

# Triangle Splatting+: Differentiable Rendering with Opaque Triangles

Jan Held<sup>1,2</sup> Renaud Vandeghen<sup>1</sup> Sanghyun Son<sup>3</sup>  
 Daniel Rebain<sup>4</sup> Matheus Gadelha<sup>6</sup> Yi Zhou<sup>6</sup>  
 Ming C. Lin<sup>3</sup> Marc Van Droogenbroeck<sup>1</sup> Andrea Tagliasacchi<sup>2,5</sup>

<sup>1</sup>University of Liège <sup>2</sup>Simon Fraser University <sup>3</sup>University of Maryland  
<sup>4</sup>University of British Columbia <sup>5</sup>University of Toronto <sup>6</sup>Adobe Research

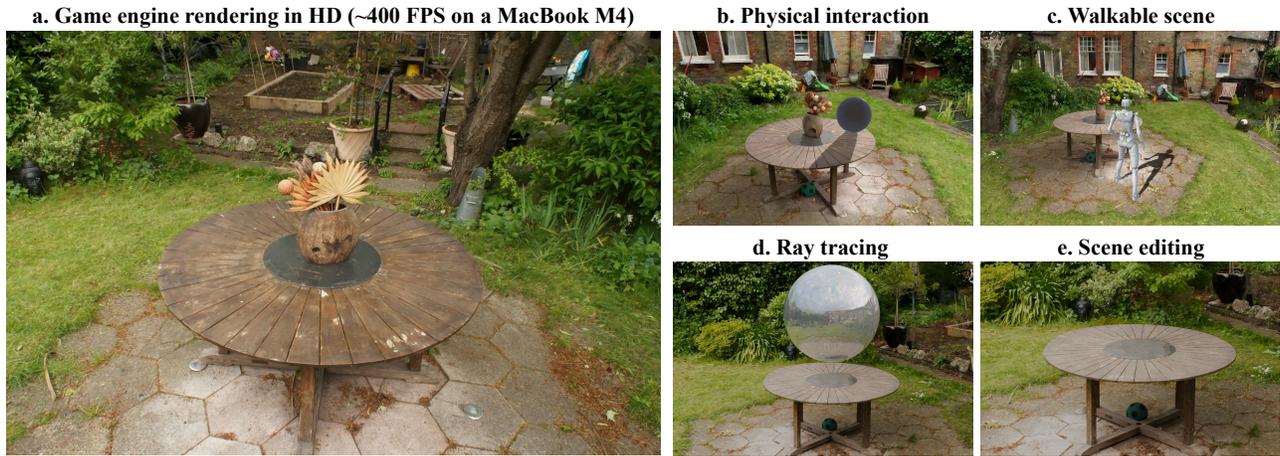


Figure 1. **Triangle Splatting+** optimizes a triangle-based representation end-to-end and achieves high visual quality using only *opaque triangles*. The resulting representation can be directly imported into any game engine without post-processing and runs at 400 FPS on a consumer laptop. Within the engine, it naturally supports (b) physical interactions (e.g., collisions), (c) walkable scene interactions as in video games, (d) ray tracing, and (e) scene editing (e.g., object removal or addition).

## Abstract

Reconstructing 3D scenes and synthesizing novel views has seen rapid progress in recent years. Neural Radiance Fields demonstrated that continuous volumetric radiance fields can achieve high-quality image synthesis, but their long training and rendering times limit practicality. 3D Gaussian Splatting (3DGS) addressed these issues by representing scenes with millions of Gaussians, enabling real-time rendering and fast optimization. However, Gaussian primitives are not natively compatible with the mesh-based pipelines used in VR headsets, and real-time graphics applications. Existing solutions attempt to convert Gaussians into meshes through post-processing or two-stage pipelines, which increases complexity and degrades visual quality. In this work, we introduce **Triangle Splatting+**, which directly

optimizes triangles, the fundamental primitive of computer graphics, within a differentiable splatting framework. We formulate triangle parametrization to enable connectivity through shared vertices, and we design a training strategy that enforces opaque triangles. The final output is immediately usable in standard graphics engines without post-processing. Experiments on the Mip-NeRF360 and Tanks & Temples datasets show that Triangle Splatting+ achieves state-of-the-art performance in mesh-based novel view synthesis. Our method surpasses prior splatting approaches in visual fidelity while remaining efficient and fast to training. Moreover, the resulting semi-connected meshes support downstream applications such as physics-based simulation or interactive walkthroughs. The project page is <https://trianglesplatting2.github.io/trianglesplatting2/>.

## 1. Introduction

In recent years, significant progress has been made in reconstructing complex scenes and generating novel views. First, Neural Radiance Fields [24] revolutionized this area by representing scenes as continuous volumetric radiance fields, which are optimized to render novel views at high-quality. Despite their impact, NeRF-based methods are hindered by long training times and slow inference, which restrict their practical applicability. To overcome these limitations, 3D Gaussian Splatting [19] was introduced, representing scenes with millions of 3D Gaussian primitives. This representation enables orders-of-magnitude faster training and supports real-time rendering, all while preserving high visual fidelity.

However, Gaussian primitives are not natively compatible with the graphics pipelines that power interactive applications such as VR headsets or real-time mesh-based renderers. Integrating them requires either modifying existing engines to directly support Gaussians, or devising conversion techniques that transform Gaussian radiance fields into mesh representations. Several recent works have pursued this direction: 2DGS [15] and RaDe-GS [36] reconstruct meshes by first optimizing a scene and then extracting surfaces via truncated signed distance fields (TSDF), or MiLo [10], which learns a surface mesh jointly during training. Although effective to some extent, all of these approaches depend on additional post-processing, which complicates the pipeline and often leads to degraded quality.

Held *et al.* proposed an initial step toward bridging classical rendering pipelines with differentiable radiance field frameworks by replacing the Gaussian primitive by *triangles*. Triangle Splatting [12] demonstrated that unstructured sets of triangles can be optimized end-to-end within a differentiable splatting framework, combining the adaptability of Gaussian splats with the efficiency and compatibility of triangles. Yet, Triangle Splatting represents only an initial step toward bridging classical rendering pipelines with differentiable radiance field frameworks. When its triangle soup is rendered in a game engine, a noticeable drop in visual quality occurs because training does not enforce opaque triangles but instead relies on soft, semi-transparent ones. Moreover, all triangles remain isolated: due to the chosen parametrization, no connectivity can be established between neighboring triangles, even when their vertices coincide spatially or share similar color distributions.

In this work, we introduce **Triangle Splatting+**, which unifies radiance field optimization with traditional computer graphics. We reformulate the parametrization of triangles to enable connectivity: starting from a shared set of vertices, triangles can naturally connect through common vertices rather than remaining isolated. In addition, we design a training strategy that enforces opaque triangles, ensuring that the final representation integrates seamlessly into mesh-

based rendering pipelines. Although the resulting mesh is only semi-connected, it is sufficient for a wide range of downstream applications, including physics-based simulation, or interactive walkthroughs. Triangle Splatting+ thus paves the way for immediate integration into VR/AR environments and modern video games.

