Multi-Scale 3D Gaussian Splatting for Anti-Aliased Rendering

Zhiwen Yan Weng Fei Low Yu Chen Gim Hee Lee
Department of Computer Science, National University of Singapore
yan.zhiwen@u.nus.edu {wengfei.low, chenyu}@comp.nus.edu.sg gimhee.lee@nus.edu.sg

Figure 1. The rendering quality and speed of the original 3D Gaussian splatting[12] deteriorate severely at low resolutions or from distant cameras due to aliasing. Conversely, our multi-scale 3D Gaussians representation utilizes selective rendering to achieve faster ($160\% - 2400\%$ at $4\times$-$128\times$ resolution) and more accurate rendering at lower resolutions.

Abstract

3D Gaussians have recently emerged as a highly efficient representation for 3D reconstruction and rendering. Despite its high rendering quality and speed at high resolutions, they both deteriorate drastically when rendered at lower resolutions or from far away camera position. During low resolution or far away rendering, the pixel size of the image can fall below the Nyquist frequency compared to the screen size of each splatted 3D Gaussian and leads to aliasing effect. The rendering is also drastically slowed down by the sequential alpha blending of more splatted Gaussians per pixel. To address these issues, we propose a multi-scale 3D Gaussian splatting algorithm, which maintains Gaussians at different scales to represent the same scene. Higher-resolution images are rendered with more small Gaussians, and lower-resolution images are rendered with fewer larger Gaussians. With similar training time, our algorithm can achieve 13%-66% PSNR and 160%-2400% rendering speed improvement at 4×-128× scale rendering on Mip-NeRF360 dataset compared to the single scale 3D Gaussian splatting. More results and code are released on our project page.

1. Introduction

3D Gaussian Splatting[12] has recently emerged as a highly efficient representation for novel view synthesis. Compared to the time-consuming ray marching used in most neural radiance fields (NeRF) [2, 15, 16], a high-resolution image can be rendered in real-time by rasterizing the splatted 3D Gaussians. However, this rasterization algorithm is subjected to severe aliasing effect and speed deterioration when rendering the same scene at low resolution or from distant positions as shown in Fig. 1. This limitation significantly constrain the application of the 3D Gaussian splatting algorithm in reconstructing and rendering large-scale scenes.

Aliasing effect is a consequence of inadequate sampling frequency failing to capture the continuous signal accurately. In the context of rendering, image pixels are sampled

with an interval of one-pixel size. The signal can be considered as the 3D scene represented implicitly as in NeRF or explicitly as in 3D Gaussians. When part of the 3D scene is represented with high details but rendered with low resolution or from distant positions, the disparity between the low sampling and high signal frequencies culminates in aliasing artifacts. A naive solution is to render at high resolution and subsequently down-scale the rendered image to a lower resolution. However, this solution is not viable for scenes containing both near and far regions which are very common. Due to the inability of 3D Gaussian splatting algorithm to accommodate varying resolutions within a single image, rendering the entire image with a even higher resolution for the sake of far away regions is neither time nor memory efficient.

We postulate that the pronounced aliasing artifacts observed when rendering with 3D Gaussians, as opposed to other techniques such as NeRF, are primarily attributable to the splatting of small Gaussians. 3D regions with intricate details are represented with large amount of small Gaussians. When rendering these regions with low resolution or from a distant view, many splatted small Gaussians are cramped in one pixel and therefore the pixel color of this region is dominated by the front-most Gaussian, even if this Gaussian is much smaller than others and not at the center. This problem is further aggravated by the low pass filter in [12, 19] applied to each individual Gaussian with the intention to mitigate aliasing on edges at high resolutions. This problem is explained in more detail in Sec. 3.2.

In addition to the aliasing artifacts, the rendering speed of 3D Gaussians is also affected at low resolution. The number of 3D Gaussians that need to be rendered remains constant at lower resolutions, but they are more concentrated to fewer pixels. The Gaussians that are splatted to the same pixel cannot be rendered in parallel. This means that the image rendering is even slower at lower resolution in comparison with NeRF rendering time that reduces linearly with decreasing resolution. Hence, although aliasing is not a problem exclusive to 3D Gaussian splatting, it is more prominent and more difficult to tackle.

Contributions To mitigate the aliasing problem for 3D Gaussian splatting, we propose a novel multi-scale 3D Gaussians to represent the scene at different levels of detail (LOD) as shown in Fig. 2. This is inspired by the mipmap and LOD algorithms widely used in computer graphics, which pre-computes textures and polygons at different scales to be rendered under different resolutions and distances. Similarly, we add larger, coarser Gaussians for lower resolutions by aggregating the smaller and finer Gaussians from higher resolutions. Depending on the pixel coverage of the splatted Gaussians during rendering, only a subset of the Gaussians is used. A simplified explanation for this is that the coarse Gaussians are used to render low-resolution images and the fine Gaussians are used to render high-resolution images. With fewer than 5% number of Gaussians added and a similar training time, our method can achieve 13%-66% PSNR and 160%-2400% rendering speed improvements at $4\times$-$128\times$ scale rendering on Mip-NeRF360 dataset[2], while maintaining a comparable rendering quality and speed at $1\times$ scale.

