

# MeshGS: Adaptive Mesh-Aligned Gaussian Splatting for High-Quality Rendering

Jaehoon Choi<sup>1</sup>, Yonghan Lee<sup>1</sup>, Hyungtae Lee<sup>3</sup>,  
Heesung Kwon<sup>2</sup>, and Dinesh Manocha<sup>1</sup>

<sup>1</sup> University of Maryland, College Park, USA

<sup>2</sup> DEVCOM Army Research Laboratory, Adelphi, USA

<sup>3</sup> BlueHalo, Rockville, USA

{kevchoi, lyhan12}@umd.edu, hyungtae.lee@bluehalo.com,  
heesung.kwon.civ@army.mil, dm@cs.umd.edu

**Abstract.** Recently, 3D Gaussian splatting has gained attention for its capability to generate high-fidelity rendering results. At the same time, most applications such as games, animation, and AR/VR use mesh-based representations to represent and render 3D scenes. We propose a novel approach that integrates mesh representation with 3D Gaussian splats to perform high-quality rendering of reconstructed real-world scenes. In particular, we introduce a distance-based Gaussian splatting technique to align the Gaussian splats with the mesh surface and remove redundant Gaussian splats that do not contribute to the rendering. We consider the distance between each Gaussian splat and the mesh surface to distinguish between tightly-bound and loosely-bound Gaussian splats. The tightly-bound splats are flattened and aligned well with the mesh geometry. The loosely-bound Gaussian splats are used to account for the artifacts in reconstructed 3D meshes in terms of rendering. We present a training strategy of binding Gaussian splats to the mesh geometry, and take into account both types of splats. In this context, we introduce several regularization techniques aimed at precisely aligning tightly-bound Gaussian splats with the mesh surface during the training process. We validate the effectiveness of our method on large and unbounded scene from mip-NeRF 360 and Deep Blending datasets. Our method surpasses recent mesh-based neural rendering techniques by achieving a 2dB higher PSNR, and outperforms mesh-based Gaussian splatting methods by 1.3 dB PSNR, particularly on the outdoor mip-NeRF 360 dataset, demonstrating better rendering quality. We provide analyses for each type of Gaussian splat and achieve a reduction in the number of Gaussian splats by 30% compared to the original 3D Gaussian splatting.

**Keywords:** Neural rendering · Gaussian splatting · Mesh

## 1 Introduction

Rendering realistic depictions of real-world objects and environment has long been a significant challenge with numerous practical applications in fields like

computer vision, graphics, and AR/VR. Most graphics and game engines use polygonal mesh-based representations of the objects in the scene [19], including real-time rendering and animation. Recently, Neural Radiance Fields (NeRF) [2, 3, 30, 31] have demonstrated promising capabilities for 3D reconstruction and novel-view synthesis. However, NeRFs, which rely on volumetric rendering, suffer from slow rendering speeds and are not easily compatible with modern graphics engines [7, 14, 41]. Instead, researchers have been investigating methods based on neural implicit representation to facilitate mesh-based rendering. MobileNeRF [5] utilizes a classical rasterization pipeline, incorporating z-buffers and fragment shaders, to train neural fields based on polygonal mesh representations. Subsequently, various methods [27, 29, 35, 41, 47] first reconstruct a 3D mesh and then utilize this mesh to train appearance models employing high-speed rasterization [18, 24, 39]. These techniques facilitate real-time rendering and demonstrate potential for integration into graphics and game engines. However, all of these methods are constrained by the quality of the mesh surface, particularly exhibiting a loss in rendering quality when dealing with highly detailed structures, a well-known limitation of mesh-based approaches [21, 43]. Given the inherent difficulty in achieving perfect geometric reconstruction for real-world unbounded scenes, rendering algorithms must demonstrate robustness in handling mesh artifacts such as thin-level structures.

More recently, the 3D Gaussian Splatting method [22] has attracted considerable interest due to its high-quality rendering and real-time rendering speed. Compared to mesh-based representation, 3D Gaussian splats excel in capturing intricate details of the scene with high fidelity due to their high degree of freedom in splat positions and shapes. However, 3D Gaussian splats are limited in their ability to represent the geometry of a scene and are not widely applicable to various applications. SuGaR [13] extracts mesh solely based on 3D Gaussian splats. Their method shows detailed structure at the object level and tightly binds the 3D Gaussian splats to the surface of the mesh. Other previous works adopt a similar method for tightly binding the 3D Gaussian splat to the mesh triangles [34, 42]. This is very useful for rigging, animation, or any deformation tasks. However, this assumes that the mesh can cover nearly all geometrical elements of the target object, such as the head or a human avatar. In unbounded scenes, since meshes frequently exhibit artifacts, we need better strategies to effectively bind 3D Gaussian splats to the mesh surface.

**Main Results:** In this paper, we present a new approach to integrate the strengths of mesh and 3D Gaussian splats in rendering and geometry for unbounded large-scale scenes. For geometry reconstruction, we first employ the BakedSDF [47] technique and extract a 3D mesh using Marching cube algorithm [28]. In unbounded large-scale scenes, meshes typically consist of a large number of triangles to represent complex geometric objects in the scene. Properly initializing Gaussian splats with millions of vertices and triangles becomes exceptionally challenging. Inspired by LTM [7], we employ mesh decimation and eliminate redundant triangles to extract a lightweight mesh with fewer triangles. Based on this lightweight mesh, we initialize the Gaussian splats for each mesh

triangle. Subsequently, we introduce a mesh-based Gaussian splatting method that consists of two components. First, we conduct forward splatting considering mesh geometry and eliminate any Gaussian splats that are occluded by the mesh surface. This impacts our training process to align the Gaussian splats with the explicit mesh geometry and to remove redundant Gaussian splats positioned behind the mesh surface considering all training viewpoints.

Secondly, we segregate tightly-bound Gaussian splats from loosely-bound Gaussian splats using the distance between each Gaussian splat's center and their corresponding mesh triangles. As current mesh reconstruction methods cannot fully capture the 3D geometry of real-world scenes, we introduce loosely-bound Gaussian splats to depict regions where explicit mesh-based representation is not available, thus representing their appearance. We employ different training and densification strategies for each type of splat. The tightly-bound Gaussian splats are aligned with the mesh surface through various geometric regularization techniques, whereas the optimization of loosely-bound Gaussian splats relies solely on image supervision.

Our main contributions include:

- We present a novel approach for mesh-based Gaussian splatting. We integrate Gaussian splats with the triangle mesh and introduce a new method for initializing and training Gaussian splats based on the triangle mesh.
- We differentiate between two types of Gaussian splats based on their distance from the mesh surface. Subsequently, we introduce different training strategies for each type of Gaussian splat to accommodate different rendering requirements.
- Our method surpasses the rendering quality of the state-of-the-art mesh-based neural rendering algorithms [7, 47] by 2dB PSNR on mip-NeRF360 dataset. Compared to the original 3D Gaussian splatting [22], we achieve comparable rendering quality while utilizing 30% fewer Gaussian splats for rendering.

## 2 Related Work

**Neural Rendering.** Neural Rendering has recently received considerable attention. Neural Radiance Fields (NeRF) [30] utilize a multi-layer perceptron (MLP) to encode a scene by applying differentiable volumetric rendering to represent RGB radiance and density. Various methods have been proposed to accelerate the training time and rendering speed by leveraging data structures [4, 9, 40], encoding techniques [2, 3, 31], and baking processes [16, 36]. Other researchers utilize neural implicit surface representation [12], employing MLP to map coordinates to a signed distance function (SDF) or occupancy grids. These approaches [6, 26, 32, 43, 46, 47, 49] involve converting volume density to SDF or occupancy and perform training with differentiable volumetric rendering.