**Contributions.** (i) We introduce **Triangle Splatting+**, which unifies radiance field optimization with traditional computer graphics by obtaining high-visual quality in mesh-based renderers. (ii) We re-parametrize triangles to enforce connectivity between primitives (iii) We propose a tailored training strategy that balances visual fidelity and geometric accuracy with solid and opaque triangles. (iv) Triangle Splatting+ can run on any game-engine and enables physical simulations, walkable environments for video games and many more.

## 2. Related work

**Primitive-based differentiable rendering.** Differentiable rendering enables end-to-end optimization by propagating image-based losses back to scene parameters, allowing for the learning of explicit representations such as point clouds [8, 18], voxel grids [6], polygonal meshes [18, 22, 23], and more recently, Gaussian primitives [19]. The advent of 3D Gaussian Splatting [19] showed that it is possible to fit millions of anisotropic Gaussians in just minutes, enabling real-time rendering with high fidelity. Since then, various directions have been explored to improve the Gaussian primitive, including the use of 2D Gaussians [15], generalized Gaussians [11], alternative kernels [16], and learnable basis functions [4]. Other works moved beyond Gaussians entirely, investigating different primitives such as smooth 3D convexes [13], linear primitives [32], sparse voxel fields [31], or radiance foams [7]. More recently, Held *et al.* advocated for the comeback of triangles, the most fundamental primitive in computer graphics. Many researchers have explored this direction, proposing triangle-based representations for efficient scene modeling [2, 17]. However, existing triangle-based methods usually result in an unstructured triangle soup, with no connectivity between adjacent triangles, and they fail to produce solid, opaque triangles at the end of training. Another line of work using differentiable mesh for 3D reconstruction [29, 30] produce connected, solid mesh, but they mainly handle synthetic object and do not extend to real-world scenes. In this work, we propose a method that enables triangle connectivity, producing a semi-structured mesh by the end of training. Moreover, our representation consists only of opaque triangles, making the output immediately compatible with game engines.

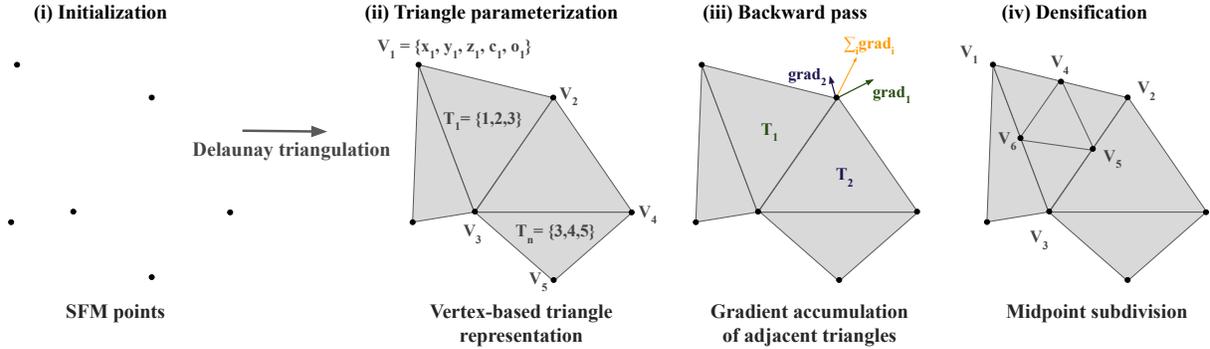


Figure 2. **Method overview of Triangle Splatting+**. (i) We start from sparse SfM points and apply 3D Delaunay triangulation to obtain an initial mesh. (ii) Triangles are parameterized through a shared vertex set, where each vertex stores position  $x_i, y_i, z_i$ , color  $c_i$ , and opacity  $o_i$ . Each triangle is defined by the three indices in the vertex set that compose it. (iii) During training, gradients from all adjacent triangles accumulate at shared vertices. (iv) Densification is performed by midpoint subdivision, introducing new vertices and triangles while preserving connectivity.

**Mesh reconstruction from images.** Implicit and explicit methods have made significant progress in reconstructing 3D scenes, but they remain largely incompatible with traditional mesh-based renderers and game engines. Some methods propose strategies to convert implicit radiance fields into meshes. BakedSDF [33] learns a neural SDF and appearance then bakes them into a textured triangle mesh, Binary Opacity Fields [27] drives densities toward near binary opacities so surfaces can be extracted as a mesh and rendered efficiently, and MobileNeRF [5] distills a NeRF into a compact set of textured polygons for real-time mobile rendering. However, these methods introduce overhead and increase the overall training time. More recently, several methods have built on Gaussian Splatting and proposed ways to extract a mesh from the optimized Gaussian scene. 2DGS [15] and RaDe-GS [36] rely on Truncated Signed Distance Fields (TSDF) for mesh extraction. Other approaches extract meshes by sampling a surface-aligned Gaussian level set followed by Poisson reconstruction [9], or by defining a Gaussian opacity level set and applying Marching Tetrahedra on Gaussian-induced tetrahedral grids [35]. All of these methods, however, treat mesh extraction as a separate post-processing step. More recently, MiLo [10] integrates surface mesh extraction directly into the optimization, jointly refining both the mesh and the Gaussian representation. However, while MiLo optimizes the mesh geometry during training, color still has to be learned separately, for instance through a neural color field applied in a post-processing step. In contrast, Triangle Splatting+ directly optimizes opaque triangles together with their vertex colors, making the result immediately compatible with any mesh-based renderer without requiring additional post-processing.

### 3. Methodology

Kerbl *et al.* [19] introduced 3D Gaussian Splatting, representing scenes as large sets of Gaussian primitives that are directly optimized from multi-view observations. Building on this idea, subsequent works have explored alternative primitives: 3D Convex Splatting represents scenes using convex shapes [13], 2DGS employs flat 2D Gaussians [15] and most recently, Held *et al.* proposed representing a scene using *triangles*. While Triangle Splatting demonstrated that unstructured sets of triangles can be optimized end-to-end, it left an open question of whether triangles can also serve as a practical and high-quality representation for mesh-based rendering.

In this work, we show that triangles can indeed be optimized end-to-end as opaque primitives, yielding a representation that integrates seamlessly into any game engine. Specifically, Sec. 3.1 and Sec. 3.2 introduce the new parametrization of Triangle Splatting+ to enable connectivity, Sec. 3.3 details how triangles are projected into image space, Sec. 3.4 outlines the training strategy for achieving high mesh-based visual quality, and finally, Sec. 3.5 provides additional information on initialization and training losses. An overview of these steps is illustrated in Fig. 2.

