SPIRE: A DIFFERENTIABLE MESH RENDERER FOR CLOSING THE EXPORT GAP

Anonymous authors
Paper under double-blind review



Figure 1: **SPIRE optimizes opaque meshes to preserve visual detail in Blender.** Under matched Blender settings, the enlarged scene-editing views show smoothed surface detail after mesh extraction in 2DGS (Huang et al., 2024) and visible triangular artifacts in MeshSplatting (Held et al., 2025a), which transitions from softened to opaque faces during training. SPIRE instead differentiates pixel-area coverage with faces kept fully opaque, directly optimizing mesh geometry to preserve fine surface detail and sharp boundaries. The left panel demonstrates fluid simulation, and the right panels show scene editing with a full view and three matching crops.

ABSTRACT

Triangle meshes are the standard representation for graphics and geometry processing, yet high-fidelity novel-view synthesis commonly relies on neural fields or Gaussian primitives. Converting these representations to meshes can reduce rendering fidelity, creating an export gap between native rendering and the reusable asset. Direct mesh optimization is challenging due to hard surface boundaries and occlusion. Conventional mesh renderers make binary coverage and visibility decisions at discrete pixel samples, yielding zero derivatives almost everywhere. A real camera, by contrast, integrates radiance over each pixel. This pixel-area integration can yield continuous, differentiable image values even for opaque meshes with hard boundaries. We introduce SPIRE, a differentiable mesh renderer that uses analytic anti-aliasing to realize pixel-area integration. The renderer integrates clipped triangle footprints, combines adjacent faces without mutual occlusion, and spatially tracks the remaining visible footprint when compositing depth-separated surfaces. A sparse backward pass efficiently differentiates these operations with respect to mesh geometry and spherical harmonic appearance. Given multiview RGB images, SPIRE reconstructs conventional opaque meshes with shared vertices and vertex colors, without external normal, depth, or diffusion priors. Our meshes achieve the highest post-export fidelity across the evaluated Blender and Filament settings on synthetic and real scenes, with small export gaps. Controlled ablations establish the importance of pixel-area coverage and surface-aware compositing for direct mesh optimization.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

1 INTRODUCTION

Scene reconstruction from images should ideally enable both novel-view synthesis (NVS) and the creation of reusable 3D assets. Triangle meshes are well suited to both purposes because they can be rendered efficiently while supporting editing, animation, geometry processing, and simulation in standard graphics software. Recent high-fidelity NVS methods often optimize neural fields or 3D Gaussian primitives using representation-specific renderers (Mildenhall et al., 2021; Kerbl et al., 2023). Using these representations as meshes in standard graphics software, such as Blender, Autodesk Maya, and Unreal Engine, requires converting them to opaque meshes (Blender Development Team, 2026; Autodesk, Inc., 2026; Epic Games, Inc., 2026). This conversion can alter geometry, appearance, or visibility, degrading rendering quality even when the native representation renders highly accurate novel views. We refer to this discrepancy between native rendering quality and that of the exported mesh as the *export gap*.

Despite their practical advantages, meshes are rarely used as the underlying scene representation for high-fidelity NVS because differentiable rendering of meshes remains challenging. A conventional mesh renderer evaluates a finite number of samples per pixel and makes hard decisions about coverage and visibility. These discrete decisions have zero derivatives almost everywhere and thus provide little useful gradient signal for optimizing mesh geometry.

Existing methods address this difficulty by modifying either the scene representation or the rendering gradients. Neural-field and Gaussian methods optimize a representation suited to continuous optimization and extract meshes afterward (Wang et al., 2021; Huang et al., 2024; Guédon & Lepetit, 2024; Yu et al., 2024b). When extraction is deferred to post-processing, image losses do not directly supervise the mesh used downstream or account for the changes introduced by conversion. Consequently, extraction can remove thin structures or displace surface boundaries even when the optimized representation fits the input images.

Differentiable rasterizers instead enable vertex optimization by softening coverage or approximating boundary derivatives (Loper & Black, 2014; Kato et al., 2018; Liu et al., 2019; Chen et al., 2019; Laine et al., 2020; Pidhorskyi et al., 2024). Recent triangle-based methods also obtain training gradients through softened support, semitransparency, stochastic masks, or surrogate edge gradients (Held et al., 2026; 2025a; Tojo et al., 2026). These advances enable mesh optimization, but semitransparent or softened primitives contribute differently to the image during training and during final opaque rendering. For example, a misplaced foreground face that contributed through partial transparency during optimization can occlude the surface behind it when made opaque, producing visible triangular artifacts. Surrogate derivatives, on the other hand, approximate the effect of geometry changes rather than differentiating through the forward rendering operations themselves. Errors in this approximation can misdirect vertex updates near face boundaries, making geometric refinement less reliable. Local coverage approximations can also miss the contribution of subpixel triangles (Laine et al., 2020). Our goal is to obtain useful geometry gradients directly from rendering the opaque mesh, without softening its faces or introducing separate surrogate derivatives. The mesh can thus be optimized in the opaque form used downstream, without a later transition from softened or semitransparent faces. Table 1 compares the optimization and rendering properties of representative methods.

We introduce SPIRE (**S**urface-aware **P**ixel-area **I**ntegration **RE**nderer), a mesh renderer that remains differentiable even for fully opaque, shared-vertex triangle meshes. Our intuition is that a real camera integrates radiance over each pixel, allowing moving opaque boundaries to produce continuous, differentiable image values. SPIRE realizes this principle through analytic anti-aliasing, combining exact triangle-footprint integration with surface-aware compositing that accumulates adjacent faces together and approximates the remaining visible region through moment matching. A sparse backward pass differentiates these same operations with respect to vertex positions and spherical harmonic appearance, without softening the opaque faces in the forward pass.

In calibrated multiview reconstruction, SPIRE jointly optimizes vertex positions and appearance from RGB supervision while keeping every face opaque. The resulting mesh is compatible with graphics software that supports vertex-colored triangle meshes, including Blender and Filament, without requiring our renderer. We evaluate exported meshes in Blender and Filament, separating the fidelity of reusable assets from native rendering quality.

Table 1: Comparison of mesh optimization and rendering capabilities of methods spanning mesh extraction with 2DGS (Huang et al., 2024), differentiable rasterization with nvdiffrast (Laine et al., 2020), and triangle-based reconstruction with MeshSplatting (Held et al., 2025a). SPIRE is a differentiable mesh renderer that uses analytic anti-aliasing to obtain exact face-footprint coverage gradients without softening mesh visibility, enabling direct optimization of opaque meshes to reduce the export gap. ✓ and × indicate support and no support, respectively.

Method	Direct mesh optimization	Differentiable opaque mesh rendering	Coverage gradients	Coverage anti-aliasing
2DGS	×	×	×	×
Gaussian Blending	×	×	×	✓
nvdiffrast	✓	✓	Silhouette only	Silhouette only
MeshSplatting	✓	Soft window	Soft window	SSAA
SPIRE (Ours)	✓	✓	✓	✓

We make three main contributions.

- We show that exact pixel-area integration turns the motion of hard opaque boundaries into analytic coverage gradients, enabling SPIRE to directly optimize fully opaque, shared-vertex meshes without semitransparency or softened support. Its surface-aware anti-aliasing integrates connected faces as one surface while preserving inter-surface occlusion.
- As native renderers entangle asset quality with representation-specific appearance and compositing, our post-export protocol uses matched settings in standard graphics software to measure the NVS fidelity preserved in deployable meshes.
- Applied to calibrated multiview reconstruction, SPIRE optimizes deployable meshes, achieving the highest post-export NVS fidelity among evaluated methods on synthetic and real scenes while retaining small export gaps.

2 Related Work

2.1 Mesh reconstruction and extraction

Neural-field and Gaussian methods reconstruct surfaces from images using representations that support continuous optimization. Neural fields yield meshes through isosurface extraction (Yariv et al., 2020; Oechsle et al., 2021; Wang et al., 2021; Yariv et al., 2021; Wang et al., 2023; Li et al., 2023), while Gaussian methods use surface regularization, depth fusion, or opacity-field extraction (Guédon & Lepetit, 2024; Huang et al., 2024; Dai et al., 2024; Chen et al., 2024; Zhang et al., 2026a; Yu et al., 2024b). Converting the learned representation can change surface locations, appearance, and visibility, so native rendering quality alone does not establish the fidelity of the exported mesh. MeshSplats (Tobiasz et al., 2026) and other mesh-based view-synthesis methods convert or bake learned scene representations into meshes with explicit appearance information for efficient rendering in graphics pipelines (Chen et al., 2023b; Yariv et al., 2023; Tang et al., 2023; Reiser et al., 2024).

Differentiable extraction brings mesh rendering into training rather than leaving meshing entirely to post-processing. nvdiffrec and FlexiCubes optimize meshes through grid-based isosurface extraction (Munkberg et al., 2022; Shen et al., 2023), while IMLS-Splatting, MILo, and SDFRaster derive meshes from point, Gaussian, or signed-distance representations (Yang et al., 2025; Guédon et al., 2025; Cui et al., 2026). Meshtryoshka instead optimizes an underlying signed-distance field through opacity-based compositing of extracted mesh shells (Charatan et al., 2026). Our focus is the renderer needed to optimize shared-vertex meshes directly, without an intervening extraction step.

2.2 Differentiable mesh and triangle rendering

Differentiable rasterizers supply geometry gradients through smoothed coverage or approximate boundary derivatives (Loper & Black, 2014; Kato et al., 2018; Liu et al., 2019; Chen et al., 2019). nvdiffrast and rasterized edge gradients provide local visibility gradients at projected boundaries

(Laine et al., 2020; Pidhorskyi et al., 2024), and Large Steps improves the conditioning of vertex updates (Nicolet et al., 2021). Together, these advances enable image-based mesh optimization without requiring a volumetric scene representation.

Triangle-based novel-view synthesis also adapts the training primitives to obtain useful gradients. Triangle Splatting and 2D Triangle Splatting use splatted support or straight-through gradients (Held et al., 2026; Sheng et al., 2026), while Triangle Splatting+ (Held et al., 2025b) and MeshSplatting (Held et al., 2025a) introduce shared vertices and progressively enforce opacity. Mesh Splatting by Zhang et al. (2026b) instead renders semitransparent layers around a base mesh, whereas DiffSoup combines depth-tested triangles with stochastic opacity masks and rasterized edge gradients (Tojo et al., 2026). In these methods, opaque rendering and the gradients used for training are treated as separate design choices. We instead study how pixel-area integration can supply geometry gradients while retaining opaque faces, with adjacent faces contributing as one surface and depth-separated surfaces occluding one another.

2.3 Anti-aliasing and pixel integration

Accounting for a pixel's spatial extent reduces aliasing and provides a basis for differentiating moving boundaries (Li et al., 2020; Zhou et al., 2021). DiffVG uses analytical prefiltering or multisampling for vector graphics, while Mip-NeRF models conical ray footprints in neural rendering (Li et al., 2020; Barron et al., 2021). Zhou et al. (2021) maintain explicit visible polygon regions for analytic integration and differentiation. Mip-Splatting, Analytic-Splatting, and AA-2DGS filter Gaussian primitives or approximate their pixel integrals (Yu et al., 2024a; Liang et al., 2024; Younes & Boukhayma, 2026), and Gaussian Blending tracks within-pixel distributions during compositing (Koo et al., 2026). SPIRE combines exact face-footprint integration with surface grouping and a compact residual-footprint approximation, avoiding explicit visibility subdivision while supporting sparse differentiation of shared-vertex meshes.

3 Approach: SPIRE

3.1 Problem formulation and renderer overview

Given a triangle mesh and a calibrated camera, SPIRE uses analytic anti-aliasing to obtain coverage gradients while keeping mesh faces opaque. Our contribution is a surface-aware formulation that combines exact face-footprint integration with compact visibility propagation and sparse differentiation for shared-vertex meshes. For each pixel, it repeats triangle-pixel integration, same-surface accumulation, and spatial transmittance propagation for successive surfaces in a depth order (Figure 2). Accumulating adjacent faces as one surface and approximating the remaining visible region makes pixel-area integration practical without maintaining an explicit subdivision of visibility.

Let $\mathcal{M} = (V, F, H, \mathbf{w})$ denote a triangle mesh whose faces may share vertices. Its geometry is $V = [\mathbf{v}_i]_{i=1}^{|V|} \in \mathbb{R}^{|V|\times 3}$, where $\mathbf{v}_i$ is a vertex position, and its connectivity is $F = [\mathbf{f}_j]_{j=1}^{|F|} \in \{1, \dots, |V|\}^{|F|\times 3}$, where $\mathbf{f}_j = (f_{j1}, f_{j2}, f_{j3})$ indexes the vertices of face j. For spherical harmonics (SH) with a degree of L, view-dependent appearance is represented by RGB coefficients $H = \{\mathbf{h}_{i\ell m}\} \in \mathbb{R}^{|V|\times(L+1)^2\times 3}$, and per-vertex opacity is represented by $\mathbf{w} = [w_i]_{i=1}^{|V|} \in [0,1]^{|V|}$. The formulation supports both partially transparent and fully opaque faces. For reconstruction, we keep all faces fully opaque throughout optimization so that the resulting meshes can be used directly by any standard mesh-rendering software.

Consider a calibrated camera $\Pi = (K, R, \mathbf{t})$, with $K \in \mathbb{R}^{3\times 3}$, $R \in \mathrm{SO}(3)$, and $\mathbf{t} \in \mathbb{R}^3$. The background color is $\mathbf{C}_{\mathrm{bg}} \in [0,1]^3$. The renderer maps the mesh and camera to an RGB image:

$$(\widehat{I}, \mathcal{Z}) = \mathrm{Render}(\mathcal{M}, \Pi, \mathbf{C}_{\mathrm{bg}}), \qquad \widehat{I} \in \mathbb{R}^{H_{\mathrm{img}}\times W_{\mathrm{img}}\times 3}. \tag{1}$$

$\mathcal{Z}$ stores the active face-pixel contributions needed to differentiate the rendered image with respect to mesh geometry and appearance.

For the unit direction $\mathbf{d}_i$ from the camera center to vertex i, let $\mathbf{C}_i^V(\mathbf{d}_i)$ denote the RGB color obtained by evaluating the degree-L SH coefficients at $\mathbf{d}_i$ and clamping each channel to be nonnegative. The opacity of face j is $o_j = \min\big(w_{f_{j1}}, w_{f_{j2}}, w_{f_{j3}}\big)$.

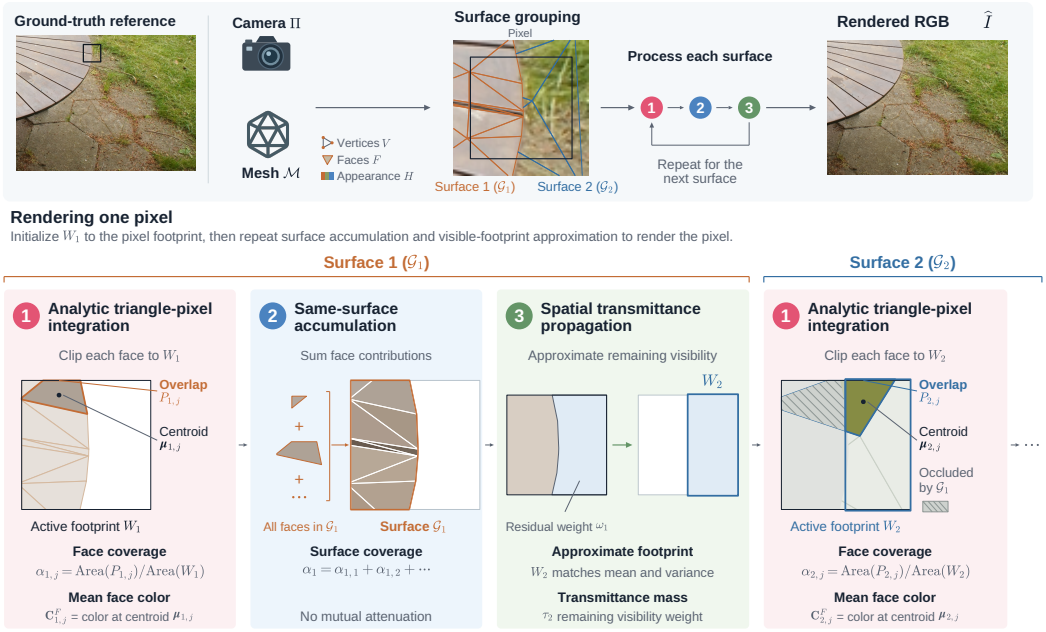


Figure 2: Overview of SPIRE for an opaque mesh. For each surface, triangle-pixel integration computes face coverage and mean color, same-surface accumulation combines contributions without mutual attenuation, and spatial transmittance propagation approximates the remaining visible footprint. These stages repeat over all surfaces contributing to the pixel in depth order.

3.2 Analytic Triangle-Pixel Integration

A camera integrates radiance over a finite image-plane area. Following this principle, we compute each face's coverage from its overlap with the active footprint representing the remaining visible region. Analytic integration of this overlap provides derivatives with respect to projected vertex positions within each clipping configuration, enabling differentiable rendering of opaque faces while also providing anti-aliasing. We describe the computation for one pixel and omit its index p. A surface group $\mathcal{G}_s$ contains adjacent faces whose covered areas should be combined rather than composited as separate occluding layers. Section 3.3 defines these groups and their depth order. At surface s, the remaining transmittance is represented by $T_s = (\tau_s, W_s)$. The transmittance mass τ_s specifies how much weight remains for deeper surfaces and the background, while the active footprint W_s approximates where that weight is distributed within the pixel. We initialize $T_1 = (1, W_1)$, with W_1 equal to the full pixel footprint. Coverage and color are integrated analytically over the active footprint W_s, while its evolution between surfaces is approximated by moment matching.