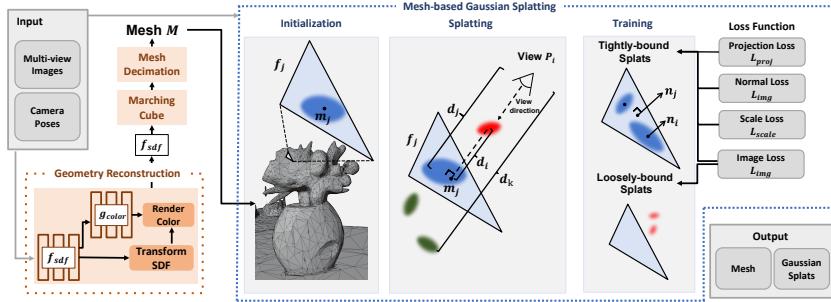
**Mesh-Based Rendering.** Polygonal mesh-based rendering [39] is a classic problem in computer graphics to visualize 3D models on a 2D screen using

rasterization. To address rendering speed issue, recent neural rendering methods also leverage explicit mesh and bake the neural appearance, enabling mesh-based neural rendering. SNeRG [16] builds on deferred shading [8] and bakes neural appearance into sparse voxel grid. MobileNeRF [5] directly optimizes the triangle faces and embeds opacities and features into texture maps through a baking process. Subsequently, other approaches involve baking the neural fields into textures and representing view-dependent effects using either a small neural network [27, 29, 35, 41] or spherical Gaussians [47]. VMesh [14] proposes hybrid representation combining mesh and volume rendering to achieve high efficiency and expressiveness. Shells [45] extract the mesh and conduct volumetric rendering within a narrow band surrounding the mesh. LTM [7] employs significant mesh decimation to achieve efficient mesh representation for large-scale scenes while simultaneously training neural appearance and geometry through differentiable rendering [24]. Many existing methods encounter limitations due to mesh artifacts stemming from 3D reconstruction techniques in real-world scenarios, posing challenges in achieving photorealistic rendering quality with neural fields. Our approach aims to address this limitation by leveraging Gaussian splatting for mesh-based rendering.

**Point-Based Rendering and Gaussian Splatting** Point-based rendering is a technique for rendering 3D scenes using point-based geometry representation instead of traditional polygon meshes. In this context, a variety of recent approaches have been proposed for the point-based rendering [23, 37], sphere rendering [25], and differentiable splatting [48]. The 3D Gaussian Splatting [22] employs 3D Gaussian splats [51], which are rendered through sorting and rasterization processes. SuGaR [13] utilizes Poisson reconstruction [20] to extract a mesh from 3D Gaussian splats and bind them to the triangle for rendering. GaussianAvatars [34] and GaMeS [42] focus on tightly binding 3D Gaussian splats to meshes and rendering dynamic scenes, including human avatars or single objects. However, these methods are mainly tailored for a single object or human avatars. Our goal is to develop a new mesh-based Gaussian splatting technique suitable for large-scale scenes, even in cases where the mesh only captures moderate levels of geometry. More recently, 2D Gaussian splatting [17] utilize the 2D Gaussian primitives instead of 3D Gaussian splats. In our paper, we focus solely on 3D Gaussian splat representation and the integration of 3D Gaussian splats with the mesh. We provide a more detailed analysis comparing 2DGs [17] with our method in the supplementary material.

### 3 Our Method

Given posed images, our goal is to reconstruct both the precise geometry and rendering high-fidelity images of unbounded large-scale scenes. An overview of our pipeline is illustrated in Fig. 1. First, we employ the BakedSDF [47] technique and extract a 3D mesh  $M$  via Marching Cubes [28]. Following the method in [7], we significantly reduce the number of triangles in the mesh to facilitate the



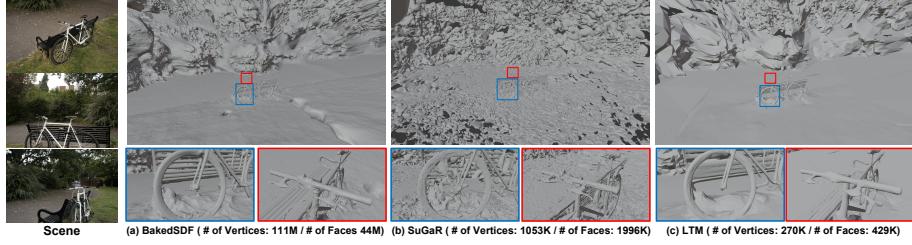
**Fig. 1: Our Approach:** In Section 3.1, we jointly train the geometry  $f_{sdf}$  and appearance  $g_{color}$  using differentiable volumetric rendering (shown in orange). Then, we extract the lightweight mesh  $M$  which is important for the initialization of Gaussian splats. Next, in Section 3.2., we present the mesh-based Gaussian splatting for training, which involves the removal of Gaussian splats (green Gaussian splats) occluded by mesh surface. Given the distance  $d_i$  between each Gaussian splat and its corresponding triangle, we distinguish between tightly-bound Gaussian splats (blue splat) and loosely Gaussian splats (red splat). In Section 3.3, we introduce the training and densification strategy for both tightly-bound Gaussian splats and loosely Gaussian splats.

integration with Gaussian splats to generate high-quality rendering. Next, we bind the Gaussian splats  $G$  to the explicit mesh to represent their appearance.

### 3.1 Geometry Reconstruction

During this stage, we leverage the neural surface representation to extract a mesh and decimate it for integrating the Gaussian splats with the mesh. The original 3D Gaussian Splatting method (3DGSS) [22] is specifically tailored to sparse point clouds generated by Structure-from-Motion (SfM) techniques [38]. Dense point clouds are not conducive to rendering and can even be detrimental to the rendering quality. Hence, it is imperative to meticulously design mesh reconstructions for the initialization of Gaussian splats. Following the approach of BakeSDF [47], we jointly train two multilayer perceptrons (MLPs) to encode both the signed distance field  $f_{sdf}$  and the appearance field  $g_{color}$ , as shown in the *Geometry Reconstruction* block in Fig. 1. Both MLPs are optimized using differentiable volumetric rendering and Eikonal regularization techniques [12]. Details are described in [47]. Then, we employ Marching Cube algorithm [28] to obtain a mesh  $M = \{V, F\}$  with vertices  $V$  and faces  $F$  extracted from  $f_{sdf}$ .

However, in large-scale scenes, the output mesh occupies a significant number of triangles (average 27 millions) [47]. Given occupying a large amount of triangles, they still cannot represent detailed geometry or reflective surfaces due to the strong smoothness regularization. Furthermore, a large number of triangles are allocated to distorted regions, especially in background regions. Several prior mesh-based Gaussian splatting methods [13, 34, 42] assume that at least one Gaussian splat is attached to the mesh surface to represent the appearance of



**Fig. 2:** The examples of extracted mesh from BakedSDF [47], SuGaR [13], and LTM [7]. We employ LTM [7] to extract a lightweight mesh with moderate triangle count, using it for the initialization of Gaussian splats. The primary reason is that LTM utilizes a minimal number of triangles while maintaining geometric quality. Zoomed-in regions visualize that the mesh frequently lacks highly detailed geometry or complex structures. “K” and “M” denotes the  $10^3$  and  $10^6$  units, respectively.

its triangles. If we strictly follow their strategy, we allocate redundant Gaussian splats to distorted regions, resulting in a rapid increase in GPU memory usage after just a few training and densification steps. In Fig.2 (a), the mesh reconstructed by BakedSDF occupies 44 million triangles to represent geometry. The original 3DGS [22] necessitates 4 million Gaussian splats to achieve high-fidelity rendering quality, yet the very large number of Gaussian splats undermines rendering quality. SuGaR also imposes constraints on the vertex count (10K~15K); however, a significant number of vertices are needed to faithfully represent the geometry, particularly in expansive, unbounded scenes. We observe that SuGaR sacrifices geometric quality, particularly in background regions (see Fig.2 (b)), and then trains Gaussian splats attached to mesh faces. However, since we aim to achieve moderate geometry quality, adopt the methodology of LTM [7]. They utilize aggressive mesh decimation based on the Quadric Error Metric [10, 11] to efficiently represent moderate geometry with a fewer number of triangles ( see Fig. 2 (c)). Using this mesh as a foundation, we can easily integrate Gaussian splatting with the explicit mesh.

