HEXELATE: COMPACT SURFACE PRIMITIVES FOR EFFICIENT SCENE RENDERING AND RECONSTRUCTION

Anonymous authors
Paper under double-blind review

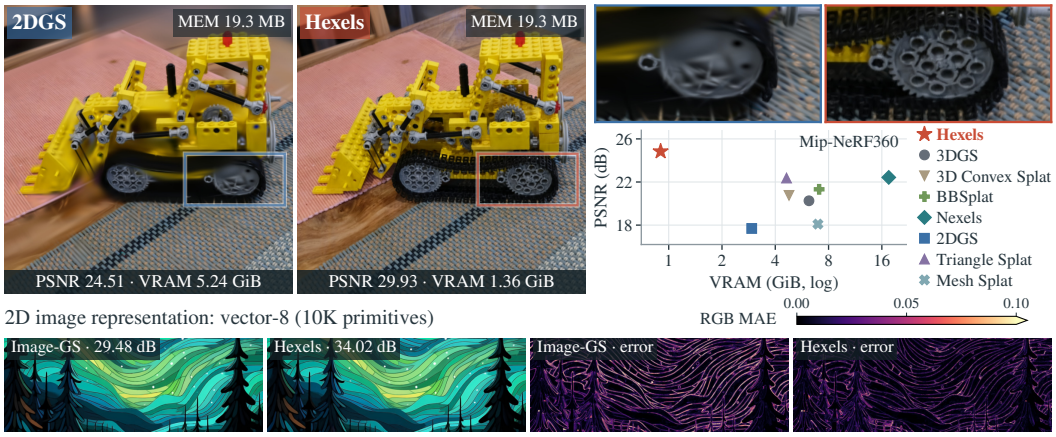


Figure 1: Hexels capture finer detail with less training memory. Top: Hexels and 2DGS at the same model size (MEM), where VRAM denotes peak training memory. The graph shows averages over nine Mip-NeRF 360 scenes under matched model-memory budgets (Table 2), with a logarithmic VRAM axis, showing that Hexels outperform all other methods while requiring significantly less cost. Bottom: Hexels are also capable of efficiently represent images and can be seen outperforming ImageGS using same number primitives.

ABSTRACT

Novel view synthesis and 3D reconstruction have advanced rapidly with radiance fields and splatting, yet their memory and computational demands limit their use in resource-constrained settings. We introduce **Hexels**, planar hexagonal surface elements with a first-order spatial color field for efficient 3D scene and 2D image representation. Hexels exploit locally planar regions with smoothly varying appearance using fewer, larger primitives. Their hexagonal support provides an area and boundary-efficient domain, while the local color field captures appearance variation within each primitive.Across 34 scenes, compact fixed-population Hexels improve PSNR over memory-matched 2DGS by 4.53–7.38 dB across dataset subsets. With densification and view-dependent appearance, Hexels improve PSNR over unconstrained 2DGS by 0.89 dB on DTU and 0.88 dB on Tanks and Temples. On Mip-NeRF360, they are 0.33 dB below 2DGS while rendering at 2.06× its speed with 83.9% less deployment storage. For all-view DTU surface reconstruction, fixed-population Hexels achieve 0.978 mm Chamfer distance, reducing error by 28.9% relative to memory-matched 2DGS, although unconstrained 2DGS remains more accurate. The same primitive also improves 2D image representation over Image-GS.

1 INTRODUCTION

Novel view synthesis and 3D reconstruction have progressed rapidly, from neural radiance fields (Mildenhall et al., 2020) to explicit Gaussian primitives that render in real time (Kerbl et al., 2023). 2DGS (Huang et al., 2024) extends this paradigm to planar surface splats, while Triangle Splatting (Held et al., 2025b) uses independent polygons. These methods differ in geometry, but all face the same question: how much of the scene can one primitive represent, and at what cost? For resource-constrained settings, efficiency must include not only rendering speed and model size, but

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.

also the memory required during training. We therefore target the quality–efficiency trade-off at the primitive itself.

Natural images and scenes are highly structured: large regions are often locally planar with smoothly varying appearance. Conventional Gaussian and triangle splats typically represent one view-conditioned color per primitive, so spatial variation emerges through many overlapping elements. BBSplat (Svitov et al., 2025) instead adds per-primitive textures, while NeST (Zhang et al., 2025a) and Nexels (Rong et al., 2025) use shared neural appearance fields. These methods increase per-element capacity, but move complexity into textures, grids, or decoders. We ask instead whether a compact, self-contained primitive can directly represent a larger structured region.

We introduce *Hexels*, planar surface elements that combine a hexagonal support with a first-order spatial color field. Natural scenes contain large regions that are locally simple in geometry and appearance; a Hexel can represent such a region with one large primitive, including smooth color variation within it, while smaller Hexels capture boundaries, curvature, and fine detail. This directly targets redundancy in splatting: rather than approximating structured regions with many overlapping, spatially constant primitives, Hexels increase what each primitive can represent. Why a hexagon? Regular hexagons tile a plane without gaps and cover more area for the same boundary length than triangles or squares (Hales, 2001). This makes them a natural shape for representing large, smooth regions with relatively few primitives. Combined with the local color field, each Hexel can model both the region itself and smooth appearance variation within it. In 3D, Hexels are rendered as oriented surface elements using perspective-correct ray-plane intersection and depth-ordered blending. The same primitive can be placed directly in the image plane for 2D image representation. Thus, Hexels provide a single compact primitive family for novel-view synthesis, surface reconstruction, and image representation, without texture maps, neural appearance fields, or decoding networks.

Our experiments separate compact fixed-population models from densified models that prioritize quality. At the V1 representation-memory budget, Hexels improve PSNR over 2DGS by 4.53–7.38 dB and over Triangle Splatting by 1.70–2.90 dB across the three benchmarks (Table 2). Under shared primitive-count ceilings, V2 improves dataset-mean PSNR over 2DGS by 2.09–4.74 dB across the three benchmarks, while storing half as many learned scalars per primitive. Densified V3 gains 0.89/0.88 dB over native 2DGS on DTU/Tanks and Temples; on Mip-NeRF360 it trades 0.34 dB for $2.06\times$ the rendering speed and 83.9% less deployment storage. These benefits are not universal: some textured and volumetric baselines achieve higher PSNR, and V3 densification can increase training VRAM above 2DGS. For all-view DTU reconstruction, fixed V3 reaches 0.978 mm Chamfer distance versus 1.375 mm for memory-controlled 2DGS and 0.794 mm for unconstrained 2DGS. Hexels also improve 2D image representation over Image-GS.

Contributions. **(1)** We introduce *Hexels*, compact, self-contained surface primitives for resource-constrained rendering and reconstruction, combining efficient hexagonal support with a first-order local color field without textures, feature grids, neural decoders, or baking. **(2)** Hexels achieve state-of-the-art efficiency: at equal primitive counts, they outperform 2DGS by 2.09–4.74 dB with $2\times$ less learned storage, up to $2.45\times$ lower training VRAM, and up to $3\times$ higher FPS; at matched model sizes, they improve PSNR by up to 7.38 dB with up to $6.96\times$ lower training VRAM and $2.45\times$ faster rendering. **(3)** The same primitive generalizes across novel-view synthesis, surface reconstruction, and 2D image representation, achieving strong quality while remaining substantially more efficient than existing representations.

2 Related Work

Novel view synthesis and 3D reconstruction have improved rapidly since NeRF (Mildenhall et al., 2020), which represents scenes as neural fields of density and view-dependent radiance. Feature grids and explicit encodings reduced their training and rendering cost (Müller et al., 2022; Fridovich-Keil et al., 2022), while 3DGS (Kerbl et al., 2023) replaced neural fields with explicit anisotropic Gaussian primitives, enabling real-time rasterization. 2DGS (Huang et al., 2024) further replaced volumetric Gaussians with planar surfels, substantially improving surface reconstruction. These advances made rendering faster and geometry more accurate, but efficient representation requires more: high quality together with low model storage, low training VRAM, few primitives, and high FPS. In our all-view DTU evaluation, unconstrained 2DGS has the lowest surface error, making it an important geometry baseline. Appendix J discusses these trade-offs.

Subsequent work improves different parts of this trade-off, but usually with a catch. Triangle Splatting (Held et al., 2025b), Deformable Beta Splatting (Liu et al., 2025), and 3D Convex Splatting (**?**) redesign the primitive or kernel to improve expressiveness or rendering speed, but at cost of higher memory and VRAM usage. Hybrid methods such as BBSplat (Svitov et al., 2025), NeST (Zhang et al., 2025a), and Nexels (Rong et al., 2025) reduce geometric redundancy by separating geometry from appearance through textures or neural fields; this shifts capacity into additional memory, training state, and computation. Other methods retrofit existing representations through pruning, parameter sharing, quantization, distillation, or baking (Kheradmand et al., 2025; Lu et al., 2024; Fan et al., 2024; Chen et al., 2023) it still largely struggled to address the efficiency problem on all of its axes. Hexels provides a solution to the efficiency problem by redesigning the primitive itself to exploit local structure and geometry in appearance. The resulting primitive is compact in the form in which it is learned and rendered, simultaneously targeting storage, training VRAM, primitive count, FPS, and reconstruction quality. Further, unlike previous works, Hexels are general surface primitives: the same underlying design applies directly to oriented 3D surfaces and to 2D images without major changes to the representation or optimization.

3 HEXELS: HEXAGONAL SURFACE ELEMENTS

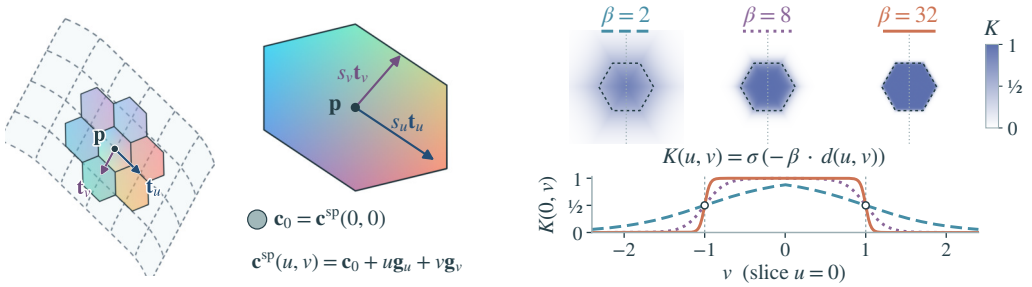


Figure 2: A Hexel is a planar patch with a soft hexagonal boundary; together, Hexels approximate a 3D surface (left). Its center $\mathbf{p}$, orthogonal unit tangents $\mathbf{t}_u, \mathbf{t}_v$, and scales s_u, s_v determine its position, orientation, and size (center). Color varies linearly from $\mathbf{c}_0$ at the center, with RGB gradients $\mathbf{g}_u, \mathbf{g}_v$ along local coordinates u, v. K (spatial weight) controls the patch's influence: increasing sharpness β sharpens the boundary (right).

We introduce *Hexels*, planar hexagonal surface elements designed to reduce overlap and overdraw, avoid the long tails of Gaussian footprints, and represent more appearance within each primitive. Each Hexel combines a soft hexagonal support with a first-order spatial color field, allowing a single primitive to cover a larger coherent surface region while modeling smooth appearance variation within it. In 3D, a Hexel is an oriented planar patch with a local 2D coordinate system. Following 2DGS (Huang et al., 2024), we evaluate each primitive at the perspective-correct ray-plane intersection and blend the resulting contributions in depth order.

Modeling A Hexel (Figure 2) is centered at $\mathbf{p}_i \in \mathbb{R}^3$ with two orthogonal unit tangent directions $\mathbf{t}_{u,i}$ and $\mathbf{t}_{v,i}$ and positive tangent scales $s_{u,i}$ and $s_{v,i}$. A local point (u, v) maps to world space as

$$P_i(u, v) = \mathbf{p}_i + s_{u,i}\mathbf{t}_{u,i}u + s_{v,i}\mathbf{t}_{v,i}v. \tag{1}$$

The tangent directions are obtained from a normalized quaternion, with $\mathbf{n}_i = \mathbf{t}_{u,i} \times \mathbf{t}_{v,i}$ defining the surface normal. This maps the hexagon to an oriented, anisotropically scaled planar patch in 3D.

Hexagonal support. In the local plane, we define

$$d(u, v) = \max\left\{|v|, \left|\tfrac{\sqrt{3}}{2}u + \tfrac{1}{2}v\right|, \left|\tfrac{\sqrt{3}}{2}u - \tfrac{1}{2}v\right|\right\} - 1. \tag{2}$$

The contour $d(u, v) = 0$ defines the canonical regular hexagon, with $d < 0$ inside and $d > 0$ outside. We convert this into a soft support weight

$$K_i(u, v) = \sigma(-\beta_i d(u, v)), \qquad \sigma(a) = \frac{1}{1 + \exp(-a)}, \tag{3}$$

where $\beta_i > 0$ controls edge sharpness. Increasing β_i produces a sharper transition around the hexagon while preserving the $K_i = 1/2$ contour. The scales in Eq. (1) control the spatial extent of

the support, while the rasterizer truncates negligible sigmoid tails below its contribution threshold (See Appendix A for details).