2. Related Works

2.1. Anti-Aliasing in Computer Graphics

Aliasing is a long-standing problem for computer graphics when rendering a scene to a discrete image. Traditional anti-aliasing techniques primarily target mesh representations. Supersampling Anti-Aliasing (SSAA) [5] renders the scene at a higher resolution before downscaling, leading to significantly more time and memory demain, and therefore is less used in real-time applications. The Multisample Anti-Aliasing (MSAA) [1, 5] algorithm selectively supersamples pixels on the edges, reducing resource and time consumption. This technique is not very suitable for 3D Gaussian splatting because of its requirement for regular grids and lack of support for variable sampling resolution at different pixels. The more recent Fast Approximate Anti-Aliasing (FXAA) [11, 14] is a post processing algorithm that smooths the jagged edges after the image is rendered. Unfortunately, this technique is also not suitable for Gaussian representation as the front-most Gaussian dominates the pixel color and produces chunky instead of jagged artifacts in mesh rendering.

In contrast to the supersampling methods mentioned above, our method takes the inspiration from hierarchical mipmap [18] and level of details (LOD) [7, 9] algorithms to address the aliasing for 3D Gaussians. Mipmap uses multi-scale textures for the rendering at different resolution or from different distances. LOD algorithm represents the models in a scene with different complexity to be rendered at different distances. Both techniques not only mitigate the aliasing effect by reducing the complexity of the scene representation, but also enhances rendering speed, particularly for large-scale scenes.

2.2. Anti-Aliasing in Neural Representation

The recent success of neural representations especially Neural Radiance Fields (NeRF) [6, 15, 16] has also inspired some works to develop algorithms against aliasing effect on neural representations beyond the traditional mesh representation. Mip-NeRF [2, 3] employ low pass filters on the positional encoding of the input spatial coordinates to reduce the scene signal frequency. Building on the hash grid representation used by InstantNGP [16] with no position encoding, Zip-NeRF [4] proposes a multi-sampling

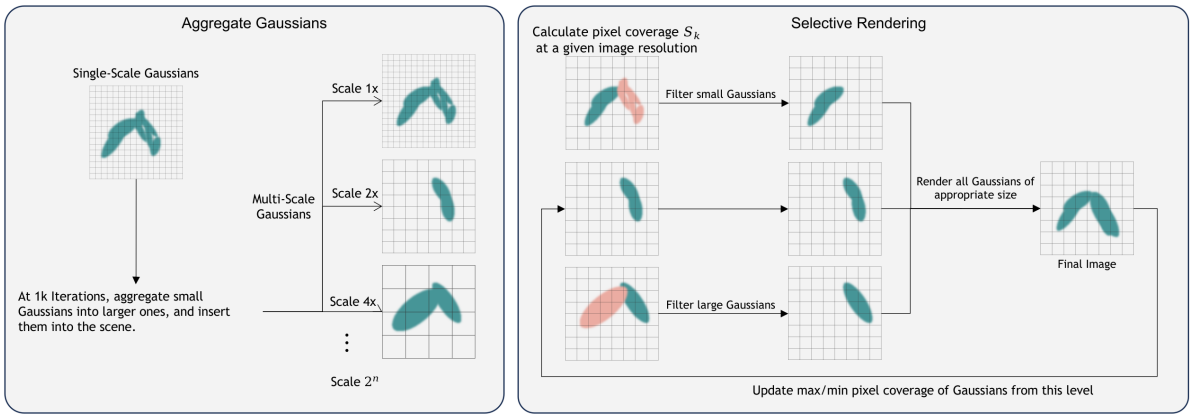

Figure 2. Overall pipeline of our algorithm. At the early stage of training (left), small Gaussians below certain size threshold in each voxel are aggregated, enlarged and inserted into the scene at different resolution scale. During rendering (right), the multi-scale Gaussians of the appropriate "pixel coverage" at the current render resolution are selected for rendering. If the rendering resolution scale equals to the scale of the Gaussians, the expected "pixel coverage" range of the Gaussians are updated accordingly.

strategy in the conical frustum instead of the camera ray, at the cost of 6× rendering time. Similar to the mipmap algorithm in mesh texture rendering, Tri-MipNeRF [10] and MipGrid [17] proposes to use multi-scale feature grids for rendering at different resolution or distance.

Conversely, 3D Gaussian splatting [12] presents unique anti-aliasing challenges due to its distinct scene representation. It does not have any positional encoding or feature grid, and its requirement for regular grids conflicts the more flexible multi-sampling strategies. The concentration of small Gaussians in detail-rich regions exacerbates aliasing and speed issues, more so than in NeRF representations. To the best of our knowledge, we are the first to propose an anti-aliasing algorithm for scene reconstruction using 3D Gaussian splatting.

3. Preliminaries

3.1. 3D Gaussian Splatting

3D Gaussian splatting is first proposed in EWA Splatting [19], and later used by [12] for scene reconstruction and novel view synthesis. The scene is represented by a set of $\mathbf{K}$ 3D Gaussians $\{\mathcal{G}_{\hat{V}^k,\hat{\mu}^k}, \sigma_k, \mathbf{c}_k \mid k \in [1, \mathbf{K}]\}$ with variance $\hat{V}^k$, center $\hat{\mu}^k$, density σ_k and color $\mathbf{c}^k$. During rendering, the 3D Gaussians are splatted to the 2D screen to by the perspective transformation to form 2D Gaussians $\mathcal{G}_{V^k,\mu^k}$. The image is then divided into 16×16 regular tiles and all 2D Gaussians touching each tile are sorted based on their original depth. The color of each pixel in the tile is then rasterized from the sequential alpha blending the 2D Gaussians from front to back.

