

DiffSoup: Direct Differentiable Rasterization of Triangle Soup for Extreme Radiance Field Simplification

Kenji Tojo¹ Bernd Bickel² Nobuyuki Umetani¹

¹The University of Tokyo ²ETH Zürich

knjtojo@g.ecc.u-tokyo.ac.jp bickelb@ethz.ch n.umetani@gmail.com

Abstract

Radiance field reconstruction aims to recover high-quality 3D representations from multi-view RGB images. Recent advances, such as 3D Gaussian splatting, enable real-time rendering with high visual fidelity on sufficiently powerful graphics hardware. However, efficient online transmission and rendering across diverse platforms requires drastic model simplification, reducing the number of primitives by several orders of magnitude. We introduce DiffSoup, a radiance field representation that employs a soup (i.e., a highly unstructured set) of a small number of triangles with neural textures and binary opacity. We show that this binary opacity representation is directly differentiable via stochastic opacity masking, enabling stable training without a mollifier (i.e., smooth rasterization). DiffSoup can be rasterized using standard depth testing, enabling seamless integration into traditional graphics pipelines and interactive rendering on consumer-grade laptops and mobile devices. Code is available at <https://github.com/kenji-tojo/diffsoup>.

1. Introduction

Realistic and efficient 3D representations are essential for a wide range of vision and graphics applications, including virtual and augmented reality, digital content creation, and robotic perception. However, it remains challenging to represent complex real-world objects such as foliage, clothing, hair, or architectural details across diverse platforms, including mobile and VR devices. Typically, the number of geometric primitives (e.g., triangles) is tightly limited for interactive rendering and smooth data transfer.

Neural radiance fields (NeRFs) [36] have demonstrated high-fidelity 3D reconstruction from multi-view RGB images via differentiable volume rendering [22]. More recently, 3D Gaussian Splatting (3DGS) [24] models the radiance field as point-based primitives, enabling real-time rendering and flexible scene editing given sufficiently pow-

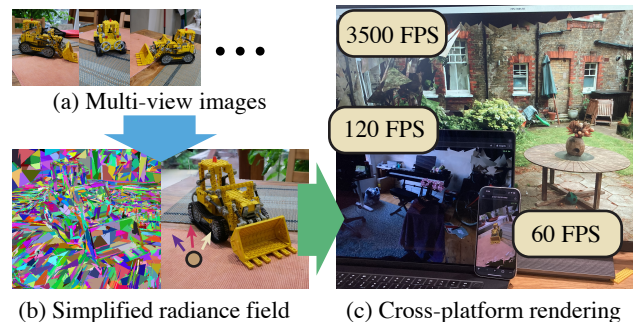


Figure 1. (a) Given multi-view RGB images, (b) we reconstruct a simplified radiance field as a textured triangle soup using differentiable rasterization. (c) The resulting scene can be rendered within a standard depth-tested rasterization pipeline, enabling seamless visualization across devices—including a high-end desktop with a dedicated GPU, a consumer-grade laptop (MacBook Pro with an M3 Pro chip), and even a smartphone (iPhone 15).

erful graphics hardware. Although these approaches—and many subsequent variants—achieve remarkable view synthesis results, they typically rely on an enormous number of primitives, often *orders of magnitude* more than the number feasible on restricted platforms and devices.

Traditional graphics pipelines are highly optimized for rendering opaque triangles with texture maps. However, opaque triangles introduce difficulties in training: unlike volumetric primitives, they fully occlude surfaces behind them and produce discontinuous color transitions at silhouettes, both of which impede smooth gradient propagation. Consequently, differentiable rasterization methods for opaque triangles [28, 41] are still not widely adopted in radiance field reconstruction pipelines.

In this paper, we address the challenging problem of *extreme model simplification* [8] for radiance field-based view synthesis. Our goal is to reconstruct a 3D scene from multi-view RGB images using a small unstructured collection (or *soup*) of opaque triangles equipped with compact neural textures that have binary opacity. The resulting representation seamlessly integrates into traditional real-time render-

ing pipelines while preserving the optimization robustness and reconstruction fidelity typically associated with volumetric approaches.

To achieve this, we introduce *DiffSoup*, a differentiable rasterization method for directly learning an opaque triangle soup. Unlike previous approaches, DiffSoup does not rely on a *molifier* (i.e., smoothed rasterization) during optimization, which often suppresses high-frequency appearance details or requires careful scheduling of blur strength [17, 32]. Instead, inspired by recent work on implicit surface reconstruction [62], we differentiate opaque triangle rasterization using *stochastic opacity masking*, which yields robust gradient signals under severe occlusion while producing the *exact* pixel color. To support this, DiffSoup further extends the silhouette-differentiation method of [41] to handle the *implicit* discontinuities introduced by opacity masking.

In summary, our contributions are as follows.

- We introduce the *extreme radiance field simplification* problem, aiming to represent a 3D scene using an unstructured soup of a small number of textured triangles.
- We present *DiffSoup*, a method for directly differentiating opaque triangle rasterization via *stochastic opacity masking*, yielding robust gradients even under severe occlusion and frequent visibility discontinuities.
- We demonstrate that DiffSoup reconstructs high-fidelity scenes using typically fewer than 20,000 triangles—where existing methods rely on millions of primitives—producing more detail-preserving novel views than baselines, while running interactively on consumer-grade laptops and mobile phones (Figure 1).

2. Related Work

Differentiable rendering A large body of work reconstructs 3D scenes from images via differentiable rendering (see survey [23]). These methods optimize geometry, texture, and material parameters using gradients from rasterization [15, 28, 32, 33, 42], light transport simulation [29, 34, 39, 61], implicit surface rendering [1, 30, 35, 47, 48, 57, 62], multi-layer transparency models [10, 15], point splatting [24, 58], and more. Our work focuses on opaque triangle rasterization with a single depth-tested layer, the dominant rendering method in real-time applications such as games, VR/AR, and mobile graphics.

Differentiable triangle rasterization Computing gradients for opaque, depth-tested rasterization is challenging due to occlusion and the discontinuities that arise at object silhouettes. Soft-rasterization methods introduce customized edge smoothing to enforce differentiability [17, 19, 32], while other approaches derive gradients directly from anti-aliasing formulations to preserve sharp silhouettes [6, 28, 33, 41]. Although these “blur-free” methods

better preserve high-frequency appearance details, their gradients are often unstable and non-smooth, leading to sub-optimal reconstruction [38]. To address this, we draw inspiration from the *stochastic surface* interpretation developed for implicit surface reconstruction [62] and inverse light transport [61]. Specifically, we extend differentiable rasterizers [28, 41] with *stochastic opacity masking*, which yields robust gradients for optimizing binary opacity under occlusion while still differentiating the exact pixel colors produced by opaque rasterization.