Let f_j^{Π} denote the screen-space region occupied by face j after projection through camera Π. Its overlap with the active footprint is $P_{s,j} = f_j^{\Pi} \cap W_s$. With $\mathrm{Area}(\cdot)$ denoting screen-space polygon area, face j contributes

$$\alpha_{s,j} = o_j \frac{\mathrm{Area}(P_{s,j})}{\mathrm{Area}(W_s)}. \tag{2}$$

The centroid and component-wise raw second moment of the clipped polygon are

$$\boldsymbol{\mu}_{s,j} = \frac{1}{\mathrm{Area}(P_{s,j})} \int_{P_{s,j}} \mathbf{x}\, d\mathbf{x}, \qquad \boldsymbol{\nu}_{s,j} = \frac{1}{\mathrm{Area}(P_{s,j})} \int_{P_{s,j}} \mathbf{x} \odot \mathbf{x}\, d\mathbf{x}, \tag{3}$$

where $\mathbf{x}$ is expressed in pixel-local coordinates and $\odot$ denotes component-wise multiplication. We obtain the overlap polygon with Sutherland–Hodgman clipping (Sutherland & Hodgman, 1974) and compute its area and moments in closed form. The centroid determines the mean interpolated color, while the second moment helps track the extent of the visible region during compositing.

Let $\beta_{j,k}(\mathbf{x})$ be the screen-space barycentric coordinate associated with vertex f_{jk}. The mean color of face j over its clipped footprint is

$$\mathbf{C}^F_{s,j} = \frac{1}{\mathrm{Area}(P_{s,j})} \int_{P_{s,j}} \sum_{k=1}^{3} \beta_{j,k}(\mathbf{x}) \mathbf{C}^V_{f_{jk}}(\mathbf{d}_{f_{jk}})\, d\mathbf{x} = \sum_{k=1}^{3} \beta_{j,k}(\boldsymbol{\mu}_{s,j}) \mathbf{C}^V_{f_{jk}}(\mathbf{d}_{f_{jk}}). \tag{4}$$

Screen-space barycentric interpolation is affine, so evaluating it at the polygon centroid gives the exact mean color without additional shading samples. The coverage and color above are therefore analytic functions of the projected vertices within each clipping configuration.

3.3 Surface-aware compositing

Individual face coverage alone does not determine how faces occlude one another. Applying the per-primitive alpha blending commonly used in splatting-based NVS directly to face coverages would attenuate each face by preceding faces, even when their projected interiors are disjoint. Adjacent faces tessellating the same surface should instead contribute additively, whereas different surfaces can overlap and occlude one another. We therefore accumulate faces within each surface without mutual attenuation and propagate an approximation of the remaining visible footprint between surfaces. We group edge-connected faces with the same projected orientation (both front-facing or both back-facing) within each image tile, then separate faces with non-overlapping vertex-depth ranges within each pixel. This preserves adjacency within a surface while distinguishing depth-separated parts of the mesh. Sorting these groups by depth gives the front-to-back sequence $\mathcal{S} = (\mathcal{G}_1, \ldots, \mathcal{G}_{|\mathcal{S}|})$, where $\mathcal{G}_s$ contains the faces assigned to surface s.

We sum same-surface coverage without mutual occlusion, $\alpha_s = \sum_{j \in \mathcal{G}_s} \alpha_{s,j}$. Front-to-back compositing gives the rendered pixel color $\widehat{\mathbf{c}}$ as

$$\widehat{\mathbf{c}} = \tau_{|\mathcal{S}|+1} \mathbf{C}_{\mathrm{bg}} + \sum_{s=1}^{|\mathcal{S}|} \tau_s \left(\sum_{j \in \mathcal{G}_s} \alpha_{s,j} \mathbf{C}^F_{s,j} \right), \qquad \text{where } \tau_{s+1} = \tau_s (1 - \alpha_s). \tag{5}$$

Between surfaces, scalar transmittance records how much weight remains but not where it lies, so it cannot capture spatial occlusion on its own. Tracking the visible region exactly requires repeated polygon subdivision and can produce quadratically many fragments as thin projected faces intersect (Appendix A). We retain a compact spatial representation by matching the first two moments of the residual footprint with an axis-aligned rectangle (Koo et al., 2026). The rectangle preserves the residual center and spatial extent while keeping the clipping boundary at four edges regardless of the number of preceding faces. Let $\mathbf{1}_{P_{s,j}}(\mathbf{x})$ indicate whether $\mathbf{x}$ lies in the clipped footprint of face j. The residual visibility has center and component-wise variance

$$\mathbf{u}^{(s+1)} = \frac{\int_{W_s} \mathbf{x}\, \omega_s(\mathbf{x})\, d\mathbf{x}}{\int_{W_s} \omega_s(\mathbf{x})\, d\mathbf{x}}, \quad \boldsymbol{\upsilon}^{(s+1)} = \frac{\int_{W_s} (\mathbf{x} - \mathbf{u}^{(s+1)}) \odot (\mathbf{x} - \mathbf{u}^{(s+1)}) \omega_s(\mathbf{x})\, d\mathbf{x}}{\int_{W_s} \omega_s(\mathbf{x})\, d\mathbf{x}}, \tag{6}$$

where $\omega_s(\mathbf{x}) = 1 - \sum_{j \in \mathcal{G}_s} o_j \mathbf{1}_{P_{s,j}}(\mathbf{x})$. Thus, ω_s is zero where an opaque face covers the active footprint and one where the pixel remains visible to deeper surfaces. We approximate the residual visibility by a uniform distribution over the axis-aligned rectangle W_{s+1} centered at $\mathbf{u}^{(s+1)}$ with component-wise widths $\sqrt{12\boldsymbol{\upsilon}^{(s+1)}}$. Together, τ_{s+1} and W_{s+1} form the next transmittance state T_{s+1}. This state gives deeper surfaces a spatially localized approximation of the pixel region left visible by the nearer opaque surface. Appendix B.2 gives the moment-subtraction formulas.

3.4 Sparse differentiation

Surface-aware compositing couples faces through a shared surface contribution, so each local coverage derivative depends on the completed surface accumulation. The backward pass therefore uses two sweeps. The first computes surface-level gradient coefficients. The second evaluates local face-pixel derivatives in parallel and combines them with the shared coefficients to accumulate gradients to vertex positions, opacity, and SH coefficients (Appendix B.3). To reduce memory usage, we retain only active face-pixel contributions in $\mathcal{Z}$ and recompute local clipping instead of storing every clipped polygon and its Jacobian. Inactive face-pixel pairs are never stored or replayed, so record storage and replay work are $\mathcal{O}(H_{\mathrm{img}} W_{\mathrm{img}} + |\mathcal{Z}|)$, including the per-pixel record offsets.

3.5 MULTIVIEW MESH RECONSTRUCTION

We apply SPIRE to reconstruct opaque meshes from calibrated multiview RGB images, allowing the image loss to supervise the geometry that will be exported. We initialize a shared-vertex mesh with GOF (Yu et al., 2024b) and jointly optimize vertex positions V and SH appearance H using SPIRE, preserving mesh connectivity F and vertex count. Appearance uses SH degree $L = 3$, with $\mathbf{w} = \mathbf{1}_{|V|}$ fixed so that every face remains opaque ($o_j = 1$).

We use the photometric loss of 3DGS (Kerbl et al., 2023) with depth and face-area regularization.

$$\mathcal{L} = 0.8\mathcal{L}_1 + 0.2\mathcal{L}_{\text{D-SSIM}} + \lambda_{\text{depth}}\mathcal{L}_{\text{depth}} + \lambda_{\text{area}}\mathcal{L}_{\text{area}}. \tag{7}$$

The depth term encourages consistency between vertex depths and the renderer's own median-depth image, without external depth supervision. The area term penalizes large visible faces relative to a fixed reference area. Appendix C gives their definitions and weights.

We export the degree-zero (DC) SH component as view-independent vertex colors, yielding an opaque mesh for standard graphics software.

4 EXPERIMENTS

We evaluate exported meshes in Blender and Filament on synthetic and real scenes, comparing their fidelity with native rendering quality to assess the export gap.

4.1 EXPERIMENTAL SETUP

Datasets and evaluation protocol. We evaluate novel-view synthesis on NeRF-Synthetic (eight synthetic scenes) (Mildenhall et al., 2021) and Mip-NeRF 360 (nine real scenes) (Barron et al., 2022). We report PSNR for native outputs and exported vertex-colored meshes, with other metrics in Appendix D. Native results use each method's renderer and appearance model. We evaluate exported meshes in Blender Cycles at 64 and one sample per pixel using an emission shader without scene lighting, and in Google Filament with unlit vertex colors and anti-aliasing disabled.

Baselines. We compare three categories of baselines, (i) mesh extraction from optimized implicit or Gaussian representations, including NeuS2 (Wang et al., 2023), SuGaR (Guédon & Lepetit, 2024), 2DGS (Huang et al., 2024), Gaussian Surfels (Dai et al., 2024), and GOF (Yu et al., 2024b), (ii) anti-aliasing-aware Gaussian methods, including Mip-Splatting (Yu et al., 2024a), AA-2DGS (Younes & Boukhayma, 2026), and Gaussian Blending (Koo et al., 2026), and (iii) hybrid and triangle-based reconstruction, including nvdiffrec (Munkberg et al., 2022), MeshSplats (Tobiasz et al., 2026), Triangle Splatting (Held et al., 2026), and MeshSplatting (Held et al., 2025a). Gaussian Surfels and MeshSplatting use their default external monocular priors, while our method uses none. Appendix Tables 6 and 7 expand comparisons and include matched baseline prior ablations.

4.2 POST-EXPORT RENDERING QUALITY

Tables 2 and 3 compare post-export quality in Blender and Filament on NeRF-Synthetic and Mip-NeRF 360, with native results as a reference. Most baselines lose substantial fidelity when their optimized representation is exported and rendered as an opaque mesh. The anti-aliasing-aware Gaussian baselines show that gains in native quality from anti-aliasing do not carry through mesh extraction. MeshSplatting also produces opaque meshes, yielding a small export gap. However, MeshSplatting's transition from semitransparent to opaque faces during training (Held et al., 2025a) can degrade native quality, while its post-export fidelity remains comparable to that of other leading baselines despite its Metric3D normal prior. Our renderer keeps faces opaque throughout optimization, retaining competitive native quality and a small export gap while achieving the highest post-export fidelity on both datasets. This advantage persists with one Cycles sample per pixel and with Filament anti-aliasing disabled. On NeRF-Synthetic, our meshes take 24 minutes to optimize after initialization and render at 849 FPS at 800×800 with Filament on an RTX A6000.

The Lego and Room crops in Tables 2 and 3 illustrate these differences in the exported meshes. 2DGS exhibits softened boundaries, GOF retains sharper geometry but yields mottled vertex colors,

Table 2: Novel-view-synthesis PSNR on NeRF-Synthetic. Post-export scores reflect mesh fidelity in standard rendering pipelines, which higher native scores need not preserve. Superscripts denote samples per pixel, and Gap is Native minus Blender^{64}. Bold and underline indicate the best and second-best results, respectively.

Method	Native	Blender^{64}	Blender^{1}	Filament	Gap ↓
NeuS2	29.47	22.65	22.26	22.29	6.82
SuGaR	31.28	22.03	21.43	21.43	9.25
2DGS	31.90	21.96	21.76	21.77	9.94
Gaussian Surfels	30.27	22.44	21.83	21.83	7.83
GOF	33.25	<u>26.01</u>	24.76	24.61	7.24
Mip-Splatting	33.42	20.68	20.23	20.22	12.74
AA-2DGS	33.39	21.90	21.67	21.67	11.49
Gaussian Blending	33.78	23.08	22.29	22.29	10.70
nvdiffrec	28.91	22.38	21.99	21.94	6.53
MeshSplats	29.51	20.35	19.50	19.41	9.16
Triangle Splatting	30.80	16.67	15.61	15.61	14.13
MeshSplatting	29.42	25.98	<u>25.03</u>	<u>25.11</u>	<u>3.44</u>
SPIRE (Ours)	32.28	**29.03**	**27.17**	**27.18**	**3.25**

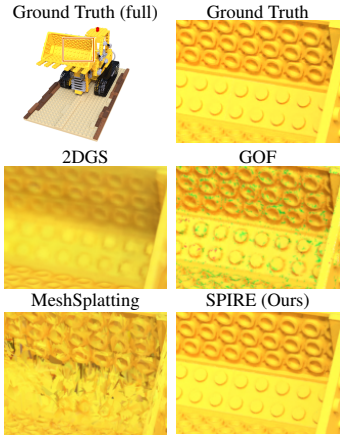
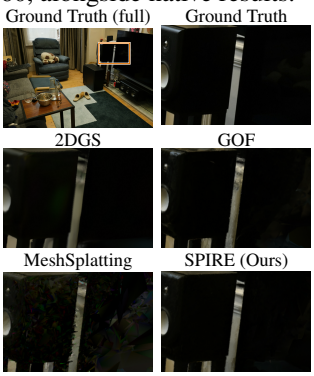


Table 3: Post-export PSNR in Blender and Filament on Mip-NeRF 360, alongside native results.

Method	Native	Blender^{64}	Blender^{1}	Filament	Gap ↓
2DGS	26.83	17.61	17.55	17.51	9.22
Gaussian Surfels	22.53	15.51	15.43	15.45	7.02
GOF	27.22	18.63	18.11	18.32	8.59
Mip-Splatting	27.98	16.72	16.64	16.58	11.26
AA-2DGS	27.44	16.58	16.51	16.46	10.86
Gaussian Blending	27.00	15.14	15.10	15.06	11.86
MeshSplats	25.02	17.48	17.14	17.13	7.54
Triangle Splatting	27.10	14.92	14.27	14.27	12.18
MeshSplatting	24.69	<u>21.17</u>	<u>20.73</u>	<u>20.82</u>	<u>3.52</u>
SPIRE (Ours)	26.23	**23.28**	**22.42**	**22.49**	**2.95**



and MeshSplatting shows conspicuous triangular facets. Our mesh preserves sharper boundaries and local color detail more consistently.

4.3 Controlled renderer comparisons and ablations

Table 4(a) evaluates the contributions of our renderer components and regularization on NeRF-Synthetic. Exact-face computes exact coverage over a fixed full-pixel footprint and composites faces as independent alpha layers. Adding grouping removes same-surface attenuation while keeping the footprint fixed, and Full additionally propagates an approximate residual visible footprint between surfaces. Hard z-buffer uses center-sampled binary coverage. Color-only freezes geometry, while the other variants jointly optimize geometry and appearance.

Unlike hard z-buffer's unstable geometry gradients from color interpolation alone, Full's coverage gradients improve fidelity over color-only fitting. Grouping improves fidelity, and footprint propagation adds further gains by accounting for spatial occlusion between surfaces. Depth and face-area regularization also improve Blender fidelity.

Table 4(b) compares renderers from identical initial meshes with fixed topology under color-only and joint optimization. Color-only fitting yields similar Blender PSNR across renderers, whereas joint optimization reveals larger differences. Even without depth and face-area regularization, our renderer outperforms the alternatives under joint optimization, supporting its advantage beyond initialization and regularization.

Table 4: Controlled comparisons on NeRF-Synthetic, reporting mean PSNR. (a) Ablations of our renderer components and regularization. DC disables higher-order SH terms. (b) Renderer comparison from identical initial meshes with fixed topology, evaluated with Blender^{64} PSNR.

(a) Ablations

Method	Native	DC	Blender^{64}
Hard z-buffer, joint	23.40	21.80	22.32
Hard z-buffer, color only	27.22	25.78	26.73
Exact-face	31.81	29.03	28.43
Exact-face + grouping	31.85	29.06	<u>28.49</u>
Full, color only	29.09	26.76	26.78
Full w/o depth and area	<u>32.10</u>	<u>29.34</u>	28.40
Full	**32.28**	**29.38**	**29.03**

(b) Renderer comparison

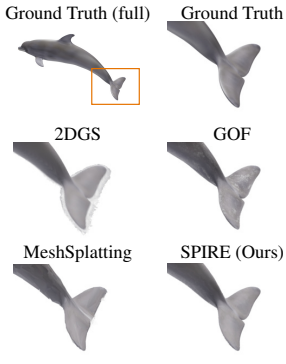
Renderer	Color-only	Joint
MeshSplatting	25.12	20.46
Triangle Splatting	25.65	13.91
nvdiffrast	**26.80**	<u>27.83</u>
MeshSplats	26.74	26.90
SuGaR	24.99	24.55
SPIRE (Ours)	<u>26.78</u>	**29.03**

Table 5: Downstream mesh editing and collision simulation. (a,b) PAPR-in-Motion editing PSNR and Dolphin examples. (c) Ficus ball collisions on reconstructed and original meshes.