3.2. Cause of Aliasing in 3D Gaussian Splatting

Aliasing can occur when sampling a continuous signal $g(x)$ with a discrete sampling function $\delta_s(x, \Delta x) = \sum_{n=-\infty}^{\infty} \delta(x - n \cdot \Delta x)$, where δ is a impulse function. The result of the sample in the spatial domain is:

$$g_s(x) = \delta_s(x, \Delta x) \cdot g(x). \tag{1}$$

This sampled function converted into the frequency domain using Fourier transform operator $\mathcal{F}$ becomes:

$$\begin{aligned} \mathcal{F}[g_s(u)] &= \frac{1}{\Delta x} \sum_{k=-\infty}^{\infty} \delta(u - \frac{k}{\Delta x}) * \mathcal{F}[g(x)] \\ &= \frac{1}{\Delta x} \sum_{k=-\infty}^{\infty} \mathbf{G}(u - \frac{k}{\Delta x}). \end{aligned} \tag{2}$$

When the highest frequency component f_{max} of the signal is greater than half of the sampling frequency $f_s = \frac{1}{\Delta x}$, $\mathbf{G}(u - \frac{k}{\Delta x})$ in the summation sequence would overlap with each other and causes the sampled signal to diverge from the actual signal. This phenomenon is the aliasing effect and the minimum sampling frequency needed to avoid aliasing is $f_{Ny} = 2 \cdot f_{max}$, known as the Nyquist frequency.

The EWA splatting [19] used by 3D Gaussian splatting [12] also tries to mitigate the aliasing problem by applying a low pass filter to the rendered color. To approximate this efficiently, it applies a Gaussian kernel $h(\mathbf{x})$ as the low pass filter on each splatted 2D signal $g_c(\mathbf{x})$ independently to produce a band limited signal:

$$\begin{aligned} g_c'(\mathbf{x}) &= g_c(\mathbf{x}) * h(\mathbf{x}) \\ &\approx \sum_k \sigma_k \mathbf{c}_k T_k \int_{\mathcal{R}^2} q_k(\eta) h(\mathbf{x} - \eta) d\eta \\ &= \sum_k \sigma_k \mathbf{c}_k T_k \cdot (q_k * h)(\mathbf{x}), \end{aligned} \tag{3}$$

where $\mathcal{R}^2$ is the range of one pixel, $q_k(\mathbf{x})$ is the 2D integrated Gaussian kernel, and $\sigma_k, \mathbf{c}_k, T_k$ are the opacity,

color, and transmittance at each Gaussian, respectively. By combining the reconstruction Gaussian kernel $\mathcal{G}_{V^k}$ and low pass Gaussian kernel $\mathcal{G}_{V^h}$ of covariance matrix V^k and V^h, the band limit function becomes:

$$\begin{aligned} g'_c(\mathbf{x}) &= \sum_k \alpha_k \cdot (\mathcal{G}_{V^k} * \mathcal{G}_{V^h})(\mathbf{x}) \\ &= \sum_k \alpha_k \cdot \mathcal{G}_{V^k+V^h}(\mathbf{x}), \end{aligned} \tag{4}$$

where α_k represents all coefficients invariant of $\mathbf{x}$ at each Gaussian and V^h is determined by the screen pixel size. A simple understanding of this is that the covariance of each 3D Gaussian is increased based on the screen pixel size.

This method of applying a low pass filter to each 3D Gaussian independently helps to smooth the edges of the Gaussians when the Gaussians are not too small compared to the pixel size. However, it also gives rise to two substantial issues at low resolutions:

1. V^h added to the original covariance V^k effectively increases the extent of each Gaussian, especially when V^h is large compared to V^k at low resolutions. Small Gaussians in the front dominate the color of the pixel and cause severe artifacts shown in Fig. 7.
2. The number of Gaussians involved in the sequential $\sum_k$ for each pixel scales increases with decreasing image resolution. Due to the incremental calculation of the transmittance T_k, the rendering even slower at lower resolutions.

4. Our Method

4.1. Multi-Scale Gaussians Based on Pixel Coverage

To mitigate the aliasing artifacts of 3D Gaussians [12] while avoiding the two problems of the EWA splatting [19], we introduce multi-scale 3D Gaussians (*cf*. Fig. 2) that tackle the problem on the scene-level instead of on each individual Gaussian. The 3D scene is represented with Gaussians from 4 levels of detail, corresponding to the $1\times$, $4\times$, $16\times$, and $64\times$ downsampled resolution. Small finer-level Gaussians are aggregated to create larger Gaussians for coarser levels during training. Each 3D Gaussian $\mathcal{G}_k^l$ belongs to one of the levels l and is included or excluded independently during the rendering based on its "pixel coverage".