Neural radiance fields Radiance-field reconstruction methods such as NeRF and its extensions [2, 3, 11, 36, 37, 50] recover high-quality 3D representations from multi-view RGB images. Despite their strong view-synthesis quality, integrating these representations into traditional real-time rendering pipelines remains difficult due to costly volumetric ray marching [22]. MobileNeRF [7] converts a radiance field into a mesh-based representation but does not directly differentiate the final opaque mesh, relying instead on NeRF optimization as a separate intermediate stage. This limits control over polygon count and final geometry, often producing excessive triangles and losing high-frequency appearance details. In contrast, we extend direct differentiable rasterization for polygonal asset creation [15, 28, 41] to radiance-field reconstruction, enabling significantly more compact, detail-preserving results with shorter training.

3D Gaussian splatting The 3DGS method uses explicit 3D Gaussian primitives for real-time radiance-field rendering [24]. To capture more accurate geometry and sharper details, follow-up works explore alternative primitives such as 2D Gaussians [20], convex hulls [18], triangles [17], and textured 2D [43, 44, 52, 55, 56] or 3D [5, 56] primitives. However, these approaches often rely on excessive primitive counts or introduce custom translucency, both of which limit practicality on consumer-grade devices such as laptops, mobile phones, and VR headsets. Concurrent work [19] proposes a scheduling scheme for the smoothing function in triangle rasterization and shows that scenes can converge to opaque triangles. Their method, however, does not optimize sub-triangle appearance using opacity and texture, and therefore requires many triangles to represent fine details. In contrast, we directly optimize an opaque triangle soup with a neural texture carrying binary opacity, enabling high-frequency details with a drastically smaller number of primitives. StochasticSplats [26] applies a stochastic differentiation strategy to optimize the *translucent* appearance of 3DGS using multiple random samples; we instead target the *fully opaque* appearance of triangle rasterization using a single depth-tested stochastic rasterization.

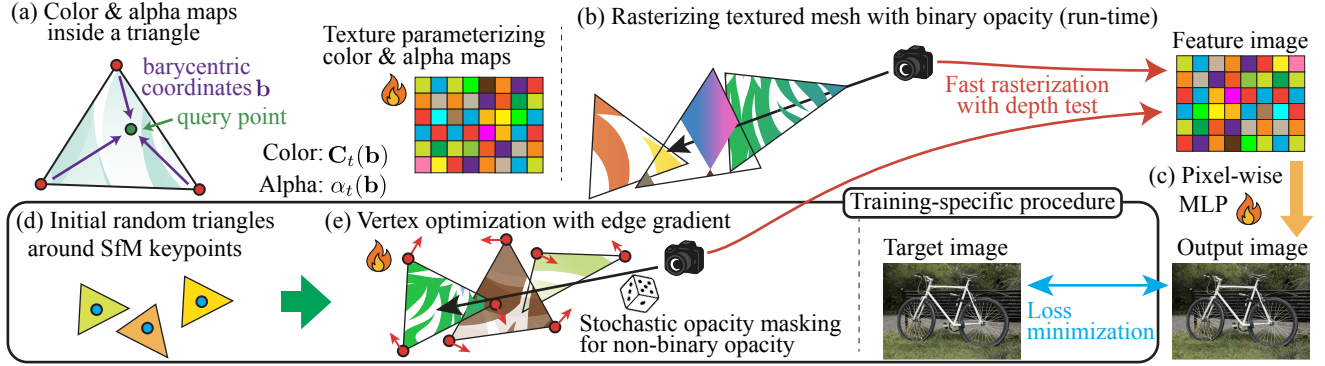


Figure 2. Overview of our algorithm. (a) We represent radiance fields as a textured triangle soup, where barycentric coordinates map to learnable color and alpha textures. (b) At run time, this representation is rendered with binary opacity, enabling efficient rasterization via standard depth testing. (c) Rasterization produces a high-dimensional color-feature image, which is then mapped to RGB values using a small shared MLP trained to minimize the error between the rendering and the ground-truth image. (d) During training, we initialize triangles around structure-from-motion (SfM) keypoints. (e) We optimize opacity through stochastic opacity masking, while vertex positions are updated using edge gradients for opaque triangles.

3. Method

Given multi-view RGB images, DiffSoup reconstructs a triangle soup augmented with neural textures and binary opacity. We first outline the necessary background and then describe our differentiable rasterizer, along with the proposed stochastic opacity masking approach. An overview of our algorithm is shown in Figure 2.

3.1. Preliminary

Triangle soup representation To represent a radiance field, we use triangles $\mathcal{T} = \{\mathbf{T}_t\}_{t=1}^{|\mathcal{T}|}$ together with per-triangle texture functions for color $\mathcal{C} = \{\mathbf{C}_t\}_{t=1}^{|\mathcal{T}|}$ and opacity $\mathcal{A} = \{\alpha_t\}_{t=1}^{|\mathcal{T}|}$. Each triangle $\mathbf{T}_t$ consists of three vertices $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3 \in \mathbb{R}^3$. Given an appropriate parameterization, the color texture $\mathbf{C}_t$ maps barycentric coordinates $\mathbf{b} \in \mathbb{R}^2$ to the color $\mathbf{C}_t(\mathbf{b})$ at the corresponding point on the triangle, while opacity is defined by a scalar function $\alpha_t(\mathbf{b}) \in [0, 1]$.

Depth-tested rasterization with binary opacity Unlike previous approaches that rely on translucent primitives, DiffSoup employs opaque triangle rasterization with binary opacity, enabling seamless integration into traditional graphics pipelines. Given triangles transformed into camera space, we perform standard rasterization to produce an array of fragments $\mathcal{F} = \{\mathbf{f}_f\}_f^{|\mathcal{F}|}$, defined for each individual pixel. Fragments are generated for all triangles intersecting the pixel and encode the pixel-center sample as

$$\mathbf{f}_f = (\mathbf{b}_f, D_f, t_f), \quad (1)$$

where $\mathbf{b}_f$ denotes the barycentric coordinates, D_f is the fragment depth, and t_f is the corresponding triangle index.

Given this information, the pixel color $\hat{\mathbf{C}}$ is computed as

$$\hat{\mathbf{C}} = \mathbf{C}_{t_{f^*}}(\mathbf{b}_{f^*}), \quad (2)$$

where $f^* = \arg \min_f D_f$ is the index of the *foremost* fragment. Hereafter, we write $\mathbf{C}_f$ and α_f as shorthands for $\mathbf{C}_{t_f}$ and α_{t_f} , respectively, to simplify notation.