We use a hexagon because it provides an efficient, nearly isotropic way to cover a plane. Regular hexagons tile the plane without gaps and, at equal area, require less perimeter than regular triangles or squares, consistent with the honeycomb theorem (Hales, 2001). More directly, under the local first-order approximation model of Appendix I, a suitably oriented and anisotropically scaled hexagon can cover about 30% more area than the corresponding square and twice the area of the corresponding triangle for the same bounded color error. (See Proposition I.2)

Spatial appearance. We model appearance within each Hexel using a first-order color field:

$$\mathbf{c}_i^{\mathrm{sp}}(u, v) = \mathbf{c}_{0,i} + u\mathbf{g}_{u,i} + v\mathbf{g}_{v,i}, \tag{4}$$

where $\mathbf{c}_{0,i}$ is the base RGB color and $\mathbf{g}_{u,i}, \mathbf{g}_{v,i}$ are RGB gradients along the two local coordinates. This allows one Hexel to represent smooth appearance variation across its surface rather than assigning a single spatial color to the primitive. Since the field is defined in local coordinates, increasing a tangent scale spreads the same color variation over a larger physical region. We refer to this view-independent model as *V1*, with 20 learned parameters per Hexel, compared with 58 for the evaluated 2DGS primitive. Our *V2* variant adds compact view dependence through a linear residual $\mathbf{r}_i(\mathbf{d}_i) = A_i\mathbf{d}_i$, where $\mathbf{d}_i$ is the normalized direction from the camera center to the Hexel center, for 29 parameters in total. We also consider an optional *V3* variant that replaces the linear residual with degree-three spherical harmonics. See Appendix A.1 for the complete parameterization details.

Splatting. Following 2DGS (Huang et al., 2024), we recover local coordinates (u_i, v_i) at each pixel using perspective-correct ray–plane intersection and composite primitives front-to-back in depth order. A Hexel contributes $\alpha_i = \min\{o_i K_i(u_i, v_i), 0.99\}$, with color evaluated from its local appearance field at (u_i, v_i). Thus, relative to 2DGS, the rasterization and compositing remain unchanged; we replace the Gaussian weight with the Hexel support and evaluate spatially varying color within each primitive. Implementation details are given in Appendix B.

Training. We optimize all Hexel parameters directly from calibrated training images using the standard photometric objective $\mathcal{L} = (1 - \lambda)\|\hat{I} - I\|_1 + \lambda(1 - \mathrm{SSIM}(\hat{I}, I))$. Hexels are initialized from either sparse COLMAP points or, where stated, a denser MVS reconstruction obtained from the training views. Unless otherwise specified, our reported models use a fixed primitive population.

Densification and Pruning. For unconstrained training, we additionally consider standard densification and pruning, following 3DGS and 2DGS (Kerbl et al., 2023; Huang et al., 2024). Primitives with large position gradients are cloned or split, with new Hexels initialized near the parent in its tangent plane. We conservatively prune mature primitives that make negligible contributions across the training views. More details are given in Appendix C. For inference, we cache camera-independent quantities and use an RGB-only Hexel rasterization path that removes unused intermediate state; the support, depth ordering, and compositing are unchanged. Full implementation details and verification are given in Appendix B.

Table 1: **2D image representation.** Image-GS dataset (45 images); constant vs. linear RGB (Eq. (4)).

Method	PSNR↑	SSIM↑	LPIPS↓	Iter.↓	Train (s)↓	FPS↑
Image-GS	33.15	.9141	.1372	10,000	238.40	260.44
Hexels (constant)	33.38	.9158	.1184	5,100	**121.61**	247.62
Hexels	**34.84**	**.9297**	**.1012**	**5,100**	176.94	**280.56**

3.1 Representing 2D Images as Hexels

Unlike most surface primitives, Hexels are general and can be used directly for both 3D scenes and 2D images. For efficient image representation, a 2D Hexel is defined by its image-plane center $\mathbf{m}m_i$, rotation θ_i, two scales $s_{x,i}, s_{y,i}$, and support sharpness β_i. The same hexagonal support from Eq. (2) is evaluated directly at each pixel, without projection, ray–plane intersection, opacity, or depth ordering. For a pixel $\mathbf{x}$, we compute

$$\mathbf{u}_i = S_i^{-1} R(\theta_i)^\top (\mathbf{m}m_i - \mathbf{x}), \qquad w_i(\mathbf{x}) = \sigma(-\beta_i d(\mathbf{u}_i)) \,,$$

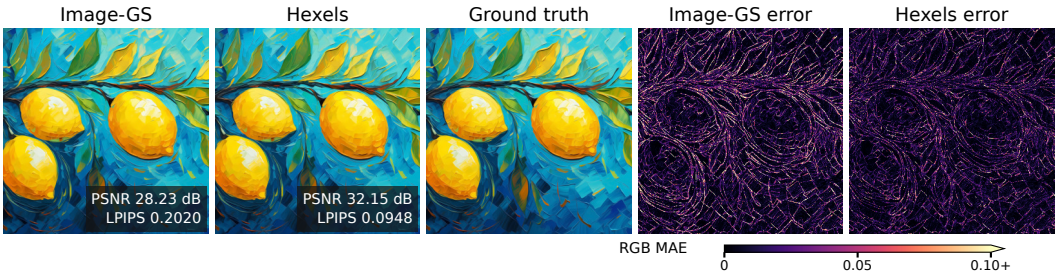


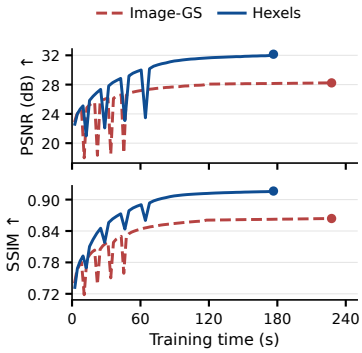
Figure 4: **2D reconstruction of `painting-8`.** At 2048×2048 and 10K primitives, Hexels improves PSNR and LPIPS.

retain the K largest support weights as in Image-GS (Zhang et al., 2025b), and normalize them to obtain

$$\mathbf{c}(\mathbf{x}) = \frac{\sum_{i \in S_K(\mathbf{x})} w_i(\mathbf{x}) \mathbf{c}_i(\mathbf{x})}{\sum_{i \in S_K(\mathbf{x})} w_i(\mathbf{x}) + \epsilon}.$$

We use the V1 first-order spatial color field as the primary 2D appearance model, expressed in image coordinates; constant-color Hexels, with a fixed shared β, match Image-GS's eight stored scalars per primitive. Initialization, loss, progressive allocation and 16-bit quantization-aware training follow Image-GS; Appendix D lists the differences. The 2D setting also provides a controlled ablation of the Hexel design: reconstruction error and primitive overlap can be measured in a straightforward manner. On the 45-image Image-GS benchmark, Hexels (V1) outperform Image-GS across all reported quality metrics, while using 5.1K rather than 10K optimization iterations, reducing training time from 238 to 177 seconds, and increasing rendering speed from 260 to 281 FPS (Table 1). Even with constant color, Hexels improve PSNR, SSIM, and LPIPS over Image-GS at matched storage, isolating the benefit of the Hexel support itself. Figure 3 shows a provisional convergence example from a separate planar ablation. Matched benchmark curves remain pending; reconstruction details are reserved in Figure 4.

Figure 3: **2D convergence on `painting-8`.** PSNR and SSIM versus fitting time on an A5000.



4 Experiments

We evaluate novel-view synthesis under matched model memory, matched primitive budgets and native baseline settings.

Implementation. Hexels start from sparse COLMAP points and train for 30K iterations. Fixed configurations retain the initial population; densified configurations use mixed clone/split growth and contribution-based pruning, with a 2M limit. We compare V3 densified and compact V2 fixed against native baselines; Appendix K.2 reports all six configurations. Training and evaluation details are in Appendix E.1.

Datasets, baselines, and metrics. We evaluate nine Mip-NeRF360 scenes (Barron et al., 2022), fifteen DTU scans (Jensen et al., 2014) and ten Tanks and Temples scenes (Knapitsch et al., 2017), using common held-out views and resolutions. We report PSNR, SSIM, LPIPS, training time, peak training VRAM, rendering FPS and deployment size, averaged over scenes. Our baselines are 3DGS (Kerbl et al., 2023), 3D Convex Splatting (**?**), BBSplat (Svitov et al., 2025), Nexels (Rong et al., 2025), 2DGS (Huang et al., 2024), Triangle Splatting (Held et al., 2025b) and MeshSplatting (Held et al., 2025a). Tables separate volumetric primitives, textured/neural surfels and surface primitives with local appearance. The last group provides the closest comparison to Hexels, without separate texture maps or neural appearance fields.

Novel View Synthesis. Matched memory. Table 2 compares methods at the V1 model-memory budget. V1 improves PSNR over 2DGS by 4.53–7.38 dB and over Triangle Splatting by 1.70–

2.90 dB across the three benchmarks. Its mean peak training VRAM is 0.55–0.70 GiB, with mean rendering rates of 382–882 FPS. Fig 5 illustrates its reconstruction of fine structures within these small memory budgets. Fig 7 compares densified version of V3 with all other baselines in an unconstrained setting with no limit on memory or primitives.

Method	Tanks and Temples (10)						Mip-NeRF 360 (9)						DTU (15)					
	Train min ↓	VRAM GiB ↓	FPS ↑	PSNR dB ↑	SSIM ↑	LPIPS VGG ↓	Train min ↓	VRAM GiB ↓	FPS ↑	PSNR dB ↑	SSIM ↑	LPIPS VGG ↓	Train min ↓	VRAM GiB ↓	FPS ↑	PSNR dB ↑	SSIM ↑	LPIPS VGG ↓
3DGS	5.6	0.80	807.5	20.39	0.683	0.477	8.2	1.42	852.6	20.46	0.528	0.614	4.5	0.43	1035.0	27.06	0.840	0.395
3D Convex	20.3	0.80	203.5	19.71	0.680	0.481	35.5	1.24	146.1	20.82	0.570	0.564	12.0	0.58	316.6	26.74	0.847	0.382
BBSplat	21.1	1.27	132.0	19.00	0.624	0.548	37.0	3.58	87.7	21.34	0.535	0.603	14.4	0.60	179.9	26.90	0.833	0.380
Nexels	72.2	2.82	99.6	21.32	0.695	0.424	82.7	5.96	49.9	22.49	0.590	0.493	29.0	2.21	155.5	28.23	0.876	0.303
2DGS	10.6	1.37	284.5	18.52	0.652	0.508	20.0	3.24	201.2	17.82	0.494	0.658	8.4	0.68	478.0	21.56	0.757	0.476
Triangle Spl.	9.4	0.74	495.2	21.35	0.731	0.409	15.8	1.12	312.7	22.36	0.588	0.538	7.4	0.60	517.5	26.04	0.835	0.381
MeshSplatting	17.2	2.15	119.6	16.29	0.565	0.567	33.1	4.49	60.8	19.31	0.475	0.634	13.3	1.85	130.7	21.14	0.666	0.505
Hexels V1	13.3	0.55	444.1	23.05	0.811	0.295	24.2	0.70	381.9	24.82	0.705	0.410	8.4	0.55	881.9	28.94	0.890	0.303

Table 2: **Matched model memory.** On matched model memory, Hexels outperform all baselines in all 3 benchmarks in PSNR, SSIM, and LPIPS, while using the lowest training VRAM, robust FPS and training time.

Matched primitive budget. Table 3 uses each scene's SfM point count as the primitive budget. V3 fixed exceeds Triangle by 0.65 dB on Mip-NeRF360, 2.80 dB on DTU and 0.86 dB on T&T. BBSplat achieves higher PSNR by 0.25–1.12 dB, using additional texture storage beyond the primitive attributes. Figure 6 shows the detail retained by V3 under this geometric budget.

Method	Tanks and Temples (10)							Mip-NeRF 360 (9)							DTU (15)						
	Train min ↓	VRAM GiB ↓	FPS ↑	PSNR dB ↑	SSIM ↑	LPIPS VGG ↓	Deploy. MB ↓	Train min ↓	VRAM GiB ↓	FPS ↑	PSNR dB ↑	SSIM ↑	LPIPS VGG ↓	Deploy. MB ↓	Train min ↓	VRAM GiB ↓	FPS ↑	PSNR dB ↑	SSIM ↑	LPIPS VGG ↓	Deploy. MB ↓
3DGS	7.5	1.28	581.8	22.95	0.773	0.361	45.15	9.8	1.49	607.2	23.38	0.599	0.527	28.45	5.1	0.55	819.6	29.29	0.893	0.324	8.43
3D Convex	29.2	1.29	114.3	22.20	0.758	0.376	46.90	45.7	1.90	83.0	23.88	0.652	0.466	30.35	15.0	0.71	213.3	28.24	0.887	0.305	8.12
BBSplat	64.7	8.85	49.8	24.41	0.842	0.224	146.64	77.7	14.45	34.1	26.50	0.767	0.296	96.28	22.0	1.79	125.4	30.19	0.913	0.228	25.91
Nexels	76.9	4.14	77.8	23.48	0.803	0.272	183.02	91.2	6.90	44.2	25.49	0.736	0.304	164.99	31.5	3.08	142.9	29.00	0.891	0.209	143.37
2DGS	12.4	2.24	193.2	21.40	0.741	0.396	40.01	22.0	5.15	161.9	21.65	0.573	0.566	27.98	9.1	1.01	364.8	24.83	0.832	0.399	8.29
Triangle Spl.	11.6	1.00	407.2	22.90	0.803	0.310	47.63	18.4	1.32	278.7	24.73	0.677	0.434	30.21	8.4	0.64	488.1	27.14	0.860	0.331	8.39
MeshSplatting	18.0	2.50	116.8	16.42	0.599	0.548	28.33	31.5	4.54	60.9	18.93	0.498	0.620	19.90	12.5	1.86	135.3	22.96	0.729	0.471	4.58
Hexels V1	13.3	0.55	444.1	23.05	0.811	0.295	12.27	24.2	0.70	381.9	24.82	0.705	0.410	7.73	8.4	0.55	881.9	28.94	0.890	0.303	2.30
Hexels V2	14.4	0.60	393.2	23.49	0.819	0.287	17.77	24.5	0.74	376.8	25.06	0.711	0.405	11.20	8.5	0.56	895.4	29.57	0.895	0.298	3.33
Hexels V3	14.6	0.71	371.1	23.76	0.824	0.282	15.94	24.3	0.81	366.3	25.38	0.716	0.402	10.09	8.5	0.58	792.5	29.94	0.899	0.294	3.11

Table 3: **Matched primitive budgets.** At the same primitive numbers, Hexels, outperform all other baselines in PSNR, SSIM, and LPLPs while at the same time using the least amount of VRAM memory and having the highest FPS among the surface primitives

Native baselines. Across Mip-NeRF360, V3 densified is 0.34 dB below native 2DGS, with 2.06× its mean FPS and 79.53 versus 494.72 MB deployment size (Table 4). On DTU and T&T, it improves PSNR by 0.89 and 0.88 dB, with 2.29× and 1.38× the FPS. BBSplat leads V3 by 0.51 dB on DTU, while 3D Convex leads by 0.28 dB on T&T. V3 also uses more training VRAM than 2DGS on T&T (12.06 versus 4.17 GiB); V2 fixed provides a lower-cost alternative. Figure 7 compares the recovered detail on Courthouse and Lighthouse.