Pixel Coverage of Gaussian. The "pixel coverage" of a Gaussian reflects the size of the Gaussian when splatted onto the screen space compared to the pixel size at the current rendering resolution. The "pixel coverage" S_k of a splatted 2D Gaussian $\mathcal{G}_{(\mu^k,V^k)}$ is defined as the length of its horizontal or vertical axis until the low opacity level set, whichever is smaller, as shown in Fig. 3. The pixel coverage is measured in pixel count and the opacity threshold σ_T is set as $\frac{1}{255}$.

$$u\colon \sigma_k \mathcal{G}_{(\mu^k,V^k)}\left(\mu_x^k + \frac{u}{2}, \mu_y^k\right) = \sigma_T$$
$$v\colon \sigma_k \mathcal{G}_{(\mu^k,V^k)}\left(\mu_x^k, \mu_y^k + \frac{v}{2}\right) = \sigma_T$$
$$S_k = \min(u, v)$$

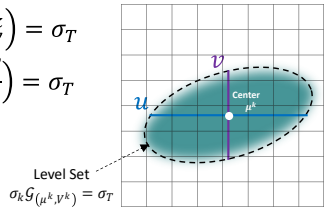

Figure 3. Pixel coverage of a 3D Gaussian is its horizontal or vertical size, whichever is smaller measured by the level set.

Figure 4. Missing parts caused by naive small Gaussian filtering at different resolution scales.

The pixel coverage approximates the extent of a 2D splatted Gaussian in the spatial domain. During the rendering from a given camera direction, the color of each splatted Gaussian is constant within this pixel coverage. As a result, the coverage of this pixel approximates the inverse of the highest frequency component $f_{max} = 1/S_k$ in this region. Compared to the sampling frequency of $f_s = 1\text{px}^{-1}$ during rasterization, a signal frequency of $f_{max} > f_s/2$ can cause the sampling to fall below the Nyquist frequency needed to avoid aliasing.

Consequently, the Gaussians with pixel coverage $S_k < S_T = 2\text{px}$ should be filtered out during rendering to avoid aliasing. Since 3D Gaussian representation does not encode the signal of different frequencies at different Gaussians, naively filtering out the small Gaussians will result in a hole or part missing in the scene as shown in Fig. 4. To address this issue, we propose to aggregate the small Gaussians to form large Gaussians that encode the low-frequency signal. These large Gaussians would appear when the small Gaussians are filtered out.

Aggregate to Insert Large Gaussians. All 3D Gaussians initialized from the input point cloud at the start of the training belong to the finest level $l = 1$. They are densified by splitting and cloning as in [12], and all the densified Gaussians would inherit the same level. After the warm-up stage of the first 1,000 iterations, we introduce coarse-level Gaussians by aggregating fine-level Gaussians that are too small as visualized in Fig. 5 and described in Algorithm 1. The procedure is outlined as follows:

1. For all levels $\{l_m \mid 2 \le l_m \le l_{max}\}$, we render all 3D Gaussians from $[1, l_m - 1]$ at the 4^{l_m-1} times downsam-

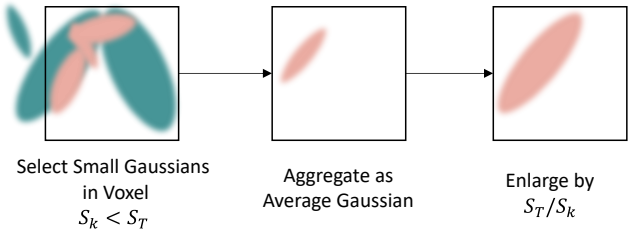

Figure 5. Large Gaussians are created by aggregating the small Gaussians in each voxel below the pixel coverage threshold, and then enlarged by the pixel coverage multiplier.

pled resolution of all training images. All 3D Gaussians with the minimal "pixel coverage" S_k smaller than the filter threshold S_T are chosen for the aggregation.

2. The chosen 3D Gaussians are binned by a $(400/l_m)^3$ resolution voxel grid based on their positions. The attributes of all Gaussians within each voxel are aggregated to create a new Gaussian using average pooling, including position, scaling, opacity and color. More details are included in the supplementary.
3. Based on the average "pixel coverage" S_{avg} of Gaussians in each voxel, the scaling of each new Gaussian created is enlarged by S_T/S_{avg} so that it is of a size suitable to be rendered at l_m. This new Gaussian belongs to level l_m.

Not all Gaussians from the fine levels are small. Many Gaussians in the background or in the textureless regions are large and do not need to be aggregated. The number of Gaussians created is often fewer than 5% of the final total number of Gaussians.