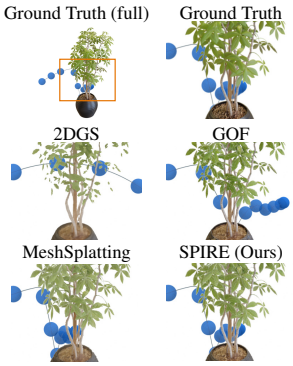
(a) Mesh editing quality

Method	Blender^{64}	Blender^{1}	Filament
NeuS2	24.42	24.06	24.04
SuGaR	26.25	25.72	25.69
2DGS	26.08	25.83	25.84
Gaussian Surfels	24.62	24.18	24.18
GOF	<u>27.87</u>	<u>27.09</u>	<u>27.09</u>
Mip-Splatting	24.52	24.09	24.09
AA-2DGS	25.74	25.53	25.53
Gaussian Blending	23.15	22.72	22.72
nvdiffrec	25.13	24.81	24.82
MeshSplats	22.87	22.24	22.16
Triangle Splatting	17.04	16.22	16.22
MeshSplatting	26.68	26.04	26.07
SPIRE (Ours)	**28.80**	**28.13**	**28.13**

(b) Dolphin deformation



(c) Ficus ball simulation



4.4 Downstream Mesh Editing and Simulation

We evaluate exported meshes for mesh editing and collision simulation. On six scenes from PAPR-in-Motion (Peng et al., 2024), we train each method on start-state views and apply the same deformation to its exported mesh. Table 5 reports editing PSNR against end-state images, which are not used for fitting. SPIRE achieves the highest mean PSNR across the three rendering settings. The Dolphin crops show that baseline geometry and color artifacts persist after editing, while our mesh retains a cleaner tail outline and appearance. In Ficus, each reconstructed mesh acts as a fixed collider for the same launched ball. The ball passes through thin leaf surfaces missing from the 2DGS mesh, while our mesh deflects it similarly to the original geometry.

5 Conclusion

We presented SPIRE, a differentiable triangle-mesh renderer based on analytic pixel-area integration and surface-aware compositing. By integrating adjacent faces as one surface and tracking visibility between surfaces, it provides geometry gradients without softening opaque meshes. Applied to calibrated multiview reconstruction, it directly optimizes reusable vertex-colored meshes from RGB supervision without external normal, depth, or diffusion priors. Our meshes achieve the highest post-export fidelity across the evaluated Blender and Filament settings on synthetic and real scenes while retaining small export gaps. These results support optimizing the mesh itself to preserve image fidelity in standard graphics workflows.

Limitations and future work. Physically based relighting would require separating reflectance from illumination. Future work includes explicit material modeling and adaptive topology to allocate vertices according to local geometric complexity.

AI USE STATEMENT

Generative AI tools were used to assist with hypothesis development and validation, implementation and experimentation, and manuscript writing. They were also used to supplement the authors' own review of related work and verification of mathematical expressions and derivations. The authors take responsibility for verifying AI-assisted analyses, numbers, text, claims, and artifacts against the underlying evidence and for the final content of this work.

ETHICS STATEMENT

This work uses publicly available synthetic and real reconstruction benchmarks, collects no human-subject data, and releases no new dataset containing personally identifiable information.

REPRODUCIBILITY STATEMENT

Section 3 describes the renderer and its differentiation. Section 4 specifies the datasets, evaluation protocol, and reported comparisons. Appendix C provides the initialization, optimization, and regularization settings for mesh reconstruction. We will release our implementation and evaluation scripts.

REFERENCES

Autodesk, Inc. Autodesk maya, 2026. URL `https://www.autodesk.com/products/maya/overview`. Version 2027, computer software.

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. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 5835–5844. IEEE, 2021.

Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5460–5469. IEEE, 2022.

Blender Development Team. *Blender*. Blender Foundation, Amsterdam, The Netherlands, 2026. URL `https://www.blender.org`. Version 5.0.1, computer software.

David Charatan, Daniel Xu, Richard Szeliski, George Kopanas, and Vincent Sitzmann. Meshtryoshka: Differentiable rendering of real-world scenes via mesh rasterization. *arXiv preprint arXiv:2606.28622*, 2026.

Danpeng Chen, Hai Li, Weicai Ye, Yifan Wang, Weijian Xie, Shangjin Zhai, Nan Wang, Haomin Liu, Hujun Bao, and Guofeng Zhang. Pgsr: Planar-based gaussian splatting for efficient and high-fidelity surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 31 (9):6100–6111, 2024.

Jun-Kun Chen, Jipeng Lyu, and Yu-Xiong Wang. Neuraleditor: Editing neural radiance fields via manipulating point clouds. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12439–12448. IEEE, 2023a.

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.

Zhiqin Chen, Thomas Funkhouser, Peter Hedman, and Andrea Tagliasacchi. Mobilenerf: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 16569–16578. IEEE, 2023b.

Jinkai Cui, Kaiwen Song, Chumeng Niu, and Juyong Zhang. Distance field rasterization for end-to-end mesh reconstruction. In *Proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers*, pp. 1–11, 2026.

Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pp. 303–312, 1996.

Pinxuan Dai, Jiamin Xu, Wenxiang Xie, Xinguo Liu, Huamin Wang, and Weiwei Xu. High-quality surface reconstruction using gaussian surfels. In *ACM SIGGRAPH 2024 conference papers*, pp. 1–11, 2024.

Akio Doi and Akio Koide. An efficient method of triangulating equi-valued surfaces by using tetrahedral cells. *IEICE TRANSACTIONS on Information and Systems*, 74(1):214–224, 1991.

Epic Games, Inc. Unreal engine, 2026. URL `https://www.unrealengine.com`. Version 5.8, computer software.

Antoine Guédon and Vincent Lepetit. Sugar: Surface-aligned gaussian splatting for efficient 3d mesh reconstruction and high-quality mesh rendering. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5354–5363. IEEE, 2024.

Antoine Guédon, Diego Gomez, Nissim Maruani, Bingchen Gong, George Drettakis, and Maks Ovsjanikov. Milo: Mesh-in-the-loop gaussian splatting for detailed and efficient surface reconstruction. *ACM Transactions on Graphics (TOG)*, 44(6):1–15, 2025.

Jan Held, Sanghyun Son, Renaud Vandeghen, Daniel Rebain, Matheus Gadelha, Yi Zhou, Anthony Cioppa, Ming C Lin, Marc Van Droogenbroeck, and Andrea Tagliasacchi. Meshsplatting: Differentiable rendering with opaque meshes. *arXiv preprint arXiv:2512.06818*, 2025a.

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 preprint arXiv:2509.25122*, 2025b.

Jan Held, Renaud Vandeghen, Adrien Deliege, Abdullah Hamdi, Silvio Giancola, Daniel Rebain, Anthony Cioppa, Bernard Ghanem, Andrea Vedaldi, Andrea Tagliasacchi, et al. Triangle splatting for real-time radiance field rendering. In *2026 International Conference on 3D Vision (3DV)*, pp. 1248–1257. IEEE, 2026.

Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger, and Shenghua Gao. 2d gaussian splatting for geometrically accurate radiance fields. In *ACM SIGGRAPH 2024 conference papers*, pp. 1–11, 2024.

Wonbong Jang, Philippe Weinzaepfel, Vincent Leroy, Lourdes Agapito, and Jerome Revaud. Pow3r: Empowering unconstrained 3d reconstruction with camera and scene priors. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1071–1081. IEEE, 2025.

Rasmus Jensen, Anders Dahl, George Vogiatzis, Engin Tola, and Henrik Aanæs. Large scale multi-view stereopsis evaluation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 406–413, 2014.

Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3d mesh renderer. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3907–3916. Ieee, 2018.

Michael Kazhdan, Matthew Bolitho, Hugues Hoppe, et al. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, volume 7, 2006.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, George Drettakis, et al. 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 42(4):139–1, 2023.

Junseo Koo, Jinseo Jeong, and Gunhee Kim. Gaussian blending: Rethinking alpha blending in 3d gaussian splatting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 5744–5754, 2026.

Samuli Laine, Janne Hellsten, Tero Karras, Yeongho Seol, Jaakko Lehtinen, and Timo Aila. Modular primitives for high-performance differentiable rendering. *ACM Transactions on Graphics (ToG)*, 39(6):1–14, 2020.

Tzu-Mao Li, Michal Lukáč, Michaël Gharbi, and Jonathan Ragan-Kelley. Differentiable vector graphics rasterization for editing and learning. *ACM Transactions on Graphics (TOG)*, 39(6): 1–15, 2020.

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 *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8456–8465. IEEE, 2023.

Zhihao Liang, Qi Zhang, Wenbo Hu, Lei Zhu, Ying Feng, and Kui Jia. Analytic-splatting: Anti-aliased 3d gaussian splatting via analytic integration. In *European conference on computer vision*, pp. 281–297. Springer, 2024.

Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. Soft rasterizer: A differentiable renderer for image-based 3d reasoning. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 7708–7717, 2019.

Matthew M Loper and Michael J Black. Opendr: An approximate differentiable renderer. In *European conference on computer vision*, pp. 154–169. Springer, 2014.

William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Seminal graphics: pioneering efforts that shaped the field*, pp. 347–353. 1998.

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.

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*, pp. 8280–8290, 2022.

Baptiste Nicolet, Alec Jacobson, and Wenzel Jakob. Large steps in inverse rendering of geometry. *ACM Transactions on Graphics (TOG)*, 40(6):1–13, 2021.

Michael Oechsle, Songyou Peng, and Andreas Geiger. Unisurf: Unifying neural implicit surfaces and radiance fields for multi-view reconstruction. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 5569–5579. IEEE, 2021.

Shichong Peng, Yanshu Zhang, and Ke Li. Papr in motion: Seamless point-level 3d scene interpolation. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 21007–21016. IEEE, 2024.

Stanislav Pidhorskyi, Tomas Simon, Gabriel Schwartz, He Wen, Yaser Sheikh, and Jason Saragih. Rasterized edge gradients: Handling discontinuities differentiably. In *European Conference on Computer Vision*, pp. 335–352. Springer, 2024.

Christian Reiser, Stephan Garbin, Pratul Srinivasan, Dor Verbin, Richard Szeliski, Ben Mildenhall, Jonathan Barron, Peter Hedman, and Andreas Geiger. Binary opacity grids: Capturing fine geometric detail for mesh-based view synthesis. *ACM Transactions on Graphics (TOG)*, 43(4):1–14, 2024.

Tianchang Shen, Jacob Munkberg, Jon Hasselgren, Kangxue Yin, Zian Wang, Wenzheng Chen, Zan Gojcic, Sanja Fidler, Nicholas Sharp, and Jun Gao. Flexible isosurface extraction for gradient-based mesh optimization. *ACM Transactions on Graphics (ToG)*, 42(4):1–16, 2023.

Kaifeng Sheng, Zheng Zhou, Yingliang Peng, and Qianwei Wang. 2d triangle splatting for direct differentiable mesh training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 285–294, 2026.

Ivan E Sutherland and Gary W Hodgman. Reentrant polygon clipping. *Communications of the ACM*, 17(1):32–42, 1974.

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. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 17693–17703. IEEE, 2023.

Rafał Tobiasz, Grzegorz Wilczyński, Marcin Mazur, Sławomir Tadeja, Weronika Smolak-Dyżewska, and Przemysław Spurek. Meshsplats: Mesh-based rendering with gaussian splatting initialization. In *International Conference on Computational Science*, pp. 386–394. Springer, 2026.

Kenji Tojo, Bernd Bickel, and Nobuyuki Umetani. Diffsoup: direct differentiable rasterization of triangle soup for extreme radiance field simplification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8353–8363, 2026.

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.

Shuzhe Wang, Vincent Leroy, Yohann Cabon, Boris Chidlovskii, and Jerome Revaud. Dust3r: Geometric 3d vision made easy. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 20697–20709. IEEE, 2024.

Yiming Wang, Qin Han, Marc Habermann, Kostas Daniilidis, Christian Theobalt, and Lingjie Liu. Neus2: Fast learning of neural implicit surfaces for multi-view reconstruction. In *2023 IEEE/CVF international conference on computer vision (ICCV)*, pp. 3272–3283. IEEE, 2023.

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.

Kaizhi Yang, Liu Dai, Isabella Liu, Xiaoshuai Zhang, Xiaoyan Sun, Xuejin Chen, Zexiang Xu, and Hao Su. Imls-splatting: Efficient mesh reconstruction from multi-view images via point representation. *ACM Transactions on Graphics (TOG)*, 44(4):1–11, 2025.

Lior Yariv, Yoni Kasten, Dror Moran, Meirav Galun, Matan Atzmon, Basri Ronen, and Yaron Lipman. Multiview neural surface reconstruction by disentangling geometry and appearance. *Advances in Neural Information Processing Systems*, 33:2492–2502, 2020.

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.

Lior Yariv, Peter Hedman, Christian Reiser, Dor Verbin, Pratul P Srinivasan, Richard Szeliski, Jonathan T Barron, and Ben Mildenhall. Bakedsdf: Meshing neural sdfs for real-time view synthesis. In *ACM SIGGRAPH 2023 conference proceedings*, pp. 1–9, 2023.

Mae Younes and Adnane Boukhayma. Anti-aliased 2d gaussian splatting. *Advances in Neural Information Processing Systems*, 38:9767–9792, 2026.

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

Zehao Yu, Torsten Sattler, and Andreas Geiger. Gaussian opacity fields: Efficient adaptive surface reconstruction in unbounded scenes. *ACM Transactions on Graphics (ToG)*, 43(6):1–13, 2024b.

Baowen Zhang, Chuan Fang, Rakesh Shrestha, Yixun Liang, Xiao-Xiao Long, and Ping Tan. Radegs: Rasterizing depth in gaussian splatting. *ACM Transactions on Graphics*, 45(2):1–14, 2026a.

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *2018 IEEE/CVF conference on computer vision and pattern recognition*, pp. 586–595. IEEE, 2018.

Ruiqi Zhang, Jiacheng Wu, and Jie Chen. Mesh splatting for end-to-end multiview surface reconstruction. *arXiv preprint arXiv:2601.21400*, 2026b.

Yang Zhou, Lifan Wu, Ravi Ramamoorthi, and Ling-Qi Yan. Vectorization for fast, analytic, and differentiable visibility. *ACM Transactions on Graphics (TOG)*, 40(3):1–21, 2021.

Appendix contents

A Motivation and Comparison of Rendering Approaches 14
A.1 Pixel-area integration 14
A.2 Surface-aware compositing 16
A.3 Coverage and visibility sampling 17
A.4 Mesh extraction and the export gap 17
A.5 Softened and opaque faces 17

B Polygon Moments, Residual Footprints, and Sparse Differentiation 17
B.1 Closed-form polygon moments 17
B.2 Residual-footprint moments 18
B.3 Two-sweep sparse backward 18

C Implementation and Evaluation Details 20

D Additional Results 22
D.1 Full STST comparisons 22
D.2 Renderer comparison under fixed mesh topology 23
D.3 Full MTMT comparisons 23
D.4 Resolution-wise MTMT results 23
D.5 Geometry 24
D.6 Runtime and resource requirements 25
D.7 Blender Cycles at one sample per pixel 25
D.8 Rendering with Google Filament 26
D.9 Scene-wise PSNR 27
D.10 Qualitative results 27
D.11 PSNR-gap decomposition 27

E Downstream Editing and Simulation 35
E.1 Endpoint mesh deformation 35
E.2 NeuralEditor deformation benchmark 35
E.3 Ball collisions with thin geometry 35
E.4 Articulation and appearance editing 35
E.5 Outdoor scene composition 35
E.6 Liquid interaction 35

A Motivation and Comparison of Rendering Approaches

Figure 3 illustrates how pixel-area integration enables differentiable opaque-mesh rendering and how existing approaches can introduce an export gap.

A.1 Pixel-area integration

A conventional mesh renderer makes binary coverage and visibility decisions at discrete samples. A real camera instead integrates radiance over each pixel, so vertex motion changes the image through the visible areas of faces. Let W_1 be the full pixel footprint and f_j^{Π} the projected region of face j, indexed from front to back. For opaque faces with consistent depth order over projected overlaps, the exact visible region and coverage are

$$R_j = (W_1 \cap f_j^{\Pi}) \setminus \bigcup_{k<j} f_k^{\Pi}, \qquad \alpha_j = \frac{\mathrm{Area}(R_j)}{\mathrm{Area}(W_1)}. \tag{8}$$

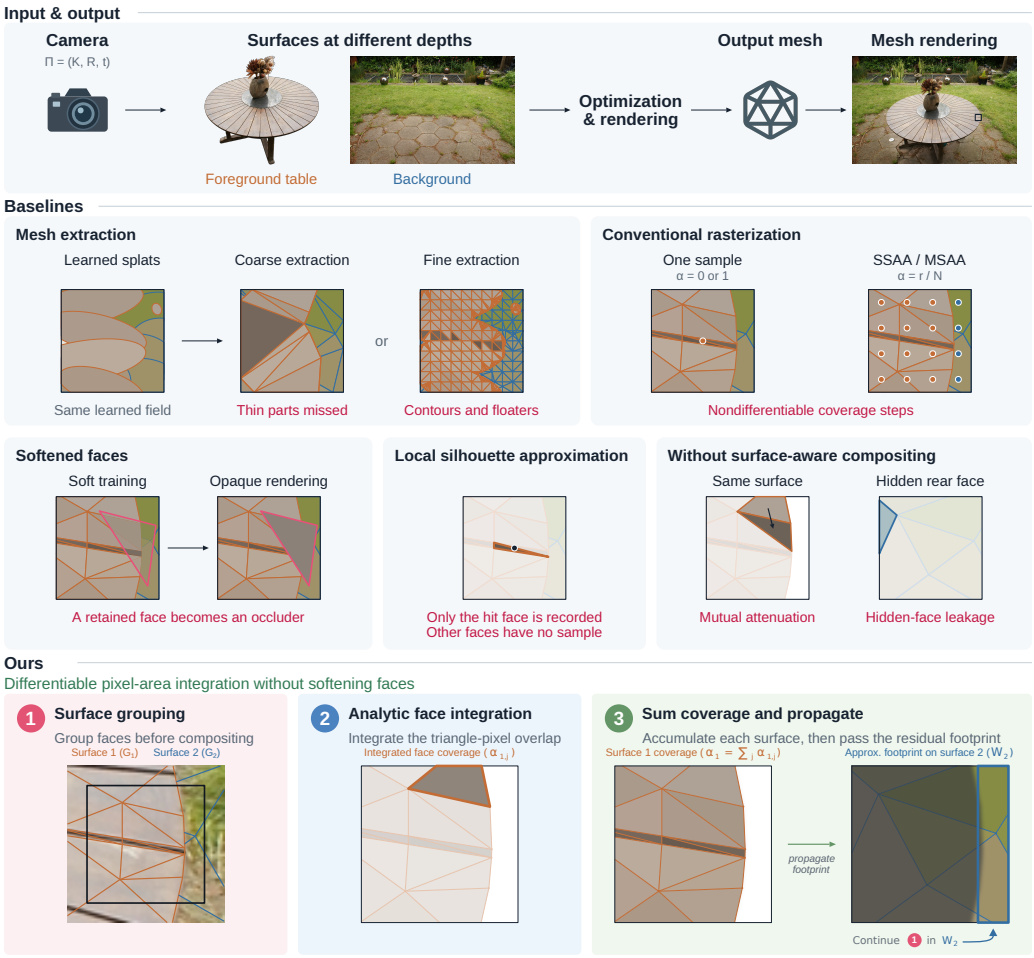


Figure 3: **Comparison of mesh reconstruction and rendering approaches.** Using the table-edge example of Figure 2, we illustrate how extraction can miss thin structures or retain primitive contours, finite sampling yields discrete coverage, and soft-to-opaque conversion can introduce unintended occluders. Local silhouette approximations can miss unsampled faces, while compositing without surface awareness can cause mutual attenuation within a surface or contributions from hidden faces. SPIRE (Ours) integrates coverage and color over the pixel area without softening faces, accumulating adjacent faces within each surface and propagating the residual footprint between surfaces to enable differentiable opaque-mesh rendering.