Efficient Surface Reconstruction. We further evaluate surface reconstruction under memory constraints on all 15 DTU scans. All methods receive the same per-scene learned-state budget, with matched MVS initialization where applicable, and are evaluated using the same mesh-extraction protocol and Chamfer distance (CD). Hexels achieve the lowest surface error at this budget, reaching 1.674 mm CD compared with 2.083 mm for Triangle Splatting, 2.757 mm for 3DGS, and 2.992 mm for 2DGS with matched MVS initialization. Thus, Hexels retain substantially better surface geometry for the same model memory.

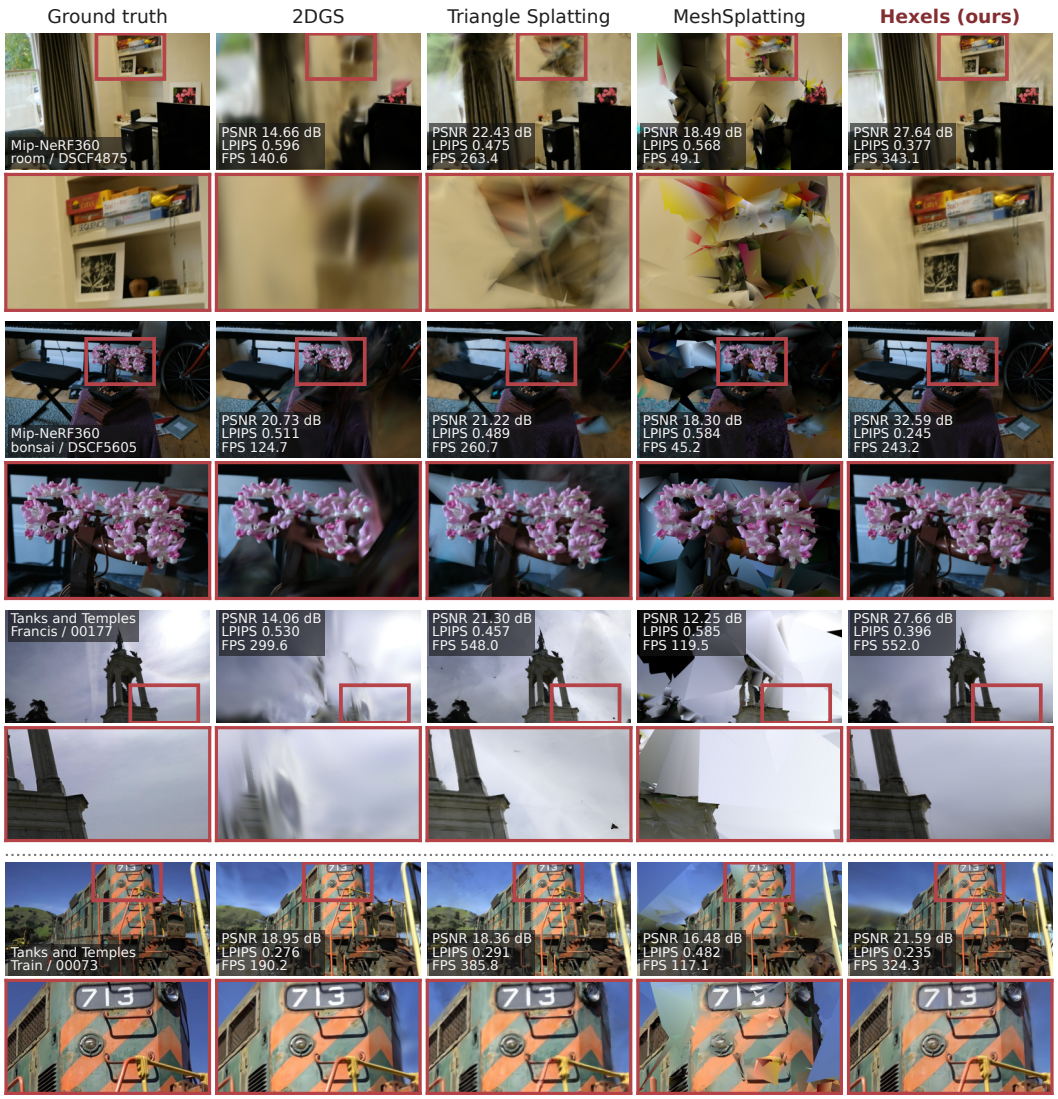


Figure 5: **Qualitative comparisons under memory and primitive budgets.** Columns compare ground truth, 2DGS, Triangle Splatting, MeshSplatting and Hexels. Above the dotted line, fixed Hexels V1 sets the memory budgets for room and bonsai (Mip-NeRF360), and Francis (Tanks and Temples). Below it, fixed Hexels V3 uses a shared 184,179-primitive budget on Train (Tanks and Temples). The common zooms highlight clearer shelving, blossoms, statue edges, and locomotive markings.

4.1 Applications

Native surface extraction. Each Hexel directly defines six 3D vertices, allowing its learned surface to be exported as a polygonal cell without Marching Cubes, voxelization, or depth fusion. The conversion is differentiable and constant-time per primitive. Moreover, the low overdraw of Hexels produces relatively little redundant surface overlap, making this direct cell representation useful for object-centric tasks such as image-to-3D generation, 3D understanding, and segmentation. Figure H.1 shows examples, and see subsection H.1 for details.

Table 5: **DTU surface reconstruction.** Means over 15 scans. Hexels reduces Chamfer distance (CD) versus both matched 2DGS controls, with lower peak GPU memory usage.

Method	CD (mm)↓	VRAM (GiB)↓
2DGS unconstrained	0.794	1.93
2DGS matched mem.	1.375	1.02
2DGS matched prim.	1.449	0.98
Triangle Splatting	1.124	2.29
MeshSplatting	1.630	2.65
Hexels (V3)	0.978	0.61

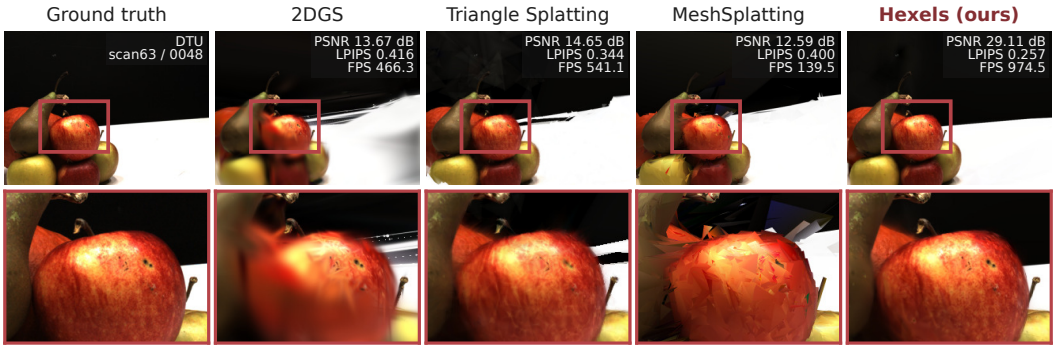


Figure 6: **Matched-primitive qualitative comparison.** Fixed-population Hexels V3, 2DGS, Triangle Splatting and MeshSplatting reconstruct DTU scan63 with a 17,763-primitive budget. Hexels better represent the fruit texture in the enlarged red-box region, showing better quality and expressiveness per primitive.

Method	Tanks and Temples (10)								Mip-NeRF 360 (9)								DTU (15)							
	Train min ↓	VRAM GiB ↓	FPS ↑	PSNR dB ↑	SSIM ↑	LPIPS VGG ↓	Deploy. MB ↓	Prims. K	Train min ↓	VRAM GiB ↓	FPS ↑	PSNR dB ↑	SSIM ↑	LPIPS VGG ↓	Deploy. MB ↓	Prims. K	Train min ↓	VRAM GiB ↓	FPS ↑	PSNR dB ↑	SSIM ↑	LPIPS VGG ↓	Deploy. MB ↓	Prims. K
3DGS	18.1	3.70	157.5	24.35	0.846	0.237	294.75	1188.5	32.8	7.33	94.7	27.55	0.817	0.259	680.48	2743.8	10.5	1.57	301.8	28.91	0.910	0.249	97.27	392.2
3D Convex*	70.2	8.64	44.4	24.80	0.841	0.224	488.65	1586.5	97.8	12.60	35.1	27.24	0.802	0.245	792.46	2572.9	21.8	1.20	136.3	28.49	0.898	0.254	22.21	72.1
BBSplat	100.5	23.51	27.4	24.52	0.844	0.218	229.48	300.0	156.7	21.77	14.0	26.75	0.783	0.274	184.02	237.8	34.3	13.26	74.5	30.26	0.914	0.221	41.99	60.0
Nexels	94.5	6.77	45.8	24.14	0.829	0.234	241.35	399.6	116.3	10.40	25.3	27.35	0.802	0.235	241.40	399.8	43.2	3.79	96.5	29.03	0.905	0.186	241.40	399.8
2DGS	19.2	4.17	103.6	23.64	0.829	0.270	177.56	727.7	41.2	10.83	48.9	26.82	0.798	0.301	494.72	2027.5	12.7	2.04	213.2	28.86	0.907	0.265	56.20	230.3
Triangle Spl.*	38.6	11.77	121.4	23.15	0.838	0.227	1039.86	2821.0	42.8	10.81	85.8	27.02	0.808	0.233	1093.15	3595.3	16.5	1.85	278.7	27.05	0.873	0.257	127.76	362.6
MeshSplatting	49.3	10.38	43.3	20.73	0.754	0.358	498.08	3694.1	73.2	13.48	23.8	24.74	0.731	0.366	707.77	5379.5	17.0	2.59	94.7	25.85	0.810	0.408	34.71	285.1
Hexels V3	24.3	12.06	142.7	24.52	0.839	0.249	64.30	739.1	37.5	13.95	100.5	26.48	0.780	0.309	79.53	917.3	10.6	1.21	487.3	29.75	0.902	0.268	8.85	100.4
Hexels V2	14.4	0.60	393.2	23.49	0.819	0.287	17.77	182.1	24.5	0.74	376.8	25.06	0.711	0.405	11.20	114.7	8.5	0.56	895.4	29.57	0.895	0.298	3.33	34.0

Table 4: **Native novel-view synthesis.** Hexels V3 and V2 versus native baseline settings. Deploy. is deployment storage or memory(Mem); Prims. is the primitive count in thousands. Mip-NeRF 360 combines five outdoor and four indoor scenes using scene-count-weighted means for every metric. Cell colors rank methods within each block.

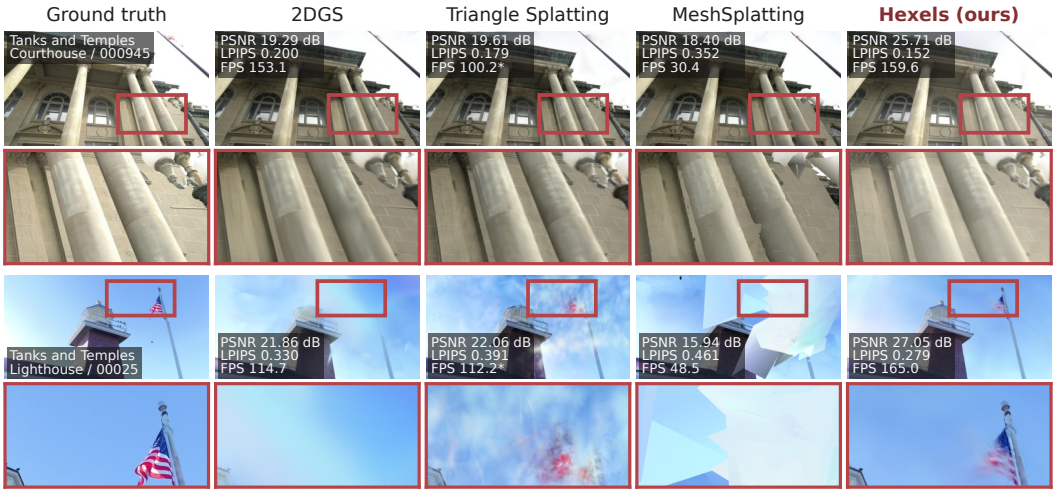


Figure 7: **Native qualitative comparison on Tanks and Temples.** Hexels (V3) is compared with native 2DGS, Triangle Splatting and MeshSplatting on Courthouse and Lighthouse. Zoomed patches highlight clearer and finer details and flag boundaries for Hexels, lower VRAM and memory usage. Overlays report full-image PSNR/LPIPS and scene rendering speed in frames/s.