In practical real-time rendering systems such as games, sub-triangle geometric detail is commonly represented using *binary opacity* stored in a texture. This modifies the selection of the foremost fragment by discarding the fragments whose opacity falls below a threshold:

$$f^* = \arg \min_{f \in \{f; \alpha(\mathbf{b}_f) > 0.5\}} D_f. \quad (3)$$

This enables fine sub-triangle detail using a small number of flat primitives, without adding complexity beyond depth-tested rendering, such as the explicit primitive sorting used in 3DGS [24].

Stochastic surface and radiance field loss Although forward rendering with opaque rasterization and binary opacity is highly efficient, *directly optimizing* such representations remains difficult because occlusion and visibility transitions are discontinuous. Prior work [7] employs a straight-through estimator [4] to approximate gradients with respect to binarized opacity values during NeRF training. However, this strategy exhibits instability in geometry optimization and requires an additional pre-training stage using a standard NeRF.

Recent work has introduced an important advancement for implicit surface reconstruction [35, 61, 62]. In particular, Zhang et al. [62] interpret *every* point in space as an infinitesimal opaque surface occurring with probability equal

to that point’s opacity. This *stochastic surface* interpretation leads to a modified NeRF training loss:

$$\mathcal{L}_{\text{sum}} = \sum_i w_i \mathcal{L}_1(\mathbf{C}_i), \quad \text{where } w_i = \bar{\alpha}_i \alpha_i. \quad (4)$$

Here, $\mathbf{C}_i$ and α_i denote the color and alpha values at the i -th sample point along the ray in depth-sorted order, and $\bar{\alpha}_i = \prod_{j=1}^{i-1} (1 - \alpha_j)$ is the cumulative transmittance. Notably, they evaluate the pixel-wise L1 loss $\mathcal{L}_1$ at the *sampled* points with weights proportional to the volumetric blending weights, rather than computing $\mathcal{L}_1$ on the final blended pixel color as in standard NeRFs. They show that this simple, theoretically grounded loss converges to binary opacity and exactly reproduces pixel colors in opaque rendering, without heuristics such as smoothness scheduling or pre-training.

However, their method seeks a potential surface via sampling in *continuous* space, which is incompatible with the *inherently discrete* rasterization pipeline, where fragments arise from explicit primitives and depth sorting introduces substantial complexity. Moreover, their formulation omits gradients with respect to the motion of explicit primitives; as a result, when applied within a triangle-based rasterization pipeline, it yields markedly suboptimal geometry.

3.2. The DiffSoup differentiable rasterizer

Our goal is to inversely optimize the previously introduced triangle-soup representation via depth-tested rasterization, given ground-truth multi-view RGB images. To this end, we build on differentiable rasterization methods for opaque triangles [28, 41], which derive gradients by differentiating an anti-aliasing model. While these methods provide a solid foundation for differentiating visibility discontinuities while preserving sharp edges, they struggle to optimize binary opacity and to handle *implicit* visibility discontinuities—those not associated with explicit triangle edges—introduced by opacity masking.

Stochastic opacity masking To derive a differentiable optimization method, we extend the stochastic surface approach [62] to our differentiable triangle rasterizer. However, a standard object-order rasterizer does not produce depth-sorted fragments per pixel, so the radiance field loss (4) cannot be directly evaluated. We therefore introduce *stochastic opacity masking*, a principled mechanism that realizes the stochastic surface process entirely within a conventional depth-tested rasterization pipeline.

The key idea is to *simulate* the stochastic surface as a *discrete* stochastic process during fragment generation (Figure 3). Algorithmically, DiffSoup achieves this by introducing stochastic opacity thresholds into (3) as

$$f_{\tau}^* = \arg \min_{\bar{f} \in \{f \in \mathcal{F} \mid \alpha(\mathbf{b}_f) > \tau_f\}} D_{\bar{f}}, \quad (5)$$

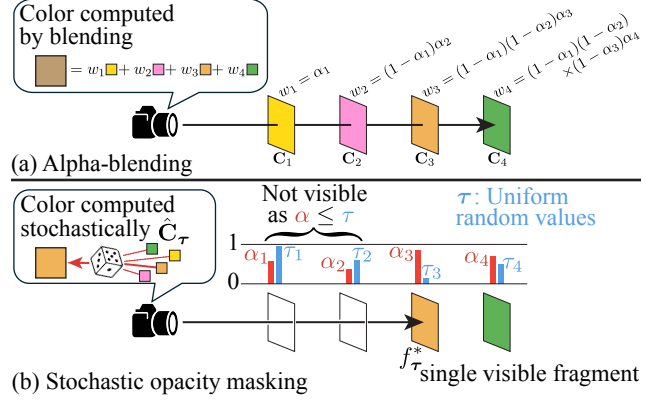


Figure 3. (a) Alpha blending computes cumulative transmittance over depth-sorted samples and produces the final color as a weighted average. (b) Our stochastic opacity masking, in contrast, simulates a discrete stochastic process over the surface, allowing the renderer to output the exact color of the selected fragment.

where $\tau = (\tau_1, \dots, \tau_{|\mathcal{F}|})$ are real numbers sampled independently and uniformly for each fragment in the range $[0, 1]$. This selects a single visible surface point for the pixel, and the resulting color is $\hat{\mathbf{C}}_{\tau} = \mathbf{C}_{f_{\tau}^*}(\mathbf{b}_{f_{\tau}^*})$. Note that the $\arg \min$ in (5) can be efficiently computed using the z-buffer method without depth sorting the fragments.

The crucial observation is that stochastic masking makes the pixel color a *random variable*: high-opacity fragments are more likely to pass the threshold and become visible, though even the most plausible surface may be skipped with a small probability. Consequently, (5) can be interpreted as drawing a single pixel color from the probability distribution induced by the stochastic surface. Specifically, for a scene parameter θ , we can show that the probability that fragment f is selected as the foremost surface is identical to the weight in (4) as

$$p(f; \theta) = w_f, \quad (6)$$

whereas our computation does not require sorting fragments by depth. With this identity, the radiance field loss (4) is computed as the *expected value* of the pixel-wise loss:

$$\mathcal{L}_{\text{exp}} = E_{p(f; \theta)}[\mathcal{L}_1(\hat{\mathbf{C}})], \quad (7)$$

as estimated by our stochastic opacity masking.