### 3.2 Mesh-based Gaussian Splatting

**3D Gaussian Splatting:** 3D Gaussian Splatting (3D-GS) [22] employs millions of 3D Gaussians  $G$  to represent a 3D scene. These 3D Gaussian splats are constructed based on a sparse point cloud obtained from Structure from Motion [38], incorporating various attributes to model both the geometry and appearance of the scene. The geometry attributes of each 3D Gaussian (ellipsoids)  $G_i$  is defined by its center (position)  $p_i \in \mathbb{R}^3$  and 3D covariance matrix  $\Sigma_i \in \mathbb{R}^{3 \times 3}$  in world space:

$$G_i(x) = e^{-\frac{1}{2}(x-p_i)^T \Sigma_i^{-1}(x-p_i)}. \quad (1)$$

To constrain valid covariance matrices, 3D-GS utilizes a semi-definite parameterization  $\Sigma_i = R_i S_i S_i^T R_i^T$  with two learnable components: a scaling matrix

$S_i \in \mathbb{R}^3$  and a rotation matrix  $R_i \in \mathbb{R}^{3 \times 3}$ , encoded by a quaternion  $q \in \mathbb{R}^4$ . For rendering, given a viewpoint defined by rotation  $R_{cw} \in \mathbb{R}^{3 \times 3}$  and translation  $t_{cw} \in \mathbb{R}^3$ , the 3D Gaussians are projected into the camera space:  $p_i^c = R_{cw}p_i + t_{cw}$  and  $\Sigma_i^c = R_{cw}\Sigma_i R_{cw}^T$ . Next, to transform camera space into ray space, 3D-GS utilize the affine transformation  $J_i \in \mathbb{R}^{2 \times 3}$  defined by  $p_i^c$  following [51], and the 2D covariance matrix in ray space can then be defined as  $\Sigma'_i = J_i\Sigma_i^c J_i^T$ . We can obtain the 2D Gaussian  $g_i$  defined by corresponding 2D covariance  $\Sigma'_i$  and 2D center  $p'_i$  as follows:

$$g_i(x) = e^{-\frac{1}{2}(x-p'_i)^T \Sigma_i'^{-1}(x-p'_i)}. \quad (2)$$

Additionally, each Gaussian includes an per-point opacity  $o_i \in [0, 1]$  and the view-dependent color  $c_i$  represented by spherical harmonic (SH) [22]. 3D-GS orders all the Gaussians that contributes to a pixel and renders 2D image via alpha blending with the sorted Gaussians using the following equation:

$$c = \sum_{i=1}^n c_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j), \quad \alpha_i = o_i g_i(x). \quad (3)$$

The blending weight  $\alpha_i$  is obtained by computing the 2D projection of the 3D Gaussian  $g_i(x)$  multiplied by a per-point opacity  $o_i$ .

**Distance-based Gaussian Splatting:** In this section, we present our approach that builds the connection between large-scale mesh and Gaussian splats. Before training, we initialize a 3D Gaussian splats at the center  $m_j$  of each triangles  $f_j$ . Given mesh  $M$ , we can render a depth  $D_i$  from a given viewpoint  $P_i$ . When rendering for each viewpoint, we utilize the corresponding viewpoint depth map to mask out the Gaussian splats occluded by the surface. In light of our assumption that the mesh surface is entirely opaque, Gaussian splats occluded by it should be excluded for rendering purposes. In the *Splatting* block shown in Fig. 1, given that the depth  $d_k$  of the green Gaussian splats is greater than that of  $d_j$ , those Gaussian splats located behind the mesh surface do not contribute to rendering pixels. Furthermore, during the training process, we assess the visibility of Gaussian splats across all training views and identify those located behind the mesh surface that are never observed by any of the training views. Then, we remove all redundant Gaussian splats situated behind the mesh surface at specific training iterations.

Given that reconstructed meshes in real-world environments often exhibit geometric artifacts, our goal is to render not only well-represented regions but also those inaccuracies resulting from severe mesh artifacts. When we tightly bind all Gaussian splats exclusively to the mesh surface, rendering performance suffers, particularly when addressing severe mesh artifacts. Therefore, during the training step, We calculate the distance  $d_i$  between each Gaussian splat's center  $p_i$  and corresponding mesh faces  $f_j$ . We treat tightly-bound Gaussian splats  $G_i^t$  ( $d_i < d_{th}$ ) and loosely-bound Gaussian splats  $G_i^l$  ( $d_i > d_{th}$ ), distinctly. If the Gaussian splats are located at a distance greater than a certain threshold  $d_{th}$  from the mesh surface, we classify them as loosely-bound Gaussian splats. We aim to flatten  $G_i^t$  and align these splats with the mesh surface.

### 3.3 Training for Tightly-bound and Loosely-bound Gaussian Splats

**Objective Function:** Following 3D-GS [22], we employ the combination of  $L_1$  loss and D-SSIM as the image loss  $L_{img}$  between the rendered image  $\hat{I}_i$  and the ground truth image  $I_i$  expressed as follows:

$$L_{img} = (1 - \lambda_{img})L_1(\hat{I}_i, I_i) + \lambda_{img}SSIM(\hat{I}_i, I_i). \quad (4)$$

We utilize the normal consistency loss  $L_{nc}$  to refine tightly-bound Gaussian splats  $G^t$  to align with the corresponding mesh surface. To estimate normals  $n_i$  for Gaussian splats, we first identify the minimum scale axis among the scale parameters  $S_i$ , and then apply the rotation matrix  $R_i$  to transform them into world coordinates. Then, we exploit the cosine similarity between the splat's normals  $n_i$  and the corresponding face normals  $n_f$  as follows:

$$L_{nc} = \sum_{i \in G^t, f \in F} (1 - n_i \cdot n_f). \quad (5)$$

Furthermore, we apply the regularization to suppress the scale parameters  $S_i = (s_1, s_2, s_3) \in \mathbb{R}^3$  of tightly-bound Gaussian splats  $G^t$  and align them with the mesh surface. We regularize the minimum scale and then the maximum scale with threshold value  $\rho$ .

$$L_{scale} = \sum_{i \in G^t} (\lambda_{min}|\min(s_1, s_2, s_3)| + \lambda_{max}|\max(s_1, s_2, s_3) - \rho|). \quad (6)$$

This ensures that the tightly-bound Gaussian splats do not expand excessively in comparison to the corresponding mesh face  $f_j$ , thus hindering the rendering process. Finally, given the mesh  $M$  consisting of face  $f_j$  and its face normal  $n_j$ , we project the Gaussian Splat's center  $P_i$  to the closest point  $s_i$  on the mesh surface.

$$\begin{aligned} s_i &= P_i + ((P_i - v_i) \cdot f_j) \cdot f_j, \\ L_{proj} &= \sum_{i \in G^t} \|s_i - P_i\|_2. \end{aligned} \quad (7)$$

This loss function ensures that the tightly-bound Gaussian splats remain close to the mesh surface, preventing them from moving too far from the surface. The total loss function  $L$  is defined as:

$$L = L_{img} + \lambda_{nc}L_{nc} + L_{scale} + \lambda_{proj}L_{proj}. \quad (8)$$

where  $\lambda$  denotes the weight used to balance between the different loss terms.

**Densification Strategy** Here, we adopt two different densification strategies for loosely-bound Gaussian splats  $G^l$  and tightly-bound Gaussian splats  $G^t$ . Regarding  $G^l$ , we follow the method outlined in the original 3DGS [22]. The original Gaussian densification strategy inherently overlooks the existing 3D geometry. In terms of  $G^t$ , we find the mesh triangle close to  $G^t$  and generate new Gaussian

splats by employing random sampling from the corresponding mesh face. For both cloning and splitting, we conduct point sampling from the triangle and utilize this position to initialize new Gaussian splats. During cloning, additional attributes are copied to the new Gaussians, while during splitting, their scale is divided by a factor of 1.6.

