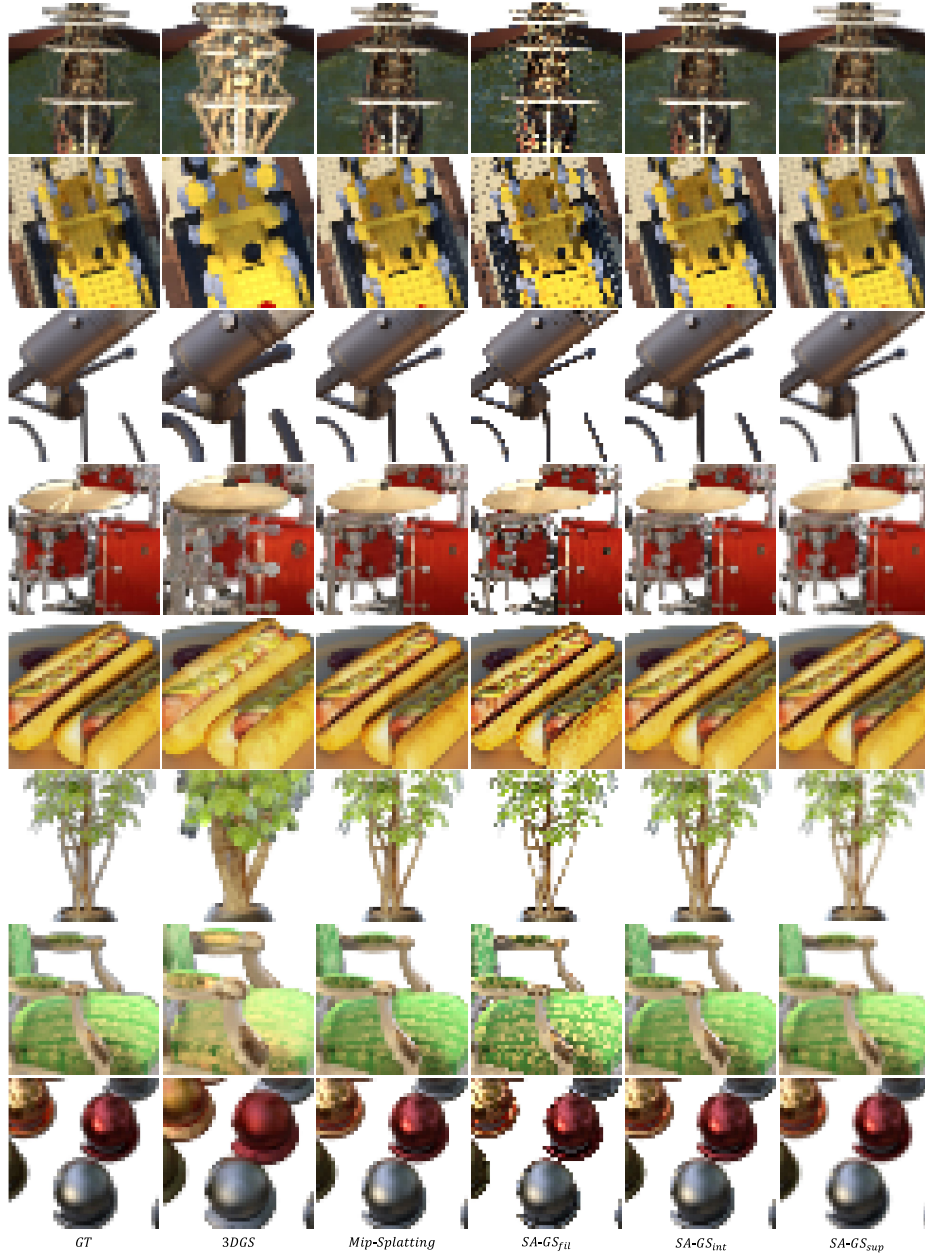


Fig.8: Single-scale Training and Multi-scale Testing on the Blender Dataset. All method are trained on full resolution($1\times$) and evaluated on smallest resolution($1/8\times$) to mimic zoom-out case. Our 2D scale-adaptive filter maintains the consistency of the Gaussian across different rendering settings. $SA-GS_{int}$ achieves results comparable to Mip-Splatting, while $SA-GS_{sup}$ surpasses Mip-Splatting, resulting in optimal performance for this setting.

