1000FPS+ Novel View Synthesis from End-to-End Opaque Triangle Optimization

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

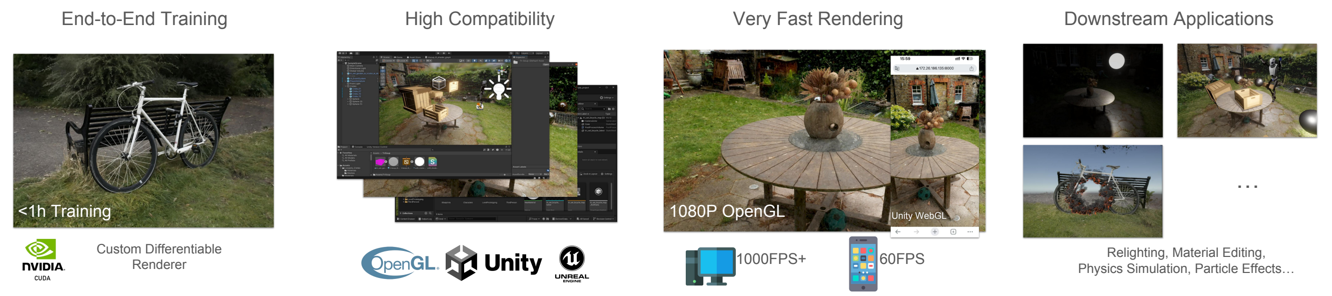


Figure 1. Our end-to-end opaque triangle optimization (OTO) method with a custom differentiable renderer demonstrates high compatibility and very fast rendering across desktop and mobile platforms. Additionally, OTO models are natively supported by conventional graphics engines, enabling a wide range of downstream applications.

Abstract

Recent implicit and primitive-based radiance field methods, such as NeRF and 3DGS, have demonstrated impressive capabilities in novel view synthesis from multi-view images. However, their custom representations are often incompatible with conventional graphics pipelines, limiting their application in areas like editing, relighting, physics simulation, and particle effects. Additionally, their volume rendering approach requires alpha-blending multiple colors per pixel, which slows down rendering. Traditional differentiable rendering methods, while offering higher compatibility and speed, rely on known mesh topology, making them unsuitable for complex scene-level reconstruction. To address these limitations, we introduce a novel end-to-end optimization process of disjoint opaque triangles, natively compatible with standard graphics engines for a wide range of applications. To enable gradient-based optimization over the highly non-differentiable rasterization process, we employ a 2D SDF approximation and a two-layer occlusion approximation. We also incorporate density controls to ensure detailed and complete scene reconstructions. Our paper tackles the challenging end-to-end optimization of scene-level novel view synthesis with opaque representation only. Our approach achieves over 1000 FPS rendering on a single desktop GPU, providing high compatibility and similar novel view synthesis quality to existing methods.

1. Introduction

Recent advancements in radiance fields have significantly improved rendering quality and speed for novel view synthesis from multi-view images. Neural Radiance Fields (NeRF) methods [1, 30, 32] have been widely adopted to model 3D scenes with implicit neural representations using multi-layer perceptrons (MLPs). Through differentiable volumetric rendering, these 3D implicit representations can be optimized with 2D image supervision. However, the need for a large number of samples per image has limited rendering speed. To address this, 3D Gaussian Splatting (3DGS) [19] was introduced, replacing the implicit representation with a primitive-based approach that directly splats semi-transparent 3D Gaussian primitives onto the image plane to achieve faster rasterization.

Despite their strengths, neither implicit representations nor 3D Gaussian primitives are natively compatible with conventional graphics engines, which restricts their applicability in content creation pipelines. This incompatibility limits support for essential downstream applications, including mobile device compatibility, relighting, physics simulation, and particle effects. Some two-stage baking methods [41, 54] attempt to bridge this gap by converting implicit or 3D Gaussian representations into mesh, compat-

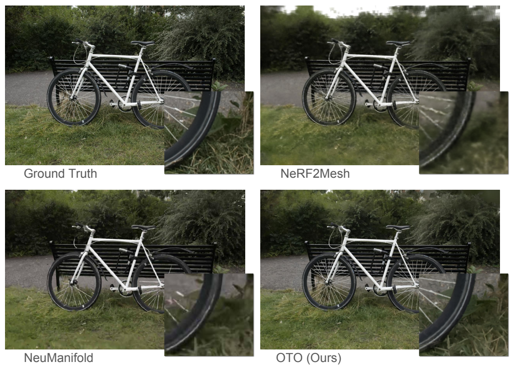


Figure 2. Comparison with two-stage baking methods, highlighting the superior detail-preserving capability of OTO.

ible with standard graphics engines. However, these two-stage baking approaches tend to lose fine structural details due to conversion resolution limitations, as illustrated in Fig. 2.

On the other hand, differentiable rendering algorithms have been developed to directly optimize mesh vertex positions and attributes, enabling the recovery of shape and surface properties from images. A key limitation of these methods, however, is their inability to scale effectively to scene-level reconstruction. Most differentiable rendering approaches [8, 9, 18, 25] rely on a known mesh topology, which is practical only for single objects. Recent methods [14, 21, 40] attempt to eliminate the topology requirement using techniques like deep marching cubes and tetrahedra, but these approaches are still limited to object-level reconstruction due to their resource-intensive dense grid design.

To address the compatibility and speed limitations of radiance field methods while avoiding the detail loss common in two-stage approaches, we introduce opaque triangle optimization (OTO), a novel representation for end-to-end optimization. Inspired by differentiable rendering techniques, OTO optimizes disjoint opaque triangles, breaking free from the constraints of fixed-topology meshes. The triangles are initialized from a sparse point cloud generated by structure-from-motion (SfM) and undergo automatic density control to enhance scene detail and completeness. This approach enables end-to-end optimization, producing models that are directly compatible with conventional graphics engines and support various downstream applications. To facilitate gradient calculation during the non-differentiable rasterization process, we employ a 2D SDF approximation and a two-layer occlusion approximation. Evaluated on the MipNeRF-360 [1] dataset, our algorithm achieves similar rendering quality, over 1000 FPS rendering speed, and seamless compatibility with conventional graphics engines for various downstream applications.

2. Related Works

2.1. Radiance Fields for Novel View Synthesis

Neural Radiance Fields (NeRF) [31] and various enhancements [3, 5, 17, 20, 24, 32, 38, 46, 55] emerged in the recent years with significant success in performing 3D reconstruction and novel view synthesis. These works represent radiance fields with multilayer-perceptrons (MLPs)[17, 31], grid structures[24, 55], or both[3, 32, 38]. Some extensions are also introduced for dynamic scene[4, 13, 34–36, 51], specular scenes[16, 44, 51], anti-aliasing[3, 17], semantics[6] and surface reconstruction[46, 53].