#### 3.1. Vertex-based representation

In previous Triangle Splatting methods [2, 12], each triangle is parameterized by three vertices  $\mathbf{v}_i \in \mathbb{R}^3$ , a color  $\mathbf{c}$ , a smoothness parameter  $\sigma$ , and an opacity  $o$ . Both  $\sigma$  and  $o$  are treated as free parameters, which prevents the triangles from being strictly opaque at the end of training. As a result, rendering the resulting triangle soup in a game engine leads to noticeable quality degradation. Moreover, the formula-

tion enforces triangles to remain isolated and unconnected. In regions of high density, many vertices lie close to each other and share similar color distributions, but cannot be merged or shared. Ideally, such cases should be represented by a single vertex connecting adjacent triangles, improving efficiency and structural consistency.

In Triangle Splatting+, we therefore define a vertex set

$$\mathcal{V} = \{\mathbf{v}_i \in \mathbb{R}^3 \mid i = 1, \dots, N\},$$

with  $N$  the total number of vertices. Each vertex is parameterized as  $\mathbf{v}_i = (x_i, y_i, z_i, \mathbf{c}_i, o_i)$ , where  $(x_i, y_i, z_i) \in \mathbb{R}^3$  denotes its 3D position,  $\mathbf{c}_i \in \mathbb{R}^3$  the vertex color and  $o_i \in [0, 1]$  the vertex opacity, similar to [29]. After training, the opacity parameter can be discarded as each triangle will be opaque.

### 3.2. Triangle representation

A triangle is then defined by a triplet of vertex indices  $\mathbf{T}_m = (i, j, k)$ ,  $i, j, k \in \{1, \dots, N\}$ , which specifies the three vertices from  $\mathcal{V}$  that build the triangle. The opacity of the triangle is computed as  $o_{\mathbf{T}_m} = \min(o_i, o_j, o_k)$ , and its color at a point inside the triangle is obtained by interpolating the vertex colors using barycentric coordinates  $\mathbf{c}_{\mathbf{T}_m}(\lambda_i, \lambda_j, \lambda_k) = \lambda_i \mathbf{c}_i + \lambda_j \mathbf{c}_j + \lambda_k \mathbf{c}_k$ , with  $\lambda_i + \lambda_j + \lambda_k = 1$  and  $\lambda_i, \lambda_j, \lambda_k \geq 0$ . Triangles can now be easily combined by connecting them through shared vertices. During training, the gradient of each triangle is propagated back to its corresponding vertices. Each vertex’s position, color, and opacity therefore receive the accumulated gradients from all connected triangles.

### 3.3. Differentiable rasterization

The rasterization process begins by projecting each 3D vertex  $\mathbf{v}_i$  onto the image plane using a standard pinhole camera model. The projection is defined by the intrinsic camera matrix  $\mathbf{K}$  and the extrinsic parameters, i.e., rotation  $\mathbf{R}$  and translation  $\mathbf{t}$ :  $\mathbf{q}_i = \mathbf{K}(\mathbf{R}\mathbf{v}_i + \mathbf{t})$ , where  $\mathbf{q}_i \in \mathbb{R}^2$  denotes the 2D coordinates of the projected vertex in image space. To determine the influence of a triangle on a pixel  $p$ , we use the same window function as used in Triangle Splatting [12]. The *signed distance field* (SDF)  $\phi$  of the 2D triangle in image space is given by:

$$\phi(\mathbf{p}) = \max_{i \in \{1, 2, 3\}} L_i(\mathbf{p}), \quad L_i(\mathbf{p}) = \mathbf{n}_i \cdot \mathbf{p} + d_i,$$

where  $\mathbf{n}_i$  are the unit normals of the triangle edges pointing outside the triangle, and  $d_i$  are offsets such that the triangle is given by the zero-level set of the function  $\phi$ . The signed distance field  $\phi$  thus takes positive values outside the triangle, negative values inside, and equals zero on its boundary.

The window function  $I$  is then defined as:

$$I(\mathbf{p}) = \text{ReLU} \left( \frac{\phi(\mathbf{p})}{\phi(\mathbf{s})} \right)^\sigma$$

$$\text{such that } I(\mathbf{p}) \begin{cases} = 1 & \text{at the triangle incenter,} \\ = 0 & \text{at the triangle boundary,} \\ = 0 & \text{outside the triangle.} \end{cases} \quad (1)$$

with  $\mathbf{s} \in \mathbb{R}^2$  be the *incenter* of the projected triangle (i.e., the point inside the triangle with minimum signed distance).  $\sigma$  is a smoothness parameter that controls the transition between the interior and exterior of the triangle. As  $\sigma \rightarrow 0$ , the representation converges to a sharp, exact triangle, while larger values of  $\sigma$  yield a smooth window function that gradually increases from zero at the boundary to one at the center. Once the triangles are projected, the color of each image pixel  $\mathbf{p}$  is computed by accumulating contributions from all overlapping triangles, in depth order given:

$$C(\mathbf{p}) = \sum_{n=1}^N c_{T_n} o_{T_n} I(\mathbf{p}) \left( \prod_{i=1}^{n-1} (1 - o_{T_i} I(\mathbf{p})) \right), \quad (2)$$

At the end of training, Eq. (2) simplifies to  $C(\mathbf{p}) = c_{T_n} I(\mathbf{p})$ , so that only a single evaluation per pixel is required, significantly accelerating the rendering process.

### 3.4. Training strategy

The training strategy is the most important part for obtaining a high-visual quality with opaque primitives. The first challenge is that during optimization, it is essential that gradients can be back-propagated. With only solid and opaque triangles (low  $\sigma$  and high  $o$ ), gradients would vanish, making optimization impossible. To ensure gradient flow, we allow triangles during training to be smooth (soft transition from inside the triangle to outside) and semi-transparent. The transition from semi-transparent and soft triangles to solid and opaque ones is crucial for achieving high visual quality in the final reconstruction. Secondly, pruning triangles becomes increasingly important. Since opacity is manually increased during training, triangles can no longer remain hidden by staying transparent. By the end of training, all triangles become opaque, even if they degrade visual quality. It is therefore essential to filter out such triangles during optimization to avoid artifacts in the final result. In the following, we first describe how  $\sigma$  and opacity are optimized during training to ensure gradient flow while still yielding fully opaque triangles at convergence. We then introduce our pruning and densification strategy designed to achieve high visual quality.

**Triangles smoothness.** Unlike Triangle Splatting, where  $\sigma$  is freely optimized and defined per triangle, we initialize it to 1.0 (corresponding to a linear transition from the incenter

to the boundary) and treat it as a single parameter shared across all triangles. During training, we gradually anneal  $\sigma$  to 0.0001, providing strong gradient flow in the early stages and converging to sharp, well-defined triangles at the end.

**Triangles opacities.** In traditional splatting methods, opacity is mapped to the  $[0, 1]$  domain using a Sigmoid activation. In our approach, however, we enforce a gradually increasing mapping domain by redefining the activation as  $\text{opacity}(x) = O_t + (1 - O_t)\sigma(x)$ , where  $O_t$  denotes the opacity floor. After the first 5k training iterations, we update the mapping to constrain opacities to  $[O_t, 1]$ , and anneal  $O_t$  over the course of training until all triangles become opaque. Unlike the smoothness parameter  $\sigma$ , opacity values remain free variables throughout optimization but is constrained to lie within  $[O_t, 1]$ , instead of fixing them to a constant. After training,  $\sigma$  and the opacity can be discarded.