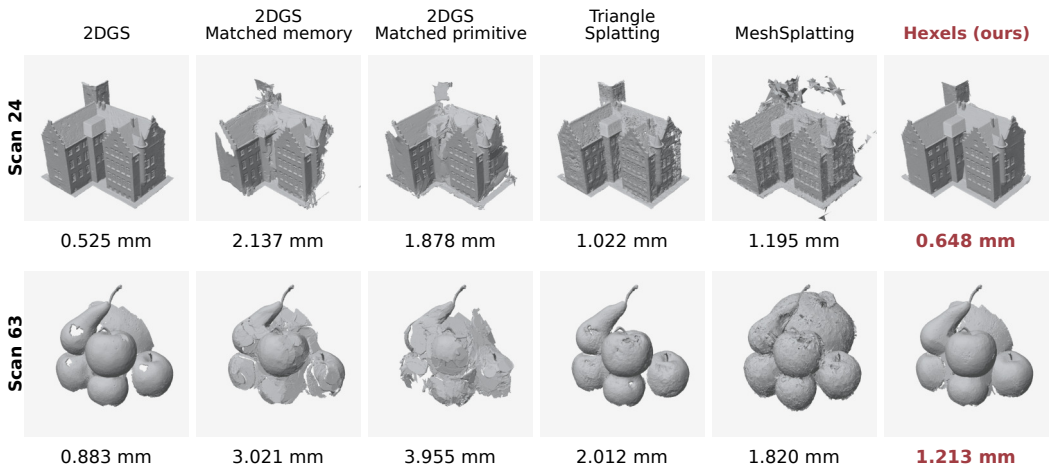


Figure 8: Matched memory surface reconstruction. We find that Hexels outperform all baselines in matched memory, and matched primitive (2DGS) surface reconstruction.

Resource-constrained reconstruction. The low model memory, low training VRAM, fast convergence, and high rendering rates of Hexels make them particularly suitable for 3D reconstruction and representation on edge and resource-constrained hardware. We will release an implementation for local training and rendering in this setting, with mobile deployment as a trivial application.

5 Conclusion

We introduced Hexels, a general surface primitive that exploits the local structure of the data it represents: hexagonal support efficiently covers coherent surface regions, while a first-order spatial color field models appearance variation within each primitive. Across our three 3D benchmarks, Hexels establish a new quality–efficiency frontier among the evaluated native surface-aligned representations, improving PSNR by up to 7.38 dB at matched model memory, using up to 6.96× less training VRAM and 2× less learned state at matched primitive count, while rendering up to 2.45× faster than 2DGS. The same primitive also performs strongly for memory-constrained surface reconstruction and directly generalizes to efficient 2D image representation. Importantly, these gains are obtained in full precision, without post-hoc pruning, quantization, compression, texture baking, or learned decoders. Hexels therefore move efficiency into the primitive itself, opening a path toward high-quality novel-view synthesis and explicit surface reconstruction on edge and resource-constrained hardware.

Limitations and Future Work. A first-order color model cannot represent arbitrary texture within a single primitive. High-frequency or uncorrelated appearance requires smaller primitives or a richer color model, reducing the expected benefit of exploiting local smoothness. Sparse initialization can also leave parts of a scene poorly covered, and the independent surface cells do not form a watertight mesh. Sharper footprints do not by themselves prevent sampling artifacts, especially at small projected sizes or grazing views. Hexels does not outperform every baseline on every metric; the appendix provides the full comparisons. The proposed growth and pruning extension may address limited coverage, but its effect on quality and resource use remains to be evaluated. Our current primitives are planar. A natural extension is to developable Hexels: curved patches with zero Gaussian curvature that remain locally isometric to the planar canonical hexagon. Such primitives could bend along one principal direction, for example as cylindrical or conical patches, while preserving the simple 2D footprint and affine appearance parameterization. This may allow individual primitives to cover larger regions of locally near-developable geometry without introducing additional overlap.

AI Statement : Large Language Models (LLMs) were utilized strictly for grammatical corrections and language refinement during the drafting of this manuscript. They did not contribute to the conceptualization or ideation of the research.

Ethics Statement: Hexels represents scenes and images for rendering and reconstruction. Such models can reproduce identifiable places or personal content; their use should respect privacy, consent and dataset licenses, and synthetic views should not be presented as unaltered photographs.

Reproducibility Statement: The appendix specifies scene lists, camera splits, initialization, optimization, resource measurement and surface evaluation. We report all six Hexels configurations, deployment compression results and per-scene results for densified V3. The NVS and surface benchmarks use separate checkpoints and evaluation protocols.

REFERENCES

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.

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

Albert Cohen and Jean-Marie Mirebeau. Adaptive and anisotropic piecewise polynomial approximation. 2011. URL https://arxiv.org/abs/1101.1587.

Zhiwen Fan, Kevin Wang, Kairun Wen, Zehao Zhu, Dejia Xu, and Zhangyang Wang. LightGaussian: Unbounded 3D gaussian compression with 15x reduction and 200+ FPS. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.nips.cc/paper_files/paper/2024/hash/fd881d3b625437354d4421818f81058f-Abstract-Conference.html.

Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *CVPR*, 2022.

Antoine Guédon and Vincent Lepetit. Sugar: Surface-aligned gaussian splatting for efficient 3d mesh reconstruction and high-quality mesh rendering. *arXiv preprint arXiv:2311.12775*, 2023.

Antoine Guédon, Diego Gomez, Nissim Maruani, Bingchen Gong, George Drettakis, and Maks Ovsjanikov. MILo: Mesh-in-the-loop gaussian splatting for detailed and efficient surface reconstruction. *arXiv preprint arXiv:2506.24096*, 2025. URL https://arxiv.org/abs/2506.24096.

Thomas C. Hales. The honeycomb conjecture. *Discrete & Computational Geometry*, 25:1–22, 2001. doi: 10.1007/s004540010071.

Abdullah Hamdi, Luke Melas-Kyriazi, Jinjie Mai, Guocheng Qian, Ruoshi Liu, Carl Vondrick, Bernard Ghanem, and Andrea Vedaldi. GES: Generalized exponential splatting for efficient radiance field rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 19812–19822, 2024. doi: 10.1109/CVPR52733.2024.01873. URL https://arxiv.org/abs/2402.10128.

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. 2025a. URL https://arxiv.org/abs/2512.06818.

Jan Held, Renaud Vandeghen, Adrien Deliege, Abdullah Hamdi, Anthony Cioppa, Silvio Giancola, Andrea Vedaldi, Bernard Ghanem, Andrea Tagliasacchi, and Marc Van Droogenbroeck. Triangle splatting for real-time radiance field rendering. *arXiv*, 2025b.

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*, 2024. doi: 10.1145/3641519.3657428.

Laki Iinbor, Zhiyang Dou, and Wojciech Matusik. Soft anisotropic diagrams for differentiable image representation. *ACM Transactions on Graphics*, 2026. SIGGRAPH 2026.

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.

Neel Kelkar, Simon Niedermayr, Kaloian Petkov, Klaus Engel, and Rüdiger Westermann. Bake it till you make it: Ultrafast spatial texture-atlas splatting, 2026. URL https://arxiv.org/abs/2607.13808.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), July 2023. URL https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/.

Shakiba Kheradmand, Daniel Rebain, Gopal Sharma, Weiwei Sun, Jeff Tseng, Hossam Isack, Abhishek Kar, Andrea Tagliasacchi, and Kwang Moo Yi. 3d gaussian splatting as markov chain monte carlo. 2025. URL https://arxiv.org/abs/2404.09591.

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.

Rong Liu, Dylan Sun, Meida Chen, Yue Wang, and Andrew Feng. Deformable beta splatting. *arXiv preprint arXiv:2501.18630*, 2025. URL https://arxiv.org/abs/2501.18630.

Tao Lu, Mulin Yu, Linning Xu, Yuanbo Xiangli, Limin Wang, Dahua Lin, and Bo Dai. Scaffold-GS: Structured 3D gaussians for view-adaptive rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 20654–20664, 2024. URL https://city-super.github.io/scaffold-gs/.

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

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

Simon Niedermayr, Josef Stumpfegger, and Rüdiger Westermann. Compressed 3d gaussian splatting for accelerated novel view synthesis, 2024. URL https://arxiv.org/abs/2401.02436.

Christian Reiser, Richard Szeliski, Dor Verbin, Pratul P. Srinivasan, Ben Mildenhall, Andreas Geiger, Jonathan T. Barron, and Peter Hedman. Merf: Memory-efficient radiance fields for real-time view synthesis in unbounded scenes. 2023. URL https://arxiv.org/abs/2302.12249.

Victor Rong, Jan Held, Victor Chu, Daniel Rebain, Marc Van Droogenbroeck, Kiriakos N. Kutulakos, Andrea Tagliasacchi, and David B. Lindell. Nexels: Neurally-textured surfels for real-time novel view synthesis with sparse geometries. *arXiv preprint arXiv:2512.13796*, 2025. URL https://arxiv.org/abs/2512.13796.

David Svitov, Pietro Morerio, Lourdes Agapito, and Alessio Del Bue. BillBoard Splatting (BBSplat): Learnable textured primitives for novel view synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2025. URL https://arxiv.org/abs/2411.08508.

Sebastian Weiss and Derek Bradley. Gaussian billboards: Expressive 2D gaussian splatting with textures. *arXiv preprint arXiv:2412.12734*, 2024. URL https://arxiv.org/abs/2412.12734.

Xin Zhang, Anpei Chen, Jincheng Xiong, Pinxuan Dai, Yujun Shen, and Weiwei Xu. Neural shell texture splatting: More details and fewer primitives. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2025a. URL https://arxiv.org/abs/2507.20200.

Xinjie Zhang, Xingtong Ge, Tongda Xu, Dailan He, Yan Wang, Hongwei Qin, Guo Lu, Jing Geng, and Jun Zhang. GaussianImage: 1000 FPS image representation and compression by 2D gaussian splatting. In *European Conference on Computer Vision*, pp. 327–345. Springer, 2024.

Yunxiang Zhang, Bingxuan Li, Alexandr Kuznetsov, Akshay Jindal, Stavros Diolatzis, Kenneth Chen, Anton Sochenov, Anton Kaplanyan, and Qi Sun. Image-gs: Content-adaptive image representation via 2d gaussians. In *Proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers*, pp. 1–11. ACM, 2025b. doi: 10.1145/3721238.3730596. URL http://dx.doi.org/10.1145/3721238.3730596.

Ziyu Zhang, Binbin Huang, Hanqing Jiang, Liyang Zhou, Xiaojun Xiang, and Shunhan Shen. Quadratic gaussian splatting: High quality surface reconstruction with second-order geometric primitives. 2025c. URL https://arxiv.org/abs/2411.16392.

APPENDIX CONTENTS

A Hexel Parameterization **15**
A.1 Appearance 15
A.2 Derivatives 15

B Rasterization **16**

C Densification and Pruning **17**

D 2D Image Representation **18**

E Implementation Details **19**
E.1 Datasets and Metrics 19
E.2 Initialization and Training 19
E.3 Baseline Settings 19
E.4 Timing and Memory 20

F Spatial Appearance Ablations **20**

G Surface Reconstruction Ablations **20**

H Applications **22**
H.1 Direct Surface-Cell Extraction 22
H.2 Deployment on Memory-Limited Devices 23

I Why Hexagons and Spatial Color? **23**
I.1 Linear Color 24
I.2 Patch Shape 25
I.3 3D Alpha Compositing 26

J Extended Related Work **26**

K Full NVS Results **29**
K.1 Matched Model Memory 29
K.2 Fixed and Densified Hexels 32
K.3 Deployment Compression 33

L Per-Scene NVS Results **34**

M Additional Qualitative Comparisons **36**

A HEXEL PARAMETERIZATION

Stored parameters. A 3D Hexel stores its center $\mathbf{p}_i \in \mathbb{R}^3$, two positive tangent scales $s_{u,i}$ and $s_{v,i}$, a quaternion defining its orientation, opacity o_i, and support sharpness β_i. The quaternion is normalized before constructing the tangent frame, and its first two axes define the orthogonal tangent directions $\mathbf{t}_{u,i}$ and $\mathbf{t}_{v,i}$. Thus, the geometric and support parameters require 11 learned scalars: three for position, two for scale, four for orientation, one for opacity, and one for support sharpness.

The V1 appearance model adds a base RGB value and two RGB spatial gradients, for nine additional scalars. A V1 Hexel therefore contains 20 learned scalars in total. V2 adds nine directional coefficients, while V3 adds 45 spherical-harmonic coefficients, giving 29 and 65 learned scalars, respectively.