3D Gaussian Splatting(3DGS)[19] was introduced later to use 3D Gaussian primitives for scene representation. Its fast splatting algorithm surpasses the sample-based method used in NeRFs in terms of rendering efficiency while maintaining high rendering quality. Many follow-up works were introduced for improved quality[27, 52, 56], dynamic scenes[28, 29, 49], compression[12] and more[7].

Despite the excellent novel view synthesis quality of NeRF and 3DGS methods, they all rely on their own custom representation and rendering pipeline. Unlike mesh models, they do not natively support conventional graphics engines, limiting their rendering speed and downstream compatibility. Some works [10, 15, 22] attach a neural texture or 3D Gaussians to an optimized mesh. They often provide geometry editing capability, but the rendering pipeline is still not natively compatible with conventional graphics engines.

2.2. Mesh Baking from Radiance Fields

To address the incompatibility issue discussed earlier, several works [11, 23, 37, 41] have explored two-stage methods that involve baking or distilling a radiance field into a mesh model. These approaches typically involve training a NeRF or 3DGS model [7], followed by mesh extraction using techniques like marching cubes [26] or other mesh extraction algorithms. The extracted mesh's attributes, including vertex positions and texture colors, are often fine-tuned to improve rendering quality. Some methods [11] further simplify the mesh to reduce memory usage and enhance rendering efficiency.

However, the primary drawback of these two-stage methods is the loss of detail during the extraction phase. NeRF and 3DGS are highly effective at representing fine structures in a scene due to their flexible, topology-free nature. Converting these representations into a mesh format makes it challenging to retain intricate details, as meshes have limited resolution and tend to impose smoothness constraints during extraction. While the rendering of the resultant mesh is typically very fast, it often comes at the cost of overly smoothed images.

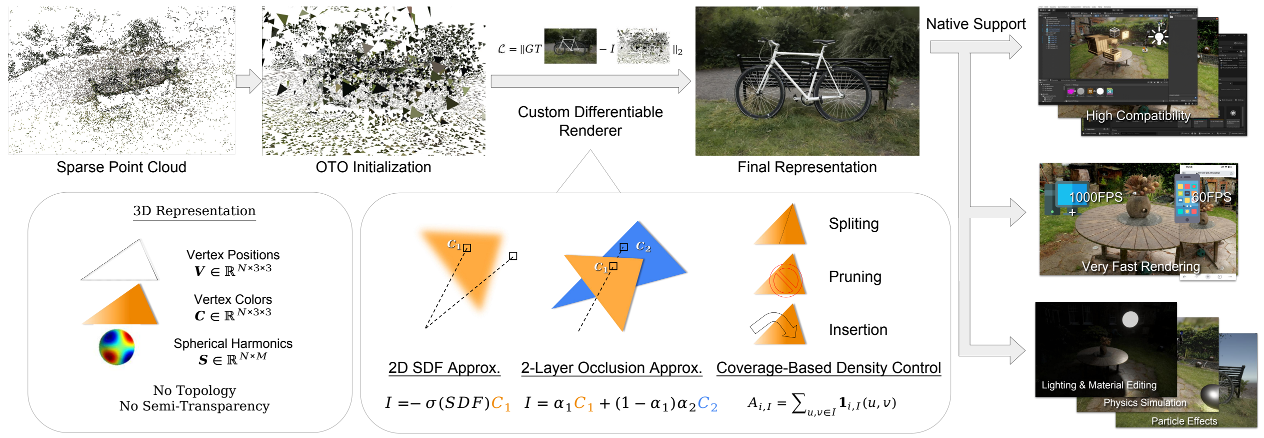

Sparse Point Cloud
OTO Initialization
$\mathcal{L} = \|GT - I\|_2$
Custom Differentiable Renderer
Final Representation
Native Support
High Compatibility
1000FPS
60FPS
Very Fast Rendering
Lighting & Material Editing
Physics Simulation
Particle Effects
3D Representation
Vertex Positions $\boldsymbol{V} \in \mathbb{R}^{N\times3\times3}$
Vertex Colors $\boldsymbol{C} \in \mathbb{R}^{N\times3\times3}$
Spherical Harmonics $\boldsymbol{S} \in \mathbb{R}^{N\times M}$
No Topology
No Semi-Transparency
C_1
C_2
Spliting
Pruning
Insertion
2D SDF Approx.
2-Layer Occlusion Approx.
Coverage-Based Density Control
$I = -\sigma(SDF)C_1$
$I = \alpha_1 C_1 + (1-\alpha_1)\alpha_2 C_2$
$A_{i,I} = \sum_{u,v\in I} \mathbf{1}_{i,I}(u,v)$

Figure 3. Our OTO representation, parameterized by vertex positions, colors, and spherical harmonics, is initialized from a sparse point cloud. Leveraging 2D SDF approximation, two-layer occlusion approximation, and coverage-based density control, OTO is optimized using photometric rendering loss and our custom differentiable rendering pipeline. The resulting model natively supports various graphics engines, downstream applications, and enables ultra-fast rendering.

2.3. Reconstruction using Differentiable Rendering

Many works have explored the direct optimization of explicit mesh representations through differentiable rendering. Some methods initialize a spherical or cubic mesh with a fixed topology for optimization [8, 9, 18, 25, 45], while more recent approaches introduce deep marching cubes or tetrahedra to overcome the limitations of fixed topology [14, 21, 40]. These techniques optimize mesh vertices, colors, and sometimes materials and lighting conditions through a differentiable rendering pipeline. However, a common limitation of these methods is their focus on object reconstruction, making them less suitable for complex scenes. Acquiring the topology of an intricate scene before optimization is often infeasible, and performing deep marching cubes or marching tetrahedra at high enough resolutions to capture scene details can be computationally prohibitive. Consequently, directly optimizing a highly discrete representation, like a mesh, for complex scenes remains largely unexplored.

In contrast, our proposed OTO representation addresses these limitations by decomposing the mesh into disjoint, opaque triangle primitives. With carefully designed approximations and density control, OTO combines the compatibility advantages of mesh representations with the ability to preserve fine details for novel view synthesis, overcoming the constraints of fixed topology in complex scene reconstruction.

3. Our Method

3.1. Differentiable Rendering

Following the approach of existing novel view synthesis methods, we render our 3D representation into a 2D image in a forward pass, then apply a backward pass using the photometric 2D loss to update the OTO. We utilize only the mean squared error, defined as $L_{MSE} = ||\bar{I} - I||$ between ground truth $\bar{I}$ and rendered image I. To ensure that the exported OTO model is compatible with conventional graphics engines, we design the forward pass to closely mimic the traditional graphics pipeline, while the backward pass approximates the gradient.

