MESHSPLATBENCH: A UNIFIED BENCHMARK FOR TRIANGLE- AND MESH-BASED NEURAL RENDERING

Anonymous authors
Paper under double-blind review

ABSTRACT

Triangle- and mesh-based neural rendering aims to bridge neural scene representations and existing graphics engines (*e.g.*, Unity, Blender) by leveraging *triangle* primitives compatible with standard rasterization hardware. There have been several such methods driven by parallel efforts, developed and evaluated under inconsistent settings, with little or no comparison with each other; critically, most even have never used graphics engines for evaluation nor considered deployability in practice – significantly undermining the objective and motivation. To address these issues, we introduce **MeshSplatBench**, the very first benchmark of its kind that enables both systematic evaluation of *graphics engine* based deployment and Non-Engine Render evaluation for triangle- and mesh-based neural rendering methods. Importantly, a deployment protocol with two rendering options is introduced: (1) *Standard deployment* – a conventional opaque mesh pipeline with vertex colors and hardware Z-buffering; (2) *Dedicated deployment* – adding method-specific engine implementations supporting the retained appearance and compositing features (*e.g.*, alpha blending), so that the characteristics of each specific model can be taken into account. For mesh splatting, we further propose a structural audit of exported surfaces, diagnosing topological and geometric integrity toward downstream graphics assets. We highlight several results: (1) Graphics engine deployment would incur image-quality degradation under both deployment options. Among all methods tested, mesh splatting methods degrade the least under standard deployment. (2) The way of deployment matters – dedicated deployment can keep the majority of fidelity at about $6\sim30\times$ slowdown. (3) About mesh splatting, the current approaches to explicit connectivity and shared indexing are still limited: shared vertex indexing alone does not ensure manifoldness or global connectivity. From this benchmark, we validate that *rasterizability is merely part of graphics readiness*, and highlight the significance of assessing the graphics engine deployment process (*e.g.*, the degree of engine compatibility). Source code will be released.

1 INTRODUCTION

Triangle- and mesh-based neural rendering is attractive because it optimizes the *triangle* primitives already supported by standard graphics hardware (Held et al., 2026b; Sheng et al., 2025; Held et al., 2026a; Tojo et al., 2026). Compared with representations that rely on volumetric ray marching (Mildenhall et al., 2020) or specialized Gaussian splatting pipelines (Kerbl et al., 2023), this design appears to offer a more direct route to graphics engines such as Unity [1] and Blender [2]. However, the use of triangles alone does not make a learned representation directly deployable. Existing methods associate triangles with substantially different rendering mechanisms, including continuous soft coverage (Sheng et al., 2025), learned opacity (Held et al., 2026b), view-dependent spherical harmonics (SHs) (Held et al., 2026a), neural texture decoders (Tojo et al., 2026), and order-dependent alpha compositing (Sheng et al., 2025; Held et al., 2026b). A conventional opaque mesh pipeline discards many of these mechanisms, whereas retaining them inside a graphics engine requires method-specific shaders, buffers, sorting strategies, and runtime state. Consequently, a representation may

[1] https://www.unity.com
[2] https://www.blender.org

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.

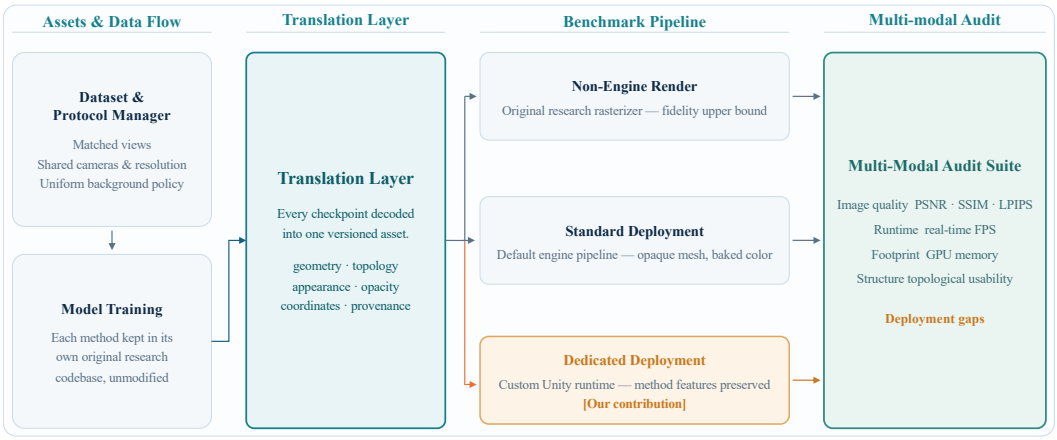


Figure 1: **MeshSplatBench overview.** Under identical evaluation controls, every trained representation is decoded into a single versioned asset and then evaluated along three matched paths: *non-engine rendering*, *standard* and *dedicated engine deployment*. Then MeshSplatBench quantifies deployment gaps in quality, runtime, and topological usability.

use rasterizable primitives in its non-engine renderer while still losing image quality, efficiency, or both after deployment.

Despite growing interest in triangle- and mesh-based neural rendering, current evaluation practice does not provide a systematic way to study this gap. First, existing methods are developed and reported under different dataset preprocessing, background policies, metric implementations, and non-engine renderers, making direct cross-method comparison difficult. Second, most evaluations stop at the source-code renderer and do not test the learned representation after export to a graphics engine. Non-Engine Render novel-view-synthesis (NVS) quality therefore does not reveal how much of the learned appearance can be retained under practical engine rendering, nor what computational cost is required to preserve it. Third, for methods that explicitly reconstruct connected geometry, rendering quality alone does not establish whether the exported representation forms a structurally useful mesh: vertex sharing, for example, does not necessarily imply manifoldness, watertightness, or global connectivity. These limitations leave three practical questions unresolved: *how existing methods compare under a common Non-Engine Render protocol, how their quality and efficiency change after engine deployment, and whether their exported geometry constitutes a structurally usable graphics asset*.

To address these questions, we introduce **MeshSplatBench**, a unified benchmark that evaluates triangle- and mesh-based neural rendering from Non-Engine Render evaluation to graphics-engine deployment (see Fig. 1). MeshSplatBench preserves each method's original optimization procedure and rendering semantics while standardizing datasets, cameras, backgrounds, and image metrics. The resulting *Non-Engine Render* evaluation provides a common reference for cross-method comparison. Each trained representation is then exported through a unified, versioned asset interface and evaluated in Unity under two deployment settings. *Dedicated deployment* retains supported method-specific appearance and compositing mechanisms through custom engine implementations, whereas *Standard deployment* reduces the representation to a conventional opaque vertex-colored mesh rendered with hardware Z-buffering. Using identical cameras, ground truth, and metric implementations across all three conditions allows the quality and efficiency changes introduced by deployment to be measured directly. For mesh-based representations, MeshSplatBench additionally performs a structural audit of the exported geometry, measuring connectivity, boundary structure, and non-manifold behavior.

Our evaluation produces three findings. First, graphics-engine deployment degrades image quality under both deployment options, and MeshSplatting (Held et al., 2026a) loses the least quality on the Standard opaque path (Tab. 4). Second, Dedicated deployment retains more source-renderer appearance than Standard deployment, but this added support costs about 6–30$\times$ in rendering efficiency and can change method rankings after export (Appendix Fig. 5). Third, a topological audit of MeshSplatting (Held et al., 2026a) shows that explicit vertex sharing alone does not ensure manifoldness or global connectivity: exported surfaces contain many open and non-manifold elements

Method	Connectivity	Continuous coverage	Binary opacity	SHs	Neural decoder	Alpha compositing
2DTS	✗	✓	✗	✓	✗	✓
Triangle-Splatting	✗	✓	✗	✓	✗	✓
MeshSplatting	✓	✗	✗	✓	✗	✓
DiffSoup	✗	✗	✓	✗	✓	✗

Table 1: Native representation and Non-Engine rendering attributes of the evaluated methods. Deployment-specific feature support is reported separately in Tab. 2.

and split into hundreds of thousands of edge-connected components (Tab. 5). These results reveal that *rasterizability is only one part of graphics readiness*.

Our contributions are as follows: (**I**) We propose **MeshSplatBench**, the first unified benchmark for representative triangle- and mesh-based neural rendering methods that standardizes Non-Engine Render evaluation while preserving each method's original optimization procedure. (**II**) We implement method-specific shader paths for all benchmarked methods, retaining each method's supported appearance and compositing features for Dedicated deployment. (**III**) We conduct a joint evaluation of reconstruction quality, runtime efficiency, training cost, and deployment retention that exposes the quality–efficiency trade-off between Dedicated and Standard deployment. (**IV**) We establish a structural audit protocol for exported meshes and analyze MeshSplatting, separating vertex reuse from manifoldness and global connectivity.