**Pruning.** In traditional splatting methods, pruning primarily serves to reduce memory consumption and improve rendering speed. In the worst case, a primitive simply converges to a low opacity and becomes invisible in the final 3D representation. In our setting, however, pruning plays a far more critical role. During training, our representation gradually converges to opaque triangles. Unlike in traditional splatting, our triangles cannot hide behind low opacities, since this would contradict our training objective. Consequently, it is essential to employ an effective pruning strategy to ensure that redundant triangles are removed before they become solid and opaque, as otherwise they would persist in the final scene representation and significantly degrade visual quality. During the first 5k iterations, opacity is not restricted, allowing important triangles to converge toward high opacities while less important ones naturally decay to low opacity. After 5k iterations, we apply a *hard pruning step* that removes all triangles with opacity below a threshold  $T_o \approx 0.2$ . This eliminates roughly 70% of the triangles and vertices, but ultimately improves the final visual quality. Using a lower threshold  $T_o$  reduces the immediate drop in quality, but in later stages, when triangles become opaque, many redundant primitives remain. Fig. 3 illustrates the effect of this hard pruning step. After 5k iterations, pruning continues. However, removing triangles solely based on low opacity  $o$  becomes ineffective, since opacities are explicitly pushed toward high values and our mapping function no longer permits low opacities. To identify triangles that have little influence or are completely occluded, we instead compute the maximum volume rendering weight  $T \cdot o$  (with  $T$  denoting transmittance and  $o$  opacity) for each triangle during rasterization. Triangles whose maximum weight falls below a threshold  $\tau_{\text{prune}}$  across all training views are discarded. Toward the end of training, as triangles become increasingly opaque and sharp, many triangles are pruned because they lie behind others and no longer contribute to the rendering. Vertices are pruned once



Figure 3. **Hard pruning step.** We eliminate unnecessary triangles (right) that would otherwise remain in the scene (left).

they are no longer connected to any triangle. A detailed analysis of the pruning impact is provided in Sec. 4.3.

**Densification.** Since the initial representation does not contain enough triangles and vertices to achieve high visual quality, we adopt a probabilistic MCMC-based framework [20] to progressively introduce additional triangles. At each densification step, candidate triangles are selected by sampling from a probability distribution constructed directly from their opacity  $o$  using Bernoulli sampling. New triangles and vertices are then generated through *midpoint subdivision*: the midpoints of the three edges of a selected triangle are connected, splitting it into four smaller triangles. The new midpoints are added to the vertex set  $\mathcal{V}$  and assigned the average color and opacity of their two adjacent vertices. This operation preserves both the total area and the spatial extent of the original primitive, and ensures that the subdivided triangles remain connected.

### 3.5. Optimization

**Initialization.** Our method starts from a set of images and their corresponding camera parameters, calibrated via SfM [28], which also provides a sparse point cloud. We apply 3D Delaunay triangulation on this point cloud to obtain a tetrahedralization, from which we extract all unique triangles. This initial mesh provides a well-connected set of triangles that serves as the starting point.

**Parameters & losses.** We optimize the 3D vertex positions  $\mathbf{v}_i$ , opacity  $o_i$ , and spherical harmonic color coefficients  $\mathbf{c}_i$  of all vertices by minimizing the rendering error from the given posed views. Our training loss combines the photometric  $\mathcal{L}_1$  and  $\mathcal{L}_{\text{D-SSIM}}$  terms [19], the opacity loss  $\mathcal{L}_o$  [20], and normal  $\mathcal{L}_n$  losses. The final loss  $\mathcal{L}$  is given by:

$$\mathcal{L} = (1 - \lambda)\mathcal{L}_1 + \lambda\mathcal{L}_{\text{D-SSIM}} + \beta_1\mathcal{L}_o + \beta_2\mathcal{L}_n. \quad (3)$$

For the normal loss  $\mathcal{L}_n$ , we supervise using a normal estimation model [14].

**Anti-aliasing.** To mitigate aliasing artifacts, we render at  $s \times$  the target resolution and then downsample to the final resolution using area interpolation, which averages over input pixel regions and acts as an anti-aliasing filter.

Dataset Method—Metric	Mip-NeRF360 Dataset							Tanks & Temples			
	E2E mesh	E2E colored	Mesh Con.	PSNR <sup>↑</sup>	LPIPS <sup>↓</sup>	SSIM <sup>↑</sup>	#Verts <sup>↓</sup>	PSNR <sup>↑</sup>	LPIPS <sup>↓</sup>	SSIM <sup>↑</sup>	#Verts <sup>↓</sup>
3DGS[19]	/	/	/	27.21	0.214	0.815	/	23.14	0.183	0.841	/
2DGS [15]	✗	✗	✓	15.36	0.474	0.498	2M	14.23	0.485	0.569	16M
GOF [35]	✗	✗	✓	20.78	0.465	0.573	33M	21.69	0.326	0.690	12M
RaDe-GS [36]	✗	✗	✓	23.56	0.361	0.668	31M	20.51	0.344	0.659	10M
MiLo [10]	✓	✗	✓	24.09	0.323	0.688	7M	21.46	0.348	0.706	4M
Triangle Splatting [12] †	✓	✓	✗	21.05	0.462	0.558	3M	17.27	0.402	0.600	6M
Ours	✓	✓	~	25.21	0.294	0.742	2M	20.91	0.249	0.773	2M

Table 1. **Mesh-based novel view synthesis on the Mip-NeRF360 dataset.** *E2E mesh* indicates whether a method directly produces a mesh. *E2E colored* denotes whether the mesh is already colored or requires post-processing. *Mesh Con.* specifies whether the reconstructed mesh consists of a connected component. ~ stands for semi-connectivity. † with only opaque triangles.

## 4. Experiments

**Implementation details.** For all experiments, we set the spherical harmonics to degree 3, which yields 51 parameters per vertex (48 from the SH coefficients and 3 from the vertex position) and 3 parameters per triangle. In comparison, a single Gaussian in 3DGS requires 59 parameters.

**Task.** Prior work [19, 24] evaluates reconstruction quality by comparing synthesized and ground-truth images, independent of whether the representation is implicit, explicit, or semi-transparent. We instead focus on the visual fidelity of the reconstructed mesh itself, using only opaque triangles. To this end, we follow MiLo [10] and adopt the task of *Mesh-Based Novel View Synthesis*, which assesses how well reconstructed meshes represent complete scenes.