Representation Our OTO representation consists of a set of N disjoint opaque triangles, parameterized by vertex positions $\mathbf{V} \in \mathbb{R}^{N\times3\times3}$, vertex diffuse colors $\mathbf{C} \in \mathbb{R}^{N\times3\times3}$, and triangle view-dependent color parameters $\mathbf{S} \in \mathbb{R}^{N\times M}$. Each triangle has 3 vertex position XYZ, 3 vertex diffuse color RGB, and M view-dependent color parameters. In differentiable rendering, view-dependent colors are typically modeled with various lighting and material parameters. However, for scene-level reconstruction with complex and unknown lighting conditions, we adopt a simple spherical harmonics function instead.

The OTO initialization process begins with a sparse point cloud generated by COLMAP Structure-from-Motion (SfM)[39] similar to 3DGS[19]. One triangle is spawned for each point, with its vertex diffuse color taking the point color and the view-dependent parameters initialized with zeros. To determine the exact position of the three vertices, we first estimate the surface normal of each point using Open3D[58], and the size of the triangle using the nearest neighbor distance. A maximum triangle size limit is imposed based on its estimated image coverage on the nearest camera to prevent large floaters from covering the camera. For unbounded scenes, we incorporate a skybox, represented by a six-faced cube with each face subdivided into a 50×50 array of triangles. While the colors of the skybox triangles are optimized during training, their vertex positions remain fixed. Additional details on the initialization stage can be found in the supplementary material.

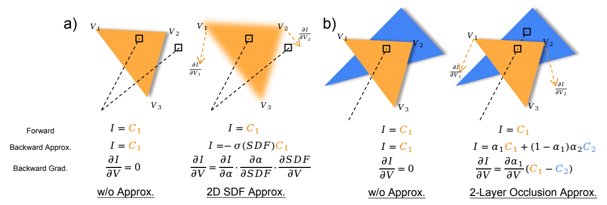

a)
V_1
V_2
V_3
$\frac{\partial I}{\partial V_1}$
$\frac{\partial I}{\partial V_2}$
b)
Forward
Backward Approx.
Backward Grad.
$I = C_1$
$I = C_1$
$\frac{\partial I}{\partial V} = 0$
w/o Approx.
$I = C_1$
$I = -\sigma(SDF)C_1$
$\frac{\partial I}{\partial V} = \frac{\partial I}{\partial \alpha} \cdot \frac{\partial \alpha}{\partial SDF} \cdot \frac{\partial SDF}{\partial V}$
2D SDF Approx.
$I = C_1$
$I = C_1$
$\frac{\partial I}{\partial V} = 0$
w/o Approx.
$I = C_1$
$I = \alpha_1 C_1 + (1-\alpha_1)\alpha_2 C_2$
$\frac{\partial I}{\partial V} = \frac{\partial \alpha_1}{\partial V}(C_1 - C_2)$
2-Layer Occlusion Approx.

Figure 4. An illustration of gradient estimation using our (a) 2D SDF approximation and (b) two-layer occlusion approximation.

Vertex Processing Vertex processing is the first step in the rendering pipeline, converting 3D triangle vertices into their 2D projections. Our forward pass projects 3D vertex positions to 2D and evaluates the vertex color based on the current camera direction. Specifically, the 2D vertex position and depth $\mathbf{V}_{2D}^{i,j} = \{x^{i,j}, y^{i,j}, z^{i,j}\}$ of triangle i and vertex j are calculated by transforming 3D vertex position $\mathbf{V}^{i,j}$ using camera extrinsics $[\mathbf{R}|\mathbf{t}]$ and intrinsics $\mathbf{K}$. The 2D view-dependent color $\mathbf{C}_{2D}^{i,j} = \mathbf{C}^{i,j} + sh(\mathbf{S}^i, \mathbf{D}^i)$, where sh denotes the spherical harmonics function and $\mathbf{D}^i$ is the direction from camera center to the triangle center. To reduce the parameter count, we model each triangle with a uniform view-dependent appearance due to their small size. All operations in the vertex processing step are differentiable, so no approximation is required in the backward pass.

Fragment Interpolation Fragments are potential pixels. Fragment interpolation calculates fragment attributes based on vertex attributes $\mathbf{V}_{2D}^{i,1:3}$, $\mathbf{C}_{2D}^{i,1:3}$ and pixel position $\{u, v\}$. We employ the barycentric interpolation but adapt it to a clamped version to handle pixels outside the triangle, which are necessary for the backward pass. The adjusted barycentric weight $\hat{b}_j$ is computed using the original value b_j of the fragment with respect to the triangle:

$$\hat{b}_j = max(b_j, 0) / \sum_{\bar{j}=1}^{3} max(b_{\bar{j}}, 0). \tag{1}$$

Using these weights, the fragment position and colors are interpolated as $\mathbf{V}_{frag}^{i,u,v} = \sum_j^3 \hat{b}_j \mathbf{V}_{2D}^{i,j}$ and $\mathbf{C}_{frag}^{i,u,v} = \sum_j^3 \hat{b}_j \mathbf{C}_{2D}^{i,j}$.

Rasterization Rasterization is one of the most challenging aspects of scene optimization due to its inherently non-differentiable nature. As illustrated in Fig. 4, a pixel takes the color of the triangle fragment if and only if the fragment lies within the triangle, and is the nearest fragment to the camera among all other fragments at that pixel location. This can be written as:

$$\begin{aligned} \mathbf{I}_{u,v} = \sum_{i=1}^{N} & is\_inside(x^i, y^i, u, v) \\ & \cdot is\_nearest(z_{frag}^{i,u,v}) \cdot \mathbf{C}_{frag}^{i,u,v}. \end{aligned} \tag{2}$$

As both the $is\_inside()$ and $is\_nearest()$ functions take either value 1 or 0, they are discrete and non-differentiable. The vertex positions of the triangle receive no gradient and the shape cannot be updated.