## 4 Experiments

### 4.1 Experiment Settings

**Datasets:** For comparison, we assess the rendering quality of our method using the extensively utilized real-world large-scale scenes published by Mip-NeRF360 [3]. Furthermore, to assess the generalization capability, we employ a different large-scale dataset which are two scenes from Deep Blending [15].

**Implementation Details:** For geometry reconstruction section in Sec 3.1, we utilize the LTM [7] code for extracting mesh from neural field and decimating those mesh. In Sec 3.2, when applying distance-based Gaussian splatting, we add the 0.01 to the depth  $D_i$ . We prune the Gaussian splats located behind the mesh at every 500 training iterations. To distinguish the tightly-bound and loosely-bound Gaussian splats, we set the distance threshold  $d_{th}$  to 0.01. For training Gaussian splatting, we generally follow the original 3DGS paper [22]. we run all experiments for 30k iterations. We set the loss weights to  $\lambda_{img}=0.001$ ,  $\lambda_{nc}=0.1$ ,  $\lambda_{min}=0.1$ ,  $\lambda_{max}=10$ , and  $\lambda_{proj}=50$ . We set the threshold  $\rho$  to 0.1. Following the train/test split in Mip-NeRF360, we subsampled every eight frames for test sets. We evaluate all our method trained with 30K iterations.

### 4.2 Experimental Results

We compare our proposed method with three different approaches: LTM [7], which represents the state-of-the-art method for reconstructing lightweight meshes and training a neural field for the corresponding mesh. SuGaR [13], which builds upon the original 3DGS [22] by reconstructing the mesh and binding the Gaussian splats to the corresponding mesh. 3DGS [22], which relies on sparse point clouds instead of an explicit mesh representation. As BakedSDF [47] has not released public code, we include only metrics reported in their original paper.

In Table 4, we report the rendering quality using the standard PSNR, SSIM [44], and LPIPS [50] on outdoor scenes in mip-NeRF 360 dataset [3]. Generally, mesh-based Gaussian splatting methods, including our method, SuGaR [13], and GaMeS [42], exhibit superior performance (with around 2 PSNR increase) compared to state-of-the-art mesh-based neural rendering techniques such as LTM [7], BakedSDF [47], and others. Our method also demonstrates better quality compared to other recent mesh-based Gaussian splatting methods such as SuGaR [13] (with a 1.3 PSNR increase) and GaMeS [42] (with a 0.9 PSNR increase). Table 5 reports the rendering quality on indoor scenes in mip-NeRF 360 dataset [3]. Our method also exhibits higher rendering quality compared

Method	Mesh	Rendering	Outdoor (PSNR ↑ / SSIM ↑ / LPIPS ↓)					
			Bicycle	Garden	Stump	Mean		
Instant-NGP [31]	X	Volume	22.1 / 0.49 / 0.49	24.5 / 0.65 / 0.31	23.6 / 0.57 / 0.45	23.4 / 0.57 / 0.42		
mip-NeRF 360 [3]	X	Volume	24.4 / 0.68 / 0.30	26.9 / 0.81 / 0.17	26.4 / 0.74 / 0.26	25.9 / 0.75 / 0.24		
3DGS [22]	X	Splat	25.2 / 0.77 / 0.20	27.4 / 0.86 / 0.10	26.5 / 0.77 / 0.21	26.4 / 0.70 / 0.17		
MobileNeRF [5]	O	Mesh	21.7 / 0.43 / 0.51	18.8 / 0.59 / 0.36	23.9 / 0.56 / 0.43	21.5 / 0.53 / 0.43		
NeRF2Mesh [41]	O	Mesh	22.1 / 0.48 / 0.51	23.4 / 0.55 / 0.40	22.5 / 0.54 / 0.46	22.7 / 0.52 / 0.46		
BakedSDF [47]	O	Mesh	22.0 / 0.57 / 0.37	24.9 / 0.75 / 0.21	23.6 / 0.59 / 0.37	23.5 / 0.64 / 0.32		
LTM [7]	O	Mesh	22.4 / 0.52 / 0.44	23.5 / 0.64 / 0.30	23.7 / 0.57 / 0.42	23.2 / 0.58 / 0.39		
SuGaR [13]	O	Splat	23.1 / 0.64 / 0.34	25.3 / 0.77 / 0.22	24.7 / 0.68 / 0.34	24.4 / 0.70 / 0.30		
GaMeS [42]	O	Splat	23.4 / 0.67 / 0.33	26.3 / 0.83 / 0.14	24.6 / 0.66 / 0.35	24.8 / 0.72 / 0.39		
MeshGS (Ours)	O	Splat	24.4 / 0.71 / 0.26	26.8 / 0.85 / 0.12	25.8 / 0.74 / 0.25	25.7 / 0.77 / 0.21		
MeshGS* (Ours)	O	Splat	24.9 / 0.75 / 0.25	26.8 / 0.86 / 0.12	26.3 / 0.76 / 0.22	26.0 / 0.79 / 0.20		

**Table 1: Quantitative Comparison on outdoor mip-NeRF 360 Dataset.** Splat-based rendering demonstrates superior performance compared to mesh-based neural rendering techniques such as LTM [7] and BakedSDF [47]. MeshGS\* did not utilize any regularization techniques to tightly align Gaussian splats; instead, it solely applied image loss  $L_{img}$  for training. Compared to GaMeS [42] and SuGaR [13], our method enhances the PSNR by 0.9 dB and 1.3 dB, respectively.

Method	Mesh	Rendering	Indoor (PSNR ↑ / SSIM ↑ / LPIPS ↓)					
			Room	Counter	Kitchen	Bonsai	Mean	
Instant-NGP [31]	X	Volume	29.2 / 0.85 / 0.30	26.43 / 0.80 / 0.34	28.5 / 0.82 / 0.25	30.3 / 0.89 / 0.23	28.6 / 0.84 / 0.28	
mip-NeRF 360 [3]	X	Volume	31.6 / 0.91 / 0.21	29.5 / 0.89 / 0.20	32.3 / 0.92 / 0.13	33.5 / 0.94 / 0.18	31.7 / 0.92 / 0.18	
3DGS [22]	X	Splat	30.6 / 0.91 / 0.26	28.7 / 0.90 / 0.29	30.3 / 0.92 / 0.22	31.9 / 0.94 / 0.29	30.4 / 0.92 / 0.19	
MobileNeRF [5]	O	Mesh	28.9 / 0.85 / 0.28	25.1 / 0.72 / 0.29	26.8 / 0.79 / 0.79	23.8 / 0.71 / 0.72	26.2 / 0.77 / 0.52	
NeRF2Mesh [41]	O	Mesh	25.7 / 0.79 / 0.35	23.9 / 0.71 / 0.35	24.0 / 0.61 / 0.36	25.0 / 0.77 / 0.29	24.7 / 0.72 / 0.34	
BakedSDF [47]	O	Mesh	28.7 / 0.87 / 0.25	25.7 / 0.81 / 0.28	26.7 / 0.82 / 0.24	27.2 / 0.85 / 0.26	27.0 / 0.84 / 0.26	
LTM [7]	O	Mesh	29.3 / 0.88 / 0.23	25.1 / 0.77 / 0.27	26.5 / 0.80 / 0.21	27.3 / 0.84 / 0.22	27.1 / 0.82 / 0.23	
SuGaR [13]	O	Splat	30.0 / 0.91 / 0.25	27.6 / 0.89 / 0.23	29.6 / 0.91 / 0.16	30.5 / 0.93 / 0.22	29.4 / 0.91 / 0.22	
GaMeS [42]	O	Splat	28.8 / 0.89 / 0.26	26.4 / 0.84 / 0.29	27.2 / 0.86 / 0.22	27.8 / 0.89 / 0.29	27.6 / 0.87 / 0.26	
MeshGS (Ours)	O	Splat	30.3 / 0.90 / 0.24	28.0 / 0.89 / 0.21	29.7 / 0.90 / 0.17	30.7 / 0.93 / 0.20	29.6 / 0.91 / 0.20	
MeshGS* (Ours)	O	Splat	30.9 / 0.93 / 0.23	27.8 / 0.90 / 0.21	29.3 / 0.90 / 0.17	30.5 / 0.93 / 0.19	29.6 / 0.91 / 0.20	