**Baselines.** We compare against Triangle Splatting (restricted to solid and opaque triangles), and meshes derived from MiLo [10], 2DGS [15], Gaussian Opacity Fields (GOF) [35], and RaDe-GS [36], following the protocol of [10]. For MiLo, surface mesh extraction is integrated into the optimization itself, so no additional post-processing is required to obtain a mesh output. In contrast, 2DGS, GOF, and RaDe-GS rely on mesh extraction after optimization. Nevertheless, all these methods require an additional post-processing stage to color the mesh, achieved by training a neural color field for 5k iterations (see MiLo for details [10]). For Triangle Splatting, we adopt the *game engine training strategy*, which produces opaque triangles, and no additional post-processing is needed. For reference, we also compare against the original 3D Gaussian Splatting [19] to highlight the contrast between radiance-field rendering quality and mesh-based novel view synthesis. For our method, the final output consists of opaque, colored triangles, making it immediately compatible with mesh renderers without requiring any post-processing.

**Datasets & Metrics.** We compare our method to concurrent approaches on the standard benchmarks Mip-NeRF360 [1] and Tanks and Temples (T&T) [21]. We

evaluate the visual quality using standard metrics: SSIM, PSNR, and LPIPS. We report training time on an NVIDIA A100 and the total number of used vertices.

### 4.1. Mesh-Based Novel View Synthesis

Tab. 1 reports quantitative results on the Mip-NeRF360 and Tanks & Temples datasets for mesh-based novel view synthesis. Triangle Splatting+ consistently outperforms all concurrent methods across all metrics. Compared to 2DGS and Triangle Splatting, our method uses a similar number of vertices yet achieves a 4–10 dB higher PSNR. GOF, RaDe-GS, and MiLo require 2–10× more vertices while still obtaining lower PSNR and SSIM and higher LPIPS. In terms of LPIPS (the metric that best correlates with human visual perception), Triangle Splatting+ significantly outperforms all concurrent methods. Furthermore, 2DGS, GOF, and RaDe-GS require two additional post-processing steps after training: first extracting a mesh, and then coloring it. MiLo directly outputs a surface mesh after training, but still relies on a post-processing stage to texture the mesh. These extra steps limit the practicality of such methods, and increase the overall pipeline complexity. In contrast, our method directly produces colored, opaque triangles that are immediately compatible with any game engine, without requiring additional steps. Although 2DGS, GOF, RaDe-GS, and MiLo produce watertight and fully connected meshes, our representation is still sufficient for key downstream uses such as collision-based physical simulation or walkable environments in interactive games. Furthermore, Simplificits [25] shows that elastic simulation can be performed directly on non-watertight, even mesh-free representations. Triangle Splatting+ representation enables the same downstream applications without requiring a fully watertight mesh. Fig. 4 shows qualitative results, demonstrating that Triangle Splatting+ produces sharp renderings and reconstructs fine details.

**Training speed.** The training speed of Triangle Splatting+ is considerably higher than that of concurrent approaches. On the Mip-NeRF360 dataset, Triangle Splatting+ trains in

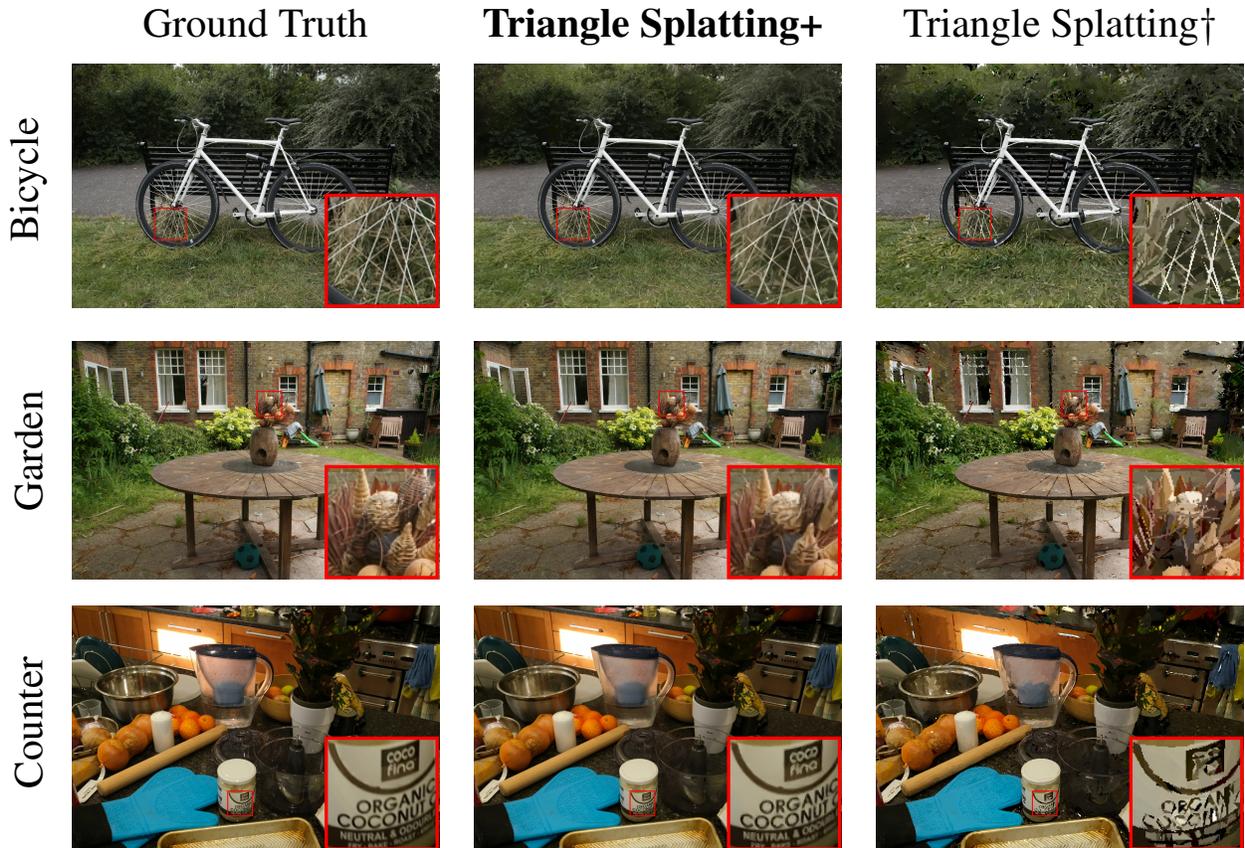


Figure 4. **Qualitative results.** Comparison of our method with Ground Truth and opaque Triangle Splatting [12]. Our approach produces renderings that are closer to the ground truth, with sharper details and more faithful recovery of fine structures. <sup>†</sup> Opaque version of Triangle Splatting.

39 minutes, and on the T&T dataset in 25 minutes. By comparison, MiLo requires around 45 minutes on T&T and up to 2 hours on Mip-NeRF360. While Triangle Splatting achieves shorter training times (17 minutes on Mip-NeRF360 and 20 minutes on T&T), this comes at the cost of substantially lower visual quality.