To enable the gradient-based optimization of OTO, we draw inspiration from volume rendering to redefine the rasterization process as:

$$\mathbf{I}_{u,v} = \sum_{i=1}^{N} \alpha_i \cdot \mathbf{C}_{frag}^{i,u,v} \cdot \prod_{k=1}^{i-1} (1 - \alpha_k) \tag{3a}$$

$$\alpha_i = -\ step(SDF(x^i, y^i, u, v)), \tag{3b}$$

with an "opacity" value α and triangles $i \in \{1 : N\}$ sorted by depth. Unlike NeRF or 3DGS, our opaque triangles only have opacity values of 1. However, we introduce this value calculated with a 2D signed distance function (SDF) and a step function for later use. The 2D SDF function is defined as $SDF(x^i, y^i, u, v) = d \cdot s$, where d is the shortest distance from fragment (u, v) to the triangle boundary, and $s \in \{-1, +1\}$ indicates whether the fragment is inside $(s = -1)$ or outside $(s = +1)$ the triangle

During the backward pass, we approximate the rasterization equation to:

$$\mathbf{I}_{u,v} \approx \alpha_1 \cdot \mathbf{C}_{frag}^{1,u,v} + (1 - \alpha_1) \cdot \alpha_2 \cdot \mathbf{C}_{frag}^{2,u,v} \tag{4}$$

Only the nearest triangle $(i = 1)$ covering or close to the pixel, and the second nearest triangles $(i = 2$ containing this pixel location are considered, forming a two-layer approximation to simplify occlusion as shown in Fig. 4b. Although the "opacity" value α for opaque triangles is always 1, this allows us to calculate the gradient $\frac{\partial \mathbf{I}_{uv}}{\partial \alpha_1} = \mathbf{C}_{frag}^{1,u,v} - \alpha_2 \cdot \mathbf{C}_{frag}^{2,u,v}$ taking occlusion into account. Intuitively, this pushes the first-layer triangle to expand if its color is closer to the ground truth than that of the second-layer triangle, or to shrink otherwise.

To approximate the hard step function, we use a soft sigmoid function defined as $\sigma(x) = (1 + exp(-\tau \cdot x))^{-1}$, with temperature τ, as shown in Fig. 4a:

$$\alpha_1 \approx -\sigma(SDF(x^1, y^1, u, v), \tau). \tag{5}$$

This provides a "spread" around the triangle boundaries, with the scale controlled by temperature τ, allowing all nearby fragments to receive gradients. Empirically, We find a higher temperature τ for a shaper sigmoid function and a more accurate approximation results in better quality.

By replacing the is_inside and $is\_nearest$ functions with the proposed approximation, the forward pass remains unchanged, while the backward pass enables gradient-based optimization of vertex positions and triangle shapes. To validate the effectiveness of this approach, we include a 2D example where a random image is approximated using uniformly initialized opaque triangles. As shown in Fig. 5, the

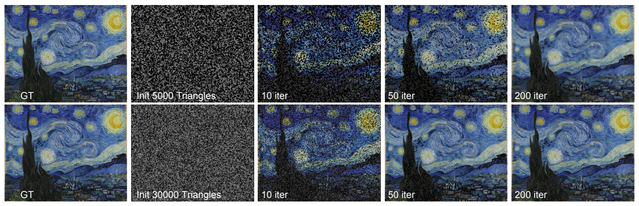


Figure 5. An example of fitting a 2D image using our OTO method with the proposed approximations.

triangles quickly adjust to closely resemble the target image, capturing fine details within seconds.

Additional Rendering Details To mitigate aliasing effects for small triangles, we apply super-sampling anti-aliasing (SSAA) with a factor of 2. To prevent large triangles from slowing down the rendering process for the entire image, we implement tiled rendering, dividing triangles into smaller 16×16 tiles before fragment processing. During tiling, we also add a small margin around each triangle to capture pixels outside the triangle, which are required for the backward pass. The margin size is determined by the temperature of the sigmoid function used for the 2D SDF.

3.2. Density Control

The initial point cloud generated by structure-from-motion is often sparse, incomplete, or contains errors, necessitating the addition and removal of certain primitives—similar to the approach used in 3DGS [19]. To address this, we propose three types of density control: splitting, pruning, and insertion. Leveraging the characteristics of our opaque representation, we introduce an efficient, intuitive coverage-based density control method. This approach evaluates the pixel coverage of each triangle at predetermined intervals during training, allowing us to dynamically adjust the density by adding, removing, or refining triangles as needed.

Splitting A triangle is split if it appears too large on the image or covers a region with high color variation. Unlike semi-transparent primitives, our opaque triangles allow each pixel to be directly associated with a specific triangle. The area $A_{i,I}$ for a triangle i on an image I can be easily calculated by counting the pixel rendered by the triangle $A_{i,I} = \sum_{u,v \in I} \mathbf{1}_{i,I}(u,v)$, where $\mathbf{1}_{i,I}(u,v)$ indicates triangle i is the nearest triangle covering pixel (u,v) on image I. The coverage color variation is defined as the variation of all ground truth pixel colors $\hat{I}_{u,v}$ covered by a triangle $\mathcal{V}_{i,I} = var(\{\hat{\mathbf{I}}_{u,v}, \forall(u,v) \in I \wedge \mathbf{1}_{i,I}(u,v)\})$. A triangle is split if its highest coverage count or average coverage color variation across all training images exceeds a percentile threshold. During splitting, two new triangles are formed to replace the original one using the mid-point at the longest edge as an additional vertex.

Pruning A triangle should be pruned if it appears too small in the image or has extremely high error. Since opaque triangles do not have an opacity parameter, some triangles may become very small but never fully disappear. To avoid aliasing artifacts and performance overhead, any triangle whose maximum coverage area $A_{i,I}$ across all images falls below a certain threshold will be pruned. Additionally, some floater triangles may be difficult to optimize due to incorrect initialization position. We detect these triangles using the coverage error $\mathcal{E}_{i,I} = \sum_{u,v \in I} (\hat{\mathbf{I}}_{u,v} - \mathbf{I}_{u,v})^2 \cdot \mathbf{1}_{i,I}(u,v)$ and remove those with coverage error above a certain percentile threshold.