Notably, viewing the rasterization as a discrete stochastic process enables unbiased estimation of the gradient of $\mathcal{L}_{\text{exp}}$ using the DiffSoup rasterizer as a sampler. Specifically, we apply the likelihood-ratio gradient identity [14, 53]:

$$\begin{aligned} \partial_{\theta} E_{p(f; \theta)}[\mathcal{L}_1(\hat{\mathbf{C}})] &= E_{p(f; \theta)}[\partial_{\theta} \mathcal{L}_1(\hat{\mathbf{C}})] \\ &\quad + E_{p(f; \theta)}[\mathcal{L}_1(\hat{\mathbf{C}}) \partial_{\theta} \log p(f; \theta)], \end{aligned} \quad (8)$$

where ∂_{θ} denotes the partial derivative with respect to the scene parameter θ . The first term is the standard loss gradi-

ent for the *exact* color of the sampled fragment. The second term propagates gradients with respect to opacity, where the score term $\partial_\theta \log p(f; \theta)$ is weighted by the pixel-wise color loss. For a fragment f' , this score term reduces to the partial derivative with respect to its opacity:

$$\frac{\partial}{\partial \alpha_{f'}} \log p(f; \theta) = \begin{cases} 1/\alpha_f, & f' = f, \\ -1/(1 - \alpha_{f'}), & D_{f'} < D_f, \end{cases} \quad (9)$$

and no gradient is produced for fragments with $D_{f'} > D_f$. Crucially, the opacity gradient for each fragment f' depends only on the sampled pixel fragment f and the fragment itself, yielding an efficient, sort-free gradient estimator that operates entirely within the depth-tested rasterizer (pseudocode in the supplementary material).

Handling visibility discontinuities Stochastic opacity masking yields gradients for binary opacity but not for the *motion* of explicit primitives. The approach of [62], which relies on dense spatial sampling for implicit fields, is incompatible with our primitive-based setting. To obtain motion gradients, we extend the *edge-gradient* formulation of differentiable rasterization [41]. Conventional edge gradients detect visibility boundaries by examining changes in triangle IDs between neighboring pixels. In DiffSoup, however, stochastic opacity masking introduces *intra-triangle* visibility discontinuities that do not alter triangle IDs and therefore remain undetected. To handle these internal discontinuities, we evaluate the rasterized edge-gradient expression for *all* horizontally and vertically adjacent pixel pairs in the stochastic intermediate image when computing the color loss. This naturally generalizes edge gradients to implicit, opacity-induced boundaries. Averaged over training iterations, this procedure produces stable and meaningful motion gradients that align with the effective sub-triangle silhouettes emerging from converged binary opacity, enabling accurate optimization of triangle geometry and placement.

3.3. Parameterization

Learnable multi-resolution triangle texture A straightforward way to represent per-triangle textures $\mathbf{C}_t$ and α_t is to map the triangle soup onto a 2D texture atlas and store texture values directly in image space. However, such an atlas may place unrelated triangles adjacent in texture space, which interferes with the use of *multi-resolution* parameterizations that have proven highly effective in gradient-based optimization, such as multi-level 3D grids [37, 45] and texture mip-mapping [28, 51]. To avoid these issues and to fully leverage multi-resolution optimization, we adapt these parameterizations to operate directly on our triangle soup.

Specifically, we extend the “mesh color” approach [59, 60] for atlas-free texturing to support *learnable* feature vectors. At each resolution level R , we place learnable features

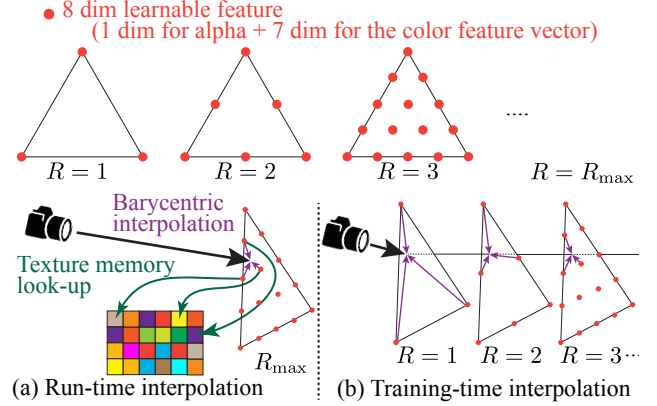


Figure 4. Multi-resolution triangle texture. Top: We parameterize the texture using multiple grid resolutions, corresponding to different triangle subdivision levels. Bottom: (a) At run time, the renderer queries only the highest-resolution triangle texture. (b) At training time, this highest-resolution texture is over-parameterized with lower-resolution grids to improve optimization stability.

at the vertices of a recursively subdivided triangle grid: the base level $R = 0$ corresponds to the original triangle vertices, and each subsequent level R refines the previous one, yielding $(2^{R-1} + 1)(2^R + 1)$ feature vertices (Figure 4). These multi-resolution per-triangle features are queried via barycentric interpolation within the micro-triangle containing the evaluation point.

To enable a multi-resolution representation of the texture at target level $R_{\max}$, we maintain learnable feature grids for $R_{\min}, \dots, R_{\max} - 1$. All lower-resolution grids are queried and accumulated on the finest grid before rendering, yielding an *overparameterized* high-resolution texture field, similar in spirit to [51] but applied to a triangular grid rather than a texture image. A sigmoid activation is applied to constrain the resulting features to $[0, 1]$ prior to rasterization. This strategy preserves the benefits of multi-resolution optimization while avoiding multiple per-level texture lookups during rendering. After training, only the finest-resolution texture (after sigmoid activation) is retained and stored as 8-bit PNG images.

Neural deferred shading To model view-dependent appearance without increasing texture storage or relying on high-order bases (e.g., spherical harmonics), we employ neural deferred shading [46, 54] within our binary-opacity rasterization framework. Rather than storing RGB values directly, each triangle carries an N_{feat} -dimensional learnable feature vector in its color texture, which is rasterized to yield per-pixel feature maps. A lightweight shared MLP is then applied per pixel, taking as input the concatenation of the rasterized feature vector and the viewing direction, and producing the final shaded color. Opacity, on the con-