A.1 APPEARANCE

The spatial appearance field used by all three variants is

$$\mathbf{c}_i^{\mathrm{sp}}(u, v) = \mathbf{c}_{0,i} + u\,\mathbf{g}_{u,i} + v\,\mathbf{g}_{v,i}, \tag{5}$$

where $\mathbf{c}_{0,i} \in \mathbb{R}^3$ is the base color and $\mathbf{g}_{u,i}, \mathbf{g}_{v,i} \in \mathbb{R}^3$ are RGB gradients along the two local Hexel coordinates. V1 uses this spatial field directly. V2 and V3 add a view-dependent RGB residual:

$$\mathbf{c}_i(u, v, \mathbf{d}_i) = \mathbf{c}_i^{\mathrm{sp}}(u, v) + \mathbf{r}_i(\mathbf{d}_i), \tag{6}$$

where $\mathbf{d}_i$ is the normalized world-space direction from the camera center to the Hexel center $\mathbf{p}_i$, and

$$\mathbf{r}_i(\mathbf{d}_i) = \begin{cases} 0, & \text{V1}, \\ \mathbf{A}_i \mathbf{d}_i, & \text{V2}, \\ \displaystyle\sum_{\ell=1}^{3} \sum_{m=-\ell}^{\ell} \mathbf{a}_{i,\ell m} Y_{\ell m}(\mathbf{d}_i), & \text{V3}. \end{cases} \tag{7}$$

Here $\mathbf{A}_i \in \mathbb{R}^{3\times 3}$ gives the nine coefficients of the linear V2 residual. For V3, $Y_{\ell m}$ denotes a real spherical-harmonic basis function and $\mathbf{a}_{i,\ell m} \in \mathbb{R}^3$ its RGB coefficient. Degrees one through three contain 15 basis functions, giving 45 learned coefficients; the base RGB $\mathbf{c}_{0,i}$ provides the view-independent constant component.

The viewing direction is evaluated once at the Hexel center. Therefore, for a fixed camera, the V2 and V3 residuals are constant over the primitive: view dependence changes its RGB offset but not the spatial gradients $\mathbf{g}_{u,i}$ and $\mathbf{g}_{v,i}$.

A.2 DERIVATIVES

Support derivatives. Let $\mathbf{u} = (u, v)^\top$ and define the three unit normals

$$\mathbf{n}_0 = (0, 1)^\top, \qquad \mathbf{n}_1 = (\sqrt{3}/2, 1/2)^\top, \qquad \mathbf{n}_2 = (\sqrt{3}/2, -1/2)^\top. \tag{8}$$

Together, $\mathcal{N} = \{\pm\mathbf{n}_0, \pm\mathbf{n}_1, \pm\mathbf{n}_2\}$ are the outward normals of the six sides of the canonical hexagon. The hexagonal function can therefore be written as

$$d(\mathbf{u}) = \max_{\mathbf{n}\in\mathcal{N}} \mathbf{n}^\top \mathbf{u} - 1 = \max_{e\in\{0,1,2\}} |\mathbf{n}_e^\top \mathbf{u}| - 1. \tag{9}$$

Its zero level set defines the canonical hexagonal boundary, with $d < 0$ inside and $d > 0$ outside.

At a point where one projection is strictly larger than the other five, let $\mathbf{n}^\star$ denote the corresponding normal. For sufficiently small changes in $\mathbf{u}$, this projection remains the largest, so $d(\mathbf{u}) = \mathbf{n}^{\star\top}\mathbf{u} - 1$ and

$$\nabla_{\mathbf{u}} d = \mathbf{n}^\star. \tag{10}$$

When two or more projections tie for the largest value, small movements in different directions can select different side normals. The function remains continuous, but has no ordinary gradient at that point. Any of the tied maximizing normals is a valid subgradient: one of the slopes that can be used for optimization at this corner of the function. These ties form a zero-area set, so the ordinary

gradient exists almost everywhere. Although d gives the exact hexagonal contour and inside-outside sign, it is not the Euclidean signed distance outside the hexagon near its vertices.

For the soft support $K(\mathbf{u}) = \sigma(-\beta d(\mathbf{u}))$, the chain rule gives

$$\nabla_{\mathbf{u}} K = -\beta K(1-K)\nabla_{\mathbf{u}} d, \qquad \frac{\partial K}{\partial \beta} = -d(\mathbf{u})K(1-K). \tag{11}$$

The spatial derivative holds wherever d is differentiable. At a tie, substituting a selected maximizing normal gives the corresponding gradient choice for optimization. The derivative with respect to β exists at every point.

The rendered opacity contribution is $\alpha = \min\{oK, 0.99\}$. For $oK < 0.99$,

$$\frac{\partial \alpha}{\partial o} = K, \qquad \frac{\partial \alpha}{\partial K} = o. \tag{12}$$

For $oK > 0.99$, α is locally constant and both derivatives are zero. At $oK = 0.99$, our backward implementation uses the unclamped-branch derivatives.

Color and compositing derivatives. For the spatial color field, $\partial \mathbf{c}_i^{\mathrm{sp}}/\partial u = \mathbf{g}_{u,i}$ and $\partial \mathbf{c}_i^{\mathrm{sp}}/\partial v = \mathbf{g}_{v,i}$. Changes in local coordinates therefore affect both the support weight and the color within the Hexel.

Let $\mathbf{c}_{>i}$ be the color obtained by compositing only the retained Hexels behind primitive i, including the background and starting with transmittance one. With the retained contributions and their order fixed, let $T_i = \prod_{j<i}(1-\alpha_j)$ be the transmittance reaching primitive i. Then

$$\frac{\partial \mathbf{c}(\mathbf{x})}{\partial \mathbf{c}_i} = T_i \alpha_i \mathbf{I}_3, \qquad \frac{\partial \mathbf{c}(\mathbf{x})}{\partial \alpha_i} = T_i\big(\mathbf{c}_i - \mathbf{c}_{>i}\big), \tag{13}$$

where $\mathbf{I}_3$ is the 3×3 identity matrix. The first expression weights the color derivative by the primitive's blending weight. The second describes how increasing its opacity replaces some of the color behind it with its own color.

A back-to-front traversal accumulates these derivatives. Gradients propagate to appearance and opacity, and through the local coordinates and ray–plane intersection to geometry. For V2 and V3, moving the Hexel center also changes its view-dependent color. We differentiate through the numerically stabilized view direction $\mathbf{d}_i = (\mathbf{p}_i - \mathbf{o})/\sqrt{\|\mathbf{p}_i - \mathbf{o}\|^2 + 10^{-8}}$, where $\mathbf{o}$ is the camera center.

B RASTERIZATION

Perspective-correct coordinates. Following 2DGS (Huang et al., 2024), we recover local Hexel coordinates through ray–plane intersection. Let $\mathbf{T}_i \in \mathbb{R}^{3\times 3}$ map local homogeneous coordinates $(u, v, 1)^\top$ to homogeneous image coordinates, combining Eq. (1) with the calibrated camera. Denote its rows by $\mathbf{t}_x^\top$, $\mathbf{t}_y^\top$, and $\mathbf{t}_z^\top$. For a pixel (x, y), define

$$\mathbf{k} = x\mathbf{t}_z - \mathbf{t}_x, \qquad \mathbf{l} = y\mathbf{t}_z - \mathbf{t}_y, \qquad \mathbf{h} = \mathbf{k} \times \mathbf{l}. \tag{14}$$

The intersection must satisfy $\mathbf{k}^\top(u, v, 1)^\top = \mathbf{l}^\top(u, v, 1)^\top = 0$. Since $\mathbf{h}$ is orthogonal to both $\mathbf{k}$ and $\mathbf{l}$, the local coordinates are $(u, v) = (h_x/h_z, h_y/h_z)$ when $h_z \neq 0$. This recovers the perspective-correct intersection without a per-pixel matrix inverse. We skip intersections with $h_z = 0$ or camera-space depth below the near-plane threshold of 0.2.

The same local coordinates are used to evaluate the hexagonal support and spatial color field. Although the local-to-world map is affine, its perspective projection onto the image is generally projective.

Support bounds. The sigmoid is nonzero outside the canonical hexagon, so we bound the region that can contribute above the rasterizer's alpha threshold $\tau = 1/255$. Since $\alpha_i \leq K_i$, a contribution with $\alpha_i \geq \tau$ must satisfy $K_i \geq \tau$. Substituting $K_i = \sigma(-\beta_i d)$ gives

$$d(u, v) \leq \frac{1}{\beta_i} \log \frac{1-\tau}{\tau} = \frac{\log 254}{\beta_i}. \tag{15}$$

This defines a larger regular hexagon with center-to-edge distance $1 + \log 254/\beta_i$. It is enclosed by a circle with radius

$$r_i = \frac{2}{\sqrt{3}} \left(1 + \frac{\log 254}{\beta_i} \right). \tag{16}$$

The radius is in local Hexel coordinates, before tangent scaling and projection. We use this enclosing circle with the projective bounding routine inherited from 2DGS to determine the image tiles assigned to each primitive. The bound is conservative in the local plane because it uses the maximum possible opacity, rather than the learned opacity of each Hexel.

Visibility and compositing. We divide the image into 16×16-pixel tiles and assign each Hexel to the tiles intersecting its projected bound. Within each tile, primitives are sorted by their centers' camera-space depth. This order is shared by the pixels in the tile; we do not sort intersections separately at each pixel.

For each candidate intersection, we evaluate $\alpha_i = \min\{o_i K_i(u_i, v_i), 0.99\}$ and skip contributions with $\alpha_i < 1/255$. Let T be the remaining transmittance before processing the candidate. If $T(1 - \alpha_i) < 10^{-4}$, we stop without accumulating that candidate or updating T. Otherwise, its color contribution is $T\alpha_i \mathbf{c}_i(u_i, v_i, \mathbf{d}_i)$, and we update $T \leftarrow T(1 - \alpha_i)$.

For the m contributions actually accumulated at a pixel, indexed in their compositing order, the final color is

$$\begin{aligned} \mathbf{c}(\mathbf{x}) &= \sum_{i=1}^{m} T_i \alpha_i \mathbf{c}_i(u_i, v_i, \mathbf{d}_i) + T_{\text{end}} \mathbf{c}_{\text{bg}}, \\ T_i &= \prod_{j<i} (1 - \alpha_j), \qquad T_{\text{end}} = \prod_{i=1}^{m} (1 - \alpha_i). \end{aligned} \tag{17}$$

Here $\mathbf{c}_{\text{bg}}$ is the background color. Its weight is the transmittance remaining after the contributions actually accumulated, not the tentative value that triggered early termination.

C Densification and Pruning

The extension in Sec. 3 combines mixed growth with conservative contribution-based pruning, separately from the reported fixed-population configurations.

Mixed growth. The trainer averages screen-space positional-gradient norms over visible observations. A gradient threshold and minimum observation count select parents; an optional event limit retains the largest scores. Maximum tangent scale relative to scene extent separates cloning from splitting. Tangential offsets are Gaussian samples scaled by the parent's two scales and a jitter factor. Clones retain the other parameters, with opacity capped at a configured low value. Splits remove the parent and create two children, dividing both scales by 1.6 in the implemented rule. Appearance coefficients are copied as an initialization, not an exact rendering-preserving transfer.

Weak-pruning score. For the evaluated grown configurations, let $\mathcal{V}_{\text{train}}$ be the training cameras and Ω_v the pixels of view v. Define

$$q_i = \frac{\displaystyle\sum_{v \in \mathcal{V}_{\text{train}}} \sum_{\mathbf{x} \in \Omega_v} T_{i,v}(\mathbf{x}) \alpha_{i,v}(\mathbf{x})}{\displaystyle\sum_{v \in \mathcal{V}_{\text{train}}} |\Omega_v|}. \tag{18}$$

Rejected or invisible contributions have zero weight. Only mature primitives below a contribution threshold are eligible, subject to a per-event removal limit. The extra full-camera sweeps belong in the measured training cost.

Evaluated schedule. The reported grown NVS configurations use mixed growth every 500 iterations between 5K and 15K, with a 2M population ceiling. Parents require a mean position-gradient norm above 2×10^{-4} and at least 20 observations. Clone and split jitter multipliers are 0.25 and 0.5;

clone opacity starts at most at 0.02. Coordinated contribution pruning runs every 1K iterations, uses a 500-iteration newborn grace period and a normalized mean-contribution threshold of 10^{-7}, and removes at most 2% per event. The opacity culling threshold is 0.005; periodic opacity resets are disabled. This combined implementation is evaluated in all grown NVS rows, rather than an unmeasured extension. It is not the separate residual-driven insertion experiment or the older three-seed ablation.

D 2D IMAGE REPRESENTATION

Hexels can represent an image directly by placing the primitives in the image plane. We retain the hexagonal support and first-order spatial color model, but replace 3D projection and depth-ordered compositing with direct image-space evaluation and normalized blending.

Geometry and support. For a pixel $\mathbf{x} = (x, y)^\top$, let $\mathbf{m}_i = (m_{x,i}, m_{y,i})^\top$ be the Hexel center, $\mathbf{R}(\theta_i)$ its planar rotation, and $\mathbf{S}_i = \mathrm{diag}(s_{x,i}, s_{y,i})$ its positive scale matrix. We compute local coordinates and the soft support weight as

$$\mathbf{u}_i(\mathbf{x}) = \mathbf{S}_i^{-1}\mathbf{R}(\theta_i)^\top(\mathbf{m}_i - \mathbf{x}), \qquad w_i(\mathbf{x}) = \sigma(-\beta_i d(\mathbf{u}_i(\mathbf{x}))), \tag{19}$$

where d is the hexagonal function from Eq. (2) and $\beta_i > 0$ is the support sharpness. This evaluates the support directly at each pixel, without projection or ray–plane intersection. Since $d(\mathbf{u}) = d(-\mathbf{u})$, reversing the displacement does not change the support weight. Scales are optimized as log-scales. Hexels (V1) learn $\beta_i = \mathrm{softplus}(\tilde{\beta}_i)$, initialized near 10; the constant-color ablation uses a fixed $\beta = 10$ shared by all primitives and not stored.