#### 4.2. Downstream applications.

**Removing or extracting objects.** Current 3D Gaussian Splatting methods for object extraction or removal [3, 34] face a fundamental challenge: a single pixel is influenced by the accumulated contributions of many primitives, rather than being assigned to a single one. This makes it non-trivial to decide whether a primitive belongs to a given object. To address this, prior work learns object associations during optimization [3, 34]. In contrast, Triangle Splatting+ requires no additional optimization during training. Since each pixel is determined by exactly one triangle, mapping objects from image space to 3D space becomes straightforward: given a 2D mask of an object, all triangles contributing to pixels within that mask are directly identified as

part of the object. By iterating over all training views, we recover the complete set of triangles belonging to the object. The object masks are generated using Segment Anything 2 [26], which enables the selection of single or multiple objects that can then be removed or extracted from a scene with minimal additional processing. Fig. 5 presents qualitative examples of two objects extracted from the Mip-NeRF360 and Tanks & Temples datasets.



Figure 5. **Straightforward extraction of objects.** With Triangle Splatting+, objects can be extracted or removed from a scene.

**Physical simulation and walkable environments.** As our representation contains no transparent or nearly invisible

primitives, the triangles may directly be interpreted as hard surfaces for the purpose of physics simulation. We demonstrate this by using an off-the-shelf non-convex mesh collider implementation, specifically the one provided in the Unity game engine. With no post-processing, our mesh can be loaded into a Unity scene and used for physics interaction with dynamic objects and characters. Additional visualizations are available on our project page. We also provide the PLY files for *Garden*, *Bicycle*, and *Truck*, along with a *Unity* project for exploring physics-based interactions and walkable scenes.

### 4.3. Ablation study

**Pruning and training strategy.** An effective pruning strategy is crucial for achieving high visual quality with opaque primitives. Tab. 2 summarizes the impact of the two pruning strategies on Triangle Splatting+. Without the hard pruning step at iteration  $5k$ , visual quality degrades noticeably. Unnecessary triangles remain in the scene and can no longer be removed. Similarly, when relying only on opacity as the pruning criterion, triangles can no longer be removed once the opacity floor has increased, since none of them are allowed to fall below the threshold. In contrast, using the blending weight as the criterion still enables pruning: triangles hidden behind others receive progressively lower transmittance as the front triangles become solid and opaque, and their blending weight eventually drops below the pruning threshold. Instead of initializing with soft triangles (i.e.,  $\sigma = 1.0$ ) and gradually converging towards solid triangles, we ablate the case where training starts directly with solid triangles, such that gradients can only be propagated through opacity. In this setting, performance and visual quality drop significantly, as optimization barely progresses due to vanishing gradients.

Method	PSNR $\uparrow$	LPIPS $\downarrow$	SSIM $\uparrow$
Baseline	25.28	0.289	0.751
w/o hard pruning step	-0.46	+0.029	-0.025
w/o blending weight pruning	-0.51	+0.069	-0.045
w/o sigma decay	-6.84	+0.243	-0.282

Table 2. **Pruning and training strategy ablated on Mip-NeRF360.** We isolate the impact of each design choice by removing them individually.

**Soft and semi-transparent triangles.** Tab. 3 shows how visual quality is affected when using solid and semi-transparent triangles, soft and opaque triangles, or soft and semi-transparent triangles, compared to the baseline with solid and opaque triangles. While higher final  $\sigma$  values yield slight improvements in visual quality, the gains are significantly smaller than those obtained by allowing opacity to remain optimizable rather than forcing it to be fully opaque. When the only objective is to maximize visual

$\sigma$	$o$	PSNR $\uparrow$	LPIPS $\downarrow$	SSIM $\uparrow$
Hard	Opaque	25.28	0.289	0.751
Soft	Opaque	+0.31	-0.018	+0.013
Hard	Free	+1.26	-0.048	+0.032
Soft	Free	<b>+1.34</b>	<b>-0.054</b>	<b>+0.038</b>

Table 3. **Relative improvements for different  $\sigma$  and opacity settings.** Here, *soft* denotes  $\alpha = 0.1$ , *hard* corresponds to a sharp transition, *opaque* enforces a fixed opacity of 1, and *free* allows opacity to be optimized.



Figure 6. **Limitations.** Accurately recovering backgrounds (left), particularly under limited viewpoints, and handling transparent objects (right) remain challenging.

quality, training with free opacity yields the best results, as semi-transparent triangles can be effectively blended and their colors accumulated to produce higher-quality renderings. However, in game engines, rendering *efficiency* depends on skipping sorting. For this reason, enforcing fully opaque triangles is essential for achieving the fastest rendering performance.

**Triangle connectivity.** During optimization, we split triangles and connect subsets of them, but the optimization process does not strictly enforce full connectivity. Mainly because of pruning, connectivity is only partially preserved. On average, each vertex is connected to 1.5 triangles. Overall, 80% of the triangles are connected to at least one other triangle, with some triangles connected to as many as six.

**Limitations.** Triangle Splatting+ achieves high visual quality and accurate reconstruction in regions where the initial point cloud is dense. In contrast, background areas with sparse coverage still exhibit incomplete geometry and reduced fidelity. Moreover, when moving outside the orbit of training views, the visual quality degrades. While the softness and opacity of Gaussian-based approaches may still provide slightly plausible results in such cases, our use of opaque triangles makes the artifacts more pronounced. Future work could address those limitation by initializing with a more complete point cloud, or by incorporating alternative additional representation such as a triangulated sky dome. Finally, transparent objects such as glasses or bottles remain difficult to represent using only opaque triangles, as illustrated in Fig. 6.

## 5. Conclusion

We introduced Triangle Splatting+ , a differentiable rendering framework that optimizes opaque triangles with shared-vertex connectivity. With a tailored training strategy for opaque primitives, our method achieves state-of-the-art performance in mesh-based novel view synthesis while remaining efficient to train. Unlike Gaussian-based approaches, the resulting representation is immediately compatible with game engines, enabling downstream applications such as relighting, physical simulation, and interactive walkthroughs. Triangle Splatting+ bridges radiance field optimization with traditional graphics pipelines, paving the way for practical integration of radiance field representations into interactive VR applications, game engines, and simulation frameworks