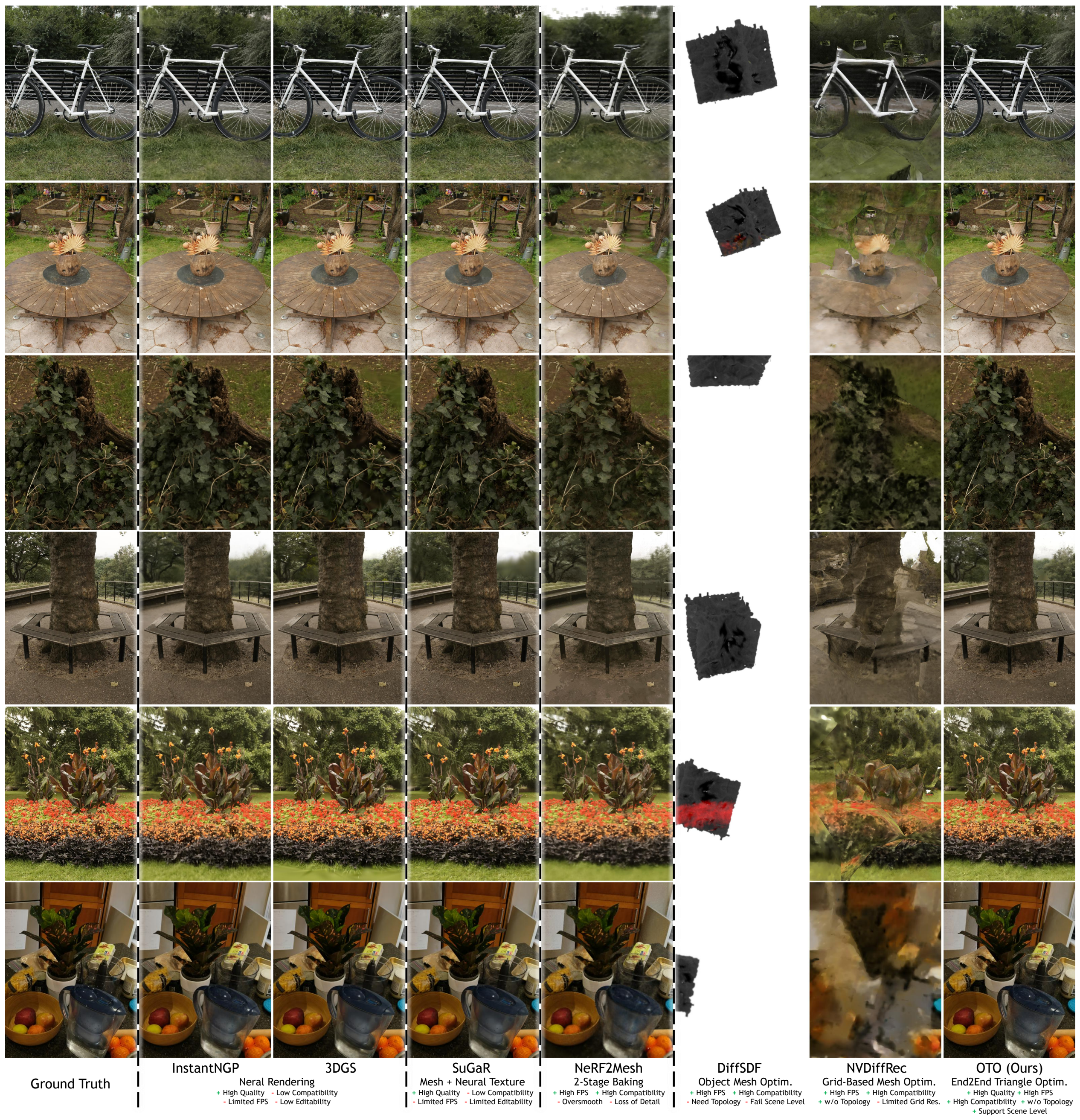
Insertion To ensure scene completeness, additional triangles are inserted. Some pixels covered by the skybox retain high error values even after many optimization iterations, indicating that they may belong to the foreground rather than the distant background. For such pixels, we search within a $W \times W$ 2D window on the training image to find pixels covered by a foreground triangle with a similar color in the ground truth training image. A new triangle, matching this color, is then inserted along the ray direction of the pixel and positioned at the depth of the selected foreground triangle. Intuitively, this duplicates the foreground triangle to fill up the hole.

4. Experiments

In this section, we present both qualitative (Fig. 6) and quantitative (Tab. 1) comparisons between our proposed OTO method and four categories of existing methods: neural rendering methods, mesh geometry with neural texture methods, two-stage baking methods, and end-to-end mesh optimization methods. Each category showcases unique strengths and weaknesses, and we analyze them individually to highlight the distinct advantages of OTO.

Given the diverse set of baseline methods, we use the MipNeRF-360 dataset [2] as the primary benchmark for rendering quality and speed, as it has been widely adopted across these categories, ensuring a fair comparison. Quantitative metrics (including PSNR, SSIM [47], LPIPS[57], training and rendering speed) are drawn directly from the respective papers, while qualitative results are generated using officially released implementations unless otherwise specified. We also present an ablation study to evaluate the contribution of each proposed component to the rendering quality in Tab. 2.

To assess the compatibility of different methods with downstream applications, we also evaluate their support within conventional graphics engines and various practical use cases. Compatibility is marked as ”possible” when explicit support is not reported in the original paper but appears feasible based on our knowledge of the method’s


Ground Truth
InstantNGP
3DGS
Neral Rendering
+ High Quality - Low Compatibility
- Limited FPS - Low Editability
SuGaR
Mesh + Neural Texture
+ High Quality - Low Compatibility
- Limited FPS - Limited Editability
NeRF2Mesh
2-Stage Baking
+ High FPS + High Compatibility
- Oversmooth - Loss of Detail
DiffSDF
Object Mesh Optim.
+ High FPS + High Compatibility
- Need Topology - Fail Scene Level
NVDiffRec
Grid-Based Mesh Optim.
+ High FPS + High Compatibility
+ w/o Topology - Limited Grid Res.
OTO (Ours)
End2End Triangle Optim.
+ High Quality + High FPS
+ High Compatibility + w/o Topology
+ Support Scene Level
Figure 6. A qualitative comparison of novel view synthesis results across different method categories on the MipNeRF360 dataset [2].

structure. We also demonstrate a few application visualizations in Fig. 7.

Compare with Neural Rendering Methods We select Plenoxel [55], Instant NGP (iNGP) [32], MipNeRF [1], and 3D Gaussian Splatting (3DGS)[19] as the baselines in the neural rendering category. Plenoxel and iNGP are optimized versions of NeRF [30] that prioritize faster training and rendering, while MipNeRF improves NeRF's quality at the cost of slower training and rendering. Meanwhile, 3DGS uses 3D Gaussian primitives to achieve both high-quality and efficient training and rendering. As shown in Tab. 1, our OTO method achieves rendering quality comparable to iNGP and slightly lower than MipNeRF and 3DGS, while providing a significant speedup, with rendering acceleration from $9.4\times$ to $20,000\times$.

All methods in this category rely on custom representations, making them incompatible with conventional graphics engines. This limitation constrains their rendering speed and makes downstream applications challenging. Although some specialized approaches have been developed for neural representations, such as relighting [42] and physics sim-