Algorithm 1 Aggregate Small Gaussians

procedure AGGREGATEGAUSSIANS($\mathcal{G}_{1:K}^{1:l_{max}}$)
  for $l_m \leftarrow 2$ **to** l_{max} **do**
    $S_{1:K}$=PixelCoverage($\mathcal{G}_{1:K}^{1:l_m-1}$, scale 4^{l_m-1})
    $G_{small} = \{\mathcal{G}_k | S_k < S_T, \forall k \in [1:K]\}$
    for $n \leftarrow 1$ **to** $(400/l_m)^3$ **do**
      $G_n = \{\mathcal{G}_k | \mathcal{G}_k \text{ in voxel } n, \forall \mathcal{G}_k \in G_{small}\}$
      $\mathcal{G}_{new,n}^{l_m} =$ Enlarge(Average(G_n))
      InsertIntoScene($\mathcal{G}_{new,n}^{l_m}$)
    end for
  end for
end procedure

Multi-Scale Training and Selective Rendering. After the large Gaussians are added, the model is trained with both the original images and the downsampled images. A maximum pixel coverage S_k^{max} and a minimum pixel coverage S_k^{min} of each Gaussian are stored for the selective rendering. If the rendering downsample scale equals to

Algorithm 2 Selective Rendering Based on Pixel Coverage

procedure SELECTIVERENDER($\mathcal{G}_{1:K}^{1:l_{max}}$, scale l_r)
  $S_{1:K}$ =PixelCoverage($\mathcal{G}_{1:K}^{1:l_{max}}$)
  $G_1 = \{\mathcal{G}_k | S_k/S_k^{max} \leq S_{rel}^{max}, \forall k\}$
  $G_2 = \{\mathcal{G}_k | S_k/S_k^{min} \geq S_{rel}^{min} \vee S_k \geq S_T, \forall k\}$
  $G_{l_r} = \{\mathcal{G}_k^l | l = l_r, \forall k\}$
  for $\mathcal{G}_k^l \in G_{l_r}$ **do**
    UpdateRange($S_k^{max}, S_k^{min}, S_k$)
  end for
  return Render($G_1 \cap G_2, l_r$)
end procedure

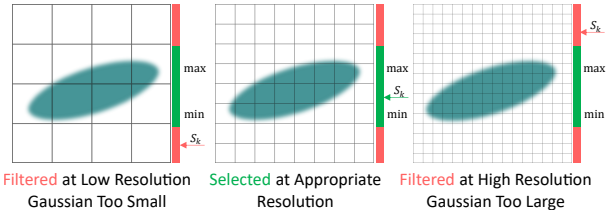

Figure 6. Based on the rendering resolution, the current pixel coverage of a Gaussian relative to its minimum and maximum pixel coverages determines whether it is selected for rendering.

the downsample scale when the Gaussian $\mathcal{G}_k$ is created, its S_k^{max} and S_k^{min} values are updated with the new pixel coverage S_k:

$$\begin{aligned} S_k^{'max} &= max(\lambda_1 S_k^{max}, S_k), \\ S_k^{'min} &= min(\lambda_2 S_k^{min}, S_k), \end{aligned} \tag{5}$$

where λ_1 and λ_2 are decay coefficients taking the empirical value of 0.95 and 1.05, respectively.

During rendering at any resolution or camera distance, a Gaussian is selected for rendering if its pixel coverage S_k on the screen satisfies the following condition:

$$(\frac{S_k}{S_k^{max}} \leq S_{rel}^{max}) \wedge (\frac{S_k}{S_k^{min}} \geq S_{rel}^{min} \vee S_k \geq S_T), \tag{6}$$

where S_{rel}^{max} and S_{rel}^{min} are the maximum and minimum relative pixel coverage taking the empirical values of 1.5 and 0.5 respectively. If the pixel coverage of a Gaussian is too much larger than the S_k^{max}, it is filtered out from rendering. Similarly, if it is too much smaller than the S_k^{min} and is smaller than S_T, it is filtered out from rendering (*cf*. Fig. 6). The absolute S_T threshold is used to preserve the large Gaussians from the lower scales, as they do not cause the aliasing problem if their screen size is not sufficiently small. This selective rendering procedure is described in Algorithm 2. Additionally, even if the Gaussians from the finest level are too large or the Gaussians from the coarsest level are too small, they are not filtered to render beyond the maximum and below the minimum training resolutions.

Table 1. Quantitative comparison and ablation study on the 360 dataset [3] at various downsampled scales, with time in "ms".

Scale	1x			4x			16x			64x		
Metric	PSNR↑	LPIPS↓	Time↓	PSNR↑	LPIPS↓	Time↓	PSNR↑	LPIPS↓	Time↓	PSNR↑	LPIPS↓	Time↓
3D Gaussian[12]	**27.52**	**0.142**	10.5	22.50	0.137	9.3	17.79	0.149	27.9	15.23	N.A.	103.3
3DGS + MS Train	27.35	0.155	11.3	23.50	**0.126**	7.7	20.21	0.115	22.8	19.38	N.A.	84.8
3DGS + Filter Small	27.40	0.153	10.0	23.81	0.149	5.4	20.02	0.186	4.8	17.38	N.A.	**4.6**
3DGS + Insert Large	18.02	0.604	9.7	18.75	0.531	**2.5**	20.23	0.256	**2.7**	21.53	N.A.	7.1
Our Full Method	27.39	0.155	**9.1**	**24.82**	0.132	5.4	**24.75**	**0.066**	4.9	**25.35**	N.A.	4.9

Table 2. Quantitative comparison and ablation study on Tank and Temples dataset [13] at various downsampled scales, with time in "ms".