2 Related Work

Neural Scene Representations. Neural radiance fields (NeRFs) represent a scene as a continuous function and synthesize views through volumetric integration (Mildenhall et al., 2020). Later methods improve reconstruction quality and anti-aliasing (Barron et al., 2021; 2022; 2023) or accelerate rendering through explicit, factorized, and hashed structures (Reiser et al., 2024; Yu et al., 2021; Fridovich-Keil et al., 2022; Chen et al., 2022; Müller et al., 2022). 3DGS replaces repeated network queries with optimized anisotropic Gaussians and a dedicated splatting rasterizer (Kerbl et al., 2023). Follow-up work refines Gaussian optimization or encourages surface-aligned geometry (Kheradmand et al., 2024; Huang et al., 2024; Guédon & Lepetit, 2024). These methods establish strong NVS baselines, but their outputs remain coupled to specialized renderers or require a separate conversion stage before entering a conventional graphics pipeline.

Neural Asset Deployment and Graphics Conversion. A parallel line of research investigates extracting deployable polygon meshes from implicit fields or explicit splats. Methods such as MobileNeRF (Chen et al., 2023) and BakedSDF (Yariv et al., 2023) bake radiance fields into textured polygon meshes with feature textures executed via lightweight fragment shaders. For explicit splats, SuGaR (Guédon & Lepetit, 2024) and GaMeS (Waczyńska et al., 2024) extract mesh surfaces from 3D Gaussians or bind Gaussian primitives to deformable mesh faces. Although these conversion pipelines produce standard mesh assets, they often rely on lossy post-processing (*e.g.*, Marching Cubes, Poisson surface reconstruction, or heuristic simplification) that decouples the optimization target from the final rendering objective. In contrast, triangle-based neural rendering optimizes the deployable primitive end-to-end, raising the fundamental question of whether natively optimized triangles are truly graphics-ready without lossy approximations.

Triangle-Based Neural Representations. To bridge the gap between neural primitives and standard graphics hardware, triangle-based neural representations optimize geometry directly expressible by hardware rasterizers (Held et al., 2026b; Sheng et al., 2025; Held et al., 2026a; Tojo et al., 2026). These approaches diverge into two distinct paradigms: (i) *discontinuous triangle soups* (*e.g.*, 2DTS (Sheng et al., 2025) and Triangle-Splatting (Held et al., 2026b)) that parameterize independent planar facelets optimized with continuous coverage kernels and splatting-style alpha blending; and (ii) *structured or indexed meshes* (*e.g.*, MeshSplatting (Held et al., 2026a)) that introduce explicit local connectivity or neural texture parameterizations. As categorized in Tab. 1, their core differences extend far beyond geometric representation. However, these methods are developed and reported under different evaluation pipelines, making direct cross-method comparison difficult. Meanwhile, most evaluations stop at the source-code renderer and do not test the learned representation after export to a graphics engine and MeshSplatBench aims to bridge these gaps.

Method	Standard Deployment		Dedicated Deployment					
	Support	Appearance	Support	View-dep.	Opacity	Soft cov.	Depth sort	Appearance
3DGS	N/A				N/A			
2DGS	Fallback	baked color			N/A			
2DTS	Fallback	baked SH-DC	Method-aware	✓	✓	✓	✓	SH
Triangle-Splatting	Fallback	baked SH-DC	Method-aware	✓	✓	✓	✓	SH
MeshSplatting	Fallback	baked SH-DC	Method-aware	✓	✓	–	✓	SH
DiffSoup	N/A		Method-aware	✓	✓	–	✓	MLP

Table 2: Condition-level engine support and retained rendering semantics. *Fallback* denotes opaque static-color rendering; *Method-aware* supports the listed functions. N/A: unsupported.

Neural Rendering Benchmarks. Standardized benchmarking has become essential for rigorous evaluation in NVS. However, existing benchmarks (Kulhanek & Sattler, 2025; Liu et al., 2026) operate exclusively within non-engine research environments, evaluating image-space metrics while leaving post-deployment behavior unexamined. MeshSplatBench complements these efforts by extending evaluation beyond Non-Engine Render evaluation: it standardizes non-engine workflows while introducing a matched multi-condition engine deployment protocol and a comprehensive structural topology audit, systematically quantifying the fidelity drop, hardware overhead, and geometric integrity.

3 MeshSplatBench Framework

MeshSplatBench establishes an end-to-end benchmark from source research codebases to graphics-engine deployment. As illustrated in Fig. 1, it evaluates each trained representation under common camera, background, and metric controls along three matched paths: Non-Engine Render, Standard Deployment, and Dedicated Deployment. This design isolates deployment-induced differences in visual quality and runtime cost; for exported indexed meshes, it additionally audits structural usability. The supporting modular software architecture is detailed in Appendix Fig. 4.

3.1 Evaluation Protocol Standardization

Rigorous cross-method evaluation requires eliminating confounding discrepancies across research environments (Kulhanek & Sattler, 2025; Liu et al., 2026). MeshSplatBench establishes three standardized protocols:

Camera and Viewport Alignment: For a held-out camera set $\mathcal{C}$, the benchmark standardizes camera coordinate frames, optical axes, field of view, and projection matrices.

Background Compositing Policy: Prior works adopt inconsistent backgrounds (*e.g.*, on DTU (Jensen et al., 2014), Triangle-Splatting trains over black backgrounds whereas 2DTS composites over solid white). We enforce a uniform alpha-composited white-background protocol for masked objects (see Appendix Sec. A.3.2).

Perceptual Metric Standardization: To resolve implementation discrepancies (*e.g.*, unnormalized vs. $[-1, 1]$ normalized tensors), we establish a fixed evaluation suite: PSNR (Hore & Ziou, 2010), SSIM (Wang et al., 2004), and LPIPS (Zhang et al., 2018) computed strictly using a VGG (Simonyan & Zisserman, 2015) backbone with image normalized to $[-1, 1]$.

3.2 Universal Translation Layer and Asset Interface

Existing triangle-based representations store parameters in proprietary PyTorch state dictionaries coupled to custom CUDA rasterization kernels: 2DTS (Sheng et al., 2025) and Triangle-Splatting (Held et al., 2026b) maintain independent triangle soups, MeshSplatting (Held et al., 2026a) binds SHs to 3D Delaunay vertices, and DiffSoup (Tojo et al., 2026) embeds a multi-resolution feature lattice with a coordinate MLP decoder.

To bridge this divide, MeshSplatBench introduces a **translation layer** (the method adapter $\mathcal{E}_m$). For any trained model $\mathcal{S}_m$, the adapter executes an automated decoding pass that serializes the model

into a unified, versioned engine asset container $\mathcal{A}_m$:

$$\mathcal{S}_m \xrightarrow{\mathcal{E}_m} \mathcal{A}_m. \tag{1}$$

The resulting asset interface standardizes primitive geometry, view-dependent appearance, neural feature lattices, and opacity parameters for downstream graphics runtimes. Executing entirely on host CPU without CUDA drivers or PyTorch runtime dependencies, $\mathcal{A}_m$ supports direct ingestion by downstream graphics runtimes (see Appendix Sec. A.2 for implementation details).

3.3 Two-Tier Graphics Engine Deployment Runtime

As illustrated in Fig. 1, MeshSplatBench organizes rendering into three distinct execution pathways, separating a non-engine research baseline from two engine deployment tiers:

Non-engine render (I_m^{ne}): Evaluates each trained checkpoint $\mathcal{S}_m$ directly within its official codebase using native CUDA rasterizers: $I_m^{\mathrm{ne}}(c) = \mathcal{R}_m^{\mathrm{ne}}(\mathcal{S}_m, c), \forall c \in \mathcal{C}$, where c indexes a held-out camera, $\mathcal{C}$ is the matched camera set, $\mathcal{S}_m$ is the trained representation of method m, and $\mathcal{R}_m^{\mathrm{ne}}$ is its source renderer. This establishes the algorithmic upper bound of visual fidelity in the absence of engine constraints.

Standard deployment (I_m^{std}): Evaluates representations when treated as conventional 3D game assets without proprietary shader support. The standard tier strips neural decoders and alpha blending: diffuse radiance is baked into static per-vertex RGB colors, and geometry is rendered through standard mesh renderer with native hardware Z-Buffering and opaque rasterization.

Dedicated deployment (I_m^{ded}): To evaluate appearance preservation when an engine provides tailored runtime support, we implement custom renderers inside Unity, preserving method-specific features (Tab. 2).

3.4 Multi-Modal Audit: Gaps, Efficiency, and Topology

MeshSplatBench formalizes a three-dimensional audit protocol across visual, computational, and structural dimensions:

Visual fidelity and deployment gaps: Following Sec. 3.1, we report NVS quality using PSNR (Hore & Ziou, 2010), SSIM (Wang et al., 2004), and LPIPS (Zhang et al., 2018) using a VGG backbone with image normalization. For any metric q, mean fidelity over camera set $\mathcal{C}$ is $Q_m^r = \frac{1}{|\mathcal{C}|}\sum_{c\in\mathcal{C}} q(I_m^r(c), I^{\mathrm{gt}}(c))$ for $r \in \{\mathrm{ne}, \mathrm{ded}, \mathrm{std}\}$. We quantify deployment loss through three orthogonal gaps:

$$\Delta Q_m^{\mathrm{adapt}} = \left|Q_m^{\mathrm{ne}} - Q_m^{\mathrm{ded}}\right|, \quad \Delta Q_m^{\mathrm{port}} = \left|Q_m^{\mathrm{ded}} - Q_m^{\mathrm{std}}\right|, \quad \Delta Q_m^{\mathrm{deploy}} = \left|Q_m^{\mathrm{ne}} - Q_m^{\mathrm{std}}\right|. \tag{2}$$

The *adaptation gap* $\Delta Q_m^{\mathrm{adapt}}$ measures fidelity lost from CUDA kernels to engine shaders; the *portability gap* $\Delta Q_m^{\mathrm{port}}$ measures the collapse from specialized shaders to standard opaque rasterization; and the total *deployment gap* $\Delta Q_m^{\mathrm{deploy}} = \Delta Q_m^{\mathrm{adapt}} + \Delta Q_m^{\mathrm{port}}$ reflects end-to-end degradation from research prototype to standard game asset.