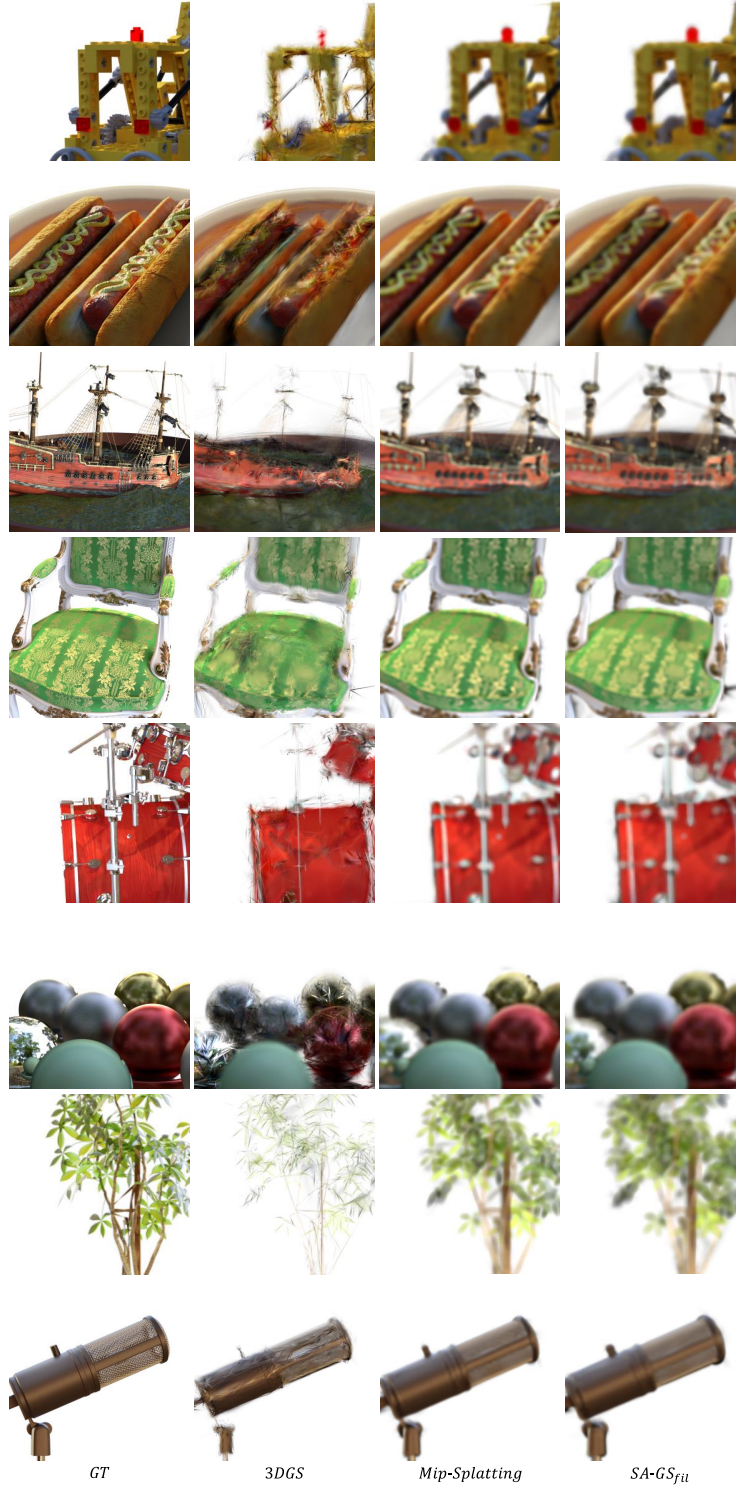


Fig.9: Single-scale Training and Multi-scale Testing on the Blender Dataset. All method are trained on smallest resolution($1/8\times$) and evaluated on full resolution($1\times$) to mimic zoom-in case. $SA-GS_{fil}$ achieves performance comparable to Mip-Splatting. However, Our 2D scale-adaptive filter is training-free and does not add any computational burden.