## References

- [1] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-NeRF 360: Unbounded anti-aliased neural radiance fields. In *IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pages 5460–5469, New Orleans, LA, USA, Jun. 2022. doi: 10.1109/cvpr52688.2022.00539. URL <https://doi.org/10.1109/CVPR52688.2022.00539>. 6
- [2] Nathaniel Burgdorfer and Philippos Mordohai. Radiant triangle soup with soft connectivity forces for 3D reconstruction and novel view synthesis. *arXiv*, abs/2505.23642, 2025. doi: 10.48550/arXiv.2505.23642. URL <https://doi.org/10.48550/arXiv.2505.23642>. 2, 3
- [3] Jiazhang Cen, Jiemin Fang, Chen Yang, Lingxi Xie, Xiaopeng Zhang, Wei Shen, and Qi Tian. Segment any 3D Gaussians. In *AAAI Conf. Artif. Intell.*, volume 39, pages 1971–1979, Philadelphia, PA, USA, Apr. 2025. Assoc. Adv. Artif. Intell. (AAAI). doi: 10.1609/aaai.v39i2.32193. URL <https://doi.org/10.1609/aaai.v39i2.32193>. 7
- [4] Haodong Chen, Runnan Chen, Qiang Qu, Zhaoqing Wang, Tongliang Liu, Xiaoming Chen, and Yuk Ying Chung. Beyond Gaussians: Fast and high-fidelity 3D splatting with linear kernels. *arXiv*, abs/2411.12440:1–14, 2024. doi: 10.48550/arXiv.2411.12440. URL <https://doi.org/10.48550/arXiv.2411.12440>. 2
- [5] Zhiqin Chen, Thomas Funkhouser, Peter Hedman, and Andrea Tagliasacchi. MobileNeRF: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures. In *IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pages 16569–16578, Vancouver, Can., Jun. 2023. doi: 10.1109/cvpr52729.2023.01590. URL <https://doi.org/10.1109/CVPR52729.2023.01590>. 3
- [6] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinlong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pages 5491–5500, New Orleans, LA, USA, Jun. 2022. doi: 10.1109/cvpr52688.2022.00542. URL <https://doi.org/10.1109/CVPR52688.2022.00542>. 2
- [7] Shrisudhan Govindarajan, Daniel Rebain, Kwang Moo Yi, and Andrea Tagliasacchi. Radiant foam: Real-time differentiable ray tracing. *arXiv*, abs/2502.01157, 2025. doi: 10.48550/arXiv.2502.01157. URL <https://doi.org/10.48550/arXiv.2502.01157>. 2
- [8] Markus Gross and Hanspeter Pfister. *Point-Based Graphics*. Morgan Kaufmann Publ. Inc., San Francisco, CA, USA, Jun. 2007. doi: 10.1016/B978-0-12-370604-1.X5000-7. URL <http://dx.doi.org/10.1016/B978-0-12-370604-1.X5000-7>. 2
- [9] Antoine Guédon and Vincent Lepetit. SuGaR: Surface-aligned Gaussian splatting for efficient 3D mesh reconstruction and high-quality mesh rendering. In *IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pages 5354–5363, Seattle, WA, USA, Jun. 2024. doi: 10.1109/cvpr52733.2024.00512. URL <https://doi.org/10.1109/CVPR52733.2024.00512>. 3
- [10] Antoine Guédon, Diego Gomez, Nissim Maruani, Bingchen Gong, George Drettakis, and Maks Ovsjanikov. MILo: Mesh-in-the-loop Gaussian splatting for detailed and efficient surface reconstruction. *arXiv*, abs/2506.24096, 2025. doi: 10.48550/arXiv.2506.24096. URL <https://doi.org/10.48550/arXiv.2506.24096>. 2, 3, 6
- [11] Abdullah Hamdi, Luke Melas-Kyriazi, Jinjie Mai, Guocheng Qian, Ruoshi Liu, Carl Vondrick, Bernard Ghanem, and Andrea Vedaldi. GES: Generalized exponential splatting for efficient radiance field rendering. In *IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pages 19812–19822, Seattle, WA, USA, Jun. 2024. doi: 10.1109/cvpr52733.2024.01873. URL <https://doi.org/10.1109/CVPR52733.2024.01873>. 2
- [12] Jan Held, Renaud Vandeghen, Adrien Delière, Daniel Hamdi, Abdullah Rebain, Silvio Giancola, Anthony Cioppa, Andrea Vedaldi, Bernard Ghanem, Andrea Tagliasacchi, and Marc Van Droogenbroeck. Triangle splatting for real-time radiance field rendering. *arXiv*, abs/2505.19175, 2025. doi: 10.48550/arXiv.2505.19175. URL <https://doi.org/10.48550/arXiv.2505.19175>. 2, 3, 4, 6, 7
- [13] Jan Held, Renaud Vandeghen, Abdullah Hamdi, Adrien Delière, Anthony Cioppa, Silvio Giancola, Andrea Vedaldi, Bernard Ghanem, and Marc Van Droogenbroeck. 3D convex splatting: Radiance field rendering with 3D smooth convexes. In *IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pages 21360–21369, Nashville, TN, USA, Jun. 2025. doi: 10.1109/CVPR52734.2025.01990. URL <https://doi.org/10.1109/CVPR52734.2025.01990>. 2, 3
- [14] Mu Hu, Wei Yin, Chi Zhang, Zhipeng Cai, Xiaoxiao Long, Hao Chen, Kaixuan Wang, Gang Yu, Chunhua Shen, and Shaojie Shen. Metric3d v2: A versatile monocular geometric foundation model for zero-shot metric depth and surface normal estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(12):10579–10596, Dec. 2024. ISSN 1939-3539. doi: 10.1109/tpami.2024.3444912. URL <http://dx.doi.org/10.1109/TPAMI.2024.3444912>. 5
- [15] Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger, and Shenghua Gao. 2D Gaussian splatting for geometrically accurate radiance fields. In *ACM SIGGRAPH Conf. Pap.*, volume 35, pages 1–11, Denver, CO, USA, Jul. 2024. ACM. doi: 10.1145/3641519.3657428. URL <https://doi.org/10.1145/3641519.3657428>. 2, 3, 6
- [16] Yi-Hua Huang, Ming-Xian Lin, Yang-Tian Sun, Ziyi Yang, Xiaoyang Lyu, Yan-Pei Cao, and Xiaojuan Qi. Deformable radial kernel splatting. In *IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pages 21513–21523, Nashville, TN, USA, Jun. 2025. doi: 10.1109/cvpr52734.2025.02004. URL <https://doi.org/10.1109/CVPR52734.2025.02004>. 2
- [17] Changjian Jiang, Kerui Ren, Linning Xu, Jiong Chen, Jiangmiao Pang, Yu Zhang, Bo Dai, and Mulin Yu. HaloGS: Loose coupling of compact geometry and Gaussian splats for 3D scenes. *arXiv*, abs/2505.20267, 2025. doi: 10.