Computational overhead and inference profiling: Beyond visual quality, we audit computational overhead across pipelines. During training, we track peak VRAM (MiB) and optimization time (min). In Unity deployment, automated Linux builds under Vulkan (Appendix Sec. A.1) query GPU timestamp registers to profile median (P50) and tail (P95) GPU frame latencies, wall-clock throughput (FPS), and runtime VRAM consumption.

Surface geometry and mesh structural integrity: We evaluate 3D surface geometry via Chamfer Distance (CD) on DTU against reference scans. For exported meshes, rendering as triangles does not guarantee a valid geometric surface. MeshSplatBench audits structural integrity across three complementary axes: (i) *Local reuse* (V/F ratio, vertex valence); (ii) *Local structure* (boundary edge ratio, non-manifold edge/vertex ratios); and (iii) *Global connectivity* (LCC face/area ratios, total component count). See Appendix Sec. A.4 for formal definitions.

4 BENCHMARK EVALUATION

Methods. We evaluate four representative triangle- and mesh-based representations: 2DTS (Sheng et al., 2025), Triangle-Splatting (Held et al., 2026b), DiffSoup (Tojo et al., 2026), and MeshSplatting (Held et al., 2026a). The first three produce *triangle soups*, comprising independent triangles without explicit shared-vertex connectivity. 2DTS and Triangle-Splatting use continuous soft coverage, learned opacity, and splatting-style alpha blending, whereas DiffSoup combines direct differentiable rasterization with neural texture features and a color decoder. In contrast, MeshSplatting produces an *indexed mesh*, introducing shared vertices and explicit local connectivity through 3D Delaunay triangulation, with SH-based view-dependent appearance. However, this mesh structure does *not* itself guarantee manifoldness or global connectivity, motivating the structural audit in Sec. 4.3. Together, these methods cover distinct geometry and rendering designs (Tab. 1) for evaluating graphics-engine deployment.

Datasets. We evaluate on four complementary datasets spanning real-world environments and object-centric captures. Mip-NeRF 360 (Barron et al., 2022) comprises nine real scenes, with surrounding camera trajectories, cluttered geometry, and large depth variations that challenge reconstruction of both foreground and distant content. For Tanks and Temples (Knapitsch et al., 2017), we use the two outdoor scenes *train* and *truck*, featuring large real-world objects, complex backgrounds, and occlusions. DTU (Jensen et al., 2014) provides laboratory captures of varied objects from 49 or 64 calibrated camera positions per scene, together with structured-light reference scans, enabling geometric evaluation alongside image fidelity. NeRF-Synthetic (Mildenhall et al., 2020) contains eight path-traced object-centric scenes with detailed geometry and realistic non-Lambertian materials; its original release provides 100 training and 200 test views per scene at 800×800 resolution. Known camera poses and controlled rendering make it complementary to real captures, while fine structures and view-dependent reflections test the representations' ability to reproduce complex appearance.

Implementation details. Mip-NeRF 360 images are $2\times$ and $4\times$ downsampled for indoor and outdoor scenes; the remaining datasets use $2\times$ downsampling. For DiffSoup, following its original implementation, we uniformly downsample all images by $4\times$ to accommodate its memory requirements. DTU is evaluated under the standardized white-background protocol (Sec. 3.1 and Appendix Sec. A.3.2). To ensure fair comparisons, we preserve the original optimization pipeline of each method, including its official training hyperparameters, primitive or topology updates, densification or refinement schedule, renderer, and checkpointing behavior. Thus, all methods share the same evaluation conventions while retaining their core algorithmic designs. All training and evaluation use a single NVIDIA RTX 4090 with 24 GB memory.

Reproducibility. Before cross-method comparisons, we benchmark MeshSplatBench against reported values and public codebases. Across all 27 Mip-NeRF 360 method–scene pairs with published per-scene results, our reproduction achieves tight agreement, with a mean absolute PSNR difference of only 0.075 dB and a maximum relative deviation below 0.8%[3]. This confirms solid baseline fidelity on the evaluated pairs without claiming identical geometry or optimization trajectories (see Fig. 6 and Tab. 11 in Appendix for more details).

4.1 STANDARDIZED NON-ENGINE RENDER BENCHMARK

Tab. 3 brings the standardized Non-Engine Render results across four datasets (see per-dataset breakdowns in Appendix Sec. A.3). Across Mip-NeRF 360 (Barron et al., 2022), NeRF-Synthetic (Mildenhall et al., 2020), and DTU (Jensen et al., 2014), 2DTS achieves the highest reconstruction fidelity (28.16 dB, 33.65 dB, and 29.66 dB PSNR), demanding the lowest peak training memory on synthetic and object scenes ($3{,}172$ MiB and $2{,}470$ MiB). Triangle-Splatting yields the sharpest perceptual fidelity, attaining the lowest LPIPS on Tanks and Temples (0.171) and DTU (0.080). DiffSoup achieves the highest throughput (up to 371 FPS) and fastest training (12.2–16.8 min), but its rigid planar approximation without adaptive densification limits fidelity (17.45 dB on Tanks and Temples). MeshSplatting enforces indexed connectivity, converging fastest on DTU (10.7 min) with lower photometric fidelity on unbounded scenes (24.72 dB). Thus, *no* representation dominates fidelity, throughput, memory, and training time simultaneously.

[3] DiffSoup is omitted since it does not report per-scene metrics in its original paper.

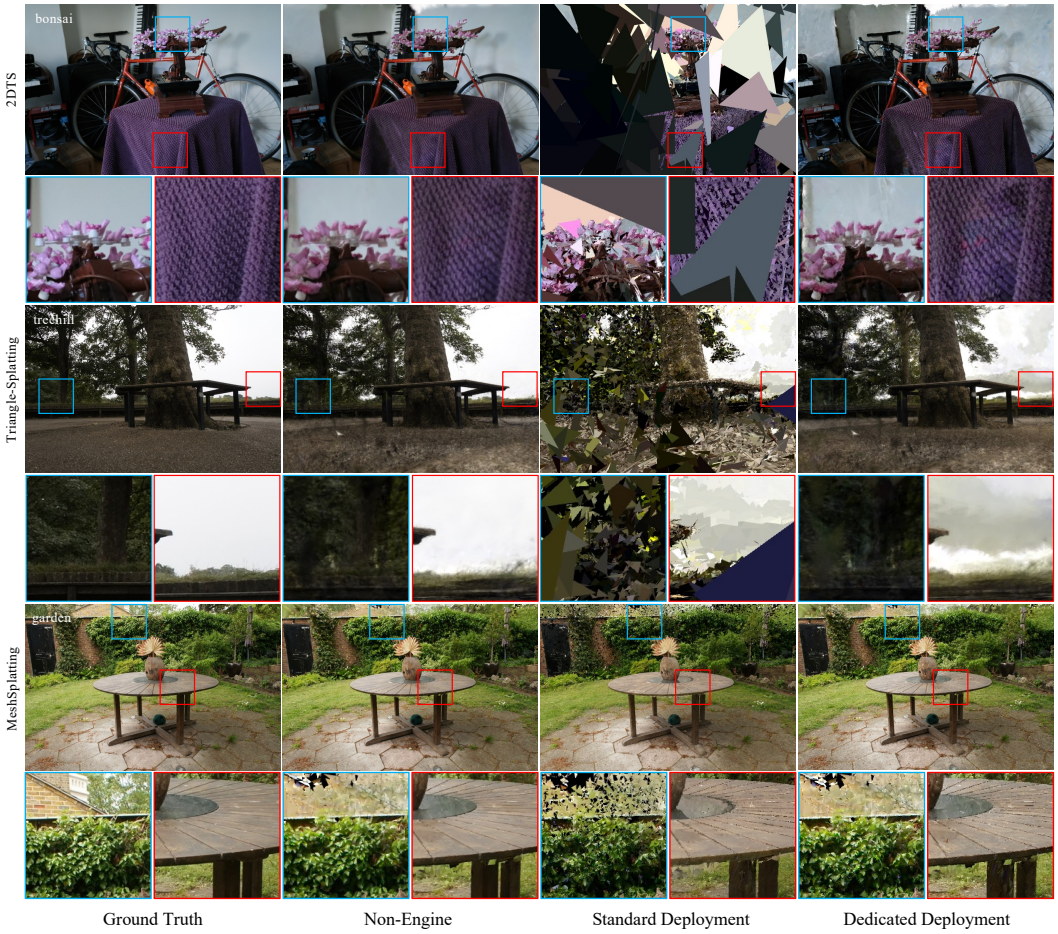


Figure 2: Qualitative NVS results of 2DTS (Sheng et al., 2025), Triangle-Splatting (Held et al., 2026b) and MeshSplatting (Held et al., 2026a) on the different scenes with different rendering protocols.