		MipNeRF360			Speed			Engine Supported	Applications Supported			
		PSNR$\uparrow$	SSIM$\uparrow$	LPIPS$\downarrow$	Training$\downarrow$	FPS(PC)$\uparrow$	FPS(Mobile)$\uparrow$		Editing	Relighting	Physics	Particle
Neural Rendering	Plenoxel[55]	23.08	0.626	0.463	25 mins	6.79	-	No	no	no	no	no
	iNGP[32]	25.59	0.699	0.331	7 mins	9.43	-	No	no	no	no	no
	MipNeRF[1]	**27.69**	0.792	0.237	48 hours	0.06	-	No	no	no	no	no
	3DGS[19]	27.21	**0.815**	**0.214**	41 mins	134	-	No	no	no	no	no
Mesh Geometry + Neural Texture	MobileNeRF[10]	21.95	0.470	0.470	-	279	22.2	WebGL(Custom Script)	no	no	no	no
	DLM[22]	**27.54**	**0.843**	**0.212**	28 mins	-	-	No	yes	no	no	no
	SuGaR[15]	27.27	0.820	0.253	85 mins	-	-	Blender(Custom Add-on)	yes	no	no	no
Two Stage Baking/ Distillation	NeRF2Mesh[41]	22.74	0.523	0.457	41 mins	-	-	Possible	yes	yes	possible	possible
	NeuManifold[48]	24.53	0.666	0.355	-	93	-	Possible	yes	possible	yes	possible
	BakedSDF[54]	**24.76**	**0.710**	0.303	-	72	-	Possible	yes	yes	possible	possible
	LTM [11]	24.66	0.695	**0.290**	-	212	-	Possible	yes	yes	possible	possible
End-to-end Mesh/ Triangle Optim.	Diff. SDF Render[45]	3.46	0.227	0.707	41 mins	-	-	Possible	yes	yes	possible	possible
	NVDiffRec[33]	20.07	0.477	0.560	27 mins	-	-	Possible	yes	yes	yes	possible
	OTO (Ours)	**25.32**	**0.702**	**0.295**	56 mins	1261	58.7	All (OpenGL, WebGL, Unity, Unreal)	yes	yes	yes	yes

Table 1. A quantitative evaluation of rendering quality, speed, engine compatibility, and native support for downstream applications across different method categories. Compatibility is marked as ”Possible” if not explicitly reported in the respective paper but appears feasible based on our understanding of the method’s structure.

ulation [50], these are niche solutions requiring custom code and environments. In contrast, our OTO method is compatible with most existing graphics engines, allowing it to be used seamlessly in downstream applications without custom implementations. This high compatibility enables OTO to integrate easily into established production pipelines, such as those in games and films.

Compare with Mesh Neural Texture Methods Some works use a mesh as their geometry representation and attach a neural texture, rendered with custom scripts. For instance, MobileNeRF [10] uses neural features as textures, which are later decoded by small multi-layer perceptrons (MLPs) during rendering. In contrast, DLM [22] and SuGaR [15] attach 3D Gaussians to the mesh surface, which are rendered using the standard 3DGS pipeline. As shown in Tab. 1, our OTO method achieves significantly higher rendering quality and speed (on both PC and mobile platforms) compared to MobileNeRF. While DLM and SuGaR achieve slightly higher rendering quality than OTO due to their reliance on the 3DGS pipeline, their rendering speed is similarly constrained by the limitations of 3DGS.

None of these methods offer native support for conventional graphics engines. MobileNeRF and SuGaR partially address this by providing custom scripts for rendering in WebGL and Blender, respectively, but their solutions are mostly limited to rendering only. While DLM and SuGaR support shape editing and animation due to their mesh-based geometry, they still fall short in other downstream applications. Since these methods rely on the same rendering pipeline as NeRF and 3DGS, they inherit the same compatibility limitations. In contrast, our OTO method produces opaque triangles, offering native compatibility with all conventional graphics engines and supporting a broad range of downstream applications.

Compare with Two-Stage Baking Methods Two-stage methods begin by training a neural 3D representation, which is then baked or distilled into a mesh format for rendering and downstream applications. Unlike the previous two categories, the final mesh produced by these two-stage methods is typically compatible with conventional graphics engines and downstream applications. However, the baking process often results in a loss of details. As shown in Fig. 2 and Fig. 6, delicate structures, such as grass and leaves, appear blurry in NeRF2Mesh [41], whereas our end-to-end OTO approach retains much of this fine detail. This preservation is partly due to OTO’s flexibility, as it is not restricted by a fixed topology, making it easier to optimize geometry to capture small, intricate structures. Our method’s quality advantage is further reflected in Tab. 1, where OTO surpasses all two-stage baselines. Additionally, OTO demonstrates the feasibility of end-to-end triangle optimization for scene-scale novel view synthesis, avoiding the need to bake an intermediate neural representation.

Compare with End-to-End Mesh Optimization End-to-end mesh optimization and inverse rendering primarily focus on object-level reconstruction. These methods are closely related to our proposed approach in terms of rendering and optimization pipelines, but they lack the flexibility of OTO, which supports complex scenes. To compare the performance, we adapted the official code for DiffSDF [45] and NVDiffRec [33] on the MipNeRF-360 [2] dataset.

DiffSDF is a classic differentiable rendering method that initializes with a cubic spherical mesh and iteratively updates the geometry and texture while keeping the topology fixed. However, as shown in Fig. 6, it struggles to adapt this simple mesh to complex scene structures, even with prolonged optimization, likely due to the significant topology changes required. NVDiffRec[33], a more recent approach, leverages differentiable marching tetrahedra, enabling optimization without a fixed topology. This allows it to fit larger scenes, as illustrated in Fig. 6. However, its dense 3D grid structure limits scalability to high resolutions, making it challenging to achieve fine details across the entire scene.

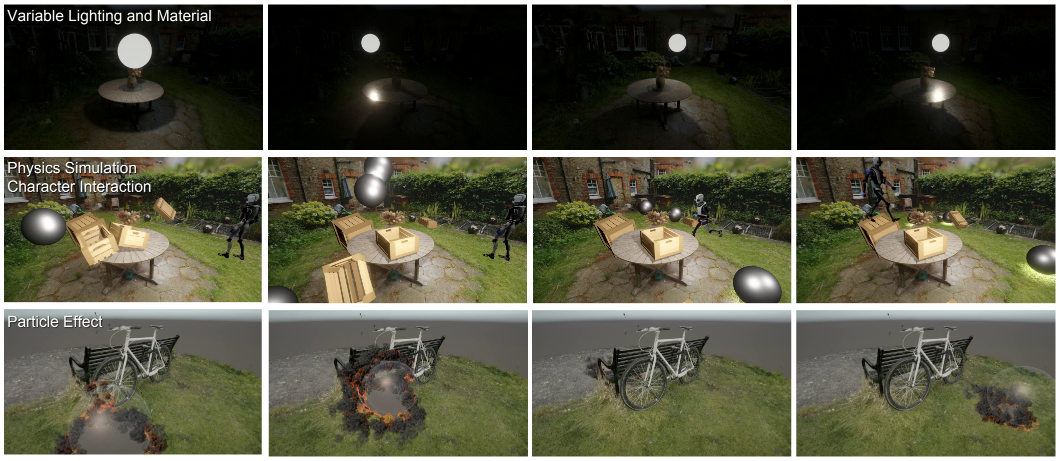


Figure 7. Examples of downstream application scenarios using our reconstructed model, with native support in the Unity engine[43].