**Table 2: Quantitative Comparison on indoor mip-NeRF 360 Dataset.** Splat-based rendering exhibits superior performance when compared to mesh-based neural rendering techniques. MeshGS\* did not utilize any regularization techniques to tightly align Gaussian splats; instead, it solely applied image loss  $L_{img}$  for training. Compared to GaMeS [42] and SuGaR [13], our method enhances the PSNR by 2.0 dB and 0.2 dB, respectively.

to GaMeS [42] (with a 2.0 PSNR increase) and SuGaR [13] (with a 0.2 PSNR increase). Overall, our method shows slightly lower rendering quality compared to the original 3DGS. Without considering geometry, 3D Gaussian splats are more flexible in representing the scene based solely on image loss. This is why most mesh-based splatting methods are not superior to the original 3DGS. Our goal is to find a balance between rendering quality and geometric alignment. Thus, we emphasize the performance by solely comparing mesh-based neural rendering and mesh-based Gaussian splatting.

Furthermore, we train and evaluate on Playroom and DrJohnson scenes from the Deep Blending dataset [15]. These two scenes are collected from bounded indoor scenes and have many complex textures on Playroom scene and contain many textureless regions on DrJohnson scene. These two scenes were captured from bounded indoor environments, with the Playroom scene consisting of intricate textures and the DrJohnson scene comprising several textureless regions.

Method	Mesh	Rendering	Deep Blending [15]	
			DrJohnson	Playroom
Instant-NGP [31]	X	Volume	27.7 / 0.84 / 0.38	19.5 / 0.78 / 0.46
mip-Nerf360 [3]	X	Volume	29.1 / 0.90 / 0.24	29.6 / 0.90 / 0.25
3DGS [22]	X	Splat	29.1 / 0.90 / 0.24	30.0 / 0.90 / 0.24
MobileNeRF [5]	O	Mesh	25.9 / 0.78 / 0.31	28.7 / 0.85 / 0.28
NeRF2Mesh [41]	O	Mesh	24.0 / 0.75 / 0.51	26.6 / 0.83 / 0.41
LTM [7]	O	Mesh	26.5 / 0.82 / 0.40	29.1 / 0.87 / 0.32
SuGaR [22]	O	Splat	28.6 / 0.88 / 0.27	30.3 / 0.89 / 0.26
MeshGS (Ours)	O	Splat	<b>28.8 / 0.90 / 0.24</b>	<b>30.5 / 0.90 / 0.25</b>

**Table 3: Quantitative Comparison on Deep Blending Dataset.** We report PSNR  $\uparrow$ , SSIM  $\uparrow$ , and LPIPS  $\downarrow$  on Deep Blending datasets [15]

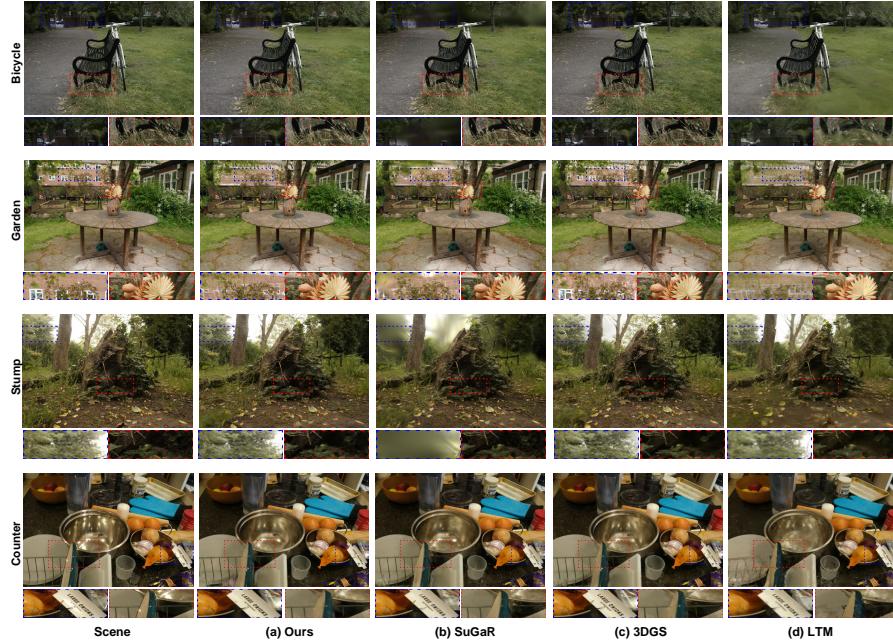
In Table 3, Our method demonstrates comparable rendering quality to both the original 3DGS [22] and SuGaR [13].

In Fig. 3, we visualize the rendering results for comparison. Our method builds on the mesh reconstructed by LTM [7]. Compared to LTM, the Gaussian splatting in our method exhibits sharper and more high-frequency details than the neural appearance field. As meshes struggle to represent highly detailed geometry such as dense foliage, tree leaves, and thin iron structures, LTM may not adequately capture these appearances for rendering (see the zoomed-in regions in Fig. 3). SuGaR is based on the original 3DGS and reconstruct the mesh solely based on Gaussian splats. However, compared to our geometry reconstruction, the mesh generated by SuGaR exhibits numerous geometric artifacts, particularly in the background region, where defects are prevalent due to weak smoothness regularization (see the background region in Fig. 2 and Fig. 4). Due to the geometric artifacts present in the background region, SuGaR often produces blurry rendering results, especially in unbounded outdoor scenes (see the background regions of Bicycle, Garden, and Stump in Fig. 3).

### 4.3 Ablation Study

In Table 4 and Table 5, MeshGS\* (ours) denotes we only use image loss  $L_{img}$  to train our mesh-based Gaussian splatting in Section 3.3. Compared to MeshGS (ours), MeshGS\* did not employ any regularization technique to tightly align Gaussian splats with the mesh surface. We observe that MeshGS\* shows slightly better rendering quality compared to our method. This is a trade-off between aligning splats geometrically and achieving high-quality rendering.

In Table 5, we show the proportion of tightly-bound Guassian splats and loosely-bound Gaussian splats. Compared to MeshGS\*, our approach employs fewer Gaussian splats overall. Furthermore, our method utilizes Gaussian splats that are 6% less tightly bound compared to MeshGS\*. It is noteworthy that the Stump scene generates a significantly higher proportion of loosely-bound Gaussian splats (42%) compared to other scenes (22%). In the Stump scene, we observe that the majority of background regions are covered by loosely-bound Gaussian splats. This is due to the inability of the mesh to accurately cover all



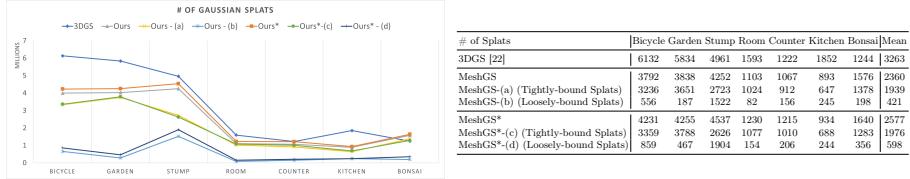
**Fig. 3: Qualitative Comparisons with Existing Methods.** We visually compare our method with SuGaR [13], 3DGS [22], and LTM [7]. For a detailed description, we visualize the zoomed-in blue and red regions.