Dataset	Method	PSNR ↑	SSIM ↑	LPIPS ↓	FPS ↑	Peak mem. (MiB) ↓	Train time (min) ↓	CD ↓
Mip-NeRF 360	2DTS	**28.16**	**0.841**	**0.215**	70	**10069**	46.1	–
	Triangle-Splatting	27.13	0.812	0.227	94	18901	40.5	–
	MeshSplatting	24.72	0.729	0.365	38	23134	41.7	–
	DiffSoup	23.54	0.689	0.322	**168**	23484	**16.8**	–
Tanks and Temples	2DTS	**23.42**	0.853	0.203	101	16061	25.9	–
	Triangle-Splatting	23.08	**0.856**	**0.171**	159	**11157**	24.5	–
	MeshSplatting	20.53	0.756	0.319	66	21321	24.1	–
	DiffSoup	17.45	0.645	0.389	**226**	23890	**15.4**	–
NeRF-Synthetic	2DTS	**33.65**	**0.970**	**0.034**	126	**3172**	16.3	–
	Triangle-Splatting	29.87	0.956	0.055	243	12257	19.1	–
	MeshSplatting	29.50	0.945	0.071	86	23386	13.4	–
	DiffSoup	28.65	0.910	0.076	**371**	16334	**12.2**	–
DTU	2DTS	**29.66**	**0.951**	0.110	**312**	**2470**	21.2	**0.63**
	Triangle-Splatting	27.78	0.944	**0.080**	165	4163	13.1	1.27
	MeshSplatting	27.82	0.933	0.106	114	7680	**10.7**	0.80
	DiffSoup	17.91	0.760	0.263	170	23930	14.5	3.89

Table 3: Standardized Non-Engine Render results across four benchmark datasets.

2D novel-view fidelity vs. 3D surface geometry. Incorporating DTU into Tab. 3 provides a complementary evaluation of NVS alongside physical 3D surface geometry. The empirical results show that 2D photometric fidelity does not strictly track 3D surface accuracy. While 2DTS leads in both PSNR (29.66 dB) and CD (0.63 mm), Triangle-Splatting attains the lowest LPIPS (0.080) and comparable PSNR (27.78 dB) but yields a noticeably higher CD of 1.27 mm. Meanwhile, MeshSplatting

Use engine	Deployment	Metric	3DGS	2DGS	2DTS	Triangle-Splatting	Diffsoup	MeshSplatting
✗	-	PSNR ↑	**27.21**	**26.79**	**28.16**	**27.13**	**23.54**	**24.72**
		SSIM ↑	**0.815**	**0.796**	**0.841**	**0.812**	**0.689**	**0.729**
		LPIPS ↓	**0.214**	**0.252**	**0.215**	**0.227**	**0.322**	**0.365**
		FPS ↑	109	38	70	94	168	38
✓	Standard	PSNR ↑	N/A	8.88	12.52	14.77	N/A	19.83
		SSIM ↑		0.076	0.311	0.246		0.491
		LPIPS ↓		0.699	0.661	0.656		0.518
		FPS ↑		**1933**	**3754**	**2294**		**943**
✓	Dedicated	PSNR ↑	N/A	N/A	23.28	22.96	23.15	22.52
		SSIM ↑			0.699	0.695	0.647	0.589
		LPIPS ↓			0.313	0.301	0.331	0.457
		FPS ↑			545	70	**1995**	157

Table 4: Comparisons on the Mip-NeRF 360 dataset.

achieves competitive surface accuracy (0.80 mm CD) despite lower 2D fidelity on unbounded scenes (24.72 dB on Mip-NeRF 360), and DiffSoup exhibits higher surface error (3.89 mm CD). These divergences indicate that high 2D NVS fidelity does not guarantee accurate 3D surface recovery, motivating joint geometric and photometric evaluation before graphics deployment.

4.2 Renderer-Conditioned Deployment

Tab. 4 details quality and throughput profiles on Mip-NeRF 360, while Appendix Fig. 5 summarizes Non-Engine–Dedicated–Standard PSNR trajectories across datasets. Under our engine contract, 3DGS (Kerbl et al., 2023) cannot deploy to either deployment tier; 2DGS (Huang et al., 2024) lacks dedicated shaders because TSDF meshing discards source-renderer appearance variables (though its mesh runs under Standard deployment); and DiffSoup lacks a Standard deployment opaque path due to its neural feature decoder.

Deployment quality. NVS under engine conditions reveals tier-dependent losses in visual fidelity. Dedicated renderers preserve much of the Non-Engine appearance but still incur adaptation loss (ΔQ^{adapt}): the opacity-splatting soup methods 2DTS and Triangle-Splatting lose 4.17–4.88 dB, while DiffSoup and MeshSplatting lose 0.39 dB and 2.20 dB, respectively. Standard deployment removes custom shading and relies on hardware Z-buffering; this causes severe portability loss (ΔQ^{port}) for the soup methods, with drops of 8–11 dB. For example, 2DTS produces giant opaque shards on *bonsai* (a 15.64 dB drop), and Triangle-Splatting exhibits the same failure mode on *treehill* (Fig. 2). In contrast, MeshSplatting has the smallest Standard-deployment loss, at 2.69 dB from Dedicated to Standard (Fig. 2). The ranking therefore changes across deployment tiers: 2DTS and Triangle-Splatting lead under Non-Engine and Dedicated rendering, whereas MeshSplatting is best under Standard deployment across datasets (Appendix Fig. 5 and Appendix Tab. 9). These results characterize visual usability in an opaque engine, not whether an exported mesh is manifold, globally connected, or suitable for editing; Sec. 4.3 evaluates this separate structural axis. Supplementary videos compare the methods across the rendering protocols.

Quality–efficiency trade-off. Comparing Dedicated and Standard deployment exposes a direct trade-off between visual fidelity and engine execution cost. Dedicated deployment preserves substantial appearance over the Standard opaque path via soft coverage and learned transparency. However, fidelity retention costs 6.0–32.8× in throughput relative to Standard deployment (*e.g.*, 2294 vs. 70 FPS in Triangle-Splatting). Under Dedicated deployment, per-camera depth sorting, alpha blending, and neural shading incur median GPU frame times (P50) of 0.5–6.4 ms on Mip-NeRF 360 (70–1995 FPS) and demand 288–1559 MiB VRAM. While narrow P50–P95 spreads (< 0.2 ms) indicate low GPU-latency variability in these runs, standard opaque rasterization reduces GPU frame times to 0.3–1.1 ms, with throughput of 943–3754 FPS and VRAM of 54 MiB (Appendix Sec. A.3.1). Dedicated deployment thus retains appearance at a substantial end-to-end runtime cost relative to the Standard path.

Benchmarking protocol and interpretation boundaries. Engine evaluations run on Linux under Vulkan at canonical viewports. End-to-end FPS and GPU latency diverge due to CPU–GPU serialization: in Dedicated Triangle-Splatting, median GPU latency is 3.8 ms ($\sim$ 263 theoretical FPS), but throughput is 70 FPS (14.3 ms) because CPU depth sorting and buffer streaming account for approximately 10.5 ms. Standard deployment renders static buffers under hardware Z-buffering at 2294 FPS (0.44 ms). The resulting 6.0–32.8× throughput difference is an end-to-end renderer-condition comparison, not an attribution to a single implementation choice (Appendix Sec. A.1).

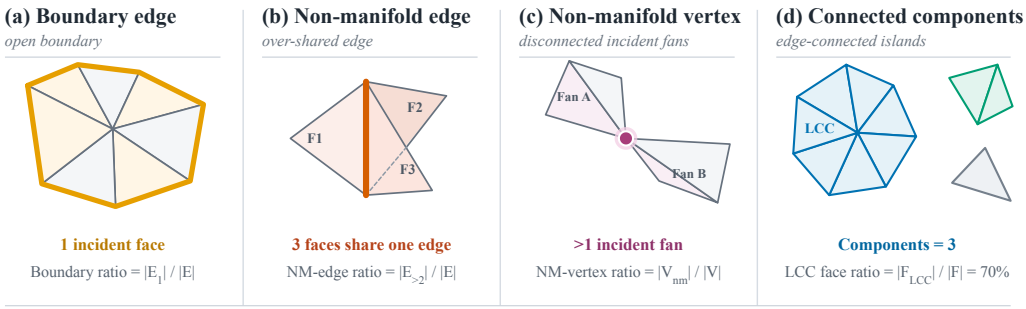


Figure 3: Topology diagnostics used in the structural audit. (a) Boundary edges have exactly one incident face. (b) Non-manifold edges are shared by more than two faces. (c) Non-manifold vertices contain multiple disconnected incident-face fans around the same vertex. (d) Edge-connected components are maximal face sets joined through complete shared edges.