Scale	1x			4x			16x			64x		
Metric	PSNR↑	LPIPS↓	Time↓	PSNR↑	LPIPS↓	Time↓	PSNR↑	LPIPS↓	Time↓	PSNR↑	LPIPS↓	Time↓
3D Gaussian[12]	23.74	**0.096**	6.5	19.70	0.105	11.1	15.61	0.068	43.4	13.88	N.A.	82.6
3DGS + MS Train	22.97	0.118	6.0	21.46	**0.086**	9.6	18.56	0.049	37.4	16.54	N.A.	71.7
3DGS + Filter Small	**23.78**	0.100	5.6	20.12	0.107	4.5	17.41	0.072	4.4	14.95	N.A.	4.7
3DGS + Insert Large	10.84	0.697	**5.1**	11.15	0.703	**1.7**	11.73	0.447	**1.7**	12.62	N.A.	**2.5**
Our Full Method	23.46	0.111	7.6	**21.92**	0.087	4.7	**20.91**	**0.034**	4.8	**19.67**	N.A.	5.9

Table 3. Quantitative comparison and ablation study on the Deep Blending dataset [8] at various downsampled scales, with time in "ms".

Scale	1x			4x			16x			64x		
Metric	PSNR↑	LPIPS↓	Time↓	PSNR↑	LPIPS↓	Time↓	PSNR↑	LPIPS↓	Time↓	PSNR↑	LPIPS↓	Time↓
3D Gaussian[12]	29.65	**0.094**	8.6	27.48	0.066	7.5	22.06	0.067	20.7	17.75	N.A.	59.7
3DGS + MS Train	29.46	0.102	6.6	28.18	**0.062**	5.3	24.13	0.055	14.3	20.03	N.A.	41.3
3DGS + Filter Small	29.68	0.095	6.7	28.26	0.064	4.2	24.52	0.078	3.6	18.29	N.A.	**3.2**
3DGS + Insert Large	20.59	0.379	**4.6**	20.83	0.336	**1.6**	21.29	0.143	**2.1**	20.10	N.A.	4.2
Our Full Method	**29.70**	0.096	7.4	**28.43**	0.064	3.9	**27.66**	**0.036**	3.4	**25.70**	N.A.	3.4

The pixel coverage range of each Gaussian allows the model to maintain multi-scale Gaussians for different levels of detail. The appropriate subset of Gaussians is chosen for rendering at different resolutions and distances. More smaller Gaussians encoding the high-frequency information are rendered at high resolution, and fewer and larger Gaussians encoding the low-frequency information are rendered at low resolution for less aliasing effect and faster speed.

5. Experiments

In this section, we present a comprehensive evaluation of our proposed model, which is grounded on the implementation framework of the official release of the 3D Gaussian Splatting code. To achieve a similar training time as the baseline model, our models are trained for 40000 iterations with all other hyper-parameters unchanged. All rendering speed are measured on a single RTX3090 GPU. We evaluate the performance of the vanilla 3D Gaussian Splatting[12] algorithm and our model on the multi-scale 360[3], Tank And Temples[13], and Deep Blending[8] dataset, aligned with the data used by the original paper. These datasets cover a wide range of object centric, indoor, and ourdoor scenes.

Our evaluation focuses on the rendering quality and speed at multiple downsampling scales of 1x, 4x, 16x, and 64x derived from the test views. The rendering quality is measured in PSNR and LPIPS, while the speed is measured in per-image rendering time. This multi-scale evaluation is aimed at simulating the rendering performance in scenarios of low-resolution imaging or when captured from distant cameras. More detailed evaluations, including the results for more resolution scales and per-scene decomposition, are included in the supplementary materials due to the space constraint. Additionally, the supplementary materials include a video that offers an intuitive qualitative comparison of the two algorithms, vividly demonstrating the improvement of our algorithm in quality and speed from multiple viewpoints.

Quantitative Comparison. As shown in Tab. 1, Tab. 2, and Tab. 3, our method can achieve substantial quality and speed improvements compared to the original 3D Gaussian