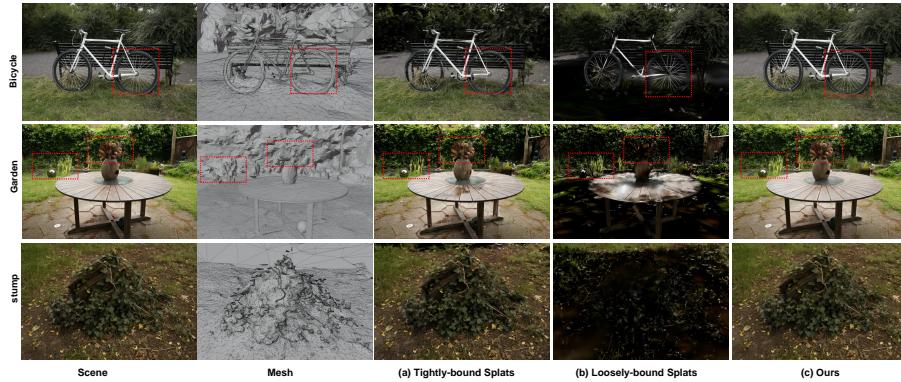
**Fig. 4: Qualitative Comparisons of Our Method and SuGaR [13].** We show shading mesh without texture.

background regions, causing Gaussian splats to often move far from the mesh surface.

Figure 6 presents a visualization of both loosely-bound and tightly-bound Gaussian splats using the outdoor mip-NeRF 360 dataset. In Fig. 6-(a), tightly-bound Gaussian splats can cover the most regions for rendering. Nevertheless, due to limitations in mesh geometry, they fail to address all mesh artifacts. In Fig. 6, the red dotted box highlights regions where tightly-bound Gaussian splats cannot accurately represent details such as the bicycle wheel, thin trunk around the vase, and dense tall grass, which closely resemble mesh artifacts. Loosely-bound Gaussian splats can serve as a supplementary method to cover these artifacts. In the Stump scene, loosely-bound Gaussian splats demonstrate their ability to accurately depict the detailed thin grass and leaves covering the trunk, thereby facilitating high-fidelity rendering. Moreover, given that the mesh geometry shows more geometric defects in the background region, it necessitates the use of numerous loosely-bound Gaussian splats to achieve high-quality ren-



**Fig. 5: The number of Gaussian Splats** The unit for the number of Gaussian splats is  $10^3$ . Compared to the original 3DGS, we utilize 30% fewer Gaussian splats. MeshGS\* did not utilize any regularization techniques to tightly align Gaussian splats; instead, it solely applied image loss  $L_{img}$  for training. MeshGS-(a) and MeshGS\*-(c) shows the number of tightly-bound Gaussian splats. MeshGS-(b) and MeshGS\*-(d) denotes the number of loosely-bound Gaussian splats.

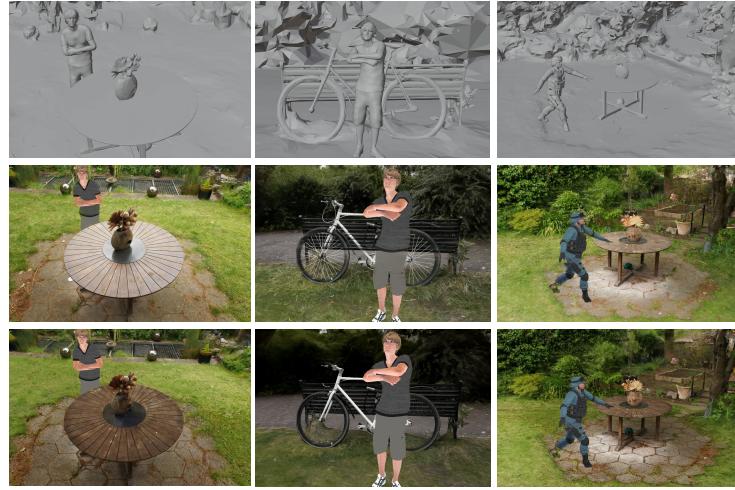


**Fig. 6: Visualization results of both loosely-bound and tightly-bound Gaussian splats.** The second column shows wireframe mesh extracted without texture. The third column visualizes only the tightly-bound Gaussian splats, while the fourth column illustrates the loosely-bound Gaussian splats. The fifth column presents the final rendering results achieved by our method.

dering. Furthermore, in the mesh geometry, the process of mesh decimation significantly reduces the number of triangles in the background region. As a result, the initialization of Gaussian splats becomes more sparse, posing challenges in rendering images.

#### 4.4 Compositional scenes

In Fig. 7, we visualize the practical application of our method for scene composition. We incorporate an artist-designed human model from Mixamo [1] into the scene reconstructed by our method. This human asset consists of high-polygon meshes with photorealistic materials and textures. We show the composition of a synthetic human into a different scene. The background region rendered by our method shows photorealistic image quality of compositional scenes. Thanks to the mesh, we can composite the object and scene. Furthermore, the composited scene can be rendered using Gaussian splatting without causing severe artifacts around the human's boundary.



**Fig. 7: Scene Composition:** Rendering obtained by embedding the synthetic human to a reconstructed scene. The first row visualizes the shading mesh without texture. The second row shows the composition of a human in the scene reconstructed by our method. The third row visualizes the composition of the same human in the scene generated by LTM [7].

## 5 Conclusion, Limitations, and Future Work

This paper presents a practical approach for integrating mesh representation with 3D Gaussian splatting rendering. Our method attains a high level of photorealistic rendering quality by employing Gaussian splats bound to the mesh surface and executing distance-based Gaussian splatting to ensure alignment with the mesh surface. Additionally, we differentiate between two types of Gaussian splats and present training strategies tailored to their respective roles. We present that a combination of mesh-based representation and Gaussian splatting is crucial for enhancing rendering quality and ensuring compatibility with game engines in practical applications. However, our method does have limitations. Compared to textured mesh, our approach necessitates large disk storage and a high-end GPU for real-time rendering. Moreover, we require a more sophisticated method for assessing the quality of Gaussian splats and ensuring their tight binding to the surface mesh. Additionally, our method is constrained by the underlying mesh shape, particularly when dealing with excessively large faces that cannot appropriately initialize Gaussian splats. In the future, our goal is to develop surface extraction techniques from loosely-bound splats to address mesh defects and evaluate geometric quality. Furthermore, we should explore adaptive methods for distributing Gaussian splats to individual triangles, akin to mesh subdivision or remeshing techniques.

**Acknowledgements** This work was supported in part by ARO Grant W911N F2310352 and Army Cooperative Agreement W911NF2120076.

## A Implementation Details

We utilize the geometry reconstruction and mesh processing part from the LTM [7]. Leveraging the strong smoothness regularization [12], this method shows consistently smooth surfaces on both ground and wall structures. Additionally, due to the incorporation of contraction function [47], background regions are effectively reconstructed. However, we do not utilize the optimization step, resulting in a mesh surface that lacks detailed structures and often appears smoothed due to the limited number of vertex. Our method is implemented using PyTorch [33] and builds on Gaussian rasterization provided by the original 3DGS [22]. For scene composition, we initially obtain the textured human mesh from Mixamo and export the compositional mesh result from the Blender engine. Subsequently, we employ our rasterization code to render the compositional scene using the exported mesh composition.

## B Comparison to 2D Gaussian Splatting

Recently, a 2DGS [17] was released, showing remarkable performance in mesh reconstruction and novel view synthesis. In Table 4 and Table 5, we compare our method with 2D GS in terms of rendering performance. Unlike approaches that bind 3D splats to an underlying mesh, this research prioritizes aligning 2D Gaussian splats with the underlying geometry without requiring prior 3D geometry. This paper demonstrate that 2D Gaussian splats surpass 3D Gaussian splats in representing geometry. Our Gaussian splats binding method can leverage the 2D Gaussian splats representation to bind splats to a given mesh.