With a fixed camera, uniform pixel response, and fixed background color $\mathbf{C}_{\mathrm{bg}}$, the integrated pixel color is

$$\begin{aligned}\widehat{\mathbf{c}} &= \mathbf{C}_{\mathrm{bg}} + \frac{1}{\mathrm{Area}(W_1)} \sum_j \int_{R_j} (\mathbf{c}_j(\mathbf{x}) - \mathbf{C}_{\mathrm{bg}})\, d\mathbf{x} \\ &= \mathbf{C}_{\mathrm{bg}} + \frac{1}{\mathrm{Area}(W_1)} \sum_j \mathrm{Area}(R_j)(\mathbf{C}_j^F - \mathbf{C}_{\mathrm{bg}}) \\ &= \sum_j \alpha_j \mathbf{C}_j^F + \left(1 - \sum_j \alpha_j\right) \mathbf{C}_{\mathrm{bg}},\end{aligned} \tag{9}$$

where $\mathbf{c}_j(\mathbf{x})$ is the RGB radiance from face j and $\mathbf{C}_j^F$ is its mean over the nonempty visible region R_j. For color affine in screen space, including barycentric color interpolation, $\mathbf{C}_j^F = \mathbf{c}_j(\boldsymbol{\mu}_j)$, the

color at the area centroid $\boldsymbol{\mu}_j$ of R_j. This follows because averaging an affine function replaces its coordinates by their area-weighted mean, including for disconnected regions and regions with holes.

Even if R_j contains disconnected polygons or holes, its area is a signed sum over boundary loops,

$$A_j = \mathrm{Area}(R_j) = \frac{1}{2} \sum_{\ell} \sum_{r} (x_{\ell r} y_{\ell,r+1} - x_{\ell,r+1} y_{\ell r}), \tag{10}$$

where $(x_{\ell r}, y_{\ell r})$ are cyclically ordered vertices of loop ℓ, counterclockwise for outer boundaries and clockwise for holes. For nondegenerate projection and fixed intersection and depth-order configurations, these boundary coordinates are differentiable in the mesh vertices. The chain rule therefore gives $\partial \alpha_j / \partial v_{i,d} = \mathrm{Area}(W_1)^{-1} \partial A_j / \partial v_{i,d}$ for coordinate d of vertex $\mathbf{v}_i$.

Differentiating the area-weighted color gives

$$\frac{\partial \widehat{\mathbf{c}}}{\partial v_{i,d}} = \sum_j \left[(\mathbf{C}_j^F - \mathbf{C}_{\mathrm{bg}}) \frac{\partial \alpha_j}{\partial v_{i,d}} + \alpha_j \frac{\partial \mathbf{C}_j^F}{\partial v_{i,d}} \right]. \tag{11}$$

The two terms account for changes in visible coverage and mean color. For affine face color, the centroid and its color are differentiable under the same regularity conditions, giving geometry gradients despite binary pointwise visibility.

A.2 Surface-aware compositing

Maintaining every component and hole of R_j requires repeated polygon intersections and differences. For example, N sufficiently thin projected faces with different orientations can be arranged like lines to divide a pixel into $N(N+1)/2 + 1$ disconnected visible regions. We simplify this computation by grouping adjacent faces into surfaces and approximating residual visibility between surfaces. Faces that tessellate a locally continuous surface generally meet at shared edges without overlapping in projected area. Under this assumption, their coverages add without mutual occlusion.

Gaussian Blending uses the spatial coherence of primitives representing continuous object surfaces to approximate transmittance by a 2D uniform distribution (Koo et al., 2026). We adopt this assumption at the surface level. Near a typical surface boundary, adjacent faces cover a contiguous region, leaving visibility mainly beside that surface rather than through many isolated openings. This motivates approximating the remaining transmittance by an axis-aligned rectangle W_s with uniform density and total mass τ_s. Initially, W_1 is the full pixel footprint and $\tau_1 = 1$.

For surface $\mathcal{G}_s$, each face with opacity $o_j \in [0, 1]$ is clipped to the active footprint and its opacity-weighted coverage is accumulated as

$$P_{s,j} = f_j^{\Pi} \cap W_s, \qquad \alpha_{s,j} = o_j \frac{\mathrm{Area}(P_{s,j})}{\mathrm{Area}(W_s)}, \qquad \alpha_s = \sum_{j \in \mathcal{G}_s} \alpha_{s,j}. \tag{12}$$

Coverage and interpolated color are integrated analytically over $P_{s,j}$. Both same-surface accumulation and the mass update $\tau_{s+1} = \tau_s (1 - \alpha_s)$ preserve differentiability.

Subtracting the opacity-weighted face moments from the incoming moments gives the residual mass, center $\mathbf{u}^{(s+1)}$, and component-wise variance $\boldsymbol{\upsilon}^{(s+1)}$. Following Gaussian Blending, moment matching sets the next rectangle's half-width to $\boldsymbol{\rho}^{(s+1)} = \sqrt{3 \boldsymbol{\upsilon}^{(s+1)}}$,

$$W_{s+1} = [u_x^{(s+1)} - \rho_x^{(s+1)}, u_x^{(s+1)} + \rho_x^{(s+1)}] \times [u_y^{(s+1)} - \rho_y^{(s+1)}, u_y^{(s+1)} + \rho_y^{(s+1)}]. \tag{13}$$

These moment-subtraction, normalization, and square-root operations define the forward footprint update. Appendix B.2 gives the formulas, and Appendix B.3 describes their treatment in the backward pass. Faces of the next surface are then integrated only over their overlap with W_{s+1}, and this process repeats in depth order. The footprint approximates residual visibility, while face integration within it remains exact.

The same integration also handles texture detail represented by vertex colors within a single surface. All intersecting faces contribute their integrated colors without mutual attenuation, even when no silhouette crosses the pixel. With multiple surfaces, this color integration is restricted to each surface's active footprint before compositing.

A.3 Coverage and visibility sampling

A conventional mesh renderer assigns each sample to the nearest covering face. With N fixed, equally weighted samples, the estimated visible coverage is

$$\alpha_{N,j} = \frac{1}{N}\sum_{k=1}^{N} \mathbf{1}_{R_j}(\mathbf{x}_k) = \frac{r_j}{N}, \tag{14}$$

where r_j counts samples at which face j is visible. Supersampling and multisample anti-aliasing reduce aliasing, but coverage still changes in integer steps and has zero derivatives almost everywhere. Barycentric color interpolation supplies geometry gradients within the selected face, without accounting for visibility changes.

nvdiffrast approximates boundary derivatives using projected silhouette edges and neighboring point-sampled colors (Laine et al., 2020). The center sample in Figure 3 records only its visible face. Neighboring samples provide additional information, but subpixel faces missed by all samples can also be missed during anti-aliasing. Direct triangle-pixel integration accounts for these faces through their covered areas.

A.4 Mesh extraction and the export gap

Neural-field and Gaussian methods optimize representation-specific rendering, then obtain meshes through conversion (Mildenhall et al., 2021; Kerbl et al., 2023; Huang et al., 2024; Yu et al., 2024b). This conversion can use techniques such as Marching Cubes, Marching Tetrahedra, or Poisson surface reconstruction (Lorensen & Cline, 1998; Doi & Koide, 1991; Kazhdan et al., 2006). When extraction is deferred to post-processing, image losses do not directly supervise the final mesh. Low-resolution grids can miss thin structures, while high-resolution grids can retain primitive contours, surface noise, and detached fragments, as illustrated in Figure 3. Displaced vertices can also sample colors from incorrect locations.

SPIRE differentiably renders opaque meshes through pixel-area integration and surface-aware compositing. Gradients of the image loss therefore flow directly to mesh vertex positions and appearance, without an intervening extraction step during optimization.

A.5 Softened and opaque faces

Soft rasterization and triangle splatting obtain geometry gradients through softened coverage or semitransparent faces (Liu et al., 2019; Held et al., 2026; 2025a). A misplaced face can contribute weakly during training yet occlude a surface behind it when made opaque, producing a triangular artifact without changing position. Figure 3 illustrates this transition. MeshSplatting gradually increases opacity until the mesh becomes opaque (Held et al., 2025a). SPIRE instead renders opaque meshes differentiably throughout optimization, without a later transition from softened or semitransparent faces.

B Polygon Moments, Residual Footprints, and Sparse Differentiation

B.1 Closed-form polygon moments

The renderer uses the area, centroid, and component-wise raw second moment of each clipped polygon. Let the ordered vertices of a nondegenerate polygon P be $\mathbf{q}_k = (x_k, y_k)$, with cyclic indexing, and define

$$\Delta_k = x_k y_{k+1} - x_{k+1} y_k, \qquad A_P = \frac{1}{2}\sum_k \Delta_k. \tag{15}$$

Here, A_P is the signed area for the chosen vertex order, and the nonnegative area used for coverage in Section 3 is $|A_P|$. Using the same orientation throughout, the centroid is

$$\boldsymbol{\mu}_P = \frac{1}{6A_P}\sum_k \begin{bmatrix} x_k + x_{k+1} \\ y_k + y_{k+1} \end{bmatrix} \Delta_k, \tag{16}$$

and the component-wise raw second moment is

$$\boldsymbol{\nu}_P = \frac{1}{12A_P} \sum_k \begin{bmatrix} x_k^2 + x_k x_{k+1} + x_{k+1}^2 \\ y_k^2 + y_k y_{k+1} + y_{k+1}^2 \end{bmatrix} \Delta_k. \tag{17}$$

These expressions give the moments used for barycentric color interpolation and residual-footprint matching.

B.2 Residual-footprint moments

Equation 6 describes the residual visibility after a surface. We evaluate its center and variance by subtracting the moments of the covered polygons. In the expressions below, the parenthesized superscript denotes moment order. The normalized moments of the active footprint are

$$\mathbf{m}_s^{(1)} = \frac{1}{\mathrm{Area}(W_s)} \int_{W_s} \mathbf{x}\, d\mathbf{x}, \qquad \mathbf{m}_s^{(2)} = \frac{1}{\mathrm{Area}(W_s)} \int_{W_s} \mathbf{x} \odot \mathbf{x}\, d\mathbf{x}. \tag{18}$$

The corresponding moments covered by surface s are

$$\mathbf{M}_s^{(1)} = \sum_{j \in \mathcal{G}_s} \alpha_{s,j} \boldsymbol{\mu}_{s,j}, \qquad \mathbf{M}_s^{(2)} = \sum_{j \in \mathcal{G}_s} \alpha_{s,j} \boldsymbol{\nu}_{s,j}. \tag{19}$$

Subtracting the covered portion gives the residual mass and moments

$$r_s^{(0)} = 1 - \sum_{j \in \mathcal{G}_s} \alpha_{s,j}, \qquad \mathbf{r}_s^{(1)} = \mathbf{m}_s^{(1)} - \mathbf{M}_s^{(1)}, \qquad \mathbf{r}_s^{(2)} = \mathbf{m}_s^{(2)} - \mathbf{M}_s^{(2)}. \tag{20}$$

For $r_s^{(0)} > 0$, the residual center and variance are computed as

$$\mathbf{u}^{(s+1)} = \frac{\mathbf{r}_s^{(1)}}{r_s^{(0)}}, \qquad \boldsymbol{\upsilon}^{(s+1)} = \frac{\mathbf{r}_s^{(2)}}{r_s^{(0)}} - \mathbf{u}^{(s+1)} \odot \mathbf{u}^{(s+1)}. \tag{21}$$

If $r_s^{(0)} = 0$, the visible footprint is exhausted and compositing terminates. These expressions evaluate the weighted integrals in Equation 6 using the polygon moments above.

B.3 Two-sweep sparse backward

Surface-aware compositing couples the derivatives of faces that share a surface. All faces in $\mathcal{G}_s$ use the same incoming transmittance mass τ_s, which is updated after their contributions have been accumulated. The backward pass uses two dependent sweeps over the recorded contributions, first computing shared surface-level gradient coefficients and then using them for local face derivatives.

Following Gaussian Blending (Koo et al., 2026), we apply stop-gradient to footprint centers and widths for numerical stability and efficiency, with spatial occlusion already encoded in the active footprints.

Define the total surface coverage and coverage-weighted color sum as

$$\alpha_s = \sum_{j \in \mathcal{G}_s} \alpha_{s,j}, \qquad \mathbf{Q}_s = \sum_{j \in \mathcal{G}_s} \alpha_{s,j} \mathbf{C}_{s,j}^F. \tag{22}$$

Let $\mathbf{B}_s$ be the color composited from all surfaces behind s, including the background, with unit incoming transmittance, and let $\mathbf{c}_{<s}$ contain the contributions from nearer surfaces. Using the recorded footprints, the pixel color can be written as

$$\widehat{\mathbf{c}} = \mathbf{c}_{<s} + \tau_s \left[\mathbf{Q}_s + (1 - \alpha_s) \mathbf{B}_s \right]. \tag{23}$$

Let $\mathbf{g} = \partial \mathcal{L} / \partial \widehat{\mathbf{c}}$ be the loss gradient with respect to the pixel color. Within each smooth clipping and depth-order configuration, the local partial derivatives are

$$g_{s,j} = \frac{\partial \mathcal{L}}{\partial \alpha_{s,j}} = \tau_s\, \mathbf{g}^\top \mathbf{C}_{s,j}^F + K_s, \qquad K_s = -\tau_s\, \mathbf{g}^\top \mathbf{B}_s, \qquad \mathbf{g}_{s,j}^C = \frac{\partial \mathcal{L}}{\partial \mathbf{C}_{s,j}^F} = \tau_s \alpha_{s,j} \mathbf{g}. \tag{24}$$

Every face in a surface shares K_s, which accounts for how its coverage changes the contribution of deeper surfaces.

The first sweep obtains K_s from the accumulated color $\mathbf{c}_{\le s} = \mathbf{c}_{<s} + \tau_s \mathbf{Q}_s$ and the final pixel color. For $\alpha_s < 1$, rearranging the compositing equation gives

$$K_s = \frac{\mathbf{g}^\top (\mathbf{c}_{\le s} - \widehat{\mathbf{c}})}{1 - \alpha_s}. \tag{25}$$

Both $\mathbf{c}_{\le s}$ and α_s require accumulation over all faces in $\mathcal{G}_s$, so K_s becomes available when the first sweep finishes that surface. At each surface boundary, the sweep assigns this coefficient to the completed surface's records and continues accumulating the next surface. Once the first sweep is complete, the second uses the stored coefficients to differentiate the active face-pixel contributions in parallel. Separating the sweeps makes the complete surface accumulation available to every face without repeating it for each local derivative. For the final contributing surface, the background gives $K_s = -\tau_s \mathbf{g}^\top \mathbf{C}_{\mathrm{bg}}$ directly, avoiding division by the remaining mass. For other surfaces, the implementation guards the division when the remaining mass is close to zero.