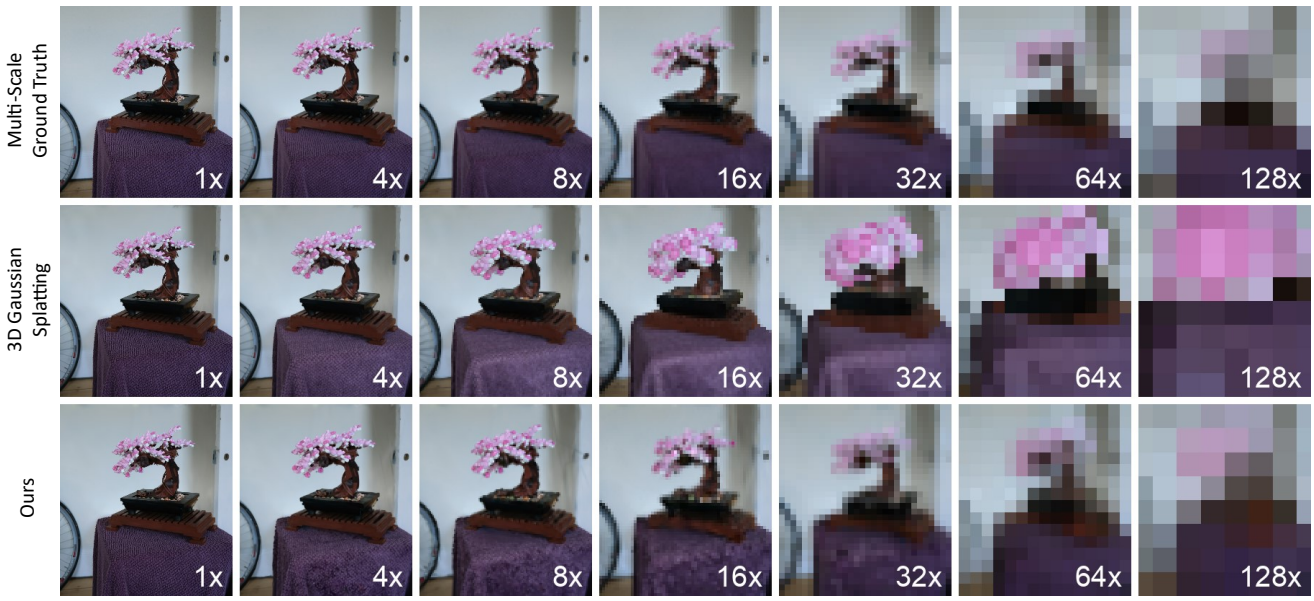

Figure 7. Qualitative Comparison on 360 dataset[3] for different resolution scales.

Figure 8. Qualitative Comparison on Tank and Temples dataset[13] for different resolution scales.

Splatting [12] at lower resolutions. The quality and speed improvements become more pronounced as the resolution reduces, with the most noticeable 6-10dB PSNR and 20-30× speed gain at the 64× resolution scale. As the resolution reduces, the original splatting algorithm slows down while our method accelerates. The rendering quality and speed at the original resolution (1×) remain comparable, indicating the effectiveness of our multi-scale Gaussians in representing both the high and low resolutions together.

Qualitative Comparison. We present the qualitative comparison with the original 3D Gaussian Splatting [12] shown in Fig. 7, Fig. 8, and Fig. 9. At higher resolutions (1×-8×), both ours and the original algorithm can render the novel view rather faithfully. However, as the resolution reduces further(16×-64×), the original splatting algorithm produces severe artifacts, where the foreground becomes larger and larger, dominating the pixel colors as explained in Sec. 3.2. In contrast, the images rendered by our method closely resemble the ground truth across all resolution scales.

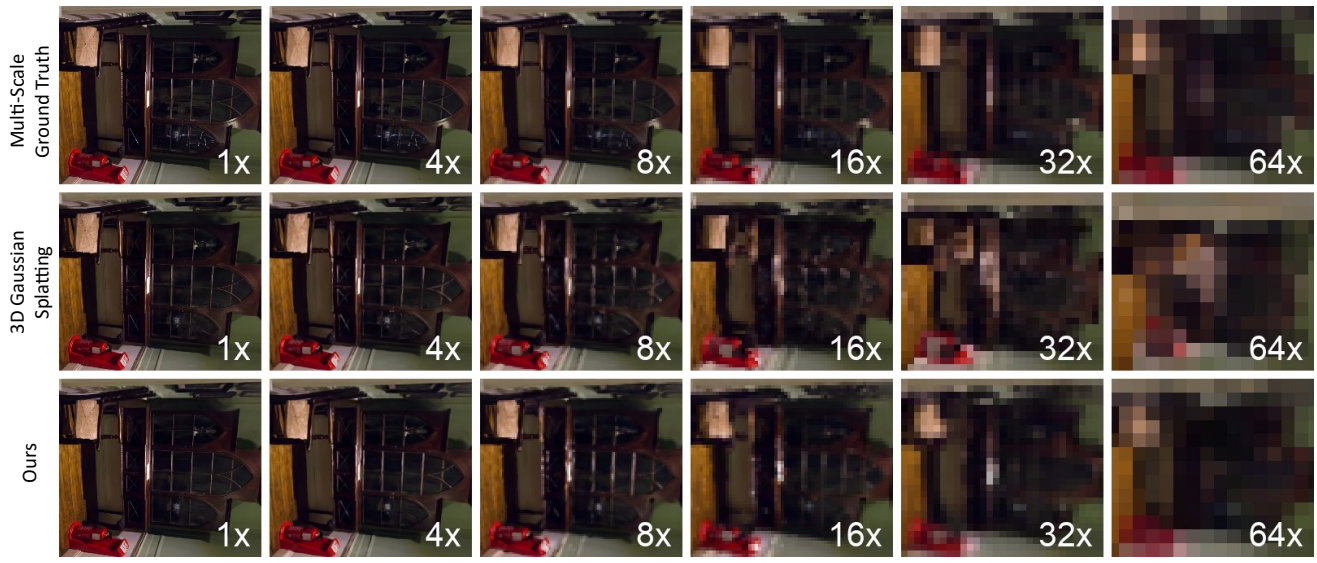

Figure 9. Qualitative Comparison on Deep Blending dataset[8] for different resolution scales.