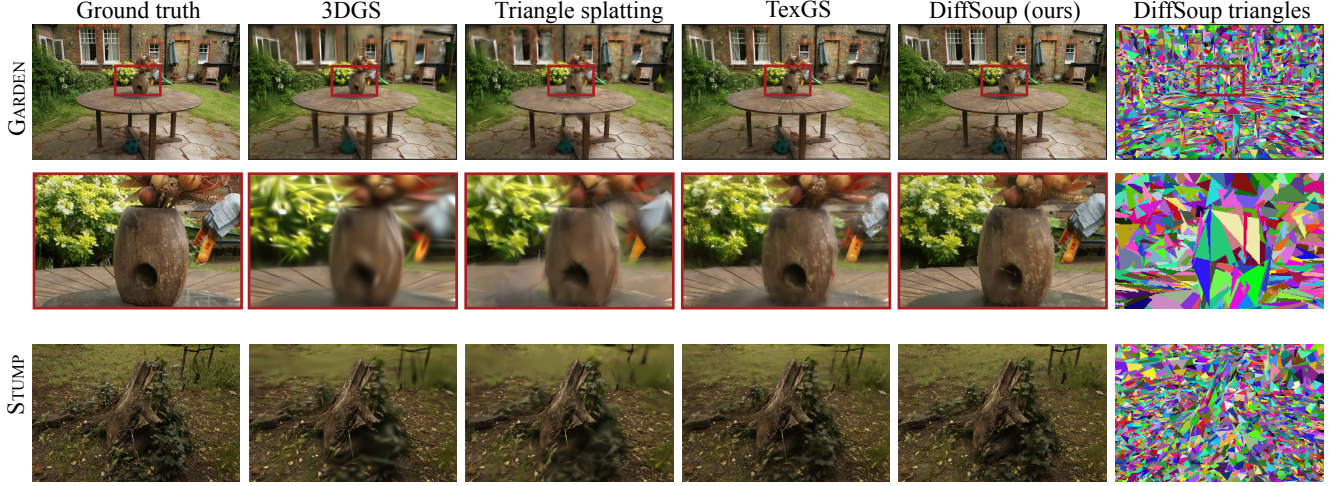


Figure 5. Qualitative results on the *MipNeRF360* [3] dataset. (Left to right) Ground-truth test images; views synthesized by 3DGS [24], Triangle Splatting [17], TexGS [5], our method; and a visualization of our reconstructed triangle geometry. Rows correspond to different scenes, and all models use 15K primitives.

trary, is stored as a scalar texture and kept *outside* the neural network. This separation reflects its view-independent nature and ensures efficient access during stochastic opacity masking, where opacity must be evaluated for *all* fragments, while only the “winning” fragment’s color participates in differentiation. We use $N_{\text{feat}} = 7$, allowing all features and opacity to fit in two 8-bit RGBA textures after training.

3.4. Optimization

Loss function To optimize our triangle soup representation, we minimize a photometric objective,

$$\lambda \mathcal{L}_1 + (1 - \lambda) \mathcal{L}_{\text{SSIM}}, \quad (10)$$

where $\mathcal{L}_{\text{SSIM}}$ denotes the structural similarity term [49]. We set $\lambda = 0.8$ following 3DGS [24]. The loss is evaluated on the stochastic rendering, analogous to (7).

Initialization and training details To better capture high-frequency details through texture, we sample two-thirds of the target points using farthest point sampling (FPS) [9] on the SfM points and draw the remaining one-third uniformly at random. We then randomly initialize triangles around these sampled points with a radius set to one quarter of the average nearest-neighbor distance. We use VectorAdam [31] for optimizing vertex positions and Adam [27] for all other parameters. Training is run for 10,000 iterations. Additional hyperparameters are provided in the supplementary material. We compute gradients for four views in parallel at each optimization step.

Coarse-to-fine optimization We find that coarse-to-fine texture optimization improves the convergence of both ge-

Table 1. Quantitative evaluation on the *MipNeRF360* [3] dataset. For PSNR, SSIM, and LPIPS, we report scene-level averages. The number of primitives is strictly enforced for all methods.

Method	PSNR $\uparrow$	SSIM $\uparrow$	LPIPS $\downarrow$	# Prims
3DGS [24]	23.72	0.664	0.420	15K
TS [17]	22.81	0.634	0.430	15K
TexGS [5]	24.80	0.697	0.270	15K
Ours	24.76	0.748	0.204	15K

ometry and color. Specifically, we use $R_{\min} = R_{\max} = 3$ (i.e., vertex features) for the first 5000 iterations, and switch to $R_{\min} = 2$ and $R_{\max} = 5$ for the final 5000 iterations.

Adaptive control of primitives To balance reconstruction fidelity with a compact primitive budget, we introduce an adaptive mechanism for dynamically adjusting the triangle set. To avoid overly large primitives that compromise high-frequency detail, we periodically subdivide edges whose view-space length exceeds a prescribed fraction of the image height (1/5 in our experiments). This encourages approximately uniform view-space edge lengths and improves the effective use of texture resolution. Conversely, we enforce the target primitive count by pruning triangles with low pixel coverage across the training views until the desired budget is met. In practice, edge splitting and triangle removal are performed every 100 iterations. For edge splitting, view-space edge lengths are computed on a randomly sampled subset of 20 training views, as evaluating all views would introduce significant overhead.