The first sweep also accounts for the upper limit on accumulated geometric coverage. Let $a_{s,j} = \mathrm{Area}(f_j^{\Pi} \cap W_s)/\mathrm{Area}(W_s)$ be the raw geometric coverage, and let $i \prec j$ denote the recorded face order within a surface. Writing $\widetilde{a}_{s,j}$ for the coverage after applying this limit, the contribution of a face with opacity o_j is

$$\widetilde{a}_{s,j} = \min\left(a_{s,j}, \max\left(0, 1 - \sum_{i \prec j} \widetilde{a}_{s,i}\right)\right), \qquad \alpha_{s,j} = o_j \widetilde{a}_{s,j}. \tag{26}$$

Under the non-overlap assumption in Appendix A, the limit leaves the geometric coverage unchanged. If face m is the first face whose raw coverage exceeds the remaining positive budget, increasing an earlier face's coverage reduces the geometric coverage assigned to face m by the same amount. Away from branch boundaries, the coverage derivatives are

$$\kappa_{s,j} = \frac{\partial \mathcal{L}}{\partial a_{s,j}} = \begin{cases} o_j g_{s,j}, & \text{if no face is budget-clipped,} \\ o_j g_{s,j} - o_m g_{s,m}, & j \prec m, \\ 0, & j = m \text{ or } m \prec j. \end{cases} \tag{27}$$

The opacity derivative is $\partial \mathcal{L}/\partial o_j = \widetilde{a}_{s,j} g_{s,j}$ for this pixel. A face whose coverage is limited still receives opacity and color gradients. Setting $o_j = 1$ recovers the opaque reconstruction case.

The second sweep recomputes local clipping and color interpolation using W_s. Treating projected vertex positions, vertex colors, and face opacities as inputs, let θ denote one of their scalar components and $P_{s,j} = f_j^{\Pi} \cap W_s$. The pixel loss derivative is

$$\frac{\partial \mathcal{L}}{\partial \theta} = \sum_{s=1}^{|\mathcal{S}|} \sum_{j \in \mathcal{G}_s} \left[\frac{\kappa_{s,j}}{\mathrm{Area}(W_s)} \frac{\partial \,\mathrm{Area}(P_{s,j})}{\partial \theta} + g_{s,j} \widetilde{a}_{s,j} \frac{\partial o_j}{\partial \theta} + (\mathbf{g}^C_{s,j})^\top \frac{\partial \mathbf{C}^F_{s,j}}{\partial \theta} \right]. \tag{28}$$

For a clipped polygon $P = P_{s,j}$, let A_P denote its signed area as above and $\mathbf{N}_P = \sum_k (\mathbf{q}_k + \mathbf{q}_{k+1}) \Delta_k$ its centroid numerator. With cyclic indices, the polygon formulas yield

$$\begin{aligned} \frac{\partial A_P}{\partial \mathbf{q}_k} &= \frac{1}{2} \begin{bmatrix} y_{k+1} - y_{k-1} \\ x_{k-1} - x_{k+1} \end{bmatrix}, \quad \mathrm{d}\,\mathrm{Area}(P) = \mathrm{sgn}(A_P)\,\mathrm{d}A_P, \\ \mathrm{d}\boldsymbol{\mu}_P &= \frac{\mathrm{d}\mathbf{N}_P}{6A_P} - \frac{\boldsymbol{\mu}_P}{A_P}\,\mathrm{d}A_P, \qquad \mathrm{d}\mathbf{q}_k = \sum_{\ell=1}^{3} \mathbf{J}_{k,f_{j\ell}}\,\mathrm{d}\mathbf{x}^{\Pi}_{f_{j\ell}}. \end{aligned} \tag{29}$$

Here $\mathbf{x}_i^{\Pi}$ is the projected image position of vertex i and $\mathbf{J}_{k,i} = \partial \mathbf{q}_k / \partial \mathbf{x}_i^{\Pi}$ is the clipping Jacobian. Differentiating the affine face color then gives

$$\mathrm{d}\mathbf{C}^F_{s,j} = \sum_{k=1}^{3} \left[\mathbf{C}^V_{f_{jk}}\,\mathrm{d}\beta_{j,k}(\boldsymbol{\mu}_{s,j}) + \beta_{j,k}(\boldsymbol{\mu}_{s,j})\,\mathrm{d}\mathbf{C}^V_{f_{jk}} \right], \tag{30}$$

Table 6: Full novel-view-synthesis comparison on NeRF-Synthetic under STST. Native rendering is compared with Blender rendering of the exported vertex-colored meshes. The table includes feed-forward conversions and matched runs with and without external monocular priors. PSNR gap is Native minus Blender. MeshSplatting's shared Blender result is shown only in its +SSAA row. Bold and underline mark the smallest and second-smallest gaps, respectively.

Method	Native			Blender			Gap
	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↓
DUSt3R	–	–	–	13.14	0.786	0.301	–
Pow3R	–	–	–	10.90	0.789	0.294	–
NeuS2	29.47	0.946	0.069	22.65	0.904	0.100	6.82
SuGaR	31.28	0.962	0.050	22.03	0.869	0.171	9.25
2DGS	31.90	0.965	0.047	21.96	0.874	0.170	9.94
Gaussian Surfels	29.97	0.950	0.064	22.95	0.890	0.103	7.02
GOF	33.25	0.969	0.038	26.01	0.909	0.093	7.24
PGSR	31.77	0.964	0.044	20.54	0.878	0.139	11.23
Mip-Splatting	33.42	0.969	0.038	20.68	0.876	0.141	12.74
Analytic-Splatting	33.71	0.970	0.036	20.61	0.874	0.142	13.10
AA-2DGS	33.39	0.968	0.041	21.90	0.866	0.142	11.49
Gaussian Blending	33.78	0.970	0.037	23.08	0.876	0.137	10.70
nvdiffrec	28.91	0.939	0.081	22.38	0.863	0.145	6.53
MeshSplats	29.51	0.942	0.077	20.35	0.834	0.196	9.16
Triangle Splatting	30.80	0.954	0.053	16.67	0.771	0.230	14.13
MeshSplatting	26.99	0.924	0.105	–	–	–	–
MeshSplatting (+SSAA)	29.51	0.946	0.070	26.12	0.913	0.102	<u>3.39</u>
With prior							
Gaussian Surfels	30.27	0.954	0.059	22.44	0.888	0.102	7.83
MeshSplatting	26.95	0.923	0.105	–	–	–	–
MeshSplatting (+SSAA)	29.42	0.945	0.072	25.98	0.912	0.102	3.44
SPIRE (Ours)	32.28	0.964	0.044	29.03	0.944	0.067	**3.25**

where $\mathrm{d}\beta_{j,k}$ includes changes in both the projected triangle and the polygon centroid. These terms give the local derivatives with respect to projected vertex positions and vertex colors. The second sweep combines these local derivatives with the shared coefficients and accumulates gradients at shared vertices across pixels.

We store only active face-pixel contributions in $\mathcal{Z}$, together with their footprints and incoming transmittance. Recomputing local clipping avoids storing every polygon and its Jacobian, while excluding inactive pairs keeps storage and replay work proportional to the active contributions and per-pixel indexing.

C Implementation and Evaluation Details

Initialization and optimization. We use GOF only for initialization. For NeRF-Synthetic, Multiscale Blender, and Mip-NeRF 360, we run GOF for 30k iterations and initialize our mesh with its released marching-tetrahedra extraction (Yu et al., 2024b). Before extraction, we subsample the trained GOF representation using target mesh sizes of approximately one million vertices for the two synthetic datasets and 12 million vertices for Mip-NeRF 360. For DTU, we instead initialize our mesh by fusing GOF-rendered depth maps with a truncated signed distance function (TSDF) (Curless & Levoy, 1996). We jointly optimize mesh geometry and appearance using SPIRE for 15k iterations, preserving connectivity and vertex count. The vertex learning rate decays exponentially from 2.5×10^{-4} to 2.5×10^{-5}. The degree-zero and higher-order spherical harmonic learning rates are 1.6×10^{-3} and 8×10^{-5}, respectively.

Common rendering and metrics. We refer to the main-paper evaluation as single-scale training and testing (STST), with training and testing performed at the same base resolution. We use 800×800 images for NeRF-Synthetic and the provided quarter-resolution outdoor and half-

Table 7: Full novel-view-synthesis comparison on Mip-NeRF 360 under STST. Values are nine-scene means at the standard resolution for each scene, including matched runs with and without external monocular priors. PSNR gap is Native minus Blender. MeshSplatting's shared Blender result is shown only in its +SSAA row. Bold and underline mark the smallest and second-smallest gaps, respectively.

Method	Native			Blender			Gap
	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↓
2DGS	26.83	0.797	0.298	17.61	0.463	0.594	9.22
Gaussian Surfels	22.01	0.671	0.450	15.08	0.371	0.640	6.93
GOF	27.22	0.821	0.239	18.63	0.546	0.467	8.59
PGSR	27.19	0.831	0.224	14.51	0.392	0.556	12.68
Mip-Splatting	27.98	0.838	0.214	16.72	0.449	0.591	11.26
Analytic-Splatting	27.76	0.824	0.245	16.79	0.450	0.596	10.97
AA-2DGS	27.44	0.815	0.269	16.58	0.438	0.613	10.86
Gaussian Blending	27.00	0.808	0.282	15.14	0.414	0.626	11.86
MeshSplats	25.02	0.751	0.315	17.48	0.410	0.632	7.54
Triangle Splatting	27.10	0.813	0.226	14.92	0.271	0.652	12.18
MeshSplatting	23.33	0.662	0.424	–	–	–	–
MeshSplatting (+SSAA)	24.69	0.731	0.362	21.20	0.604	0.457	<u>3.49</u>
With prior							
Gaussian Surfels	22.53	0.686	0.433	15.51	0.383	0.625	7.02
MeshSplatting	23.36	0.662	0.425	–	–	–	–
MeshSplatting (+SSAA)	24.69	0.729	0.365	21.17	0.601	0.461	3.52
SPIRE (Ours)	26.23	0.806	0.241	23.28	0.690	0.318	**2.95**

resolution indoor images for Mip-NeRF 360. We additionally evaluate multi-scale training and testing (MTMT) on Multiscale Blender and Mip-NeRF 360 at scales $\{\times 1, \times 1/2, \times 1/4, \times 1/8\}$, following prior multi-scale evaluation protocols (Barron et al., 2021; Yu et al., 2024a; Koo et al., 2026). For both datasets, the additional resolutions are obtained by box downsampling from the base resolution. We use white backgrounds for synthetic scenes and black backgrounds for Mip-NeRF 360, compositing any ground-truth alpha onto the same background. Unless otherwise specified, Blender results use 64 samples per pixel. Blender uses the Standard view transform with no additional look, zero exposure adjustment, unit gamma, and a Blackman–Harris pixel filter of width 1.5 pixels, with denoising disabled. Filament uses an unlit opaque vertex-color material with postprocessing, dithering, and anti-aliasing disabled. The full evaluations report PSNR, SSIM, and LPIPS on sRGB images (Wang et al., 2004; Zhang et al., 2018). We use an 11×11 Gaussian SSIM window with standard deviation 1.5 and VGG-based LPIPS with RGB inputs normalized to $[-1, 1]$. We average over test views and then equally over scenes, with equal weighting of the four MTMT scales.

Reconstruction regularization. The reconstruction objective combines photometric supervision with renderer-derived depth consistency and face-area regularization. Let D_Π be the rendered median-depth image, $\mathbf{x}_i^\Pi$ the projected image position of vertex i, and z_i^Π its camera-space depth. At each pixel, D_Π stores the interpolated depth of the last contributing face whose preceding transmittance exceeds one half. We sample this depth at the nearest pixel and differentiate its value while keeping the discrete face and pixel selections fixed. For the set $\mathcal{V}_\Pi$ of in-image vertices belonging to contributing faces, with finite positive sampled depth and residual below the truncation threshold $\delta = 0.5$, the depth term is

$$\mathcal{L}_{\text{depth}} = \frac{1}{|\mathcal{V}_\Pi|} \sum_{i \in \mathcal{V}_\Pi} \left| z_i^\Pi - D_\Pi\big(\mathbf{x}_i^\Pi\big) \right|. \tag{31}$$

This term encourages visible vertices to agree with the rendered surface depth without external supervision. Let $A_j(V)$ be the three-dimensional area of face j, $\overline{A}_0$ the mean face area of the initial mesh, held fixed during optimization, and $\mathcal{F}_\Pi^{\text{vis}}$ the faces contributing to the rendered image. The

face-area term is

$$\mathcal{L}_{\text{area}} = \frac{1}{|\mathcal{F}_\Pi^{\text{vis}}|} \sum_{j \in \mathcal{F}_\Pi^{\text{vis}}} \left(\frac{A_j(V)}{\bar{A}_0} \right)^2 . \tag{32}$$

This term discourages excessive growth of visible faces relative to the initial mesh scale. For SPIRE, we set $\lambda_{\text{depth}} = 2.5 \times 10^{-4}$ and $\lambda_{\text{area}} = 3 \times 10^{-5}$.

Surface grouping. The renderer constructs surface groups in 16×16-pixel tiles. Edge-connected-component labeling groups projected faces that share a mesh edge and have the same projected orientation. Faces are traversed in front-to-back order using their minimum camera-space vertex depth. Within each pixel, a new surface starts when the connected component changes or the next face lies beyond the accumulated vertex-depth range.

Mesh extraction and appearance conversion. We obtain view-independent color at each vertex from the degree-zero (DC) component of the optimized SH appearance. Let $C_{\text{DC}} = [\mathbf{C}_i^{\text{DC}}]_{i=1}^{|V|} \in [0,1]^{|V| \times 3}$, where

$$\mathbf{C}_i^{\text{DC}} = \text{clip}_{[0,1]} \left(\frac{1}{2} \mathbf{1}_3 + Y_{00} \mathbf{h}_{i00} \right), \qquad Y_{00} = 0.28209479. \tag{33}$$

This conversion yields the vertex colors of $\mathcal{M}_{\text{RGB}} = (V, F, C_{\text{DC}})$.

Our baseline runs build on the authors' public implementations, using released mesh-extraction procedures where available and the adaptations below for opaque vertex-color evaluation. For DUSt3R (Wang et al., 2024) and Pow3R (Jang et al., 2025), we align predicted depths to the known cameras and fuse them with input RGB colors using Open3D TSDF fusion, without photometric refinement (Curless & Levoy, 1996). We crop SuGaR's synthetic meshes to the scene bounding box and clean PGSR's Mip-NeRF 360 MTMT export using its released rule, removing components with fewer than 1,000 faces. Mip-Splatting, Analytic-Splatting, and Gaussian Blending share an adapted 2DGS extraction pipeline that combines added alpha-normalized depth rendering with degree-zero color, using bounded TSDF fusion on synthetic scenes and unbounded extraction on Mip-NeRF 360. The AA-2DGS export on Mip-NeRF 360 additionally requires removing non-finite Gaussians and restoring the unobserved TSDF initialization used by 2DGS. We bake nvdiffrec's learned materials and environment lighting, including approximate view-averaged specular shading, into vertex colors. MeshSplats retains its RGB vertex colors with opacity set to one, while Triangle Splatting and MeshSplatting are exported directly with face or vertex DC colors, respectively.

D Additional Results

We expand the main-paper single-scale training and testing (STST) comparisons and additionally evaluate multi-scale training and testing (MTMT). We also provide qualitative and per-scene results, per-scan geometry results, efficiency measurements, and exported-mesh evaluations with Blender and Google Filament.

D.1 Full STST comparisons

Tables 6 and 7 expand the main-paper STST comparisons with SSIM, LPIPS, and a broader set of baselines. The additional baselines are PGSR (Chen et al., 2024), Analytic-Splatting (Liang et al., 2024), and the feed-forward methods DUSt3R (Wang et al., 2024) and Pow3R (Jang et al., 2025). For the feed-forward conversions, we fuse predicted depths and input-view colors without per-scene photometric refinement, so inconsistent depths and view-dependent colors can remain in the exported asset.

The tables additionally include matched prior ablations for Gaussian Surfels and MeshSplatting. The main tables use the default prior-assisted configurations for methods designed to accept these priors. MeshSplatting uses its original $4\times$ SSAA per image dimension for native rendering in the main tables. The full STST and MTMT tables additionally report native results without SSAA, with the shared exported mesh and Blender result reported only in the +SSAA rows. The matched runs show that external priors do not consistently improve Blender fidelity, and neither configuration closes the gap to SPIRE.

Table 8: Renderer comparison on NeRF-Synthetic STST from identical initial meshes with fixed topology. Values are eight-scene Blender Cycles means. All methods optimize degree-three SH appearance for 15k iterations with the same photometric loss, with geometry regularization added for SPIRE. We use nvdiffrast's provided antialiasing and MeshSplatting's original $4\times$ SSAA per image dimension, applying the same SSAA setting to Triangle Splatting. † MeshSplats uses a vertex learning rate reduced by a factor of 250 to improve reconstruction quality, with other settings unchanged. Bold indicates the best result per column.

	Fixed vertex positions			Optimized vertex positions		
Renderer	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
MeshSplatting	25.12	0.903	0.107	20.46	0.835	0.183
Triangle Splatting	25.65	0.905	0.106	13.91	0.718	0.404
nvdiffrast	**26.80**	0.929	0.076	27.83	0.937	0.073
MeshSplats†	26.74	0.928	0.076	26.90	0.930	0.075
SuGaR	24.99	0.907	0.096	24.55	0.889	0.141
SPIRE (Ours)	26.78	**0.930**	**0.073**	**29.03**	**0.944**	**0.067**

D.2 Renderer comparison under fixed mesh topology

Table 8 expands the renderer comparison in Table 4(b) with SSIM, LPIPS, and implementation settings. All methods start from the same GOF-derived mesh for each scene and preserve vertex count and connectivity, comparing appearance-only fitting with joint geometry and appearance optimization.

Under joint optimization, SPIRE leads across all three Blender metrics. Its advantage therefore extends beyond the PSNR comparison in the main text.

Hard z-buffer gradients. The hard z-buffer variants in Table 4(a) provide geometry gradients only through barycentric color interpolation within the selected face. These gradients can become ill-conditioned for small projected triangles, and joint geometry optimization degrades image fidelity relative to fitting color on fixed geometry.