Spatial appearance. We use the first-order spatial color model of V1, expressed here in image coordinates:

$$\mathbf{c}_i(\mathbf{x}) = \mathbf{c}_{0,i} + \eta\,[(m_{x,i} - x)\mathbf{g}_{x,i} + (m_{y,i} - y)\mathbf{g}_{y,i}], \tag{20}$$

where $\mathbf{c}_{0,i}$ is the base RGB color, $\mathbf{g}_{x,i}, \mathbf{g}_{y,i} \in \mathbb{R}^3$ are the spatial color coefficients, and $\eta = 0.1$ controls their scale. The coefficients describe color changes along the image axes, using the center-minus-pixel displacement shown above. Unlike Eq. (4), these coordinates are not rotated or normalized by the Hexel scales. Both formulations represent an affine color field, but use different coordinate parameterizations. The constant-color ablation omits the two spatial terms, giving $\mathbf{c}_i(\mathbf{x}) = \mathbf{c}_{0,i}$. Neither model uses view-dependent appearance.

Rendering. Following Image-GS (Zhang et al., 2025b), we retain the $K = 10$ largest support weights among the primitives assigned to the pixel's tile. Let $\mathcal{S}_K(\mathbf{x})$ denote their indices. We blend their colors as

$$\mathbf{c}(\mathbf{x}) = \frac{\sum_{i \in \mathcal{S}_K(\mathbf{x})} w_i(\mathbf{x})\mathbf{c}_i(\mathbf{x})}{\sum_{i \in \mathcal{S}_K(\mathbf{x})} w_i(\mathbf{x}) + \epsilon}, \tag{21}$$

where $\epsilon = 10^-4$ prevents division by zero. Unlike the 3D compositor, this blend has no transmittance, depth ordering, or learned opacity: each retained primitive contributes according to its support weight relative to the others. During optimization, gradients propagate through the selected weights and colors to the Hexel parameters. The selected indices are treated as fixed when computing these gradients.

Finite support. Each Hexel is assigned to the 16×16 tiles overlapping a square of half-width $r_i = \lceil \frac{2}{\sqrt{3}} \max(s_{x,i}, s_{y,i})(1 + 4.5/\beta_i) \rceil$ pixels, which encloses the contour $d = 4.5/\beta_i$ where $w_i = \sigma(-4.5) \approx 0.011$. Every candidate in these tiles is evaluated without a weight threshold, and pixels in tiles without candidates are rendered as zero. Unlike the 3D bound of Eq. (15), this cutoff does not use $1/255$.

Primitive Overdraw. We distinguish the total number of primitives from the number covering each pixel. Since both Gaussian and Hexel supports have nonzero tails, we count a primitive as covering a pixel when its support weight reaches a common fraction $\tau \in (0, 1)$ of its own peak:

$$C_\tau(\mathbf{x}) = \sum_{i=1}^{N} \mathbf{1}\left[\frac{w_i(\mathbf{x})}{\max_{\mathbf{y}} w_i(\mathbf{y})} \geq \tau\right]. \tag{22}$$

Here N is the total primitive count and $\mathbf{1}$ is the indicator function. A value $C_\tau(\mathbf{x}) = 0$ means that no primitive covers the pixel at this threshold; $C_\tau(\mathbf{x}) = 1$ means that exactly one does, and larger values indicate overlapping supports. We count supports before top-K selection, rather than summing the normalized blending weights. This measures spatial overlap, not the number of contributions retained by the renderer or its actual computational cost.

We summarize coverage by the mean of C_τ over the full image and the fractions of pixels covered by zero or multiple supports. Both fractions matter: a mean near one can result from uncovered pixels offsetting heavily overlapping regions. Primitive count and reconstruction quality are reported alongside these measurements. The visual comparison uses identical crops, the same threshold, and a shared coverage-map scale, showing all supports intersecting each crop. Comparing methods at matched reconstruction quality additionally tests whether Hexels require fewer primitives, rather than simply reducing overlap at the expense of quality.

E Implementation Details

E.1 Datasets and Metrics

We evaluate 34 scenes: nine from Mip-NeRF360, fifteen DTU scans (24, 37, 40, 55, 63, 65, 69, 83, 97, 105, 106, 110, 114, 118 and 122), and ten from Tanks and Temples (Barn, Caterpillar, Courthouse, Francis, Horse, Ignatius, Lighthouse, Meetingroom, Train and Truck). Every eighth camera is held out, with identical cameras and images across methods. Mip-NeRF360 uses quarter-resolution outdoor and half-resolution indoor images; DTU and T&T use half resolution. PSNR and SSIM use float RGB images, and LPIPS-VGG inputs are normalized to $[-1, 1]$. We average over test views, then equally over scenes, using seed 0 for each configuration.

Table colors highlight the top three values within each representation family and dataset; displayed ties share ranks. Muted teal, pale blue and warm colors denote volumetric, textured/neural and local-appearance surface methods, respectively. F/G denote fixed/grown Hexels populations. Qualitative views are selected for large PSNR gains over the displayed baselines. Crops share the same image region; PSNR/LPIPS describe the full image and FPS the scene.

E.2 Initialization and Training

Hexels V1-V3 start from the sparse COLMAP points and colors, with one Hexel per point and its color used as the base RGB. Rotations start at the identity quaternion, without estimated surface normals. Both tangent scales are initialized to one quarter of the root-mean-square distance to the three nearest neighbors, capped at 5% of the scene camera extent. Opacity starts at 0.1 and support sharpness at 10. Spatial color gradients and view-dependent coefficients start at zero. We optimize geometry, opacity, sharpness, and appearance jointly. We train with Adam for 30K iterations using the photometric loss in the main text, with SSIM weight 0.2. Fixed arms disable population adaptation; grown arms use the measured mixed-growth schedule in Appendix C. These NVS runs do not enable depth distortion or normal regularization and do not use an MVS initializer. Initial learning rates are given in Table 6.

E.3 Baseline Settings

We use the authors' implementations of 2DGS, 3DGS, 3D Convex Splatting, Triangle Splatting, BBSplat, MeshSplatting and Nexels. BBSplat trains for 32K iterations; the other baselines train for 30K. Its released Mip-NeRF360 recipe uses 160K indoor and 300K outdoor primitives, and its deployment includes compressed textures.

Table 6: Initial learning rates for the Hexel V1-V3 models for NVS experiments. Only parameters enabled by a variant are optimized. All rates are halved after 15K, 20K, and 25K iterations.

Parameter	Learning rate
Center	1.6×10^{-4}
Log tangent scales	5×10^{-3}
Rotation quaternion	10^{-3}
Opacity logit	10^{-1}
Base RGB	10^{-2}
Spatial color gradients	5×10^{-3}
Linear view coefficients (V2)	5×10^{-3}
SH coefficients (V3)	5×10^{-4}
Log boundary sharpness	10^{-2}

Model-memory budgets include all primitive and appearance parameters, including textures, feature grids and decoders. Nexels uses a reduced feature grid within each budget. MeshSplatting is matched at initialization; native remeshing can change its final memory. Matched-primitive experiments limit final populations to the scene's SfM point count while retaining native growth and pruning. Native comparisons use each baseline's original population policy.

E.4 Timing and Memory

Hardware and rendering speed. Experiments use RTX A5000 GPUs; starred rows include RTX 6000 Ada runs where A5000 memory was insufficient. Mixed-GPU efficiency values are not ranked. Resident rendering is timed after two warmup passes, using four trials of ten passes over all test cameras with GPU synchronization. We report the median FPS per scene and average over scenes, excluding loading, image transfer and metric computation.

Model memory and deployment size. Model memory counts FP32 representation parameters, excluding optimizer state. V1/V2/V3 use 80/116/260 bytes per primitive. Deployment size counts the files needed for inference, including configuration and appearance dictionaries, in decimal MB. The compression methods and their quality effects are described in Appendix K.3; baseline files retain their native formats.

Training resources. Peak VRAM is sampled process GPU memory during training, in GiB; dataset values average the per-scene peaks. Training time includes initialization, optimization and checkpoint saving, but excludes separate evaluation, compression and video generation.

F Spatial Appearance Ablations

Tables 13 and 14 compare all six Hexels configurations (V1/V2/V3 with fixed or densified populations) on Mip-NeRF360, DTU, and Tanks and Temples. V3 achieves the highest PSNR within each population policy across all three datasets, while fixed populations require less memory and render faster. Densification improves V3's PSNR on Mip-NeRF360 and Tanks and Temples; on DTU, fixed V3 retains higher PSNR (29.94 versus 29.75 dB), whereas densified V3 improves LPIPS (0.268 versus 0.294).

G Surface Reconstruction Ablations

We ablate Hexel surface-reconstruction choices on DTU data. Every experiment uses all registered views, 30,000 training iterations and seed 0, with the same TSDF extraction and official DTU evaluation. We report the mean of accuracy and completeness in millimeters, averaged over all scans. Fixed-population experiments use the same per-scene primitive count for all baselines, so their comparison changes initialization without increasing the population. The normal-consistency and depth-distortion weights are 0.05 and 1000, respectively (used only for surface reconstruction).

Surface regularization. Table 7 shows that removing both losses raises Chamfer distance to 1.667 mm. Depth distortion alone reduces it to 1.385 mm, while normal consistency alone reaches

1.061 mm. Combining both gives 1.034 mm, a 37.9% reduction relative to neither loss and a further 2.5% reduction relative to normal consistency alone. Thus, normal consistency provides the larger individual gain, with a smaller complementary benefit from depth distortion.

Initialization, appearance and population. MVS initialization lowers mean error for all six matched combinations of appearance model and population policy. For fixed V3, it improves 1.154 mm to 1.034 mm (10.4%) at the same model memory. With fixed populations and MVS, V3 also improves on V1 (1.339 mm) and V2 (1.353 mm), although its richer appearance model uses more memory. Densified V3 with MVS obtains the lowest error, 0.989 mm, but its 32.45 MB model is 4.05 times larger than fixed V3 at 8.01 MB. The 4.4% error reduction therefore comes with a substantial memory cost; we use fixed V3 with MVS and both losses for the main compact comparison.

Table 7: **DTU surface ablations.** Mean Chamfer distance (CD) over the same six scans. Normal consistency and depth distortion are also applied; MVS improves every matched recipe. Densification improves accuracy at increased model memory.

Loss ablations: fixed V3, MVS initialization			
Normal consistency	Depth distortion	CD (mm)↓	Model (MB)↓
No	No	1.667	8.01
No	Yes	1.385	8.01
Yes	No	1.061	8.01
Yes	Yes	1.034	8.01
Appearance, population and initialization: both losses			
Variant / population	Initialization	CD (mm)↓	Model (MB)↓
V1 / fixed	SfM	1.674	2.46
V1 / fixed	MVS	1.339	2.46
V1 / densified	SfM	1.308	9.55
V1 / densified	MVS	1.179	10.66
V2 / fixed	SfM	1.715	3.57
V2 / fixed	MVS	1.353	3.57
V2 / densified	SfM	1.493	12.76
V2 / densified	MVS	1.256	14.07
V3 / fixed	SfM	1.154	8.01
V3 / fixed	MVS	1.034	8.01
V3 / densified	SfM	1.126	29.87
V3 / densified	MVS	0.989	32.45

Red, orange and yellow mark the three lowest CD values within each block; equal displayed values share ranks. Model memory is uncompressed learned state, excluding optimizer and rendering buffers.

Figure 9 shows all 15 distinct configurations on scan24, the first development scan. The common view illustrates how surface regularization and initialization affect the reconstructed building; full-scan scores are shown beneath every panel. Individual scan rankings can differ from the six-scan averages.

Unconstrained surface reconstruction. We additionally evaluate the methods without a matched memory budget. In this setting, 2DGS achieves lower Chamfer distance than Hexels, but Hexels remain competitive while requiring substantially less model memory and training resources. This gives a complementary view of the same trade-off: even when the baselines are allowed to use larger representations, Hexels retain strong surface quality with a much smaller and more efficient representation.

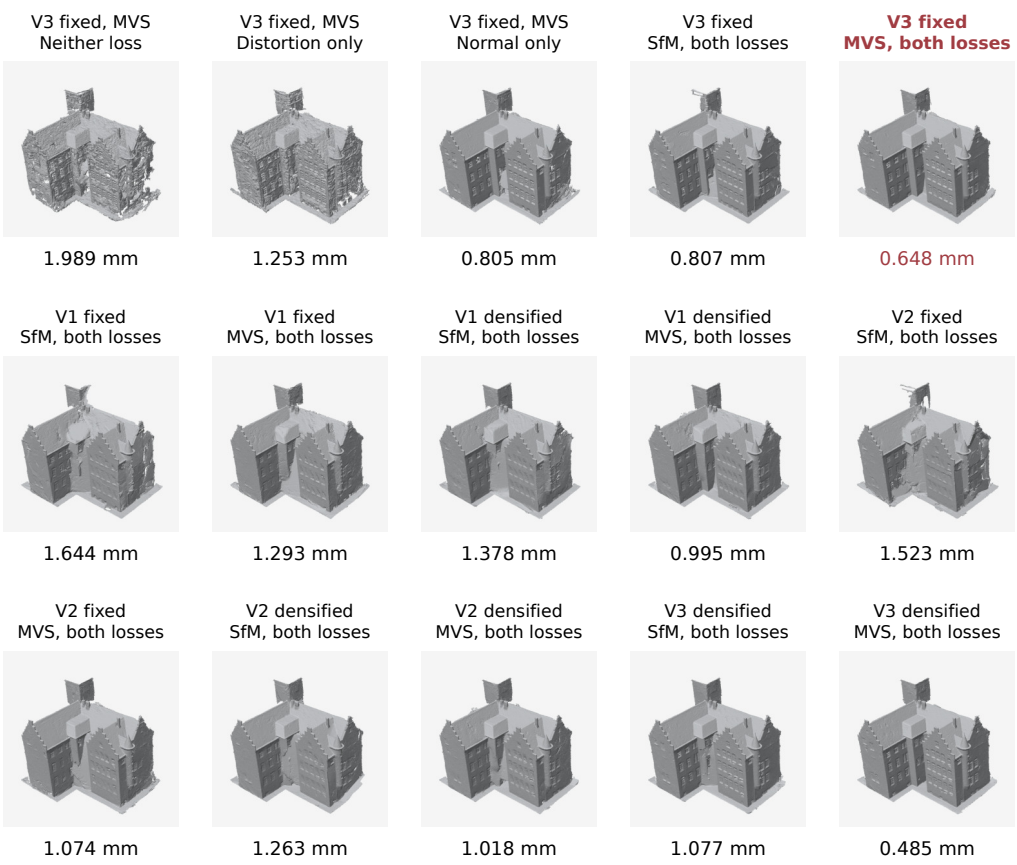


Figure 9: **Surface reconstruction ablations on DTU scan24.** All 15 distinct Hexels configurations share the same camera, crop and shading; labels below each mesh give full-scan Chamfer distance in millimeters (lower is better). The first row isolates losses and initialization around fixed V3 Hexel (no densification); remaining rows show the V1/V2/V3 recipe comparison. Except for the first three loss ablations, every panel uses both losses. Normal consistency produces smoother facade surfaces; both losses give the lowest fixed-V3 error.