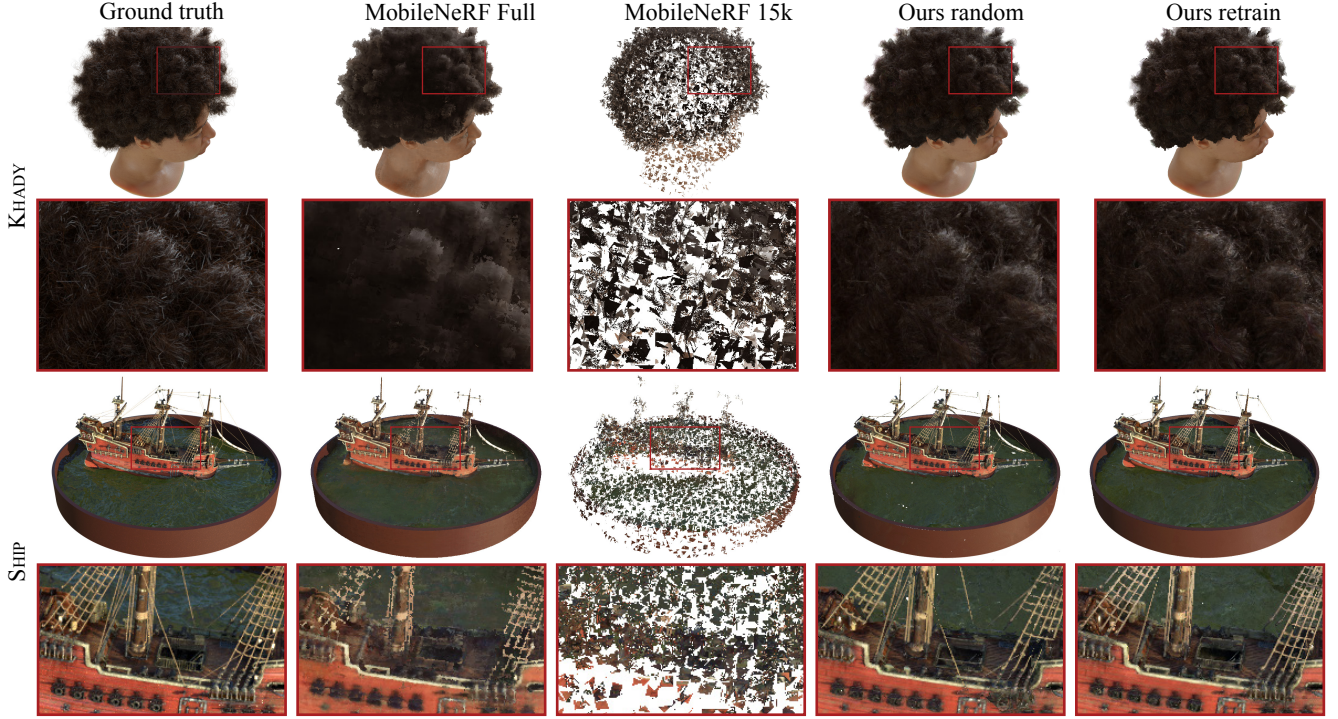


Figure 6. Qualitative results on synthetic scenes. (Left to right) Ground-truth views; view-synthesis results from MobileNeRF; a mesh decimated from the MobileNeRF output using QEM [12]; our results optimized from random initialization; and our results initialized from the MobileNeRF-decimated mesh.

Table 2. Comparison with MobileNeRF [7] on synthetic scenes from the *NeRF-Synthetic* [36] and *Shelly* [50] datasets. We report PSNR for each scene. The face count for MobileNeRF is averaged across scenes, whereas all other methods use exactly 15K faces.

Method	<i>NeRF-Synthetic</i> [36] (PSNR $\uparrow$)					<i>Shelly</i> [50] (PSNR $\uparrow$)			
	SHIP	CHAIR	MIC	DRUMS	# Faces $\downarrow$	KHADY	KITTEN	PUG	# Faces $\downarrow$
MobileNeRF [7]	26.06	31.02	29.18	23.82	159K	26.22	30.05	29.98	275K
MobileNeRF + QEM [12]	8.44	16.34	17.35	13.35	15K	11.21	14.42	11.25	15K
Ours w/ QEM initialization	26.68	32.11	30.18	23.94	15K	26.55	31.23	30.51	15K
Ours w/ random initialization	25.29	31.71	29.64	23.47	15K	26.67	30.68	30.16	15K

4. Experiments

Real-world dataset We evaluate our method on novel-view synthesis from multi-view RGB inputs using the Mip-NeRF360 dataset [3]. We compare against state-of-the-art primitive-based radiance-field reconstruction approaches. As shown in the qualitative results in Figure 5, 3DGS [24] is not well suited for low primitive budgets (e.g., 15K) and struggles to capture fine details and sharp boundaries. Because the method provides no explicit mechanism for controlling the final primitive count, we enforced a fixed budget by sub-sampling the initial SfM points and disabling adaptive densification.

TriangleSplatting [17] (TS) employs triangular primitives with a Markov chain Monte Carlo formulation [25]

to control the primitive count, but representing complex scenes without texture in this low-poly regime remains fundamentally challenging. TexturedGaussians [5] (TexGS) introduces texture fields to encode high-frequency appearance without increasing the primitive count. While TexGS significantly enhances detail reconstruction under constrained budgets, it still struggles to reproduce crisp, well-defined boundaries.

In contrast, our stochastic opacity masking drives primitive opacities toward binary values, enabling the reconstruction of the sharpest boundaries and the most detail-preserving view-synthesis results among all methods. These observations are further corroborated by the quantitative metrics in Table 1.

Rendering efficiency Beyond reducing primitive count, our approach enables hardware-accelerated rasterization by eliminating the transparency handling and depth sorting required in prior work. Table 3 reports average rendering cost on an NVIDIA RTX 4090. Although 3DGS benefits from a highly optimized CUDA renderer, our hardware-accelerated triangle rasterization achieves an order-of-magnitude speedup, despite incorporating additional texture information. Moreover, by relying on standard graphics pipelines, the same shader runs on commodity devices (e.g., the MacBook in Figure 1), yielding performance competitive with CUDA-based 3DGS on a desktop GPU. We fix primitive count across all methods, as it primarily governs the depth-sorting cost that dominates rendering time. For instance, TexGS uses $170\times$ more tunable parameters than 3DGS yet is only about $10\times$ slower, indicating that adding detail through texture is more efficient than increasing primitive count.

Synthetic scenes We further evaluate our method on synthetic scenes from the *NeRF-Synthetic* [36] and *Shelly* [50] datasets. In this setting, we compare our approach with MobileNeRF [7], which converts a trained NeRF [36] into a triangle mesh with a binary opacity texture. Although MobileNeRF can reconstruct fine structures with complex topology, it typically produces an excessively large number of faces per object (often exceeding 150K). Adjusting its training configuration to control the final face count is non-trivial, as training is computationally expensive (over 6 hours in our experiments). Due to MobileNeRF’s grid-aligned mesh topology, standard mesh simplification algorithms—such as quadratic error metric (QEM) decimation [12]—are ineffective at reducing the face count while preserving detail.

Our method naturally supports such *geometric simplification* scenarios when a coarse geometric prior is available. For instance, as shown in Figure 6, we can uniformly sample triangle primitives within the bounding box of the MobileNeRF representation and iteratively prune transparent primitives to obtain a clean triangle soup. Alternatively, we can use their low-quality decimated mesh as an *initialization* for our optimization. In both cases, our method achieves comparable reconstruction fidelity with a drastically reduced triangle count and significantly less time than retraining MobileNeRF; with QEM initialization, training took an average of 8 minutes and 40 seconds. As shown in Table 2, our simplified radiance-field representations achieve slightly higher PSNR scores across most scenes.

Additional results Figure 1 shows that scenes optimized using our method enable real-time rendering across diverse platforms and devices. We implement feature lookup and opacity masking using standard vertex and frag-

Table 3. Rendering time in the *MipNeRF360* [3] experiment. “Full” denotes the original image resolution. The number of trainable parameters is included for reference.