Individual regularizer ablations. Under the same setup as Table 4(a), removing only depth regularization yields native, DC, and Blender PSNR of 32.28, 29.38, and 29.03 dB, respectively, matching Full after rounding. Removing only face-area regularization yields 32.11, 29.35, and 28.41 dB, respectively, close to removing both regularizers. These results indicate that face-area regularization accounts for most of the regularization benefit in this setting.

D.3 Full MTMT comparisons

Tables 9 and 10 extend the evaluation to MTMT on Multiscale Blender and Mip-NeRF 360, respectively. Each entry averages all four scales and scenes under the protocol in Appendix C. Methods with strong native anti-aliasing, such as Mip-Splatting and Gaussian Blending, retain high native quality but lose much of that advantage after mesh conversion. SPIRE achieves the best Blender quality across all three metrics on both datasets, showing that its multiscale performance carries over to the mesh itself. On Multiscale Blender and Mip-NeRF 360, respectively, SPIRE reaches 29.02 and 24.43 dB Blender PSNR, compared with 27.08 and 20.64 dB for the next-best methods.

D.4 Resolution-wise MTMT results

Tables 11 and 12 report PSNR, SSIM, and LPIPS at each resolution for Multiscale Blender and Mip-NeRF 360, respectively. Across both datasets, SPIRE has the best PSNR, SSIM, and LPIPS at every evaluated resolution. The advantage therefore holds at each scale rather than arising from the average alone. It covers both fine detail at full resolution and the larger pixel footprints encountered at lower resolutions.

Table 9: MTMT NVS quality on Multiscale Blender, averaged over four resolution scales and eight scenes. PSNR gap is Native minus Blender. MeshSplatting's shared Blender result is shown only in its +SSAA row. Bold and underline indicate the best and second-best results, respectively.

Method	Native			Blender			Gap
	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↓
DUSt3R	–	–	–	13.16	0.720	0.343	–
Pow3R	–	–	–	10.73	0.594	0.491	–
NeuS2	27.59	0.936	0.088	22.71	0.897	0.098	4.88
SuGaR	26.79	0.939	0.066	21.73	0.865	0.159	5.06
2DGS	28.55	0.948	0.070	19.94	0.840	0.203	8.61
Gaussian Surfels	25.44	0.918	0.092	21.36	0.868	0.118	<u>4.08</u>
GOF	31.37	0.969	0.042	<u>27.08</u>	<u>0.929</u>	<u>0.070</u>	4.29
PGSR	28.04	0.948	0.056	20.41	0.852	0.165	7.63
Mip-Splatting	34.61	0.979	0.022	20.47	0.855	0.157	14.14
Analytic-Splatting	35.04	0.979	0.022	19.93	0.846	0.167	15.11
AA-2DGS	34.68	0.979	0.024	21.83	0.863	0.138	12.85
Gaussian Blending	36.08	0.981	0.021	20.34	0.853	0.157	15.74
nvdiffrec	25.08	0.904	0.135	23.34	0.879	0.137	**1.74**
MeshSplats	28.22	0.942	0.081	21.48	0.847	0.150	6.74
Triangle Splatting	29.34	0.952	0.063	15.75	0.726	0.289	13.59
MeshSplatting	25.01	0.905	0.126	–	–	–	–
MeshSplatting (+SSAA)	29.13	0.943	0.069	24.52	0.902	0.109	4.61
With prior							
Gaussian Surfels	25.46	0.918	0.092	21.04	0.864	0.119	4.42
MeshSplatting	25.05	0.906	0.125	–	–	–	–
MeshSplatting (+SSAA)	29.10	0.943	0.069	24.38	0.899	0.111	4.72
SPIRE (Ours)	33.54	0.973	0.031	**29.02**	**0.952**	**0.053**	4.52

Table 10: MTMT NVS quality on Mip-NeRF 360, averaged over four resolution scales and nine scenes. PSNR gap is Native minus Blender. MeshSplatting's shared Blender result is shown only in its +SSAA row. Bold and underline indicate the best and second-best results, respectively.

Method	Native			Blender			Gap
	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↓
SuGaR	25.78	0.792	0.230	20.24	0.582	0.426	5.54
2DGS	26.70	0.802	0.243	18.29	0.485	0.517	8.41
Gaussian Surfels	22.89	0.695	0.378	14.69	0.370	0.564	8.20
GOF	26.69	0.812	0.194	19.11	0.582	0.416	7.58
PGSR	26.93	0.839	0.182	19.42	<u>0.601</u>	<u>0.389</u>	7.51
Mip-Splatting	28.17	0.858	0.149	16.72	0.448	0.528	11.45
Analytic-Splatting	28.39	0.857	0.161	16.87	0.454	0.529	11.52
AA-2DGS	27.99	0.846	0.190	18.20	0.479	0.519	9.79
Gaussian Blending	28.01	0.853	0.172	16.14	0.422	0.561	11.87
MeshSplats	25.39	0.766	0.254	17.28	0.397	0.576	8.11
MeshSplatting	20.68	0.554	0.480	–	–	–	–
MeshSplatting (+SSAA)	23.85	0.698	0.355	20.60	0.581	0.442	3.25
With prior							
Gaussian Surfels	23.48	0.711	0.358	15.63	0.410	0.539	7.85
MeshSplatting	20.70	0.555	0.478	–	–	–	–
MeshSplatting (+SSAA)	23.84	0.698	0.354	<u>20.64</u>	0.582	0.440	<u>3.20</u>
SPIRE (Ours)	26.30	0.830	0.196	**24.43**	**0.779**	**0.211**	**1.87**

D.5 GEOMETRY

We evaluate mesh geometry on 15 DTU scans and report mean Chamfer distance (CD) in millimeters (Jensen et al., 2014).

Table 11: Per-resolution Blender results on Multiscale Blender under MTMT.

Method	PSNR ↑					SSIM ↑					LPIPS ↓				
	×1	×1/2	×1/4	×1/8	Mean	×1	×1/2	×1/4	×1/8	Mean	×1	×1/2	×1/4	×1/8	Mean
DUSt3R	12.92	13.02	13.20	13.49	13.16	0.773	0.735	0.702	0.672	0.720	0.302	0.328	0.362	0.382	0.343
Pow3R	10.56	10.63	10.76	10.98	10.73	0.686	0.624	0.561	0.503	0.594	0.429	0.467	0.513	0.556	0.491
NeuS2	21.89	22.43	23.02	23.50	22.71	0.875	0.891	0.906	0.917	0.897	0.130	0.106	0.085	0.072	0.098
SuGaR	20.55	21.15	22.05	23.15	21.73	0.841	0.850	0.874	0.898	0.865	0.185	0.176	0.155	0.118	0.159
2DGS	19.14	19.49	20.11	21.01	19.94	0.843	0.829	0.833	0.855	0.840	0.228	0.221	0.203	0.162	0.203
Gaussian Surfels	20.11	20.68	21.64	23.01	21.36	0.848	0.851	0.870	0.902	0.868	0.151	0.132	0.108	0.083	0.118
GOF	25.68	26.66	27.64	28.35	27.08	0.905	0.920	0.936	0.952	0.929	0.097	0.074	0.061	0.049	0.070
PGSR	19.31	19.84	20.70	21.80	20.41	0.854	0.848	0.851	0.853	0.852	0.167	0.165	0.165	0.164	0.165
Mip-Splatting	19.34	19.92	20.78	21.83	20.47	0.861	0.855	0.854	0.850	0.855	0.158	0.154	0.155	0.161	0.157
Analytic-Splatting	18.81	19.35	20.23	21.34	19.93	0.851	0.844	0.844	0.843	0.846	0.166	0.163	0.165	0.172	0.167
AA-2DGS	20.60	21.15	22.12	23.44	21.83	0.849	0.845	0.863	0.895	0.863	0.164	0.152	0.133	0.102	0.138
Gaussian Blending	19.30	19.87	20.66	21.55	20.34	0.860	0.854	0.852	0.846	0.853	0.155	0.151	0.155	0.166	0.157
nvdiffrec	22.18	22.79	23.68	24.72	23.34	0.856	0.861	0.883	0.914	0.879	0.166	0.154	0.131	0.096	0.137
MeshSplats	20.19	20.87	21.85	23.00	21.48	0.819	0.829	0.854	0.886	0.847	0.177	0.162	0.143	0.119	0.150
Triangle Splatting	14.98	15.38	15.94	16.69	15.75	0.746	0.723	0.712	0.724	0.726	0.275	0.288	0.305	0.290	0.289
MeshSplatting	23.31	24.00	24.92	25.86	24.52	0.873	0.888	0.911	0.934	0.902	0.148	0.125	0.097	0.067	0.109
With prior															
Gaussian Surfels	19.79	20.34	21.31	22.70	21.04	0.846	0.847	0.867	0.897	0.864	0.152	0.133	0.109	0.084	0.119
MeshSplatting	23.19	23.87	24.76	25.69	24.38	0.871	0.886	0.908	0.932	0.899	0.151	0.127	0.098	0.070	0.111
SPIRE (Ours)	**28.17**	**28.89**	**29.41**	**29.63**	**29.02**	**0.937**	**0.947**	**0.958**	**0.968**	**0.952**	**0.077**	**0.056**	**0.044**	**0.034**	**0.053**

Table 12: Per-resolution Blender results on Mip-NeRF 360 under MTMT.

Method	PSNR ↑					SSIM ↑					LPIPS ↓				
	×1	×1/2	×1/4	×1/8	Mean	×1	×1/2	×1/4	×1/8	Mean	×1	×1/2	×1/4	×1/8	Mean
SuGaR	19.24	19.73	20.53	21.48	20.24	0.486	0.520	0.610	0.712	0.582	0.564	0.491	0.379	0.269	0.426
2DGS	17.40	17.84	18.55	19.39	18.29	0.429	0.438	0.499	0.574	0.485	0.623	0.571	0.484	0.388	0.517
Gaussian Surfels	14.24	14.43	14.80	15.28	14.69	0.334	0.326	0.371	0.451	0.370	0.647	0.613	0.543	0.451	0.564
GOF	17.96	18.63	19.52	20.33	19.11	0.495	0.543	0.616	0.676	0.582	0.525	0.454	0.374	0.312	0.416
PGSR	18.57	19.00	19.66	20.45	19.42	0.508	0.547	0.632	0.716	0.601	0.515	0.443	0.346	0.253	0.389
Mip-Splatting	16.07	16.38	16.90	17.54	16.72	0.404	0.406	0.457	0.524	0.448	0.624	0.576	0.498	0.414	0.528
Analytic-Splatting	16.20	16.52	17.06	17.71	16.87	0.409	0.411	0.464	0.533	0.454	0.626	0.578	0.499	0.413	0.529
AA-2DGS	17.32	17.76	18.46	19.27	18.20	0.424	0.432	0.491	0.566	0.479	0.622	0.572	0.488	0.394	0.519
Gaussian Blending	15.57	15.83	16.29	16.86	16.14	0.393	0.385	0.425	0.486	0.422	0.645	0.607	0.537	0.454	0.561
MeshSplats	16.44	16.87	17.50	18.31	17.28	0.367	0.352	0.393	0.475	0.397	0.653	0.626	0.561	0.463	0.576
MeshSplatting	19.58	20.19	20.95	21.69	20.60	0.482	0.526	0.613	0.702	0.581	0.584	0.508	0.394	0.281	0.442
With prior															
Gaussian Surfels	15.12	15.34	15.75	16.30	15.63	0.367	0.362	0.413	0.499	0.410	0.632	0.591	0.516	0.417	0.539
MeshSplatting	19.61	20.22	20.99	21.74	20.64	0.482	0.527	0.615	0.705	0.582	0.584	0.507	0.391	0.277	0.440
SPIRE (Ours)	**23.22**	**24.06**	**24.94**	**25.49**	**24.43**	**0.680**	**0.751**	**0.819**	**0.865**	**0.779**	**0.336**	**0.233**	**0.158**	**0.115**	**0.211**

Table 13 reports the aggregate DTU comparison.

Table 23 expands this comparison with per-scan CD. Our mean CD is comparable to those of 2DGS and MeshSplatting, indicating that direct image-based optimization also recovers competitive surface geometry.

D.6 Runtime and resource requirements

Table 14 reports rendering speed, optimization time, storage, and mesh size on NeRF-Synthetic. Our vertex-colored meshes use less storage and fewer vertices and faces than the 2DGS and GOF meshes while achieving higher Blender fidelity. The larger training representation includes SH appearance, whereas deployment requires only the mesh and vertex colors.

D.7 Blender Cycles at one sample per pixel

Tables 15 and 16 expand the main-paper one-sample Cycles comparison with SSIM, LPIPS, additional baselines, and MTMT results. They use the same meshes, cameras, backgrounds, and vertex-color emission shader as the 64-sample evaluations. We evaluate the model–dataset combinations from the full STST and MTMT comparisons and use – for unavailable results, whether due to unsupported settings or execution failures such as out-of-memory (OOM) errors. Even with one sample per pixel and no multisample anti-aliasing, SPIRE achieves the best PSNR, SSIM, and LPIPS across all four settings while using fewer ray samples for rendering. The advantage thus persists without

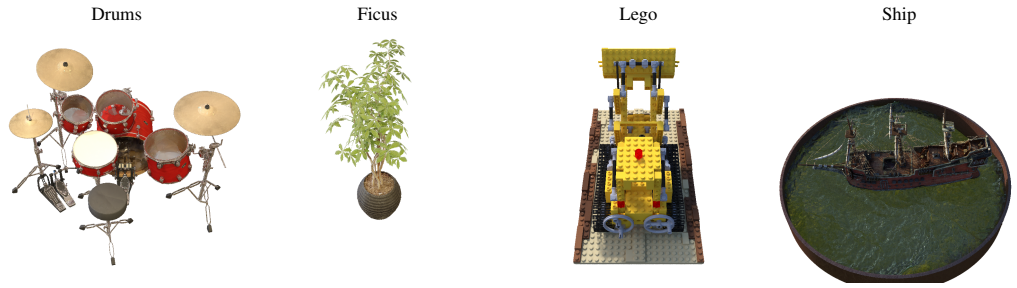


Figure 4: Exported SPIRE meshes rendered in real time with Google Filament.

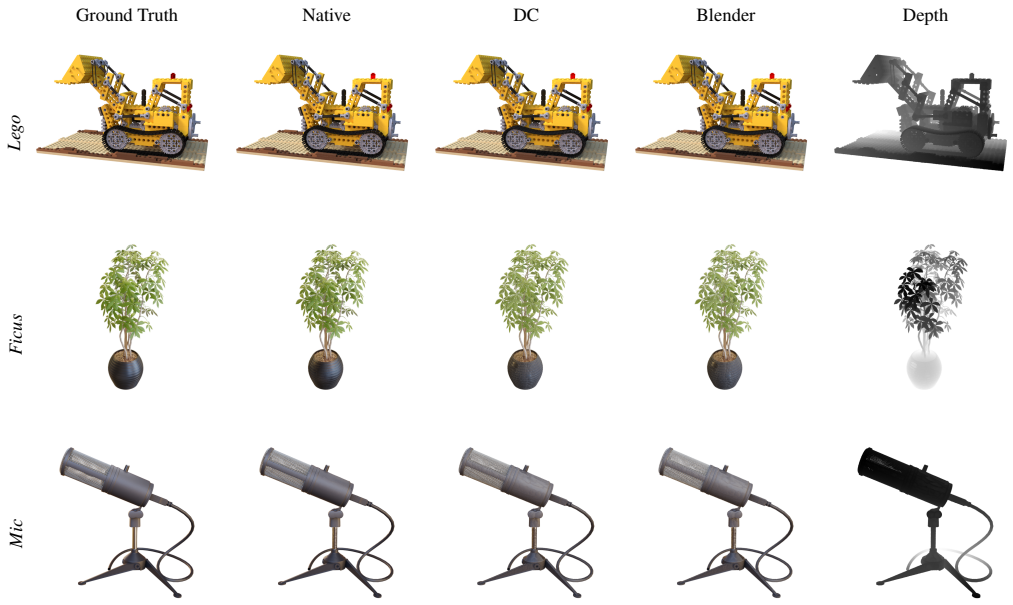


Figure 5: Native, DC, and Blender renderings of the same SPIRE meshes, with median-depth visualizations. DC disables higher-order SH terms, and depth is normalized over each view's foreground.

the dense sampling of the 64-sample Cycles evaluation, indicating that it is retained in the optimized mesh geometry and colors.

D.8 Rendering with Google Filament

Tables 17 and 18 expand the main-paper Filament comparison with SSIM, LPIPS, additional baselines, and MTMT results, using the same exported meshes and test views with anti-aliasing disabled. The model–dataset combinations and evaluation metrics follow the one-sample Cycles comparison, including all four MTMT resolutions. SPIRE achieves the best PSNR, SSIM, and LPIPS on both datasets under STST and MTMT. This shows that the rendering-quality advantage of our meshes extends beyond Blender Cycles to a production rasterizer. The agreement between one-sample Cycles and Filament further shows that our lead persists across ray tracing and rasterization without anti-aliasing.

With the same Filament configuration, our NeRF-Synthetic meshes reach an average offscreen rendering speed of 849 FPS on an NVIDIA RTX A6000 at 800×800, with per-scene medians ranging from 805 to 932 FPS. This speed measures rendering alone, excluding image saving and metric computation. Together, these results demonstrate high rendering quality and efficient rendering of our exported meshes with an optimized production renderer.