Scene	Local reuse		Local structure			Global connectivity		
	V/F	Valence	Boundary edge	Non-manif. edge ↓	Non-manif. vertex ↓	LCC face ↑	LCC area ↑	Components ↓
Outdoor scenes								
bicycle	0.57	5.90	0.45	0.16	0.57	0.69	0.67	0.72 M
flowers	0.62	5.62	0.49	0.14	0.55	0.65	0.63	0.76 M
garden	0.48	6.85	0.43	0.20	0.70	0.73	0.73	0.77 M
stump	0.57	5.92	0.45	0.16	0.57	0.68	0.64	0.76 M
treehill	0.59	5.77	0.46	0.16	0.56	0.68	0.66	0.75 M
Indoor scenes								
room	0.46	7.22	0.46	0.18	0.75	0.71	0.71	0.71 M
counter	0.46	7.08	0.46	0.20	0.73	0.70	0.70	0.62 M
kitchen	0.41	7.56	0.40	0.24	0.75	0.77	0.78	0.51 M
bonsai	0.54	6.46	0.48	0.16	0.68	0.68	0.68	0.81 M
Mean	**0.52**	**6.49**	**0.45**	**0.18**	**0.65**	**0.70**	**0.69**	**0.71 M**

Table 5: Per-scene mesh diagnostics for MeshSplatting (Held et al., 2026a) on Mip-NeRF 360.

4.3 Mesh Structural Results

Using the three mesh-structure groups introduced above, we evaluate MeshSplatting, the benchmarked method that exports a shared-vertex indexed mesh. Tab. 5 reports Local reuse, Local structure, and Global connectivity on Mip-NeRF 360; Fig. 3 illustrates the diagnosed structures, and Appendix Sec. A.4 provides exact definitions.

Local reuse. The mean vertex-to-face ratio of 0.52 and mean valence of 6.49 show substantial vertex sharing and dense local adjacency. These values are close to those of regular triangular connectivity ($V/F \approx 0.5$ and valence 6), but measure reuse rather than manifoldness or geometric correctness.

Local structure. Boundary edges account for 45% of edges on average, indicating that exported surfaces are extensively open (not necessarily erroneous for partial scenes). More critically, 18% of edges and 65% of vertices are non-manifold, revealing invalid local neighborhoods that disrupt normals, UV parameterization, subdivision, remeshing, and collision handling.

Global connectivity. The mesh averages 0.71 million edge-connected components, despite a dominant LCC containing 70% of faces and 69% of surface area (77% and 78% on *kitchen*). The exported asset thus combines a large primary surface with severe fragmentation. Coupled with strong Standard deployment quality, this separates visual usability from structural integrity: an asset may render as a coherent proxy while remaining unsuitable for topology-dependent processing.

5 Conclusion

MeshSplatBench benchmarks when rasterizable triangles become practical graphics assets. Our Non-Engine–Dedicated–Standard protocol exposes fidelity drops and ranking inversions across research and engine pipelines. Topological audits reveal open boundaries, non-manifold structures, and fragmentation despite indexed connectivity. Across the evaluated methods, achieving graphics readiness requires jointly considering geometry, topology, and appearance; no method simultaneously optimizes fidelity, efficiency, portability, and structural integrity.

ETHICS STATEMENT

MeshSplatBench uses public reconstruction datasets to improve reproducible graphics-deployment evaluation. Deployed systems should use authorized data, disclose reconstructed content, and preserve attribution.

REPRODUCIBILITY STATEMENT

We will publicly release the MeshSplatBench code and evaluation configurations. Secs. 3–4 define the Non-Engine Render protocol, exported asset interface, matched Unity conditions, metrics, datasets, preprocessing, and hardware. In the Appendix, Sec. A.1 documents renderer implementation details, Sec. A.3 provides full timing, resource, Non-Engine Render, and reproducibility results, and Sec. A.4 provides topology definitions and audit diagnostics.

AI USE STATEMENT

Generative AI tools assisted with formatting and ICLR-template migration; the authors reviewed the manuscript and take responsibility for its content.

REFERENCES

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 *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 5855–5864, 2021.

Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-NeRF 360: Unbounded anti-aliased neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5470–5479, 2022.

Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Zip-nerf: Anti-aliased grid-based neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 19697–19705, 2023.

Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. TensoRF: Tensorial radiance fields. In *Computer Vision – ECCV 2022*, pp. 333–350. Springer, 2022.

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

Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5501–5510, 2022.

Antoine Guédon and Vincent Lepetit. SuGaR: Surface-aligned gaussian splatting for efficient 3D mesh reconstruction and high-quality mesh rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5354–5365, 2024.

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. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7320–7329, 2026a.

Jan Held, Renaud Vandeghen, Adrien Deliege, Abdullah Hamdi, Daniel Rebain, Silvio Giancola, Anthony Cioppa, Andrea Vedaldi, Bernard Ghanem, Andrea Tagliasacchi, and Marc Van Droogenbroeck. Triangle splatting for real-time radiance field rendering. In *International Conference on 3D Vision*, 2026b.

Alain Hore and Djemel Ziou. Image quality metrics: Psnr vs. ssim. In *2010 20th international conference on pattern recognition*, pp. 2366–2369. IEEE, 2010.

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. ACM, 2024.

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.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3D gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4):1–14, 2023.

Shakiba Kheradmand, Daniel Rebain, Gopal Sharma, Weiwei Sun, Yang-Che Tseng, Hossam Isack, Abhishek Kar, Andrea Tagliasacchi, and Kwang Moo Yi. 3d gaussian splatting as markov chain monte carlo. In *Advances in Neural Information Processing Systems*, 2024. Spotlight Presentation.

Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics*, 36(4), 2017.

Jonas Kulhanek and Torsten Sattler. Nerfbaselines: Consistent and reproducible evaluation of novel view synthesis methods. *Advances in Neural Information Processing Systems*, 38:127378–127403, 2025.

Xiang Liu, Yimin Zhou, Jinxiang Wang, Yujun Huang, Shuzhao Xie, Shiyu Qin, Mingyao Hong, Jiawei Li, Yaowei Wang, Zhi Wang, Shu-Tao Xia, and Bin Chen. Splatwizard: A benchmark toolkit for 3d gaussian splatting compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Findings*, pp. 2261–2271, 2026.

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. In *Proceedings of the European Conference on Computer Vision*, pp. 405–421, 2020.

Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics*, 41(4):1–15, 2022.

Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. KiloNeRF: Speeding up neural radiance fields with thousands of tiny MLPs. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 14335–14345, 2024.

Kaifeng Sheng, Zheng Zhou, Yingliang Peng, and Qianwei Wang. 2D triangle splatting for direct differentiable mesh training, 2025.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

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.

Joanna Waczyńska, Piotr Borycki, Sławomir Tadeja, Jacek Tabor, and Przemysław Spurek. Games: Mesh-based adapting and modification of gaussian splatting. *arXiv preprint arXiv:2402.01459*, 2024.

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.

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.

Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. PlenOctrees for real-time rendering of neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 5752–5761, 2021.

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

A APPENDIX

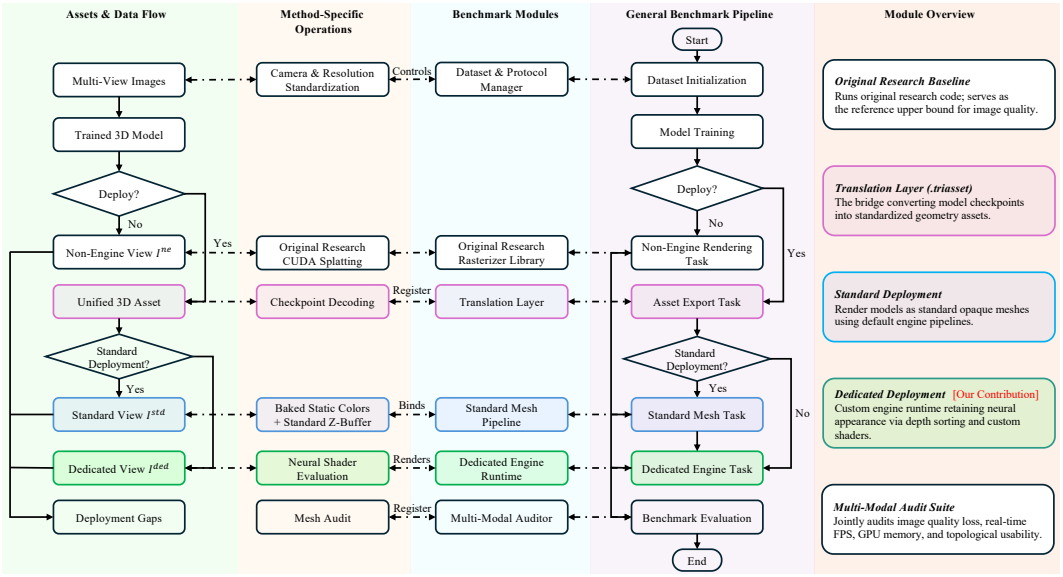


Figure 4: **Software architecture and implementation overview of MeshSplatBench.** The benchmark infrastructure resolves the academic-to-engine impedance mismatch through five modular pillars: (1) *Assets & Data Flow*, tracking representations from training datasets to multi-tier rendered views; (2) *Method-Specific Operations*, encapsulating low-level decoders, custom shaders, and sorting routines; (3) *Benchmark Modules*, providing decoupled libraries for data protocols, translation, and runtime execution; (4) *General Benchmark Pipeline*, orchestrating an automated multi-pathway evaluation; and (5) *Multi-Modal Audit Suite*, jointly quantifying visual degradation, runtime overhead, and topological surface validity.