	PSNR↑	SSIM↑	LPIPS↓
OTO Full Model	**25.32**	**0.702**	**0.295**
w/o 2D SDF Approx.	17.41	0.241	0.610
w/o 2-Layer Occlusion Approx	16.02	0.276	0.609
w/o SH colors	24.90	0.682	0.317
w/o Triangle Split	24.51	0.651	0.374
w/o Triangle Pruning	24.68	0.669	0.338
w/o Triangle Insertion	24.96	0.690	0.312

Table 2. An ablation study for novel view synthesis quality on MipNeRF360 dataset[2].

In contrast, our OTO method, producing disjoint opaque triangles, allows for high-detail fitting without wasting primitives in empty regions. As demonstrated in Tab. 1, existing end-to-end methods offer similar compatibility with conventional engines as OTO but fall short in rendering quality compared to our approach.

Downstream Applications of OTO To showcase the compatibility of our method with conventional graphics engines and related applications, we implemented several scenarios using our exported model in Unity[43], as shown in Fig. 7. For a more detailed visualization, please refer to the supplementary videos. The first row demonstrates relighting and material editing capabilities by adjusting the light source position and the material's metallic parameters, with realistic shadows generated automatically. The second row illustrates physical simulation capabilities, including object-object and object-scene collisions. We also added a controllable character, allowing users to walk and jump in the scene freely. The third row presents particle effects, one of the most commonly used special effects, by setting the model on fire wherever the effect ball is located. These experiments highlight the ease of integrating our model into diverse applications; each scenario required minimal effort, with setup times of less than an hour for an experienced game engine user.

5. Limitations

The 2D nature of triangle primitives in our OTO method leads to limited 3D view consistency. Unlike volumetric representations, such as NeRF [36] and 3DGS [19], small 2D triangles may become nearly invisible when viewed from angles parallel to their surfaces. We observe that triangles can sometimes hide themselves to mimic view-dependent effects, such as reflections. Distant regions with fewer training view angles have lower quality compared to central regions with more diverse training views.

Additionally, the geometry of our presentation is not explicitly constrained to maximize the novel view synthesis performance. Although this is not a serious problem for most applications, as shown in the supplementary video and Fig. 7, high-fidelity physics simulation and relighting might be affected by the unconstrained surface normal. Supervision from normal estimation can mitigate this issue, but it is beyond the novel view synthesis scope of this paper.

6. Conclusion

We introduced OTO, an end-to-end scene-level novel view synthesis method that is natively compatible with conventional graphics engines and downstream applications. To address the challenges of the non-differentiable rasterization pipeline, we employed 2D SDF and two-layer occlusion approximations. We implemented coverage-based density control to achieve high-detail, complete reconstructions. Our results demonstrates significantly faster rendering speeds and superior compatibility compared to various novel view synthesis methods, while preserving fine details.

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

References

[1] Jonathan T. Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P. Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. *ICCV*, 2021. 1, 2, 6, 7

[2] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. *CVPR*, 2022. 5, 6, 7, 8

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

[4] Ang Cao and Justin Johnson. Hexplane: A fast representation for dynamic scenes. *CVPR*, 2023. 2

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

[6] Hanlin Chen, Chen Li, Mengqi Guo, Zhiwen Yan, and Gim Hee Lee. Gnesf: Generalizable neural semantic fields. *Advances in Neural Information Processing Systems*, 36: 36553–36565, 2023. 2

[7] Hanlin Chen, Chen Li, and Gim Hee Lee. Neusg: Neural implicit surface reconstruction with 3d gaussian splatting guidance. *arXiv preprint arXiv:2312.00846*, 2023. 2

[8] Wenzheng Chen, Huan Ling, Jun Gao, Edward Smith, Jaakko Lehtinen, Alec Jacobson, and Sanja Fidler. Learning to predict 3d objects with an interpolation-based differentiable renderer. *Advances in neural information processing systems*, 32, 2019. 2, 3

[9] Wenzheng Chen, Joey Litalien, Jun Gao, Zian Wang, Clement Fuji Tsang, Sameh Khalis, Or Litany, and Sanja Fidler. DIB-R++: Learning to predict lighting and material with a hybrid differentiable renderer. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. 2, 3

[10] Zhiqin Chen, Thomas Funkhouser, Peter Hedman, and Andrea Tagliasacchi. Mobilenerf: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures. In *The Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023. 2, 7

[11] Jaehoon Choi, Rajvi Shah, Qinbo Li, Yipeng Wang, Ayush Saraf, Changil Kim, Jia-Bin Huang Dinesh Manocha, Suhib Alsisan, and Johannes Kopf. Ltm: Lightweight textured mesh extraction and refinement of large unbounded scenes for efficient storage and real-time rendering. In *CVPR*, 2024. 2, 7

[12] Zhiwen Fan, Kevin Wang, Kairun Wen, Zehao Zhu, Dejia Xu, and Zhangyang Wang. Lightgaussian: Unbounded 3d gaussian compression with 15x reduction and 200+ fps, 2023. 2

[13] Jiemin Fang, Taoran Yi, Xinggang Wang, Lingxi Xie, Xiaopeng Zhang, Wenyu Liu, Matthias Nießner, and Qi Tian. Fast dynamic radiance fields with time-aware neural voxels. In *SIGGRAPH Asia 2022 Conference Papers*, 2022. 2

[14] Jun Gao, Wenzheng Chen, Tommy Xiang, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Learning deformable tetrahedral meshes for 3d reconstruction. *Advances in neural information processing systems*, 33:9936–9947, 2020. 2, 3

[15] Antoine Guédon and Vincent Lepetit. Sugar: Surface-aligned gaussian splatting for efficient 3d mesh reconstruction and high-quality mesh rendering. *CVPR*, 2024. 2, 7

[16] Yuan-Chen Guo, Di Kang, Linchao Bao, Yu He, and Song-Hai Zhang. Nerfren: Neural radiance fields with reflections. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 18409–18418, 2022. 2

[17] Wenbo Hu, Yuling Wang, Lin Ma, Bangbang Yang, Lin Gao, Xiao Liu, and Yuewen Ma. Tri-miprf: Tri-mip representation for efficient anti-aliasing neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 19774–19783, 2023. 2

[18] Krishna Murthy Jatavallabhula, Edward Smith, Jean-Francois Lafleche, Clement Fuji Tsang, Artem Rozantsev, Wenzheng Chen, Tommy Xiang, Rev Lebaredian, and Sanja Fidler. Kaolin: A pytorch library for accelerating 3d deep learning research. *arXiv preprint arXiv:1911.05063*, 2019. 2, 3

[19] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42 (4), 2023. 1, 2, 3, 5, 6, 7, 8

[20] Ruilong Li, Hang Gao, Matthew Tancik, and Angjoo Kanazawa. Nerfacc: Efficient sampling accelerates nerfs. *arXiv preprint arXiv:2305.04966*, 2023. 2