Table 13: Mesh geometry quality on DTU, measured by mean Chamfer distance (CD) in millimeters. Baseline results are reported by MeshSplatting (Held et al., 2025a). The images on the right show SPIRE native RGB renderings of training views (top) and corresponding depth maps (bottom).

Method	Mean CD ↓
3DGS	1.96
2DGS	0.80
GOF	0.74
RaDe-GS	0.68
MILo	0.68
Triangle Splatting	1.06
MeshSplatting	0.79
SPIRE (Ours)	0.80

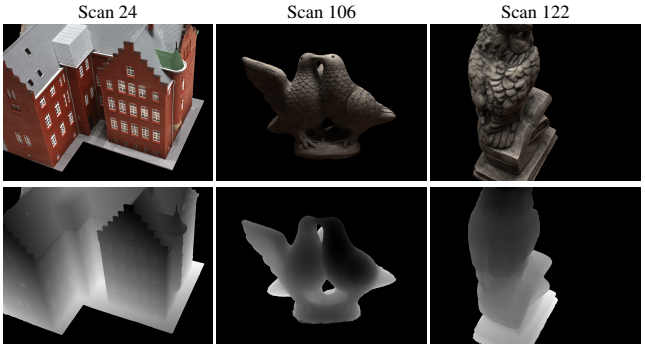


Table 14: Computational and storage costs on NeRF-Synthetic, including native-renderer speed, training time, model and mesh storage, and vertex and face counts. For SPIRE, parentheses give the additional GOF initialization time, for a total of 78.3 minutes.

Method	Renderer FPS	Train (min)	Model (MB)	Mesh (MB)	$\lvert V\rvert$ ($\times 10^6$)	$\lvert F\rvert$ ($\times 10^6$)
2DGS	323.8	11.8	25.4	65.2	1.238	2.443
GOF	31.4	48.4	41.6	67.1	1.596	3.198
MeshSplatting	171.1	27.0	78.7	18.6	0.281	0.848
SPIRE (Ours)	37.0	24.4 (+53.9)	273.1	47.3	1.050	2.105

D.9 Scene-wise PSNR

Tables 19 and 20 give the per-scene Blender PSNR under STST. Tables 21 and 22 give the corresponding MTMT results, where each scene value averages its four scales. SPIRE obtains the highest Blender PSNR on every scene in all four settings, demonstrating consistently higher image fidelity across the full set of evaluated scenes. On Mip-NeRF 360, this includes both outdoor scenes with vegetation and thin structures and indoor scenes with detailed surfaces and clutter. The gains are therefore distributed across scene types rather than concentrated in a few scenes with large improvements.

D.10 Qualitative results

Figures 6 and 7 provide full-frame Blender comparisons, including the Lego and Room views cropped in Tables 2 and 3. SPIRE preserves thin structures such as the leaves in Ficus that are missing from the 2DGS mesh, consistent with the extraction gap illustrated in Figure 3. Figure 5 also shows that thin structures and occlusion boundaries remain consistent across our native, DC, and Blender renderings. Because faces remain opaque throughout optimization, switching renderers does not expose previously softened faces as unintended occluders, avoiding the soft-to-opaque visibility change illustrated in Figure 3.

D.11 PSNR-gap decomposition

Table 24 separates the native-to-Blender change into two transitions. DC rendering retains only the degree-zero SH component for view-independent color, using the same native renderer and optimized representation. The appearance gap is the native-to-DC PSNR decrease, and the renderer gap is the subsequent DC-to-Blender decrease. These gaps are assessed alongside absolute Blender fidelity, since a small gap alone does not establish exported-mesh quality.

Both MeshSplatting and SPIRE produce opaque meshes and have small renderer gaps. MeshSplatting obtains geometry gradients through semitransparent faces that are gradually made opaque during training (Held et al., 2025a). This representation change can degrade native quality, while Mesh-

Table 15: Blender Cycles quality at one sample per pixel under STST. Values are averaged over scenes.

Method	NeRF-Synthetic			Mip-NeRF 360		
	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
DUSt3R	13.11	0.783	0.306	–	–	–
Pow3R	10.89	0.788	0.296	–	–	–
NeuS2	22.26	<u>0.897</u>	<u>0.114</u>	–	–	–
SuGaR	21.43	0.858	0.187	–	–	–
2DGS	21.76	0.871	0.177	17.55	0.460	0.594
Gaussian Surfels	22.33	0.878	0.123	15.00	0.366	0.644
GOF	24.76	0.893	0.118	18.11	0.509	0.500
PGSR	20.25	0.875	0.144	14.38	0.384	0.567
Mip-Splatting	20.23	0.871	0.150	16.64	0.442	0.597
Analytic-Splatting	20.19	0.870	0.150	16.72	0.444	0.601
AA-2DGS	21.67	0.863	0.147	16.51	0.432	0.617
Gaussian Blending	22.29	0.864	0.152	15.10	0.408	0.629
nvdiffrec	21.99	0.855	0.155	–	–	–
MeshSplats	19.50	0.808	0.221	17.14	0.387	0.645
Triangle Splatting	15.61	0.755	0.247	14.27	0.237	0.669
MeshSplatting	<u>25.12</u>	0.895	0.128	<u>20.76</u>	<u>0.557</u>	<u>0.492</u>
MeshSplatting (+SSAA)	<u>25.12</u>	0.895	0.128	<u>20.76</u>	<u>0.557</u>	<u>0.492</u>
With prior						
Gaussian Surfels	21.83	0.876	0.122	15.43	0.379	0.630
MeshSplatting	25.03	0.895	0.128	20.73	0.554	0.495
MeshSplatting (+SSAA)	25.03	0.895	0.128	20.73	0.554	0.495
SPIRE (Ours)	**27.17**	**0.927**	**0.098**	**22.42**	**0.632**	**0.383**

Splatting's exported-mesh fidelity remains comparable to that of other leading baselines despite its default Metric3D normal prior. Our analytic pixel-area integration instead differentiably renders opaque faces throughout optimization, preserving native quality while achieving both a small renderer gap and higher exported-mesh fidelity.

Table 16: Blender Cycles quality at one sample per pixel under MTMT. Values are averaged over four resolution scales and scenes.

Method	Multiscale Blender			Mip-NeRF 360		
	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
DUSt3R	13.05	0.711	0.351	–	–	–
Pow3R	10.59	0.573	0.502	–	–	–
NeuS2	21.73	0.876	0.139	–	–	–
SuGaR	20.97	0.847	0.188	19.83	0.559	0.441
2DGS	19.72	0.837	0.209	18.18	0.485	0.508
Gaussian Surfels	20.53	0.852	0.146	14.48	0.357	0.572
GOF	24.62	0.903	0.118	18.17	0.528	0.444
PGSR	19.83	0.846	0.174	19.11	0.595	0.389
Mip-Splatting	19.45	0.845	0.175	16.55	0.436	0.524
Analytic-Splatting	19.06	0.837	0.182	16.71	0.445	0.523
AA-2DGS	21.29	0.856	0.147	18.06	0.476	0.511
Gaussian Blending	19.34	0.842	0.178	16.00	0.413	0.554
nvdiffrec	22.43	0.862	0.167	–	–	–
MeshSplats	19.89	0.813	0.199	16.54	0.355	0.600
Triangle Splatting	14.64	0.702	0.316	–	–	–
MeshSplatting	23.44	0.883	0.147	19.88	0.518	0.480
MeshSplatting (+SSAA)	23.44	0.883	0.147	19.88	0.518	0.480
With prior						
Gaussian Surfels	20.17	0.848	0.147	15.40	0.396	0.547
MeshSplatting	23.34	0.881	0.149	19.91	0.519	0.479
MeshSplatting (+SSAA)	23.34	0.881	0.149	19.91	0.519	0.479
SPIRE (Ours)	**25.98**	**0.926**	**0.106**	**22.58**	**0.669**	**0.295**

Table 17: Google Filament rendering quality without anti-aliasing under STST. Values are averaged over scenes.

Method	NeRF-Synthetic			Mip-NeRF 360		
	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
DUSt3R	13.10	0.784	0.280	–	–	–
Pow3R	10.89	0.789	0.265	–	–	–
NeuS2	22.29	0.897	0.114	–	–	–
SuGaR	21.43	0.858	0.184	–	–	–
2DGS	21.77	0.871	0.157	17.51	0.460	0.602
Gaussian Surfels	22.33	0.878	0.123	15.02	0.368	0.647
GOF	24.61	0.893	0.119	18.32	0.513	0.503
PGSR	20.25	0.875	0.145	14.38	0.385	0.573
Mip-Splatting	20.22	0.872	0.150	16.58	0.441	0.602
Analytic-Splatting	20.19	0.870	0.150	16.66	0.444	0.606
AA-2DGS	21.67	0.863	0.147	16.46	0.431	0.622
Gaussian Blending	22.29	0.864	0.152	15.06	0.408	0.633
nvdiffrec	21.94	0.855	0.155	–	–	–
MeshSplats	19.41	0.808	0.214	17.13	0.388	0.648
Triangle Splatting	15.61	0.755	0.245	14.27	0.236	0.671
MeshSplatting	25.20	0.896	0.127	20.85	0.561	0.492
MeshSplatting (+SSAA)	25.20	0.896	0.127	20.85	0.561	0.492
With prior						
Gaussian Surfels	21.83	0.876	0.122	15.45	0.381	0.633
MeshSplatting	25.11	0.895	0.128	20.82	0.558	0.495
MeshSplatting (+SSAA)	25.11	0.895	0.128	20.82	0.558	0.495
SPIRE (Ours)	**27.18**	**0.928**	**0.098**	**22.49**	**0.634**	**0.386**

Table 18: Google Filament rendering quality without anti-aliasing under MTMT. Values are averaged over four resolution scales and scenes.

Method	Multiscale Blender			Mip-NeRF 360		
	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
DUSt3R	13.05	0.712	0.334	–	–	–
Pow3R	10.58	0.573	0.489	–	–	–
NeuS2	21.76	0.877	0.139	–	–	–
SuGaR	20.97	0.847	0.186	19.84	0.561	0.441
2DGS	19.72	0.837	0.184	18.07	0.480	0.513
Gaussian Surfels	20.54	0.852	0.145	14.49	0.358	0.572
GOF	24.54	0.903	0.118	18.27	0.530	0.445
PGSR	19.83	0.846	0.174	19.11	0.595	0.393
Mip-Splatting	19.44	0.845	0.175	16.42	0.431	0.526
Analytic-Splatting	19.06	0.837	0.182	16.59	0.440	0.526
AA-2DGS	21.29	0.856	0.147	17.95	0.472	0.516
Gaussian Blending	19.33	0.842	0.177	15.89	0.408	0.556
nvdiffrec	22.45	0.862	0.168	–	–	–
MeshSplats	19.85	0.813	0.198	16.56	0.355	0.602
Triangle Splatting	14.64	0.702	0.314	–	–	–
MeshSplatting	23.56	0.884	0.146	20.00	0.522	0.480
MeshSplatting (+SSAA)	23.56	0.884	0.146	20.00	0.522	0.480
With prior						
Gaussian Surfels	20.17	0.849	0.146	15.41	0.398	0.547
MeshSplatting	23.48	0.882	0.148	20.03	0.523	0.478
MeshSplatting (+SSAA)	23.48	0.882	0.148	20.03	0.523	0.478
SPIRE (Ours)	**25.98**	**0.926**	**0.106**	**22.66**	**0.671**	**0.295**

Table 19: Per-scene Blender PSNR ↑ on NeRF-Synthetic under STST.

Method	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Mean
DUSt3R	18.57	13.67	15.25	11.59	11.32	11.07	16.02	7.63	13.14
Pow3R	14.05	10.95	14.23	10.59	9.56	8.74	13.04	6.00	10.90
NeuS2	25.08	21.22	21.96	22.04	28.75	18.67	24.33	19.13	22.65
SuGaR	17.09	18.95	24.69	26.22	19.53	21.83	25.26	22.66	22.03
2DGS	22.76	19.14	18.36	26.56	23.64	19.97	23.30	22.00	21.96
Gaussian Surfels	26.40	20.90	22.95	24.16	24.03	19.49	22.87	22.76	22.95
GOF	27.05	23.39	26.23	29.87	29.55	20.14	27.79	24.08	26.01
PGSR	27.38	14.29	17.38	28.93	25.77	9.05	18.12	23.38	20.54
Mip-Splatting	25.59	14.27	17.51	27.15	26.61	9.08	24.21	21.04	20.68
Analytic-Splatting	24.73	13.72	16.99	27.58	25.89	9.05	23.98	22.92	20.61
AA-2DGS	25.11	19.84	18.10	25.97	22.68	19.06	22.81	21.63	21.90
Gaussian Blending	24.95	20.65	21.70	26.99	24.18	18.99	24.06	23.16	23.08
nvdiffrec	23.15	19.32	20.24	25.89	24.21	19.85	25.19	21.15	22.38
MeshSplats	22.68	20.41	15.89	21.71	23.39	19.23	21.34	18.15	20.35
Triangle Splatting	17.52	13.69	16.34	20.22	16.25	15.72	18.80	14.83	16.67
MeshSplatting	26.52	23.19	27.41	29.32	27.79	22.90	27.03	24.77	26.12
With prior									
Gaussian Surfels	26.38	20.93	22.37	23.29	23.52	19.25	22.66	21.16	22.44
MeshSplatting	26.31	23.08	27.40	29.23	27.66	22.62	26.87	24.66	25.98
SPIRE (Ours)	**31.01**	**24.34**	**29.88**	**32.57**	**34.32**	**24.06**	**29.95**	**26.13**	**29.03**

Table 20: Per-scene Blender PSNR ↑ on Mip-NeRF 360 under STST.

Method	Bicycle	Flowers	Garden	Stump	Treehill	Room	Counter	Kitchen	Bonsai	Mean
2DGS	19.08	16.44	18.40	20.44	19.45	23.56	14.11	12.29	14.69	17.61
Gaussian Surfels	15.61	10.95	15.00	17.09	12.65	19.00	16.34	16.63	12.47	15.08
GOF	20.45	17.87	18.58	23.77	19.89	23.69	13.59	14.91	14.95	18.63
PGSR	17.95	7.67	17.03	19.75	12.74	22.73	10.80	6.13	15.75	14.51
Mip-Splatting	18.26	14.76	17.90	19.31	18.35	22.10	13.85	12.04	13.94	16.72
Analytic-Splatting	18.66	14.45	18.04	19.07	18.47	22.29	13.93	12.08	14.10	16.79
AA-2DGS	17.73	14.28	17.82	19.36	18.12	22.04	13.75	11.97	14.14	16.58
Gaussian Blending	16.77	12.53	16.10	18.20	13.95	20.55	13.47	11.46	13.26	15.14
MeshSplats	17.11	14.26	16.75	17.86	17.15	19.68	17.86	17.69	18.94	17.48
Triangle Splatting	12.79	11.09	14.23	15.03	13.10	18.38	16.52	16.96	16.23	14.92
MeshSplatting	20.94	18.18	21.10	23.19	19.42	23.92	21.24	20.73	22.09	21.20
With prior										
Gaussian Surfels	15.94	11.15	15.20	18.04	13.31	19.78	16.84	17.07	12.28	15.51
MeshSplatting	20.90	18.17	21.02	23.07	19.44	23.90	21.24	20.69	22.07	21.17
SPIRE (Ours)	**21.19**	**18.47**	**22.43**	**25.06**	**20.98**	**26.85**	**24.19**	**24.16**	**26.16**	**23.28**

Table 21: Per-scene Blender PSNR ↑ on Multiscale Blender under MTMT.

Method	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Mean
DUSt3R	17.52	13.55	16.03	12.00	11.38	10.93	14.44	9.43	13.16
Pow3R	13.59	10.97	13.53	10.19	9.55	8.36	12.71	6.96	10.73
NeuS2	25.92	22.25	21.16	21.16	27.58	19.94	24.20	19.46	22.71
SuGaR	26.51	21.02	22.58	18.51	22.09	21.25	19.82	22.02	21.73
2DGS	20.67	18.23	17.03	25.28	20.40	17.10	20.30	20.49	19.94
Gaussian Surfels	25.23	20.35	21.29	21.39	21.75	17.08	22.12	21.65	21.36
GOF	29.51	24.83	26.57	31.16	30.19	20.82	28.57	25.00	27.08
PGSR	27.30	15.01	17.08	25.88	23.22	9.13	23.12	22.56	20.41
Mip-Splatting	26.70	13.37	17.93	25.97	24.92	9.14	23.34	22.37	20.47
Analytic-Splatting	25.82	13.43	16.79	26.17	22.57	9.09	23.60	22.00	19.93
AA-2DGS	25.82	20.45	17.40	26.13	22.00	18.44	22.84	21.57	21.83
Gaussian Blending	25.00	13.36	16.97	27.65	22.93	9.11	24.30	23.44	20.34
nvdiffrec	23.16	22.09	24.11	24.89	24.06	21.13	25.89	21.40	23.34
MeshSplats	21.14	20.93	22.56	21.39	24.27	20.00	23.30	18.25	21.48
Triangle Splatting	19.14	13.49	15.60	19.06	15.77	14.20	16.67	12.06	15.75
MeshSplatting	27.01	22.13	23.67	27.80	25.31	22.51	24.38	23.36	24.52
With prior									
Gaussian Surfels	25.22	20.26	21.37	22.11	21.40	17.28	21.95	18.69	21.04
MeshSplatting	26.84	22.07	23.37	27.71	25.25	22.39	24.10	23.28	24.38
SPIRE (Ours)	**31.62**	**25.64**	**28.50**	**32.75**	**32.28**	**25.01**	**29.57**	**26.82**	**29.02**

Table 22: Per-scene Blender PSNR ↑ on Mip-NeRF 360 under MTMT.