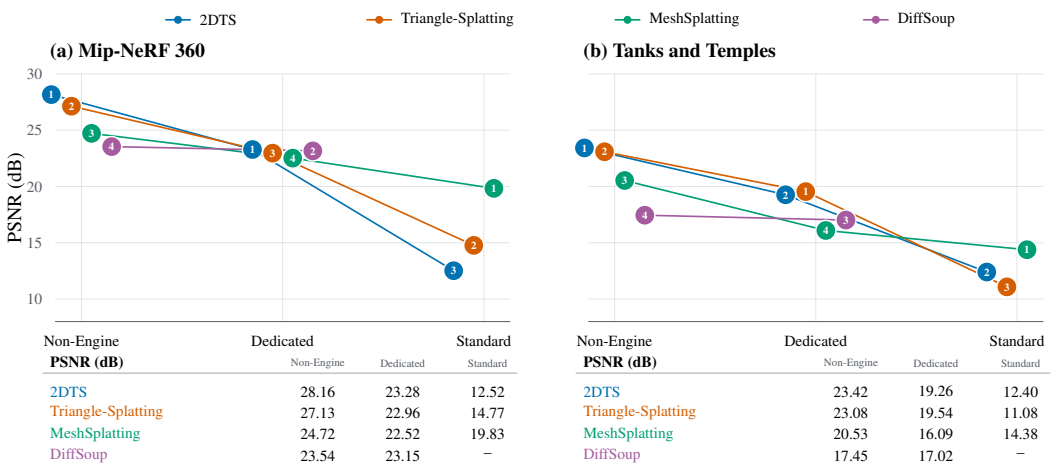
A.1 UNITY DEPLOYMENT IMPLEMENTATION DETAILS AND KNOWN MISMATCHES

Dedicated deployment. Dedicated deployment implements the appearance and compositing functions that each exported method requires. Depending on the method, this includes view-dependent SH evaluation, activated opacity, continuous coverage, ordered alpha blending, and a shader-side evaluation of a compact learned appearance decoder. Opaque terminal assets use depth-tested rendering. Tab. 2 reports the supported feature set for every method–condition pair.

Differences from Non-Engine Render. Dedicated deployment is designed to retain the relevant rendering semantics, not to reproduce the source renderer numerically. In particular, source rasterizers may use tile-local ordering and per-pixel evaluation, whereas the engine path uses engine-compatible ordering, coverage reconstruction, and attribute sampling. These differences can affect transparency, view-dependent appearance, and large or oblique primitives; they are implementation differences rather than controlled ablations of individual rendering choices.

Standard deployment. Standard deployment converts an asset to an opaque, depth-tested vertex-colored mesh. When available, it bakes the zeroth-order SH coefficient into vertex RGB as $c_0\mathbf{f}_{\mathrm{dc}} + 0.5$, where $c_0 = 0.28209$, and omits higher-order SH, opacity, coverage, neural features, and method-specific compositing. Methods without a validated static color representation, such as DiffSoup, are reported as unsupported under this condition.

Benchmarking setup and timing. Engine evaluations use a headless Linux player under Vulkan on an NVIDIA GeForce RTX 4090 (24 GB VRAM), with vsync disabled and the canonical dataset viewports. Novel-view capture and timing are separated to avoid file I/O in runtime measurements. FPS measures end-to-end wall-clock throughput; GPU timing reports P50 and P95 frame latency from timestamp queries; and runtime VRAM is measured separately. Consequently, the Dedicated–Standard efficiency gap is an end-to-end condition-level result and should not be attributed to any single implementation difference.



PSNR (dB)	Non-Engine	Dedicated	Standard
2DTS	28.16	23.28	12.52
Triangle-Splatting	27.13	22.96	14.77
MeshSplatting	24.72	22.52	19.83
DiffSoup	23.54	23.15	–

PSNR (dB)	Non-Engine	Dedicated	Standard
2DTS	23.42	19.26	12.40
Triangle-Splatting	23.08	19.54	11.08
MeshSplatting	20.53	16.09	14.38
DiffSoup	17.45	17.02	–

Figure 5: Deployment gaps on (a) Mip-NeRF 360 and (b) Tanks and Temples. Each line follows one method from its source-code Non-Engine renderer through Dedicated and Standard deployment.

A.2 Unified Engine Asset Contract (.triasset) Specification

To bridge the divide between PyTorch/CUDA research representations and standard game engine asset ingestion pipelines, MeshSplatBench establishes the versioned `.triasset` specification. This asset contract decouples training checkpoint serialization from engine consumption through a self-describing metadata manifest and memory-aligned binary raw buffers:

Metadata Manifest (`manifest.json`): Defines primitive topology (*triangle soup* vs. *indexed mesh*), facelet count F, vertex count V, axis-aligned bounding extents $[\mathbf{p}_{\min}, \mathbf{p}_{\max}]$, hierarchical bounding sphere radius R_{bound}, spherical harmonics degree ($0 \leq \ell \leq 3$), opacity kernel parameterization (*e.g.*, continuous Gaussian power-law exponents γ, ε or logistic sigmas), and required engine shader passes (*e.g.*, procedural URP forward pass vs. opaque unlit pass).

Contiguous Raw Binary Buffers: Attributes are serialized into byte-aligned, little-endian float32 and uint32 arrays for direct zero-copy GPU streaming:

- *Geometry buffers:* Vertex positions ($\mathbf{p} \in \mathbb{R}^{V\times 3}$) and triangle indices ($\mathbf{idx} \in \mathbb{N}^{F\times 3}$). For unindexed triangle soups ($V = 3F$), indices represent trivial sequential triplets.
- *Appearance buffers:* Zero-order spherical harmonics ($\mathbf{f}_{\text{dc}} \in \mathbb{R}^{V/F\times 3}$) and higher-order anisotropic coefficients ($\mathbf{f}_{\text{rest}} \in \mathbb{R}^{V/F\times 45}$ for $\ell = 3$).
- *Alpha and neural feature buffers:* Opacity logits or sigmas ($\boldsymbol{\sigma} \in \mathbb{R}^{V/F}$), continuous coverage exponents, and for neural coordinate decoders (*e.g.*, DiffSoup), multi-resolution feature lattice embeddings paired with serialized linear transformation weights and bias vectors.

Because the translation adapter $\mathcal{E}_m$ executes entirely on host CPU memory without invoking CUDA drivers or PyTorch runtime bindings, assets are fully portable across heterogeneous platforms (Windows, Linux, macOS) and can be streamed into arbitrary graphics runtimes.

A.3 Comprehensive Benchmark and Deployment Results across Datasets

This section provides the complete empirical evaluations and deployment profiles across all benchmarked datasets, expanding upon the summarized results in Sec. 4. We report per-scene quality, runtime efficiency, and resource consumption under Non-Engine Render research pipelines and standardized Unity deployment tiers, alongside a calibration check of baseline reproducibility. In addition, Fig. 5 visualizes the multi-tier PSNR deployment trajectories across Mip-NeRF 360 and Tanks and Temples, illustrating the Non-Engine–Dedicated–Standard deployment gaps and ranking inversions.

Scene	Dedicated Deployment (ms ↓)								Standard Deployment (ms ↓)					
	2DTS		Triangle-Splatting		MeshSplatting		DiffSoup		2DTS		Triangle-Splatting		MeshSplatting	
	P50	P95	P50	P95	P50	P95	P50	P95	P50	P95	P50	P95	P50	P95
bonsai	1.4	1.4	3.1	3.2	6.9	7.2	0.4	0.4	0.2	0.2	0.4	0.4	1.1	1.1
counter	1.7	1.9	3.2	3.7	5.5	5.7	0.4	0.4	0.2	0.2	0.4	0.4	1.0	1.0
kitchen	1.7	1.9	2.5	2.8	5.8	5.9	0.4	0.5	0.2	0.3	0.4	0.4	1.1	1.1
room	1.4	1.4	2.2	2.3	6.7	6.9	0.3	0.3	0.2	0.2	0.3	0.3	1.1	1.1
Indoor mean	1.6	1.7	2.8	3.0	6.2	6.4	0.4	0.4	0.2	0.2	0.4	0.4	1.1	1.1
bicycle	2.6	2.6	4.9	5.0	6.3	6.5	0.7	0.7	0.3	0.3	0.5	0.6	1.0	1.0
flowers	2.6	2.7	5.0	5.1	6.0	6.2	0.6	0.7	0.3	0.3	0.6	0.6	0.9	0.9
garden	2.6	2.6	4.5	4.5	7.9	8.2	0.7	0.8	0.3	0.4	0.5	0.5	1.3	1.3
stump	2.6	2.6	4.3	4.4	6.4	6.7	0.6	0.7	0.3	0.3	0.5	0.5	1.1	1.1
treehill	2.7	2.7	4.6	4.7	6.3	6.5	0.7	0.7	0.3	0.3	0.5	0.5	1.0	1.0
Outdoor mean	2.6	2.6	4.7	4.8	6.6	6.8	0.7	0.7	0.3	0.3	0.5	0.5	1.1	1.1
All-scene mean	2.1	2.2	3.8	4.0	6.4	6.6	0.5	0.6	0.3	0.3	0.5	0.5	1.1	1.1

Table 6: Per-scene GPU frame-time percentiles (ms) for the Mip-NeRF 360 deployment evaluations; lower is better. DiffSoup is omitted under Standard deployment due to renderer incompatibility.