H Applications

H.1 Direct Surface-Cell Extraction

Hexels provide an explicit surface representation that can be exported directly after optimization from images. Unlike the depth-fusion pipeline used for mesh extraction in 2DGS (Huang et al., 2024), this conversion requires no depth rendering, voxel grid, or Marching Cubes. The resulting surface cells can be used directly in object-centric applications such as 3D generation, segmentation, and representation learning.

For each retained Hexel, we transform the six vertices of its canonical hexagon, defined by $d = 0$, into world space using Eq. (1). Connecting consecutive vertices produces six edges and one polygonal face. Colors are obtained from the base RGB or evaluated from the spatial color field. Each cell remains associated with its parent Hexel, providing direct access to its geometry and appearance.

The conversion requires $O(1)$ work per Hexel. The vertex positions and evaluated colors are differentiable with respect to the corresponding Hexel parameters. The exported cells can be optimized independently - opening up avenues for many downstream applications, both in 3D and 4D. Figure 10 shows the resulting wireframes and surface cells.

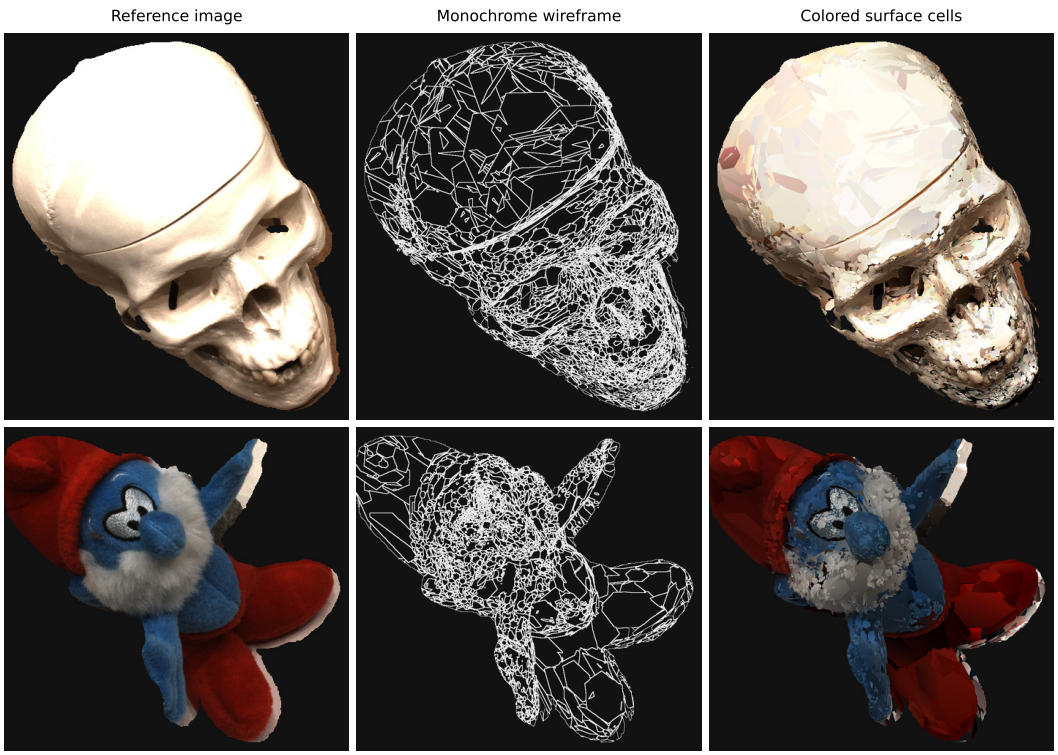


Figure 10: **Direct surface extraction from Hexels**. Hexels provide an explicit surface representation that can be directly exported. The center column shows the wireframe (that is, the edges of the surface cell), and the right column visualizes the surface cells post-extraction.

H.2 Deployment on Memory-Limited Devices

Hexels require little model memory and training VRAM while retaining high reconstruction quality and fast rendering. These advantages are obtained in full precision, without texture atlases, neural decoders, pruning, or quantization. This makes Hexels well suited to reconstruction and rendering on memory-limited and edge hardware, and provides a basis for mobile deployment without changing the underlying primitive. Pruning and quantization could further reduce storage, but are not required for the efficiency demonstrated here.

We have developed a Metal-based Hexel renderer and will release it alongside training code for memory-limited devices. The same hexagonal support and appearance model are retained, without converting the learned scene into a different representation for deployment. Hexels may have positive implications for on-device and privacy-conserving NVS and surface reconstruction.

I Why Hexagons and Spatial Color?

We want each Hexel to represent a large surface region without losing the appearance variation inside it. This requires two choices: a shape that can cover surfaces efficiently, and a color model that can describe more than a constant color. The honeycomb theorem motivates the first choice: regular hexagons minimize boundary length among equal-area partitions of the plane (Hales, 2001). The spatial color field addresses the second.

We examine these choices on one planar patch at a fixed viewing direction. First, we show that an affine color field represents the local color slope, with the remaining error controlled by how that slope changes across the patch. This determines a region where one color field stays within a chosen error tolerance. We then show that, within the same region, the largest affine hexagon covers twice the area of the largest affine triangle and about 30% more than the largest affine square. These results explain why we combine hexagonal patches with spatially varying color. Finally, we

show how errors in individual contributions combine under the 3D alpha-compositing used by the renderer.

I.1 Linear Color

We fix the viewing direction and use physical, orthonormal coordinates $\mathbf{x} \in \mathbb{R}^2$ on a local plane. Let $\mathbf{p}$ be the patch center and $\mathbf{h} = \mathbf{x} - \mathbf{p}$. In axes aligned with the primitive, $\mathbf{h} = (s_u u, s_v v)^\top$, where (u, v) are its normalized local coordinates.

Let f be one target color channel. An affine color field represents a constant and a linear change in color. The remaining approximation error depends on the second derivatives of f, which describe how its slope changes across the patch. This is the standard local approximation argument underlying piecewise-polynomial representations (Cohen & Mirebeau, 2011).

Proposition I.1 (Local error of an affine color field)**.** *Let f be twice continuously differentiable on an open convex neighborhood U containing $\mathbf{p}$. Suppose a fixed symmetric positive-definite matrix $\mathbf{Q}$ satisfies*

$$|\mathbf{z}^\top D^2 f(\mathbf{x})\mathbf{z}| \le \mathbf{z}^\top \mathbf{Q}\mathbf{z} \qquad \text{for all } \mathbf{x} \in U,\ \mathbf{z} \in \mathbb{R}^2. \tag{23}$$

Then, for every $\mathbf{p} + \mathbf{h} \in U$, the affine field $\ell_{\mathbf{p}}(\mathbf{p} + \mathbf{h}) = f(\mathbf{p}) + \nabla f(\mathbf{p})^\top \mathbf{h}$ satisfies

$$|f(\mathbf{p} + \mathbf{h}) - \ell_{\mathbf{p}}(\mathbf{p} + \mathbf{h})| \le \tfrac{1}{2}\mathbf{h}^\top \mathbf{Q}\mathbf{h}. \tag{24}$$

In particular, if $\|D^2 f(\mathbf{x})\|_{\mathrm{op}} \le M$ throughout U, the error is at most $Mr^2/2$ wherever $\|\mathbf{h}\| \le r$.

Proof. Define $g(t) = f(\mathbf{p} + t\mathbf{h})$ for $t \in [0, 1]$. Applying the fundamental theorem of calculus twice gives

$$\begin{aligned} f(\mathbf{p} + \mathbf{h}) - \ell_{\mathbf{p}}(\mathbf{p} + \mathbf{h}) &= \int_0^1 (1 - t) g''(t)\, dt, \\ g''(t) &= \mathbf{h}^\top D^2 f(\mathbf{p} + t\mathbf{h})\mathbf{h}. \end{aligned} \tag{25}$$

The segment lies in U, so Eq. (23) gives

$$\begin{aligned} |f(\mathbf{p} + \mathbf{h}) - \ell_{\mathbf{p}}(\mathbf{p} + \mathbf{h})| &\le \int_0^1 (1 - t) \left|\mathbf{h}^\top D^2 f(\mathbf{p} + t\mathbf{h})\mathbf{h}\right| dt \\ &\le \mathbf{h}^\top \mathbf{Q}\mathbf{h} \int_0^1 (1 - t)\, dt = \tfrac{1}{2}\mathbf{h}^\top \mathbf{Q}\mathbf{h}. \end{aligned} \tag{26}$$

The operator-norm statement follows by replacing the quadratic-form bound with $M\|\mathbf{h}\|^2$. □

The affine field therefore accounts for the local slope, leaving an error controlled by changes in that slope. In comparison, using only the constant value $f(\mathbf{p})$ gives an error bound of Lr when $\|\nabla f\| \le L$. A nonconstant affine color ramp is represented exactly by the affine field, but not by one constant color. This compares individual color fields; blending constant-color primitives can also produce spatial variation.

The Hexel spatial field contains this Taylor approximation. For one color channel, in primitive-aligned physical coordinates, the required coefficients are

$$c_0 = f(\mathbf{p}), \qquad g_u = s_u\, \partial_1 f(\mathbf{p}), \qquad g_v = s_v\, \partial_2 f(\mathbf{p}). \tag{27}$$

Substituting these into $c_0 + u g_u + v g_v$ gives $\ell_{\mathbf{p}}(\mathbf{p} + \mathbf{h})$. Normalized local coordinates therefore change the coefficients, not the class of affine functions that can be represented.

For RGB, apply the same argument to each channel using a common matrix $\mathbf{Q}$. Let $\mathbf{f}$ be the target RGB field and $\boldsymbol{\ell}_{\mathbf{p}}$ its channel-wise Taylor approximation. For a learned field $\mathbf{C}$, define $\eta(\mathbf{x}) = \|\mathbf{C}(\mathbf{x}) - \boldsymbol{\ell}_{\mathbf{p}}(\mathbf{x})\|_\infty$. Then

$$\|\mathbf{C}(\mathbf{p} + \mathbf{h}) - \mathbf{f}(\mathbf{p} + \mathbf{h})\|_\infty \le \eta(\mathbf{p} + \mathbf{h}) + \tfrac{1}{2}\mathbf{h}^\top \mathbf{Q}\mathbf{h}. \tag{28}$$

Here $\|\cdot\|_\infty$ is the largest absolute RGB-channel error. The term η accounts for the difference between the learned field and the Taylor field. The result establishes the available approximation capacity, rather than assuming that optimization reaches the Taylor coefficients.

I.2 PATCH SHAPE

For a tolerance $\varepsilon > 0$, define

$$E = \{\mathbf{p} + \mathbf{h} : \mathbf{h}^\top \mathbf{Q}\mathbf{h} \le 2\varepsilon\}. \tag{29}$$

Provided $E \subset U$, the Taylor color error is at most ε throughout this ellipse. Its longer directions allow larger displacements while satisfying the same error bound. The matrix $\mathbf{Q}$ bounds changes in color slope; it is not a surface curvature matrix and is not estimated by our training procedure.

The regular triangle, square, and hexagon each tile the plane. We now compare their affine images inside the same error ellipse: how much area can each patch cover while satisfying the same Taylor color-error bound?

Proposition I.2 (Patch area within one error ellipse)**.** *Let H_m be a filled regular m-gon centered at the origin, with $m \ge 3$ and unit circumradius. Among all translated, nonsingular affine images of H_m contained in E, the maximum area is*

$$A_m^{\max} = \frac{m \sin(2\pi/m)\, \varepsilon}{\sqrt{\det \mathbf{Q}}}. \tag{30}$$

A maximizer is $\mathbf{p} + \sqrt{2\varepsilon}\, \mathbf{Q}^{-1/2}\mathbf{R}H_m$, where $\mathbf{R}$ is any orthogonal matrix. Consequently,

$$\frac{A_6^{\max}}{A_4^{\max}} = \frac{3\sqrt{3}}{4} \approx 1.299, \qquad \frac{A_6^{\max}}{A_3^{\max}} = 2. \tag{31}$$

Proof. The coordinate change $\mathbf{y} = \mathbf{Q}^{1/2}(\mathbf{x} - \mathbf{p})/\sqrt{2\varepsilon}$ maps E to the unit disk.

Let $\mathbf{v}_j$, $j = 0, \dots, m-1$, be the vertices of H_m. Their equal angular spacing gives

$$\frac{1}{m}\sum_j \mathbf{v}_j = 0, \qquad \frac{1}{m}\sum_j \mathbf{v}_j \mathbf{v}_j^\top = \tfrac{1}{2}\mathbf{I}_2. \tag{32}$$

These identities follow from the sums of the sine and cosine terms at the vertex angles and at twice those angles.