- 48550/arXiv.2505.20267. URL <https://doi.org/10.48550/arXiv.2505.20267>. 2
- [18] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3D mesh renderer. In *IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pages 3907–3916, Salt Lake City, UT, USA, Jun. 2018. doi: 10.1109/cvpr.2018.00411. URL <https://doi.org/10.1109/CVPR.2018.00411.2>
- [19] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkuehler, and George Drettakis. 3D Gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 42(4):1–14, Jul. 2023. doi: 10.1145/3592433. URL <https://doi.org/10.1145/3592433.2,3,5,6>
- [20] Shakiba Kheradmand, Daniel Rebain, Gopal Sharma, Weiwei Sun, Jeff Tseng, Hossam Isack, Abhishek Kar, Andrea Tagliasacchi, and Kwang Moo Yi. 3D Gaussian splatting as Markov chain Monte Carlo. In *Adv. Neural Inf. Process. Syst. (NeurIPS)*, volume 37, pages 80965–80986, Vancouver, Can., Dec. 2024. Curran Assoc. Inc. URL <https://neurips.cc/virtual/2024/poster/94984.5>
- [21] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: benchmarking large-scale scene reconstruction. *ACM Trans. Graph.*, 36(4):1–13, Jul. 2017. doi: 10.1145/3072959.3073599. URL <https://doi.org/10.1145/3072959.3073599.6>
- [22] Shichen Liu, Weikai Chen, Tianye Li, and Hao Li. Soft rasterizer: A differentiable renderer for image-based 3D reasoning. In *IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, pages 7707–7716, Seoul, South Korea, Oct. 2019. doi: 10.1109/iccv.2019.00780. URL <https://doi.org/10.1109/ICCV.2019.00780.2>
- [23] Matthew M. Loper and Michael J. Black. OpenDR: An approximate differentiable renderer. In *Eur. Conf. Comput. Vis. (ECCV)*, volume 8695 of *Lect. Notes Comput. Sci.*, pages 154–169, Zürich, Switzerland, 2014. doi: 10.1007/978-3-319-10584-0\_11. URL [https://doi.org/10.1007/978-3-319-10584-0\\_11.2](https://doi.org/10.1007/978-3-319-10584-0_11.2)
- [24] 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 *Eur. Conf. Comput. Vis. (ECCV)*, volume 12346 of *Lect. Notes Comput. Sci.*, pages 405–421, Virtual conference, 2020. doi: 10.1007/978-3-030-58452-8\_24. URL [https://doi.org/10.1007/978-3-030-58452-8\\_24.2,6](https://doi.org/10.1007/978-3-030-58452-8_24.2,6)
- [25] Vismay Modi, Nicholas Sharp, Or Perel, Shinjiro Sueda, and David I. W. Levin. Simplicitis: Mesh-free, geometry-agnostic elastic simulation. *ACM Trans. Graph.*, 43(4):1–11, Jul. 2024. doi: 10.1145/3658184. URL <https://doi.org/10.1145/3658184.6>
- [26] Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, Ronghang Hu, Chaitanya Ryali, Tengyu Ma, Haitham Khedr, Roman Rädle, Chloe Rolland, Laura Gustafson, Eric Mintun, Junting Pan, Kalyan Vasudev Alwala, Nicolas Carion, Chao-Yuan Wu, Ross Girshick, Piotr Dollar, and Christoph Feichtenhofer. SAM 2: Segment anything in images and videos. In *Int. Conf. Learn. Represent. (ICLR)*, pages 1–44, Singapore, February 2025. URL <https://openreview.net/forum?id=Ha6RTeWMD0.7>
- [27] Christian Reiser, Stephan Garbin, Pratul Srinivasan, Dor Verbin, Richard Szeliski, Ben Mildenhall, Jonathan Barron, Peter Hedman, and Andreas Geiger. Binary opacity grids: Capturing fine geometric detail for mesh-based view synthesis. *ACM Trans. Graph.*, 43(4):1–14, Jul. 2024. doi: 10.1145/3658130. URL <https://doi.org/10.1145/3658130.3>
- [28] Johannes L. Schonberger and Jan-Michael Frahm. Structure-from-motion revisited. In *IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pages 4104–4113, Las Vegas, NV, USA, Jun. 2016. doi: 10.1109/cvpr.2016.445. URL <https://doi.org/10.1109/CVPR.2016.445.5>
- [29] Sanghyun Son, Matheus Gadelha, Yang Zhou, Matthew Fisher, Zexiang Xu, Yi-Ling Qiao, Ming C. Lin, and Yi Zhou. DMesh++: An efficient differentiable mesh for complex shapes. *arXiv*, abs/2412.16776, 2024. doi: 10.48550/arXiv.2412.16776. URL <https://doi.org/10.48550/arXiv.2412.16776.2,4>
- [30] Sanghyun Son, Matheus Gadelha, Yang Zhou, Zexiang Xu, Ming C. Lin, and Yi Zhou. DMesh: A differentiable mesh representation. In *Adv. Neural Inf. Process. Syst. (NeurIPS)*, volume 37, pages 12035–12077, Vancouver, Can., Dec. 2024. Curran Assoc. Inc. URL [https://proceedings.neurips.cc/paper\\_files/paper/2024/hash/1651f1ea5ee6213e301a89f222f3fec7-Abstract-Conference.html.2](https://proceedings.neurips.cc/paper_files/paper/2024/hash/1651f1ea5ee6213e301a89f222f3fec7-Abstract-Conference.html.2)
- [31] Cheng Sun, Jaesung Choe, Charles Loop, Wei-Chiu Ma, and Yu-Chiang Frank Wang. Sparse voxels rasterization: Real-time high-fidelity radiance field rendering. In *IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pages 16187–16196, Nashville, TN, USA, Jun. 2025. doi: 10.1109/cvpr52734.2025.01509. URL <https://doi.org/10.1109/CVPR52734.2025.01509.2>
- [32] Nicolas von Lütow and Matthias Nießner. LinPrim: Linear primitives for differentiable volumetric rendering. *arXiv*, abs/2501.16312, 2025. doi: 10.48550/arXiv.2501.16312. URL <https://doi.org/10.48550/arXiv.2501.16312.2>
- [33] Lior Yariv, Peter Hedman, Christian Reiser, Dor Verbin, Pratul P. Srinivasan, Richard Szeliski, Jonathan T. Barron, and Ben Mildenhall. BakedSDF: Meshing neural SDFs for real-time view synthesis. In *ACM SIGGRAPH Conf. Proc.*, pages 1–9, Los Angeles, CA, USA, Jul. 2023. ACM. doi: 10.1145/3588432.3591536. URL <https://doi.org/10.1145/3588432.3591536.3>
- [34] Mingqiao Ye, Martin Danelljan, Fisher Yu, and Lei Ke. Gaussian grouping: Segment and edit anything in 3D scenes. In *Eur. Conf. Comput. Vis. (ECCV)*, volume 15087 of *Lect. Notes Comput. Sci.*, pages 162–179, Nov. 2024. doi: 10.1007/978-3-031-73397-0\_10. URL [https://doi.org/10.1007/978-3-031-73397-0\\_10.7](https://doi.org/10.1007/978-3-031-73397-0_10.7)
- [35] Zehao Yu, Torsten Sattler, and Andreas Geiger. Gaussian opacity fields: Efficient adaptive surface reconstruction in unbounded scenes. *ACM Trans. Graph.*, 43(6):1–13, Dec.

2024. doi: 10.1145/3687937. URL <https://doi.org/10.1145/3687937>. 3, 6

- [36] Baowen Zhang, Chuan Fang, Rakesh Shrestha, Yixun Liang, Xiaoxiao Long, and Ping Tan. RaDe-GS: Rasterizing depth in Gaussian splatting. *arXiv*, abs/2406.01467, 2024. doi: 10.48550/arXiv.2406.01467. URL <https://doi.org/10.48550/arXiv.2406.01467>. 2, 3, 6