A.3.1 Mip-NeRF 360 Non-Engine Render and Unity Deployment Results

Tab. 8 reports standardized Non-Engine Render averages across Mip-NeRF 360 (Barron et al., 2022), Tanks and Temples (Knapitsch et al., 2017), NeRF-Synthetic (Mildenhall et al., 2020), and DTU (Jensen et al., 2014), validating baseline novel-view synthesis fidelity alongside source-training and source-rendering throughput costs. To evaluate real-world game engine execution, Tabs. 6–7 provide complementary per-scene GPU frame-time percentiles and graphics memory allocations under Dedicated deployment and Standard deployment. The indoor, outdoor, and all-scene rows in Tabs. 6–7 are arithmetic means over scenes; in particular, the reported mean P95 values are means of scene-level P95 values.

Engine memory scaling and the 54 MiB allocation floor. In Tab. 7, Dedicated deployment requires substantial runtime graphics allocations, averaging $1{,}140$ MiB for 2DTS, $1{,}559$ MiB for Triangle-Splatting, 927 MiB for MeshSplatting, and 288 MiB for DiffSoup. This substantial footprint stems from method-specific GPU buffers for view-dependent appearance, coverage or opacity, and primitive dispatch. In contrast, under Standard deployment, runtime graphics allocations drop to an almost strictly invariant ≈ 54 MiB across all supported methods, regardless of the underlying primitive counts (uniformly 57 MiB across indoor scenes, 51–52 MiB across outdoor scenes, and an all-scene average of 54 MiB).

This invariance arises from two architectural mechanisms. First, Standard deployment completely strips neural appearance representations, converting the exported scene into an unlit, opaque standard `Mesh` with baked DC vertex colors ($c_0\mathbf{f}_{\mathrm{dc}} + 0.5$). Discarding high-order SH arrays, opacity logits, and depth-sorting order buffers shrinks the raw mesh vertex and index buffers to merely several megabytes on GPU memory. Second, the remaining ≈ 50 MiB represents the static baseline allocation floor of the Unity Universal Render Pipeline (URP) engine runtime at the target render resolution, encompassing the main color target, depth-stencil buffer, swapchain swap-buffers, and driver context allocations. Because the simplified mesh geometry footprint is completely submerged beneath this fixed engine runtime floor, GPU memory usage decouples from method-specific triangle densities and varies solely with the viewport resolution across indoor and outdoor scene configurations.

A.3.2 DTU Evaluation Protocol and Standardization

Different baseline methods adopt divergent training protocols on the DTU dataset (Jensen et al., 2014): certain methods (*e.g.*, Triangle-Splatting) do not consider the alpha channel and train over an unmasked black background, whereas others (*e.g.*, 2DTS) utilize object masks and composite over a solid white background. To establish an equitable and standardized benchmark across representations, we uniformly standardize all methods to the white-background protocol, with the consolidated results reported in Tab. 8.

Scene	Dedicated Deployment VRAM (MiB) ↓				Standard Deployment VRAM (MiB) ↓		
	2DTS	Triangle-Splatting	MeshSplatting	DiffSoup	2DTS	Triangle-Splatting	MeshSplatting
bonsai	572	1159	975	298	57	57	57
counter	571	1043	779	298	57	57	57
kitchen	571	1001	779	298	57	57	57
room	572	890	887	298	57	57	57
Indoor mean	571	1023	855	298	57	57	57
bicycle	1599	2084	956	280	51	51	51
flowers	1594	2136	935	280	52	52	52
garden	1602	1930	1078	281	52	52	52
stump	1579	1838	982	280	52	52	52
treehill	1601	1950	971	280	52	52	52
Outdoor mean	1595	1988	984	280	52	52	52
All-scene mean	1140	1559	927	288	54	54	54

Table 7: Per-scene GPU runtime graphics memory allocations (VRAM in MiB) for the Mip-NeRF 360 deployment evaluations. DiffSoup is omitted under Standard deployment due to renderer incompatibility.

Dataset	Method	PSNR ↑	SSIM ↑	LPIPS ↓	FPS ↑	Peak mem. (MiB) ↓	Train time (min) ↓	CD ↓
Mip-NeRF 360	2DTS	**28.16**	**0.841**	**0.215**	70	**10069**	46.1	–
	Triangle-Splatting	27.13	0.812	0.227	94	18901	40.5	–
	MeshSplatting	24.72	0.729	0.365	38	23134	41.7	–
	DiffSoup	23.54	0.689	0.322	**168**	23484	**16.8**	–
Tanks and Temples	2DTS	**23.42**	0.853	0.203	101	16061	25.9	–
	Triangle-Splatting	23.08	**0.856**	**0.171**	159	**11157**	24.5	–
	MeshSplatting	20.53	0.756	0.319	66	21321	24.1	–
	DiffSoup	17.45	0.645	0.389	**226**	23890	**15.4**	–
NeRF-Synthetic	2DTS	**33.65**	**0.970**	**0.034**	126	**3172**	16.3	–
	Triangle-Splatting	29.87	0.956	0.055	243	12257	19.1	–
	MeshSplatting	29.50	0.945	0.071	86	23386	13.4	–
	DiffSoup	28.65	0.910	0.076	**371**	16334	**12.2**	–
DTU	2DTS	**29.66**	**0.951**	0.110	**312**	**2470**	21.2	**0.63**
	Triangle-Splatting	27.78	0.944	**0.080**	165	4163	13.1	1.27
	MeshSplatting	27.82	0.933	0.106	114	7680	**10.7**	0.80
	DiffSoup	17.91	0.760	0.263	170	23930	14.5	3.89

Table 8: Standardized Non-Engine Render results across Mip-NeRF 360 (Barron et al., 2022), Tanks and Temples (Knapitsch et al., 2017), NeRF-Synthetic (Mildenhall et al., 2020), and DTU (Jensen et al., 2014). DTU results are reported under the white-background evaluation protocol. Results are averaged across scenes within each dataset.

Specifically, the RGBA PNG alpha channel is utilized as an opacity matte to isolate the central object, compositing both ground-truth target images and novel-view renderings over a clean, solid white background:

$$\mathbf{I}_{\text{white}} = \mathbf{I}_{\text{RGB}} \odot \alpha + (1.0 - \alpha) \cdot \mathbf{1}, \tag{3}$$

with the renderer background color correspondingly set to white ($\mathbf{c}_{\text{bg}} = [1, 1, 1]$). This standardizes object-centric evaluation and prevents unconstrained background capture noise from polluting photometric error calculations.

Surface geometry (*i.e.*, Chamfer Distance) is evaluated against official DTU reference point clouds using `ObsMask` bounding volume culling to isolate the object surface. Under this standardized setting, 2DTS achieves both the highest novel-view synthesis fidelity (29.66 dB PSNR) and the best surface reconstruction accuracy (0.63 mm Chamfer distance), while Triangle-Splatting attains the lowest perceptual error (0.080 LPIPS), and MeshSplatting preserves consistent geometry (0.80 mm CD). Comparing 2D novel-view synthesis rankings with 3D Chamfer distance rankings demonstrates that 2D photometric error and 3D geometric accuracy represent complementary, partially decoupled optimization objectives.

Condition	Method	train				truck				Mean			
		PSNR ↑	SSIM ↑	LPIPS ↓	FPS ↑	PSNR ↑	SSIM ↑	LPIPS ↓	FPS ↑	PSNR ↑	SSIM ↑	LPIPS ↓	FPS ↑
Dedicated	2DTS	**18.79**	**0.650**	**0.330**	1222	**19.72**	**0.642**	**0.262**	596	**19.26**	**0.646**	**0.296**	909
	Triangle-Splatting	**18.44**	**0.705**	**0.301**	469	**20.63**	**0.733**	**0.215**	604	**19.54**	**0.719**	**0.258**	536
	MeshSplatting	**15.67**	**0.572**	**0.465**	249	**16.52**	**0.602**	**0.405**	350	**16.09**	**0.587**	**0.435**	300
	DiffSoup	**15.38**	**0.546**	**0.482**	**2431**	**18.66**	**0.666**	**0.321**	**2345**	**17.02**	**0.606**	**0.402**	**2388**
Standard	2DTS	11.68	0.313	0.655	**6643**	13.12	0.361	0.603	**4316**	12.40	0.337	0.629	**5480**
	Triangle-Splatting	9.26	0.257	0.710	**3587**	12.89	0.327	0.613	**4269**	11.08	0.292	0.661	**3928**
	MeshSplatting	13.88	0.485	0.519	**1313**	14.88	0.528	0.462	**1743**	14.38	0.506	0.490	**1528**
	DiffSoup	N/A											

Table 9: Comparison between Dedicated deployment and Standard deployment on Tanks and Temples (Knapitsch et al., 2017), reported per scene and averaged over two scenes. N/A means renderer incompatibility.