Method	FPS $\uparrow$ across resolutions			# Params
	Full	1/2	1/4	
3DGS [24]	115	482	1.32K	88.5K
TS [17]	88.8	370	1.00K	88.5K
TexGS [5]	16.8	49.1	94.8	15.1M
Ours (CUDA)	1.96K	6.11K	13.7K	6.75M
Ours (laptop)	146	447	879	6.75M

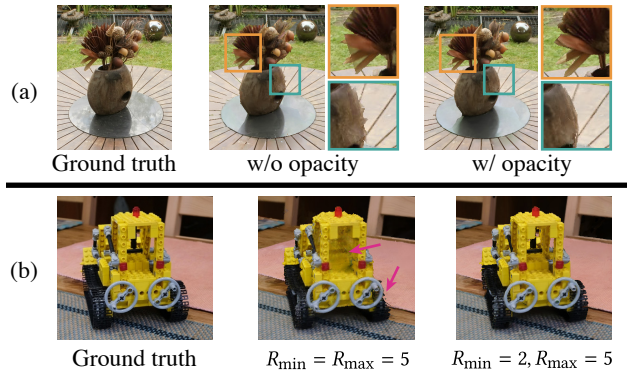


Figure 7. Ablation studies. (a) Without opacity learning, triangles are unable to represent fine sub-triangle boundaries. (b) Without our multi-resolution texture scheme, the instability of color optimization leads to noisier geometry (pink arrows). For a clearer comparison, we disable coarse-to-fine optimization in setting (b).

ment shaders, and perform MLP evaluation as a deferred-rendering pass in a separate shader. Figure 7 presents ablation studies of our key components.

5. Conclusion

We introduced a differentiable rasterization method for reconstructing high-fidelity 3D scenes from multi-view RGB images. Our opaque, single-layer shading model limits the representation of transparent objects and thin structures (e.g., hair and fur), which are more naturally handled by volumetric primitives. Because each pixel’s color is determined by a single primitive rather than a blend of many, our method does not yet match state-of-the-art volumetric pipelines (e.g., AdaptiveShells [50]). Closing this gap and achieving high-quality anti-aliasing for thin structures while retaining rendering efficiency remain important directions for future work. We believe that our stochastic opacity masking not only enables cross-platform rendering and seamless transfer of digital 3D content but also opens new possibilities for differentiable rasterization techniques that require gradients through inherently discrete operations.

Acknowledgements

We thank the anonymous reviewers for their valuable feedback. This work was supported by JSPS KAKENHI Grant Number 23KJ0699 and by JST ASPIRE Grant Number JP-MJAP2401.

References

- [1] Sai Bangaru, Michael Gharbi, Tzu-Mao Li, Fujun Luan, Kalyan Sunkavalli, Milos Hasan, Sai Bi, Zexiang Xu, Gilbert Bernstein, and Fredo Durand. Differentiable rendering of neural sdfs through reparameterization. In *ACM SIGGRAPH Asia 2022 Conference Proceedings*, New York, NY, USA, 2022. Association for Computing Machinery.
- [2] Jonathan T. Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P. Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. *ICCV*, 2021.
- [3] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. *CVPR*, 2022.
- [4] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.
- [5] Brian Chao, Hung-Yu Tseng, Lorenzo Porzi, Chen Gao, Tuotuo Li, Qinbo Li, Ayush Saraf, Jia-Bin Huang, Johannes Kopf, Gordon Wetzstein, and Changil Kim. Textured gaussians for enhanced 3d scene appearance modeling. In *CVPR*, 2025.
- [6] Wenzheng Chen, Jun Gao, Huan Ling, Edward Smith, Jaakko Lehtinen, Alec Jacobson, and Sanja Fidler. Learning to predict 3d objects with an interpolation-based differentiable renderer. In *Advances In Neural Information Processing Systems*, 2019.
- [7] 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.
- [8] Xavier Décoret, Frédo Durand, François X. Sillion, and Julie Dorsey. Billboard clouds for extreme model simplification. In *ACM SIGGRAPH 2003 Papers*, page 689–696, New York, NY, USA, 2003. Association for Computing Machinery.
- [9] Yuval Eldar, Michael Lindenbaum, Moshe Porat, and Yehoshua Y Zeevi. The farthest point strategy for progressive image sampling. *IEEE transactions on image processing*, 6(9):1305–1315, 1997.
- [10] Stefano Esposito, Anpei Chen, Christian Reiser, Samuel Rota Bulò, Lorenzo Porzi, Katja Schwarz, Christian Richardt, Michael Zollhoefer, Peter Kotschieder, and Andreas Geiger. Volumetric surfaces: Representing fuzzy geometries with layered meshes. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.
- [11] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5501–5510, 2022.
- [12] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, page 209–216, USA, 1997. ACM Press/Addison-Wesley Publishing Co.
- [13] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.
- [14] Peter W Glynn. Likelihood ratio gradient estimation for stochastic systems. *Communications of the ACM*, 33(10): 75–84, 1990.
- [15] Jon Hasselgren, Jacob Munkberg, Jaakko Lehtinen, Miika Aittala, and Samuli Laine. Appearance-driven automatic 3d model simplification. In *Eurographics Symposium on Rendering*, 2021.
- [16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [17] Jan Held, Renaud Vandeghen, Adrien Deliege, Abdullah Hamdi, Anthony Cioppa, Silvio Giancola, Andrea Vedaldi, Bernard Ghanem, Andrea Tagliasacchi, and Marc Van Droogenbroeck. Triangle splatting for real-time radiance field rendering. *arXiv*, 2025.
- [18] Jan Held, Renaud Vandeghen, Abdullah Hamdi, Adrien Deliege, Anthony Cioppa, Silvio Giancola, Andrea Vedaldi, Bernard Ghanem, and Marc Van Droogenbroeck. 3D convex splatting: Radiance field rendering with 3D smooth convexes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.
- [19] Jan Held, Renaud Vandeghen, Sanghyun Son, Daniel Rebain, Matheus Gadelha, Yi Zhou, Ming C. Lin, Marc Van Droogenbroeck, and Andrea Tagliasacchi. Triangle splatting+: Differentiable rendering with opaque triangles. *arXiv*, 2025.
- [20] Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger, and Shenghua Gao. 2d gaussian splatting for geometrically accurate radiance fields. In *SIGGRAPH 2024 Conference Papers*. Association for Computing Machinery, 2024.
- [21] Wenzel Jakob. nanobind: tiny and efficient c++/python bindings, 2022. <https://github.com/wjakob/nanobind>.
- [22] James T Kajiya and Brian P Von Herzen. Ray tracing volume densities. *ACM SIGGRAPH computer graphics*, 18(3):165–174, 1984.
- [23] Hiroharu Kato, Deniz Beker, Mihai Morariu, Takahiro Ando, Toru Matsuoka, Wadim Kehl, and Adrien Gaidon. Differentiable rendering: A survey. *arXiv preprint arXiv:2006.12057*, 2020.
- [24] 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.
- [25] Shakiba Kheradmand, Daniel Rebain, Gopal Sharma, Weiwei Sun, Yang-Che Tseng, Hossam Isack, Abhishek Kar,