Ablations. To evaluate the effectiveness of the different components proposed, we present the ablation quantitative results in Tab. 1, Tab. 2, and Tab. 3 and the ablation qualitative results in the supplementary. The three ablation methods evaluated are named "3DGS+MS Train", "3DGS+Filter Small", and "3DGS+Insert Large". "3DGS+MS Train" reports the result with multi-scale training on top of the original 3D Gaussian splatting. The "3DGS+Filter Small" reports the result with small Gaussian filtering using pixel coverage **and** the multi-scale training, which is needed to update the maximum and minimum pixel coverage. Similarly, the "3DGS+Insert Large" reports the result with large Gaussian insertion **and** the multi-scale training.

The ablation results reflect that multi-scale training marginally improves low-resolution rendering quality, but the rendering speed remains very slow. When filtering out the small Gaussians with multi-scale training, the speed at low resolution is increased by 20-30$\times$ with minimal rendering quality loss. The speed gain is caused by the considerably fewer Gaussians rendered. When inserting the large Gaussians and training with multi-scale supervision, without the small Gaussian filtering, the rendering quality drops significantly because the details of the scene are covered with large Gaussians completely for all resolutions. However, when adding the large Gaussians together with the small Gaussian filtering, the rendering quality and speed at low resolution are enhanced significantly without jeopardizing the high-resolution quality. This indicates the effectiveness of all three components and the full method proposed.

6. Limitations

Since all Gaussian filtering of our proposed method relies on the pixel coverage, it can only be done after the initial splatting process when the coverage is calculated. Although the splatting of individual Gaussians are performed in parallel and does not takes more time at lower resolution, it is still a considerable overhead when rendering at a very low resolution. Even if a very small portion of the Gaussians are used for rendering in the end, all Gaussians still need to be splatted. This is the main reason why our rendering time is not decreased linearly as the resolution decreases. In our future work, we will look into a more lightweight criteria to filter small and large Gaussians before splatting them onto the screen to achieve an even faster rendering speed.

7. Conclusion

We analyzed the cause of the severe aliasing artifact and speed degradation of the existing 3D Gaussian splatting. We identified the key challenge of mitigating the aliasing for 3D Gaussian splatting lies in representing the scene with Gaussians of appropriate scale. Based on this observation, we propose to calculate the pixel coverage of 3D Gaussians during splatting and use this as a criteria for selective rendering. Gaussians that are too large or too small at the current rendering resolution are filtered for anti-aliasing and speed improvements. We also proposed to insert large Gaussians by aggregating small Gaussians during training to preserve the low frequency details and prevent part missing. Our experiments on various datasets support the effectiveness of our algorithm in rendering quality and speed at both high and low resolution, mitigating the severe aliasing artifact of the original 3D Gaussian splatting.

Acknowledgement. This work is supported by the Agency for Science, Technology and Research (A*STAR) under its MTC Programmatic Funds (Grant No. M23L7b0021).

References

[1] Kurt Akeley. Reality engine graphics. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, page 109–116, New York, NY, USA, 1993. Association for Computing Machinery. 2

[2] 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. *ICCV*, 2021. 1, 2

[3] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. *CVPR*, 2022. 2, 6, 7

[4] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Zip-nerf: Anti-aliased grid-based neural radiance fields. *ICCV*, 2023. 2

[5] Kristof Beets and David L. Barron. Super-sampling anti-aliasing analyzed. 2000. 2

[6] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. In *European Conference on Computer Vision (ECCV)*, 2022. 2

[7] Carl Erikson. Polygonal simplification. Technical Report 96-016, 1996. 2

[8] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. Deep blending for free-viewpoint image-based rendering. *ACM Trans. Graph.*, 37(6), 2018. 6, 8

[9] Tan Kim Heok and D. Daman. A review on level of detail. In *Proceedings. International Conference on Computer Graphics, Imaging and Visualization, 2004. CGIV 2004.*, pages 70–75, 2004. 2

[10] Wenbo Hu, Yuling Wang, Lin Ma, Bangbang Yang, Lin Gao, Xiao Liu, and Yuewen Ma. Tri-miprf: Tri-mip representation for efficient anti-aliasing neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 19774–19783, 2023. 3

[11] Jorge Jimenez, Diego Gutiérrez, Jason Yang, Alexander Reshetov, Pete Demoreuille, Tobias Berghoff, Cedric Perthuis, Henry Yu, Morgan Mcguire, Timothy Lottes, Hugh Malan, and Emil Persson. Filtering approaches for real-time anti-aliasing. *ACM SIGGRAPH 2011 Courses, SIGGRAPH'11*, 2011. 2

[12] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42 (4), 2023. 1, 2, 3, 4, 6, 7

[13] 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. 6, 7

[14] Timothy Lottes. Fxaa. Technical report, Nvidia, 2011. 2

[15] 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. In *ECCV*, 2020. 1, 2

[16] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, 2022. 1, 2

[17] Seungtae Nam, Daniel Rho, Jong Hwan Ko, and Eunbyung Park. Mip-grid: Anti-aliased grid representations for neural radiance fields. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. 3

[18] Lance Williams. Pyramidal parametrics. *SIGGRAPH Comput. Graph.*, 17(3):1–11, 1983. 2

[19] M. Zwicker, H. Pfister, J. van Baar, and M. Gross. Ewa volume splatting. In *Proceedings Visualization, 2001. VIS '01.*, pages 29–538, 2001. 2, 3, 4