[21] Yiyi Liao, Simon Donné, and Andreas Geiger. Deep marching cubes: Learning explicit surface representations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 2, 3

[22] Ancheng Lin and Jun Li. Direct learning of mesh and appearance via 3d gaussian splatting. *arXiv preprint arXiv:2405.06945*, 2024. 2, 7

[23] Jeffrey Yunfan Liu, Yun Chen, Ze Yang, Jingkang Wang, Sivabalan Manivasagam, and Raquel Urtasun. Neural scene rasterization for large scene rendering in real time. In *The IEEE International Conference on Computer Vision (ICCV)*, 2023. 2

[24] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. *NeurIPS*, 2020. 2

[25] Shichen Liu, Weikai Chen, Tianye Li, and Hao Li. Soft rasterizer: A differentiable renderer for image-based 3d reasoning. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 7707–7716, 2019. 2, 3

[26] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169, 1987. 2

[27] Tao Lu, Mulin Yu, Linning Xu, Yuanbo Xiangli, Limin Wang, Dahua Lin, and Bo Dai. Scaffold-gs: Structured 3d gaussians for view-adaptive rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20654–20664, 2024. 2

[28] Zhicheng Lu, Xiang Guo, Le Hui, Tianrui Chen, Ming Yang, Xiao Tang, Feng Zhu, and Yuchao Dai. 3d geometry-aware

deformable gaussian splatting for dynamic view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024. 2

[29] Jonathon Luiten, Georgios Kopanas, Bastian Leibe, and Deva Ramanan. Dynamic 3d gaussians: Tracking by persistent dynamic view synthesis. In *3DV*, 2024. 2

[30] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020. 1, 6

[31] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021. 2

[32] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, 2022. 1, 2, 6, 7

[33] Jacob Munkberg, Jon Hasselgren, Tianchang Shen, Jun Gao, Wenzheng Chen, Alex Evans, Thomas Müller, and Sanja Fidler. Extracting Triangular 3D Models, Materials, and Lighting From Images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8280–8290, 2022. 7

[34] Keunhong Park, Utkarsh Sinha, Jonathan T Barron, Sofien Bouaziz, Dan B Goldman, Steven M Seitz, and Ricardo Martin-Brualla. Nerfies: Deformable neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5865–5874, 2021. 2

[35] Keunhong Park, Utkarsh Sinha, Peter Hedman, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Ricardo Martin-Brualla, and Steven M. Seitz. Hypernerf: A higher-dimensional representation for topologically varying neural radiance fields. *ACM Trans. Graph.*, 40(6), 2021.

[36] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-nerf: Neural radiance fields for dynamic scenes. *arXiv preprint arXiv:2011.13961*, 2020. 2, 8

[37] Marie-Julie Rakotosaona, Fabian Manhardt, Diego Martin Arroyo, Michael Niemeyer, Abhijit Kundu, and Federico Tombari. Nerfmeshing: Distilling neural radiance fields into geometrically-accurate 3d meshes. In *2024 International Conference on 3D Vision (3DV)*, pages 1156–1165. IEEE, 2024. 2

[38] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 14335–14345, 2021. 2

[39] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 3

[40] Tianchang Shen, Jun Gao, Kangxue Yin, Ming-Yu Liu, and Sanja Fidler. Deep marching tetrahedra: a hybrid representation for high-resolution 3d shape synthesis. In *Advances in Neural Information Processing Systems*, pages 6087–6101. Curran Associates, Inc., 2021. 2, 3

[41] Jiaxiang Tang, Hang Zhou, Xiaokang Chen, Tianshu Hu, Errui Ding, Jingdong Wang, and Gang Zeng. Delicate textured mesh recovery from nerf via adaptive surface refinement. *arXiv preprint arXiv:2303.02091*, 2022. 1, 2, 7

[42] Marco Toschi, Riccardo De Matteo, Riccardo Spezialetti, Daniele De Gregorio, Luigi Di Stefano, and Samuele Salti. Relight my nerf: A dataset for novel view synthesis and relighting of real world objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20762–20772, 2023. 6

[43] Unity Technologies. Unity, 2023. Game development platform. 8

[44] Dor Verbin, Peter Hedman, Ben Mildenhall, Todd Zickler, Jonathan T. Barron, and Pratul P. Srinivasan. Ref-NeRF: Structured view-dependent appearance for neural radiance fields. *CVPR*, 2022. 2

[45] Delio Vicini, Sébastien Speierer, and Wenzel Jakob. Differentiable signed distance function rendering. *Transactions on Graphics (Proceedings of SIGGRAPH)*, 41(4):125:1–125:18, 2022. 3, 7

[46] Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. *NeurIPS*, 2021. 2

[47] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4): 600–612, 2004. 5

[48] Xinyue Wei, Fanbo Xiang, Sai Bi, Anpei Chen, Kalyan Sunkavalli, Zexiang Xu, and Hao Su. Neumanifold: Neural watertight manifold reconstruction with efficient and high-quality rendering support. *arXiv preprint arXiv:2305.17134*, 2023. 7

[49] Guanjun Wu, Taoran Yi, Jiemin Fang, Lingxi Xie, Xiaopeng Zhang, Wei Wei, Wenyu Liu, Qi Tian, and Xinggang Wang. 4d gaussian splatting for real-time dynamic scene rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 20310–20320, 2024. 2

[50] Tianyi Xie, Zeshun Zong, Yuxing Qiu, Xuan Li, Yutao Feng, Yin Yang, and Chenfanfu Jiang. Physgaussian: Physics-integrated 3d gaussians for generative dynamics. *arXiv preprint arXiv:2311.12198*, 2023. 7

[51] Zhiwen Yan, Chen Li, and Gim Hee Lee. Nerf-ds: Neural radiance fields for dynamic specular objects. *arXiv preprint arXiv:2303.14435*, 2023. 2

[52] Zhiwen Yan, Weng Fei Low, Yu Chen, and Gim Hee Lee. Multi-scale 3d gaussian splatting for anti-aliased rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 20923–20931, 2024. 2

[53] Lior Yariv, Jiatao Gu, Yoni Kasten, and Yaron Lipman. Volume rendering of neural implicit surfaces. In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021. 2

[54] 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. *arXiv*, 2023. 1, 7

[55] Alex Yu, Sara Fridovich-Keil, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. *arXiv preprint arXiv:2112.05131*, 2021. 2, 6, 7

[56] Zehao Yu, Anpei Chen, Binbin Huang, Torsten Sattler, and Andreas Geiger. Mip-splatting: Alias-free 3d gaussian splatting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 19447–19456, 2024. 2

[57] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 586–595, 2018. 5

[58] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018. 3