Renderer	Method	GPU Frame Time (ms) ↓						Memory Footprint (MiB) ↓					
		train		truck		Mean		train		truck		Mean	
		P50	P95	P50	P95	P50	P95	Render	Asset	Render	Asset	Render	Asset
Dedicated	2DTS (Sheng et al., 2025)	0.8	0.8	1.7	1.7	1.2	1.3	562	234	1103	469	832	352
	Triangle-Splatting (Held et al., 2026b)	2.1	2.2	1.7	1.7	1.9	1.9	1025	556	841	440	933	498
	MeshSplatting (Held et al., 2026a)	4.0	4.1	2.9	2.9	3.4	3.5	673	434	518	346	596	390
	DiffSoup (Tojo et al., 2026)	0.4	0.4	0.4	0.4	0.4	0.4	276	258	276	258	276	258
Standard	2DTS (Sheng et al., 2025)	0.2	0.2	0.2	0.2	0.2	0.2	48	234	48	469	48	352
	Triangle-Splatting (Held et al., 2026b)	0.3	0.3	0.2	0.2	0.3	0.3	48	556	48	440	48	498
	MeshSplatting (Held et al., 2026a)	0.8	0.8	0.6	0.6	0.7	0.7	48	434	48	346	48	390
	DiffSoup (Tojo et al., 2026)	N/A											

Table 10: Per-scene GPU frame-time percentiles (ms) and renderer graphics allocation / serialized asset size (MiB) for the Tanks and Temples deployment evaluations. N/A means renderer incompatibility.

A.3.3 Tanks and Temples Benchmark and Deployment Results

As summarized in Tab. 8, on Tanks and Temples (Knapitsch et al., 2017), Non-Engine Render novel-view synthesis and efficiency metrics show strong performance for Triangle-Splatting and DiffSoup, while Tab. 9 evaluates deployment fidelity across Dedicated deployment and Standard deployment. Furthermore, Tab. 10 details the per-scene GPU frame-time percentiles and memory footprints. Across both scenes (*train* and *truck*), dedicated shaders deliver higher visual fidelity, whereas Standard deployment maximizes frame throughput. Under Dedicated deployment, Triangle-Splatting achieves the best mean image quality (19.54 dB PSNR), while DiffSoup reaches the highest throughput (2,388 FPS). Under Standard deployment, MeshSplatting retains the highest quality (14.38 dB PSNR), demonstrating that an indexed opaque mesh representation degrades substantially less without method-specific appearance shaders.

A.3.4 NeRF-Synthetic Non-Engine Render Evaluation

To demonstrate that MeshSplatBench can apply the integrated methods to an additional dataset under a shared protocol, we evaluate all four methods on NeRF-Synthetic (Mildenhall et al., 2020). As reported in Tab. 8, 2DTS achieves the strongest average image fidelity, reaching 33.65 dB PSNR, 0.970 SSIM, and 0.034 LPIPS. Compared with the second-best Triangle-Splatting, this corresponds to a 3.78 dB PSNR gain, a 0.014 SSIM improvement, and a 0.021 reduction in LPIPS. 2DTS also requires the least peak training memory at 3,172 MiB. However, its fidelity advantage does not extend to computational speed: DiffSoup achieves the highest rendering throughput at 371 FPS and the shortest training time at 12.2 minutes, whereas 2DTS reaches 126 FPS and trains in 16.3 minutes. These results reinforce that no representation dominates across all criteria and expose a clear trade-off between rendering fidelity, memory consumption, and computational efficiency.

A.3.5 Detailed Implementation Reproducibility

To ensure that benchmark conclusions rest upon faithful implementations, we calibrate MeshSplatBench against published values and official public repositories. As illustrated in Fig. 6, our reproduced models achieve tight numerical agreement across nine Mip-NeRF 360 and two Tanks and

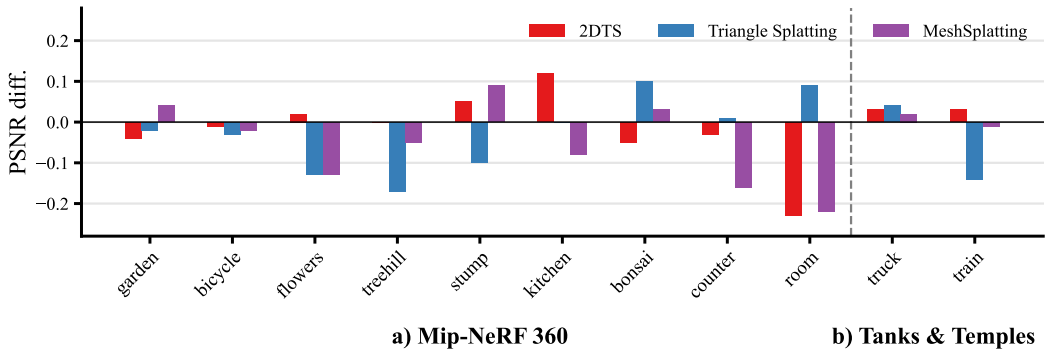


Figure 6: Per-scene PSNR reproduction differences between MeshSplatBench and paper-reported baselines ($\mathrm{PSNR}_{\mathrm{MeshSplatBench}} - \mathrm{PSNR}_{\mathrm{paper}}$). Panel (a) evaluates nine Mip-NeRF 360 scenes; panel (b) evaluates two Tanks and Temples scenes. Across all evaluated pairs, the largest relative deviation is less than 0.8%. DiffSoup is omitted since its original paper reports no per-scene metrics.

Dataset	Method	Paper			Public			MeshSplatBench		
		PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
Mip-NeRF 360	2DTS	**28.18**	**0.842**	0.218	**28.51**	**0.851**	0.215	**28.16**	**0.841**	**0.215**
	Triangle-Splatting	27.17	0.814	**0.192**	27.20	0.813	**0.190**	27.13	0.812	0.227
	MeshSplatting	24.78	0.728	0.310	24.78	0.728	0.310	24.72	0.729	0.365
	DiffSoup	24.76	0.748	0.204	23.69	0.691	0.241	23.54	0.689	0.322
Tanks and Temples	2DTS	**23.39**	0.853	0.204	**23.44**	0.853	0.171	**23.42**	0.853	0.203
	Triangle-Splatting	23.14	**0.857**	**0.143**	23.09	**0.856**	**0.144**	23.08	**0.856**	**0.171**
	MeshSplatting	20.52	0.745	0.287	20.58	0.758	0.272	20.53	0.756	0.319
	DiffSoup	–	–	–	18.38	0.671	0.299	18.90	0.685	0.352

Table 11: Quantitative comparison of implementation reproducibility on the Mip-NeRF 360 (Barron et al., 2022) and Tanks and Temples (Knapitsch et al., 2017) datasets. DiffSoup does not report official Tanks and Temples results.

Temples scenes, with a mean absolute PSNR difference of only 0.075 dB and a maximum relative deviation below 0.8%. Across all nine Mip-NeRF 360 scenes in panel (a), reproduction deviations remain bounded within ± 0.35 dB, with the largest single deviation occurring on Triangle-Splatting for *treehill* (-0.79%). On Tanks and Temples in panel (b), 2DTS, Triangle-Splatting, and MeshSplatting closely track published results within 0.05 dB. Tab. 11 provides complete metric-level comparisons across PSNR, SSIM, and LPIPS on Mip-NeRF 360 and Tanks and Temples. Minor perceptual metric differences may arise from variations in LPIPS evaluation protocols, such as the feature backbone, model version, or preprocessing. DiffSoup does not specify its LPIPS backbone, whereas MeshSplatBench uniformly adopts VGG for all methods.

A.4 Topology-Metric Definitions and Interpretation Limits

Let V, E, and F denote the sets of unique vertices, undirected edges, and triangular faces in one exported mesh. For connected-component analysis, we construct a face-adjacency graph in which two faces are adjacent only when they share a complete edge; a corner contact alone does not connect components. We exclude faces with repeated vertex indices from this graph and apply no component-size threshold. Tab. 5 in the main paper reports each metric per Mip-NeRF 360 scene and the arithmetic mean over the nine scenes.

Local reuse. The V/F ratio $|V|/|F|$ measures unique vertices per face; for a fixed face count, a lower value indicates more vertex sharing under the same export convention. UV seams, normal splits, and material boundaries can still duplicate colocated vertices. Average valence is $\frac{1}{|V|}\sum_{v\in V}\deg(v) = \frac{2|E|}{|V|}$ and summarizes local adjacency density. It approaches 6 for an ideal closed triangular 2-manifold, but neither statistic alone measures surface quality.

Local structure. The boundary-edge ratio is $|E_1|/|E|$, where E_1 contains edges incident to one face. The non-manifold-edge ratio is $|E_{>2}|/|E|$, and the non-manifold-vertex ratio is $|V_{\mathrm{nm}}|/|V|$, where incident faces at a non-manifold vertex fail to form one connected fan. A closed interior fan

and an open boundary fan are valid; bow-tie or multiple fans are not. Boundaries may be legitimate for partial scenes, so these measures diagnose structure rather than error against a watertight target.

Global connectivity. The largest connected component (LCC) is extracted from the edge-based face-adjacency graph. LCC Face and LCC Area are its fractions of total faces and triangle area, while Components is the raw number of edge-connected face components. Higher LCC fractions and fewer components indicate greater global connectivity but not geometric correctness, manifoldness, or absence of self-intersections. Component count is resolution- and scene-dependent and must therefore be interpreted with the LCC fractions and visual inspection.