Method	Mesh	Rendering	Outdoor (PSNR ↑ / SSIM ↑ / LPIPS ↓)						Mean	
			Bicycle	Garden	Stump					
2DGS [17]	O	Splat	24.6 / 0.71 / 0.30	26.6 / 0.83 / 0.16	26.0 / 0.76 / 0.29	25.7 / 0.76 / 0.25				
MeshGS (Ours)	O	Splat	24.4 / 0.71 / 0.26	26.8 / 0.85 / 0.12	25.8 / 0.74 / 0.25	25.7 / 0.77 / 0.21				
MeshGS* (Ours)	O	Splat	24.9 / 0.75 / 0.25	26.8 / 0.86 / 0.12	26.3 / 0.76 / 0.22	26.0 / 0.79 / 0.20				

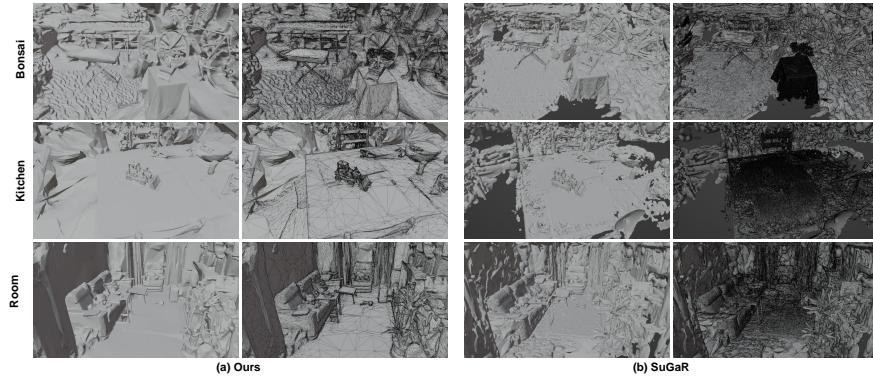
**Table 4: Quantitative Comparison on outdoor mip-NeRF 360 Dataset.**  
MeshGS\* did not utilize any regularization techniques to tightly align Gaussian splats; instead, it solely applied image loss for training.

Method	Mesh	Rendering	Indoor (PSNR ↑ / SSIM ↑ / LPIPS ↓)						Mean
			Room	Counter	Kitchen	Bonsai			
2DGS [17]	O	Splat	30.8 / 0.91 / 0.22	28.1 / 0.89 / 0.22	30.1 / 0.92 / 0.14	31.3 / 0.93 / 0.25	30.1 / 0.91 / 0.19		
MeshGS (Ours)	O	Splat	30.3 / 0.90 / 0.24	28.0 / 0.89 / 0.21	29.7 / 0.90 / 0.17	30.7 / 0.93 / 0.20	29.6 / 0.91 / 0.20		
MeshGS* (Ours)	O	Splat	30.9 / 0.93 / 0.23	27.8 / 0.90 / 0.21	29.3 / 0.90 / 0.17	30.5 / 0.93 / 0.19	29.6 / 0.91 / 0.20		

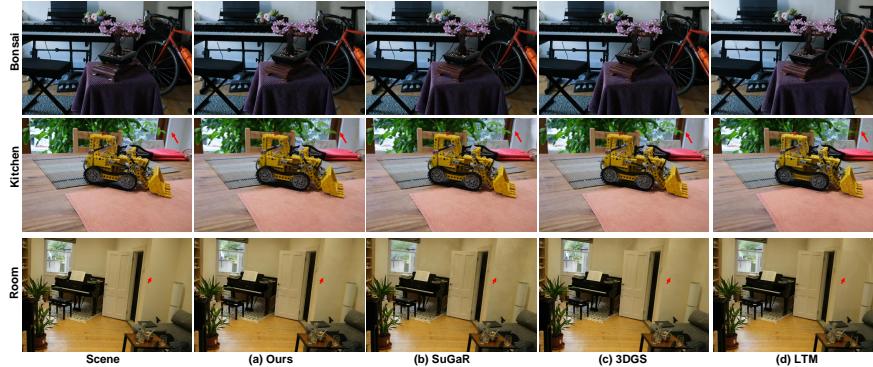
**Table 5: Quantitative Comparison on indoor mip-NeRF 360 Dataset.**  
MeshGS\* did not utilize any regularization techniques to tightly align Gaussian splats; instead, it solely applied image loss for training.

## C Additional Qualitative Comparisons

Supplementary material includes all rendering results on the test set, provided as a downloadable zip file. To accommodate file size limitations (100mb for supplementary material), we downscale the resolution of rendering images. In the manuscript, we show the mesh reconstruction results for Garden, Stump, and Bicycle scenes (refer to Fig. 2, Fig. 4, and Fig. 6) due to the limited space. we visualize the results for the remaining scenes on both Mip-NeRF 360 dataset and Deep Blending dataset. We present both the mesh reconstruction and rendering outcomes for these scenes, depicted in Fig. 8 and Fig. 9, respectively. Additionally, we show visualizations for the mesh reconstruction and rendering results of the Playroom and Dr. Johnson scenes from the Deep Blending dataset, showcased in Fig. 10 and Fig. 11, respectively.



**Fig. 8:** Qualitative comparison between (a) Ours and (b) SuGaR [13]. Our method and SuGaR show shading mesh and wireframe mesh extracted without texture, respectively. In SuGaR, the dark grey regions denote empty areas where the mesh fails to represent the scene. Compared to ours, SuGaR’s mesh inadequately captures background regions



**Fig. 9: Qualitative Comparisons with Existing Methods.** We visually compare our method with SuGaR [13], 3DGS [22], and LTM [7]. Red arrows emphasize subtle differences in rendering quality



**Fig. 10:** Qualitative comparison between (a) Ours and (b) SuGaR [13]. Our method and SuGaR show shading mesh and wireframe mesh extracted without texture, respectively.



**Fig. 11: Qualitative Comparisons with Existing Methods.** We visually compare our method with SuGaR [13], 3DGS [22], and LTM [7]. Red arrows emphasize subtle differences in rendering quality.

## References

1. Mixamo, [www.mixamo.com](http://www.mixamo.com)
2. 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)
3. 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. Chen, A., Xu, Z., Geiger, A., Yu, J., Su, H.: Tensorf: Tensorial radiance fields. In: European Conference on Computer Vision. pp. 333–350. Springer (2022)
5. Chen, Z., Funkhouser, T., Hedman, P., Tagliasacchi, A.: Mobilenerf: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 16569–16578 (2023)
6. Choi, J., Jung, D., Lee, T., Kim, S., Jung, Y., Manocha, D., Lee, D.: Tmo: Textured mesh acquisition of objects with a mobile device by using differentiable rendering. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 16674–16684 (2023)
7. Choi, J., Shah, R., Li, Q., Wang, Y., Saraf, A., Kim, C., Huang, J.B., Manocha, D., Alsisan, S., Kopf, J.: Ltm: Lightweight textured mesh extraction and refinement of