- Andrea Tagliasacchi, and Kwang Moo Yi. 3d gaussian splatting as markov chain monte carlo. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. Spotlight Presentation.
- [26] Shakiba Kheradmand, Delio Vicini, George Kopanas, Dmitry Lagun, Kwang Moo Yi, Mark Matthews, and Andrea Tagliasacchi. Stochasticplats: Stochastic rasterization for sorting-free 3d gaussian splatting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2025.
- [27] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [28] Samuli Laine, Janne Hellsten, Tero Karras, Yeongho Seol, Jaakko Lehtinen, and Timo Aila. Modular primitives for high-performance differentiable rendering. *ACM Transactions on Graphics*, 39(6), 2020.
- [29] Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. Differentiable monte carlo ray tracing through edge sampling. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, 37(6):222:1–222:11, 2018.
- [30] Zhaoshuo Li, Thomas Müller, Alex Evans, Russell H Taylor, Mathias Unberath, Ming-Yu Liu, and Chen-Hsuan Lin. Neuralangelo: High-fidelity neural surface reconstruction. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.
- [31] Selena Zihan Ling, Nicholas Sharp, and Alec Jacobson. Vectoradam for rotation equivariant geometry optimization. *Advances in Neural Information Processing Systems*, 35:4111–4122, 2022.
- [32] Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. Soft rasterizer: A differentiable renderer for image-based 3d reasoning. *The IEEE International Conference on Computer Vision (ICCV)*, 2019.
- [33] Matthew M Loper and Michael J Black. Opendr: An approximate differentiable renderer. In *Computer Vision—ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6–12, 2014, Proceedings, Part VII 13*, pages 154–169. Springer, 2014.
- [34] Guillaume Loubet, Nicolas Holzschuch, and Wenzel Jakob. Reparameterizing discontinuous integrands for differentiable rendering. *Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 38(6), 2019.
- [35] Ishit Mehta, Manmohan Chandraker, and Ravi Ramamoorthi. A theory of topological derivatives for inverse rendering of geometry. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 419–429, 2023.
- [36] 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.
- [37] 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.
- [38] Baptiste Nicolet, Alec Jacobson, and Wenzel Jakob. Large steps in inverse rendering of geometry. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 40(6), 2021.
- [39] Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. Mitsuba 2: A retargetable forward and inverse renderer. *Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 38(6), 2019.
- [40] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.
- [41] Stanislav Pidhorskyi, Tomas Simon, Gabriel Schwartz, He Wen, Yaser Sheikh, and Jason Saragih. Rasterized edge gradients: Handling discontinuities differentially. In *European Conference on Computer Vision*, pages 335–352. Springer, 2024.
- [42] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3d deep learning with pytorch3d. *arXiv preprint arXiv:2007.08501*, 2020.
- [43] Victor Rong, Jingxiang Chen, Sherwin Bahmani, Kiriakos N Kutulakos, and David B Lindell. Gstex: Per-primitive texturing of 2d gaussian splatting for decoupled appearance and geometry modeling. *arXiv preprint arXiv:2409.12954*, 2024.
- [44] David Svitov, Pietro Morerio, Lourdes Agapito, and Alessio Del Bue. Billboard splatting (bbsplat): Learnable textured primitives for novel view synthesis. *arXiv preprint arXiv:2411.08508*, 2024.
- [45] Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Neural geometric level of detail: Real-time rendering with implicit 3d shapes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11358–11367, 2021.
- [46] Justus Thies, Michael Zollhöfer, and Matthias Nießner. Deferred neural rendering: Image synthesis using neural textures. *Acm Transactions on Graphics (TOG)*, 38(4):1–12, 2019.
- [47] 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.
- [48] 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. *arXiv preprint arXiv:2106.10689*, 2021.
- [49] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.
- [50] Zian Wang, Tianchang Shen, Merlin Nimier-David, Nicholas Sharp, Jun Gao, Alexander Keller, Sanja Fidler, Thomas Müller, and Zan Gojcic. Adaptive shells for efficient neural radiance field rendering. *ACM Trans. Graph.*, 42(6), 2023.
- [51] Philippe Weier, Jérémy Riviere, Ruslan Guseinov, Stephan Garbin, Philipp Slusallek, Bernd Bickel, Thabo Beeler, and

- Delio Vicini. Practical inverse rendering of textured and translucent appearance. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 44(4), 2025.
- [52] Sebastian Weiss and Derek Bradley. Gaussian billboards: Expressive 2d gaussian splatting with textures. *arXiv preprint arXiv:2412.12734*, 2024.
- [53] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.
- [54] Markus Worchel, Rodrigo Diaz, Weiwen Hu, Oliver Schreer, Ingo Feldmann, and Peter Eisert. Multi-view mesh reconstruction with neural deferred shading. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6187–6197, 2022.
- [55] Rui Xu, Wenyue Chen, Jiepeng Wang, Yuan Liu, Peng Wang, Lin Gao, Shiqing Xin, Taku Komura, Xin Li, and Wenping Wang. Supergaussians: Enhancing gaussian splatting using primitives with spatially varying colors, 2024.
- [56] Tian-Xing Xu, Wenbo Hu, Yu-Kun Lai, Ying Shan, and Song-Hai Zhang. Texture-gs: Disentangling the geometry and texture for 3d gaussian splatting editing, 2024.
- [57] Lior Yariv, Jiatao Gu, Yoni Kasten, and Yaron Lipman. Volume rendering of neural implicit surfaces. *Advances in Neural Information Processing Systems*, 34:4805–4815, 2021.
- [58] Wang Yifan, Felice Serena, Shihao Wu, Cengiz Öztireli, and Olga Sorkine-Hornung. Differentiable surface splatting for point-based geometry processing. *ACM Transactions On Graphics (TOG)*, 38(6):1–14, 2019.
- [59] Cem Yuksel. Mesh color textures. In *Proceedings of High Performance Graphics*, New York, NY, USA, 2017. Association for Computing Machinery.
- [60] Cem Yuksel, John Keyser, and Donald H House. Mesh colors. *ACM Transactions on Graphics (TOG)*, 29(2):1–11, 2010.
- [61] Ziyi Zhang, Nicolas Roussel, and Wenzel Jakob. Many-worlds inverse rendering. *ACM Trans. Graph.*, 2025.
- [62] Ziyi Zhang, Nicolas Roussel, Thomas Muller, Tizian Zeltner, Merlin Nimier-David, Fabrice Rousselle, and Wenzel Jakob. Radiance surfaces: Optimizing surface representations with a 5d radiance field loss. In *Proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers*, New York, NY, USA, 2025. Association for Computing Machinery.