Method	Bicycle	Flowers	Garden	Stump	Treehill	Room	Counter	Kitchen	Bonsai	Mean
SuGaR	19.52	15.89	20.62	21.04	17.77	22.96	21.17	21.01	22.21	20.24
2DGS	19.95	18.57	19.86	22.05	20.67	22.86	14.01	12.31	14.36	18.29
Gaussian Surfels	15.87	11.22	14.54	17.09	12.31	17.31	15.92	16.24	11.71	14.69
GOF	22.11	20.75	15.21	24.93	21.49	22.50	15.06	15.85	14.09	19.11
PGSR	19.27	12.27	20.59	20.92	15.52	22.78	21.31	20.03	22.07	19.42
Mip-Splatting	18.35	15.42	18.81	20.26	15.80	22.19	13.96	11.68	14.02	16.72
Analytic-Splatting	18.60	15.62	18.69	20.74	15.82	22.39	13.94	12.02	14.02	16.87
AA-2DGS	19.88	18.45	19.85	21.52	20.55	22.94	14.09	12.25	14.29	18.20
Gaussian Blending	17.65	14.48	18.08	19.97	14.87	21.77	13.40	11.42	13.60	16.14
MeshSplats	16.71	14.81	16.83	17.39	17.67	19.38	17.42	16.97	18.33	17.28
MeshSplatting	20.10	18.25	21.00	21.92	18.76	22.92	20.90	20.18	21.39	20.60
With prior										
Gaussian Surfels	16.15	11.44	15.75	17.65	13.03	19.02	17.24	16.74	13.67	15.63
MeshSplatting	20.25	18.13	21.09	21.86	18.96	22.80	20.92	20.15	21.58	20.64
SPIRE (Ours)	**23.06**	**20.94**	**23.64**	**26.26**	**22.78**	**27.23**	**24.74**	**24.81**	**26.38**	**24.43**

Table 23: Per-scan Chamfer distance on DTU, measured in millimeters. Baseline per-scan values and means are reproduced as reported by MeshSplatting (Held et al., 2025a).

Method	24	37	40	55	63	65	69	83	97	105	106	110	114	118	122	Mean
3DGS	2.14	1.53	2.08	1.68	3.49	2.21	1.43	2.07	2.22	1.75	1.79	2.55	1.53	1.52	1.50	1.96
2DGS	0.48	0.91	0.39	0.39	1.01	0.83	0.81	1.36	1.27	0.76	0.70	1.40	0.40	0.76	0.52	0.80
GOF	0.50	0.82	0.37	0.37	1.12	0.74	0.73	1.18	1.29	0.68	0.77	0.90	0.42	0.66	0.49	0.74
RaDe-GS	0.46	0.73	0.33	0.38	0.79	0.75	0.76	1.19	1.22	0.62	0.70	0.78	0.36	0.68	0.47	0.68
MILo	0.43	0.74	0.34	0.37	0.80	0.74	0.70	1.21	1.22	0.66	0.62	0.80	0.37	0.76	0.48	0.68
Triangle Splatting	0.98	1.07	1.07	0.51	1.67	1.44	1.17	1.32	1.75	0.98	0.96	1.11	0.56	0.93	0.72	1.06
MeshSplatting	0.77	0.72	0.74	0.60	0.89	1.00	0.81	1.09	1.19	0.58	0.68	0.93	0.63	0.66	0.59	0.79
SPIRE (Ours)	0.53	0.68	0.42	0.43	1.06	1.19	0.90	1.09	1.70	0.48	0.67	1.06	0.78	0.50	0.53	0.80

Table 24: PSNR-gap decomposition on NeRF-Synthetic. Native uses each method's renderer, and DC rendering retains only the degree-zero SH component for view-independent color. Appearance and renderer gaps are the successive PSNR decreases across the three rendering conditions.

Method	PSNR			Gap	
	Native	DC	Blender	Appearance	Renderer
2DGS	31.90	27.39	21.96	4.51	5.43
GOF	33.25	28.21	26.01	5.04	2.20
MeshSplatting	29.42	26.52	25.98	**2.90**	0.54
SPIRE (Ours)	32.28	29.38	29.03	**2.90**	**0.35**



Figure 6: Full-frame Blender comparisons on NeRF-Synthetic. MeshSplatting uses its default external priors.

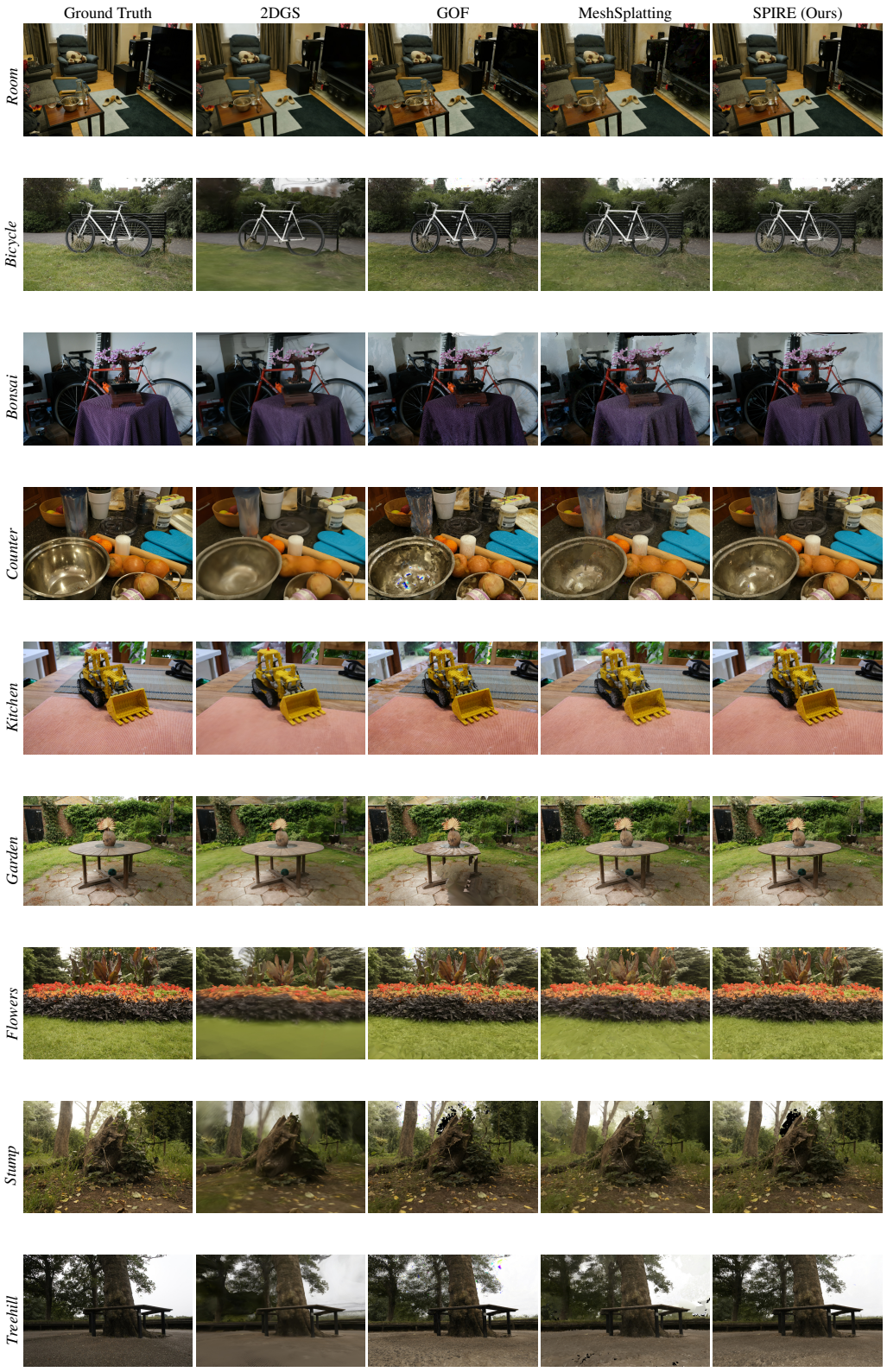


Figure 7: Full-frame Blender comparisons on Mip-NeRF 360. MeshSplatting uses its default external priors.

E Downstream Editing and Simulation

E.1 Endpoint mesh deformation

Table 25 expands the endpoint-editing comparison in Table 5 with all 15 evaluated methods and three image-quality metrics. The six scenes are Lego, Butterfly, Crab, Dolphin, Giraffe, and Lego-man from the PAPR-in-Motion data (Peng et al., 2024). We train on each start state and apply the same deformation to all reconstructed meshes, preserving topology and vertex colors without fitting to end-state images. We evaluate the edited meshes against end-state views using Blender and Filament. This evaluation reflects a downstream workflow in which a user edits a reconstructed mesh and renders it with its existing vertex colors, measuring the quality of the resulting asset after editing.

SPIRE achieves the best mean PSNR, SSIM, and LPIPS across all three rendering settings. Our PSNR is highest on every evaluated scene across the three rendering settings.

E.2 NeuralEditor deformation benchmark

Table 26 evaluates the released NeuralEditor edits (Chen et al., 2023a) on all eight NeRF-Synthetic scenes. We apply the official deformations and evaluate the edited meshes in Blender against the published views, without retraining or fitting to edited ground truth. SPIRE better preserves rendering quality even under more complex and fine-grained mesh deformations than those in the PAPR-in-Motion experiment.

E.3 Ball collisions with thin geometry

Figure 8(a) expands the Ficus comparison with PGSR and the original-geometry reference. The plant remains fixed, and all methods share the ball size, launch, and simulation settings. Mesh extraction misses thin leaf surfaces in 2DGS, allowing the ball to pass through where the original geometry and our mesh produce a collision. For other methods, extraction errors or discrepancies between the surface represented during training and the final opaque mesh can likewise alter contact geometry and make simulation behavior less reliable.

E.4 Articulation and appearance editing

Figure 8(b) applies the same arm and bucket motion to the reconstructed Lego meshes. Figure 8(c) transfers reference colors to selected Ficus leaves while preserving the pot and geometry.

E.5 Outdoor scene composition

Figure 8(d,e) demonstrates object relocation and insertion in Mip-NeRF 360 scenes. Garden follows the scene-editing setup in Figure 1, with object relocation and insertion of Lego and a chrome sphere. Bicycle combines a matte Stanford Bunny and a chrome sphere with each method's reconstructed Chair. All methods use the same edits and viewpoints to compare the usability of the reconstructed meshes in a composed scene. Together with the PAPR-in-Motion and NeuralEditor experiments, these examples show the importance of mesh reconstruction quality when objects are separated, deformed, and recomposed into a scene. SPIRE retains the highest mean rendering quality on both editing benchmarks, and its reconstructed geometry and appearance remain well preserved in these scene-composition examples.

E.6 Liquid interaction

Figure 8(f) compares liquid poured above the Garden vase under the same simulation conditions. The reconstructed meshes serve as collision geometry, showing how their surfaces guide the liquid flow.

Table 25: Full PAPR-in-Motion mesh-editing results averaged over six scenes. Higher PSNR and SSIM and lower LPIPS are better. Bold marks the best mean in each column.

	Blender^{64}			Blender^{1}			Filament		
Method	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS
NeuS2	24.42	0.9389	0.0666	24.06	0.9349	0.0766	24.04	0.9350	0.0764
SuGaR	26.25	0.9264	0.0927	25.72	0.9185	0.1066	25.69	0.9179	0.1071
2DGS	26.08	0.9364	0.0700	25.83	0.9347	0.0771	25.84	0.9348	0.0771
Gaussian Surfels	24.62	0.9367	0.0518	24.18	0.9303	0.0668	24.18	0.9304	0.0667
GOF	27.87	0.9522	0.0456	27.09	0.9428	0.0648	27.09	0.9431	0.0651
PGSR	26.54	0.9459	0.0506	26.12	0.9429	0.0589	26.12	0.9429	0.0589
Mip-Splatting	24.52	0.9353	0.0659	24.09	0.9316	0.0760	24.09	0.9316	0.0760
Analytic-Splatting	24.67	0.9357	0.0647	24.24	0.9320	0.0748	24.24	0.9320	0.0748
AA-2DGS	25.74	0.9345	0.0727	25.53	0.9328	0.0796	25.53	0.9328	0.0797
Gaussian Blending	23.15	0.9172	0.0853	22.72	0.9130	0.0950	22.72	0.9130	0.0950
nvdiffrec	25.13	0.9153	0.0950	24.81	0.9109	0.1032	24.82	0.9107	0.1032
MeshSplats	22.87	0.9070	0.1199	22.24	0.8893	0.1403	22.16	0.8894	0.1403
Triangle Splatting	17.04	0.8165	0.2257	16.22	0.8017	0.2401	16.22	0.8017	0.2401
MeshSplatting	26.68	0.9381	0.0696	26.04	0.9273	0.0880	26.07	0.9280	0.0878
SPIRE (Ours)	**28.80**	**0.9644**	**0.0320**	**28.13**	**0.9585**	**0.0493**	**28.13**	**0.9585**	**0.0492**

Table 26: NeuralEditor mesh-deformation results averaged over eight scenes, rendered in Blender^{64}. Bold marks the best result in each column.

Method	PSNR ↑	SSIM ↑	LPIPS ↓
NeuS2	22.47	0.9139	0.0805
SuGaR	21.80	0.8816	0.1474
2DGS	21.63	0.8890	0.1291
Gaussian Surfels	22.58	0.9013	0.0855
GOF	24.29	0.9131	0.0797
PGSR	20.31	0.8923	0.1142
Mip-Splatting	20.56	0.8915	0.1152
Analytic-Splatting	20.28	0.8885	0.1168
AA-2DGS	21.69	0.8822	0.1160
Gaussian Blending	22.68	0.8908	0.1109
nvdiffrec	22.48	0.8822	0.1147
MeshSplats	20.03	0.8559	0.1590
Triangle Splatting	17.42	0.8114	0.1883
MeshSplatting	24.89	0.9197	0.0807
SPIRE (Ours)	**25.27**	**0.9335**	**0.0647**

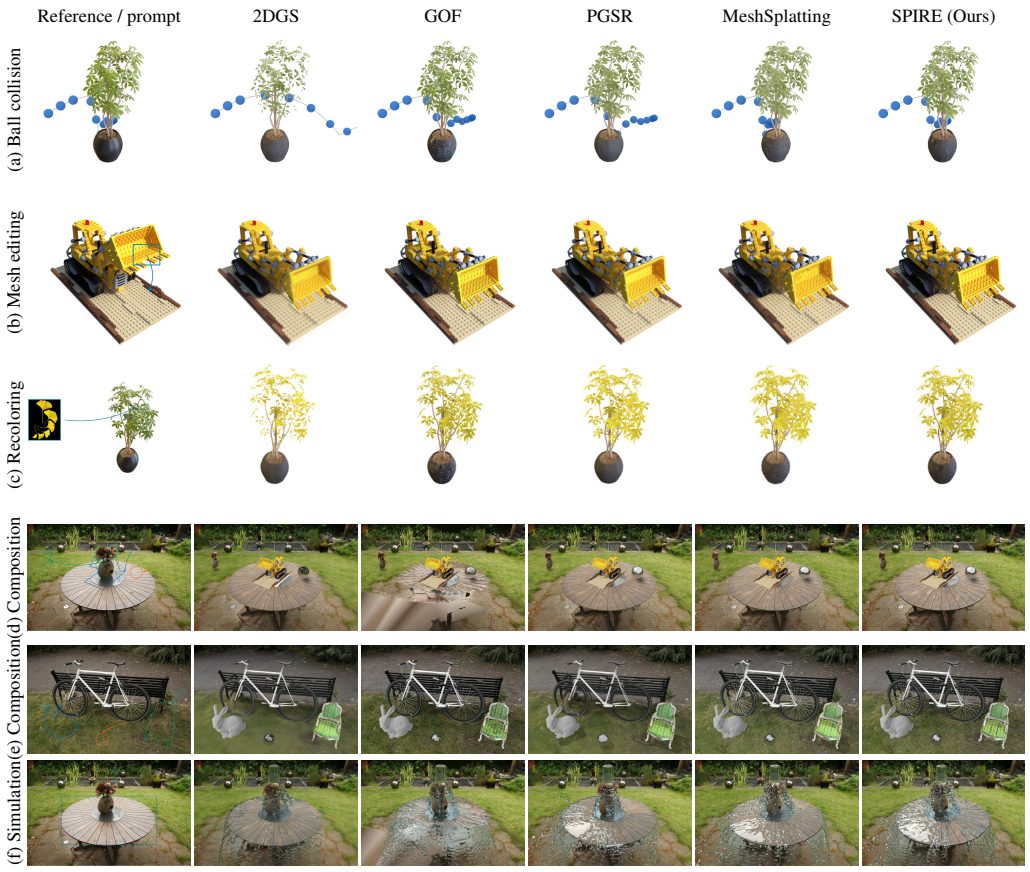


Figure 8: Downstream editing and simulation with reconstructed meshes. The first column shows the reference in (a) and editing or simulation prompts in (b)–(f). (a) Ficus ball trajectories under a common launch, with original geometry as the reference and ghost balls indicating successive positions. (b) Lego arm and bucket articulation. (c) Ficus leaf recoloring with geometry preserved. (d,e) Garden and Bicycle composition. (f) Liquid poured above the Garden vase under shared simulation conditions.