- large unbounded scenes for efficient storage and real-time rendering. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 5053–5063 (2024)
8. Deering, M., Winner, S., Schediwy, B., Duffy, C., Hunt, N.: The triangle processor and normal vector shader: a vlsi system for high performance graphics. *Acm siggraph computer graphics* **22**(4), 21–30 (1988)
  9. Fridovich-Keil, S., Yu, A., Tancik, M., Chen, Q., Recht, B., Kanazawa, A.: Plenoxtels: Radiance fields without neural networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 5501–5510 (2022)
  10. Garland, M.: Quadric-based polygonal surface simplification. Carnegie Mellon University (1999)
  11. Garland, M., Heckbert, P.S.: Surface simplification using quadric error metrics. In: Proceedings of the 24th annual conference on Computer graphics and interactive techniques. pp. 209–216 (1997)
  12. Gropp, A., Yariv, L., Haim, N., Atzmon, M., Lipman, Y.: Implicit geometric regularization for learning shapes. In: Proceedings of the 37th International Conference on Machine Learning. ICML’20, JMLR.org (2020)
  13. Guédon, A., Lepetit, V.: Sugar: Surface-aligned gaussian splatting for efficient 3d mesh reconstruction and high-quality mesh rendering. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 5354–5363 (2024)
  14. Guo, Y.C., Cao, Y.P., Wang, C., He, Y., Shan, Y., Zhang, S.H.: Vmesh: Hybrid volume-mesh representation for efficient view synthesis. In: SIGGRAPH Asia 2023 Conference Papers. pp. 1–11 (2023)
  15. Hedman, P., Philip, J., Price, T., Frahm, J.M., Drettakis, G., Brostow, G.: Deep blending for free-viewpoint image-based rendering. *ACM Transactions on Graphics (ToG)* **37**(6), 1–15 (2018)
  16. Hedman, P., Srinivasan, P.P., Mildenhall, B., Barron, J.T., Debevec, P.: Baking neural radiance fields for real-time view synthesis. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 5875–5884 (2021)
  17. Huang, B., Yu, Z., Chen, A., Geiger, A., Gao, S.: 2d gaussian splatting for geometrically accurate radiance fields. *arXiv preprint arXiv:2403.17888* (2024)
  18. Johnson, J., Ravi, N., Reizenstein, J., Novotny, D., Tulsiani, S., Lassner, C., Branson, S.: Accelerating 3d deep learning with pytorch3d. In: SIGGRAPH Asia 2020 Courses, pp. 1–1 (2020)
  19. Karis, B., Games, E.: Real shading in unreal engine 4. *Proc. Physically Based Shading Theory Practice* **4**(3), 1 (2013)
  20. Kazhdan, M., Bolitho, M., Hoppe, H.: Poisson surface reconstruction. In: Proceedings of the fourth Eurographics symposium on Geometry processing. vol. 7 (2006)
  21. Kazhdan, M., Hoppe, H.: Screened poisson surface reconstruction. *ACM Transactions on Graphics (ToG)* **32**(3), 1–13 (2013)
  22. Kerbl, B., Kopanas, G., Leimkühler, T., Drettakis, G.: 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics* **42**(4) (2023)
  23. Kopanas, G., Philip, J., Leimkühler, T., Drettakis, G.: Point-based neural rendering with per-view optimization. In: Computer Graphics Forum. vol. 40, pp. 29–43. Wiley Online Library (2021)
  24. Laine, S., Hellsten, J., Karras, T., Seol, Y., Lehtinen, J., Aila, T.: Modular primitives for high-performance differentiable rendering. *ACM Transactions on Graphics (TOG)* **39**(6), 1–14 (2020)

25. Lassner, C., Zollhofer, M.: Pulsar: Efficient sphere-based neural rendering. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 1440–1449 (2021)
26. Li, Z., Müller, T., Evans, A., Taylor, R.H., Unberath, M., Liu, M.Y., Lin, C.H.: Neuralangelo: High-fidelity neural surface reconstruction. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 8456–8465 (2023)
27. Liu, J.Y., Chen, Y., Yang, Z., Wang, J., Manivasagam, S., Urtasun, R.: Real-time neural rasterization for large scenes. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 8416–8427 (2023)
28. Lorensen, W.E., Cline, H.E.: Marching cubes: A high resolution 3d surface construction algorithm. ACM siggraph computer graphics **21**(4), 163–169 (1987)
29. Lu, F., Xu, Y., Chen, G., Li, H., Lin, K.Y., Jiang, C.: Urban radiance field representation with deformable neural mesh primitives. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 465–476 (2023)
30. 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)
31. Müller, T., Evans, A., Schied, C., Keller, A.: Instant neural graphics primitives with a multiresolution hash encoding. arXiv preprint arXiv:2201.05989 (2022)
32. Oechsle, M., Peng, S., Geiger, A.: Unisurf: Unifying neural implicit surfaces and radiance fields for multi-view reconstruction. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 5589–5599 (2021)
33. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems **32** (2019)
34. Qian, S., Kirschstein, T., Schoneveld, L., Davoli, D., Giebenhain, S., Nießner, M.: Gaussianavatars: Photorealistic head avatars with rigged 3d gaussians. arXiv preprint arXiv:2312.02069 (2023)
35. Rakotosaona, M.J., Manhardt, F., Arroyo, D.M., Niemeyer, M., Kundu, A., Tombari, F.: Nerfmeshing: Distilling neural radiance fields into geometrically-accurate 3d meshes. In: Proc. of the International Conf. on 3D Vision (3DV) (2023)
36. Reiser, C., Szeliski, R., Verbin, D., Srinivasan, P., Mildenhall, B., Geiger, A., Barron, J., Hedman, P.: Merf: Memory-efficient radiance fields for real-time view synthesis in unbounded scenes. ACM Transactions on Graphics (TOG) **42**(4), 1–12 (2023)
37. Rückert, D., Franke, L., Stamminger, M.: Adop: Approximate differentiable one-pixel point rendering. ACM Transactions on Graphics (ToG) **41**(4), 1–14 (2022)
38. Schonberger, J.L., Frahm, J.M.: Structure-from-motion revisited. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4104–4113 (2016)
39. Segal, M., Akeley, K.: The opengl graphics system: A specification (version 4.6 (core profile)-october 22, 2019). The Khronos Group Inc.[cit. 2020-12-05]. Dostupné z: <https://www.khronos.org/registry/OpenGL/specs/gl/glspec46.core.pdf> (2021)
40. Sun, C., Sun, M., Chen, H.T.: Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 5459–5469 (2022)
41. Tang, J., Zhou, H., Chen, X., Hu, T., Ding, E., Wang, J., Zeng, G.: Delicate textured mesh recovery from nerf via adaptive surface refinement. arXiv preprint arXiv:2303.02091 (2022)

42. Waczyńska, J., Borycki, P., Tadeja, S., Tabor, J., Spurek, P.: Games: Mesh-based adapting and modification of gaussian splatting. arXiv preprint arXiv:2402.01459 (2024)
43. Wang, P., Liu, L., Liu, Y., Theobalt, C., Komura, T., Wang, W.: Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. In: Beygelzimer, A., Dauphin, Y., Liang, P., Vaughan, J.W. (eds.) Advances in Neural Information Processing Systems (2021), [https://openreview.net/forum?id=D7bPRxNt\\_AP](https://openreview.net/forum?id=D7bPRxNt_AP)
44. Wang, Z., Bovik, A.C., Sheikh, H.R., Simoncelli, E.P.: Image quality assessment: from error visibility to structural similarity. IEEE transactions on image processing **13**(4), 600–612 (2004)
45. Wang, Z., Shen, T., Nimier-David, M., Sharp, N., Gao, J., Keller, A., Fidler, S., Müller, T., Gojcic, Z.: Adaptive shells for efficient neural radiance field rendering. arXiv preprint arXiv:2311.10091 (2023)
46. Yariv, L., Gu, J., Kasten, Y., Lipman, Y.: Volume rendering of neural implicit surfaces. Advances in Neural Information Processing Systems **34**, 4805–4815 (2021)
47. Yariv, L., Hedman, P., Reiser, C., Verbin, D., Srinivasan, P.P., Szeliski, R., Barron, J.T., Mildenhall, B.: Bakedsdf: Meshing neural sdf’s for real-time view synthesis. arXiv (2023)
48. Yifan, W., Serena, F., Wu, S., Öztireli, C., Sorkine-Hornung, O.: Differentiable surface splatting for point-based geometry processing. ACM Transactions on Graphics (TOG) **38**(6), 1–14 (2019)
49. Yu, Z., Peng, S., Niemeyer, M., Sattler, T., Geiger, A.: Monosdf: Exploring monocular geometric cues for neural implicit surface reconstruction. Advances in neural information processing systems **35**, 25018–25032 (2022)
50. Zhang, R., Isola, P., Efros, A.A., Shechtman, E., Wang, O.: The unreasonable effectiveness of deep features as a perceptual metric. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 586–595 (2018)
51. Zwicker, M., Pfister, H., Van Baar, J., Gross, M.: Ewa splatting. IEEE Transactions on Visualization and Computer Graphics **8**(3), 223–238 (2002)