Consider an affine image $\mathbf{t} + \mathbf{B}H_m$ contained in the unit disk. Every vertex has squared distance at most one, so

$$\|\mathbf{t}\|^2 + \tfrac{1}{2}\operatorname{tr}(\mathbf{B}^\top \mathbf{B}) = \frac{1}{m}\sum_j \|\mathbf{t} + \mathbf{B}\mathbf{v}_j\|^2 \le 1. \tag{33}$$

Let σ_1, σ_2 be the singular values of $\mathbf{B}$. Then

$$|\det \mathbf{B}| = \sigma_1\sigma_2 \le \tfrac{1}{2}(\sigma_1^2 + \sigma_2^2) \le 1 - \|\mathbf{t}\|^2 \le 1. \tag{34}$$

Thus, no translated affine image has greater area than the centered regular polygon. Equality is attained with $\mathbf{t} = 0$ and $\mathbf{B}$ orthogonal, for which the polygon lies inside the disk.

The area of H_m is $m \sin(2\pi/m)/2$. Returning to the original coordinates multiplies area by $2\varepsilon/\sqrt{\det \mathbf{Q}}$, yielding Eq. (30). □

Connection to the Hexel parameterization. The maximizing hexagon can be represented using the rotation and two tangent scales already present in Hexels. Write $\mathbf{Q} = \mathbf{U}\operatorname{diag}(\lambda_1, \lambda_2)\mathbf{U}^\top$, where $\lambda_1, \lambda_2 > 0$ and the eigenvectors in $\mathbf{U}$ are chosen to form a rotation. Align the tangent axes with these eigenvectors. Since our canonical hexagon has inradius one and circumradius $2/\sqrt{3}$, choose

$$s_u = \sqrt{\frac{3\varepsilon}{2\lambda_1}}, \qquad s_v = \sqrt{\frac{3\varepsilon}{2\lambda_2}}. \tag{35}$$

This places the six vertices on the error ellipse and attains $A_6^{\max}$. No shear parameter is needed.

Thus, among affine images of these three regular tiling cells, the hexagon covers the largest area inside the same sufficient color-error ellipse. Combined with the tiling motivation, this supports using a hexagonal patch with an affine color field: one element can represent a larger region than the corresponding triangular or square element under this local approximation bound.

I.3 3D Alpha Compositing

The preceding results concern the color represented within individual patches. In 3D rendering, the final pixel also depends on alpha compositing and the remaining background transmittance.

Proposition I.3 (Error under alpha compositing)**.** *At one pixel, let $0 \le \alpha_i \le 1$ be a finite ordered list of retained contributions, with colors C_i. Define*

$$w_i = \alpha_i \prod_{j<i}(1-\alpha_j), \qquad T = \prod_i (1-\alpha_i), \qquad I = \sum_i w_i C_i + Tb, \tag{36}$$

where b is the background color. For any target color f and any norm,

$$\|I - f\| \le \sum_i w_i \|C_i - f\| + T\|b - f\|. \tag{37}$$

Proof. Set $T_0 = 1$ and $T_i = \prod_{j\le i}(1-\alpha_j)$. Then $w_i = T_{i-1} - T_i \ge 0$, giving $\sum_i w_i + T = 1$. Hence,

$$I - f = \sum_i w_i (C_i - f) + T(b - f).$$

The result follows from the triangle inequality. □

Thus, if every contributing color differs from the target by at most e, the image error is at most $(1-T)e + T\|b-f\|$. This result is independent of the primitive shape. It also shows why local color accuracy alone does not ensure an accurate image: the contributions must approximate the same pixel target, and the remaining background contribution must be included.

Soft support and visibility. The sigmoid support extends beyond the nominal hexagon, where the local Taylor bound may no longer apply. Let $\mathcal{G}$ contain contributions with valid bounds $\|C_i - f\|_\infty \le e_i$ relative to the target color at this pixel. If the remaining contributions have error at most D and total weight $\zeta = \sum_{i\notin\mathcal{G}} w_i$, then

$$\|I - f\|_\infty \le \sum_{i\in\mathcal{G}} w_i e_i + \zeta D + T\|b - f\|_\infty. \tag{38}$$

Equation (28) can supply e_i when the local target at the intersection is the desired pixel color. An accurate approximation to a different, occluded surface does not provide that bound. Since individual Hexel colors are unclamped, $D = 1$ cannot be assumed.

These statements apply to the contributions actually accumulated by the renderer, with T their remaining transmittance.

J Extended Related Work

Novel view synthesis and surface reconstruction. NeRF (Mildenhall et al., 2020) was introduced for novel view synthesis and represents a scene as a neural field of density and view-dependent radiance. This representation enabled high-quality novel-view rendering. However, the repeated network evaluations required for volumetric rendering made the original NeRF expensive to train and slow to render. A large body of follow-up work introduced explicit feature grids, hash encodings, precomputed representations, and baked representations to reduce the sampling, rendering, and computational demands of the original neural-field models (Müller et al., 2022; Fridovich-Keil et al., 2022; Reiser et al., 2023).

3DGS (Kerbl et al., 2023) introduced an alternative for substantially more efficient rendering using explicit anisotropic Gaussian volume primitives, enabling real-time rasterization. However, because of their volumetric nature, 3D Gaussians are not naturally surface aligned and can produce inaccurate geometry. To remedy this, works such as SuGaR (Guédon & Lepetit, 2023) regularize Gaussians toward surfaces. An important solution was introduced by 2DGS (Huang et al., 2024), which instead represents the scene using oriented planar Gaussian surfels and perspective-correct ray–plane intersections, retaining fast rasterization while substantially improving the geometry.

Despite the large improvements made by 3DGS and 2DGS in training and rendering time relative to NeRF, these improvements are still not enough for truly resource-constrained settings. Efficiency requires several things at the same time: learned model storage and training VRAM should be as low as possible, primitive count should remain small relative to the cost of each primitive, and the representation should still retain high reconstruction quality and fast rendering. 3DGS and 2DGS can both require large primitive populations and substantial memory. Despite this, we find that 2DGS remains the strongest prior native surface representation in terms of the overall performance–efficiency balance when these factors are considered together, even after a number of subsequent works have attempted to improve different parts of the problem.

Primitive-based differentiable rendering. Different follow-up works have tried to make the primitive more expressive, reduce the number of primitives, increase rendering speed, reduce model storage, or address limitations of 2DGS and earlier Gaussian representations, such as the infinite support of Gaussian kernels. GES (Hamdi et al., 2024), for example, replaces the Gaussian with a more flexible generalized exponential kernel, while Deformable Beta Splatting (Liu et al., 2025) introduces bounded and deformable kernels. 3D Convex Splatting (**?**) instead uses smooth 3D convex primitives, allowing more flexible volumetric geometry and sharper boundaries, but remains a volumetric rather than surface-aligned representation.

Triangle Splatting (Held et al., 2025b), on the other hand, optimizes an independent triangle soup so that high rendering speeds can be achieved while retaining a surface-aligned primitive. However, high-quality scenes can still require very large triangle populations, increasing both model storage and the computation and memory required during optimization. The independent triangle soup also does not provide surface connectivity. MeshSplatting (Held et al., 2025a) follows this work by introducing explicit connectivity and an opaque mesh representation, trading the independence of individual splats for a conventional connected surface. Other works increase geometric complexity in different ways, including Quadratic Gaussian Splatting (Zhang et al., 2025c), which introduces curved patches, and MILo (Guédon et al., 2025), which brings mesh construction directly into Gaussian optimization.

Hexels are also primitive-based renderers, but we try to solve a more fundamental problem: can we design a more efficient primitive that is able to exploit the structure found in the world, both in geometry and appearance? Like 2DGS, we retain independent planar elements, but change the region represented by one element and the signal represented inside it. We use an affine hexagonal patch, which gives a large, nearly isotropic local region, together with a first-order color field that models smooth appearance variation across a surface. We find that much of the variation on local surfaces is smooth, making such a representation particularly efficient.

A first-order color field itself is not unique to Hexels: a vertex-colored triangle, for example, can also represent an affine color field. The domain of our primitive, however, makes an important difference. Under the same local approximation-error ellipse, our analysis shows that the largest admissible affine regular hexagon can cover twice the area of the corresponding affine triangle and about $1.3\times$ that of the square (Appendix I). Hexels therefore increase the amount represented by a simple planar element without requiring complex geometry, volumetric convexes, or mesh connectivity.

Hybrid primitives and representations. The most direct comparisons to Hexels are fundamentally native surface primitives. However, there has also been a trend toward hybrid representations that separate geometry from appearance in order to improve particular aspects of efficiency over 2DGS-like representations. Gaussian Billboards (Weiss & Bradley, 2024), for example, introduces spatially varying appearance through per-splat texture interpolation. BBSplat (Svitov et al., 2025) learns separate RGB and alpha texture maps to reduce the number of geometric primitives required. This can substantially reduce primitive count, but the textures themselves become part of the model and training cost.

NeST (Zhang et al., 2025a) follows a related idea by separating sparse surface geometry from a neural appearance representation. Nexels (Rong et al., 2025) similarly combines sparse surfels with a shared neural appearance field and per-primitive colors. These methods can substantially reduce geometric primitive count, but this capacity is moved into texture maps, feature fields, or neural networks, which introduce additional memory, training state, and computation.

Concurrent work such as Bake It Till You Make It (Kelkar et al., 2026), which is also centered on improving the efficiency of surface splats, follows this hybrid direction. It uses sparse surface geometry together with a hash-based appearance representation during optimization, and later bakes high-frequency appearance into a texture atlas for rendering. This can reduce the number of geometric surface primitives and improve rendering speed, but appearance is again moved outside the primitive and requires a separate representation and baking pipeline.

These ideas of separating geometry and appearance, neural textures, and texture baking are important, but they can in principle be applied on top of different native surface primitives. They therefore answer a somewhat different question from ours. We include these methods in our comparisons, but count their full learned representation, including textures, feature grids, and neural decoders, rather than only the number of geometric elements.

Population control and compression. Given the multifaceted nature of the efficiency problem, another body of work has developed around interventions on existing primitives to improve different aspects of their efficiency. Scaffold-GS (Lu et al., 2024), for example, reduces 3DGS redundancy through anchors that generate view-adaptive Gaussian attributes rather than storing every Gaussian independently. 3DGS-MCMC (Kheradmand et al., 2025) improves primitive allocation through stochastic updates and relocation. FastGS (**?**) instead focuses on training time, using multi-view-consistent densification and pruning to reduce unnecessary Gaussian growth.

A related line of work compresses already learned representations. LightGaussian (Fan et al., 2024) prunes low-importance Gaussians, distills appearance into lower-order spherical harmonics, and applies vector quantization to reduce model size. Compact 3D Gaussian Representation (**?**) combines learnable masking with shared neural appearance, vector quantization, and entropy coding. Compressed 3DGS (Niedermayr et al., 2024) similarly uses quantization-aware clustering and codebooks to reduce storage. Related ideas have also been developed for surface-based Gaussian representations.

These methods are useful and complementary, but they start with an already chosen primitive and add additional methodology around it. They may greatly improve one aspect of efficiency without necessarily reducing the memory and computation required elsewhere, particularly during training. Our focus is therefore the more fundamental question: we can add pruning, quantization, codebooks, texture atlases, neural decoders, and other post-hoc compression, but can we also start with a more efficient native surface primitive? Hexels address this question before these additional techniques are applied; many of them could still be applied on top of Hexels afterward.

Neural Image Representations. A similar line of research has developed independently in 2D, where neural image representations are used to encode visual media. Earlier approaches often relied on autoencoders or implicit neural functions. More recently, GaussianImage (Zhang et al., 2024) adapted ideas from Gaussian splatting to represent images using simple 2D Gaussian primitives, demonstrating fast fitting and rendering. Image-GS (Zhang et al., 2025b) further improves this representation using content-adaptive initialization and progressive optimization of anisotropic 2D Gaussians. SAD (Iinbor et al., 2026) instead represents the image through a soft anisotropic partition whose regions depend on neighboring sites rather than independently parameterized primitives.

We show that Hexels can also be used directly for 2D image representation, where they outperform Image-GS in reconstruction quality and training convergence under our evaluated settings. More importantly, Hexels are designed as a general surface primitive rather than separate 2D and 3D representations. In 2D, the primitive is placed directly in the image plane; in 3D, it is oriented on a surface patch. While the compositor changes where required by visibility and depth ordering, the basic hexagonal support and appearance model remain the same. This gives Hexels a general capability across image representation, novel-view synthesis, and surface reconstruction, rather than requiring a separate representation for each setting.

Overall efficiency. A common pattern across methods that have tried to make surface-aligned representations more efficient is that the cost is often moved to another part of the problem. Faster rasterization may still require a very large primitive population. Sparse geometry may move capacity into external texture maps or neural fields, increasing representation complexity, training VRAM, or final model storage. Compression can make the deployed scene small while having much less effect

on the size and memory required during training. Richer geometry can reduce primitive count while increasing the cost of each primitive.

Hexels are designed to alleviate these problems together, without first resorting to post-hoc retrofitting and before pruning or compression is applied. We use the fact that natural images, surfaces, and scenes contain local structure. Large regions are often approximately planar and vary smoothly in appearance. A Hexel covers a bounded, efficient local region and models first-order appearance variation inside it, allowing simple regions to be represented by larger and fewer elements. This allows us to target high quality, low learned storage, low training VRAM, few primitives, and high FPS in the same native surface representation, in both 2D and 3D. Pruning, quantization, neural appearance models, and other post-hoc improvements remain possible additions rather than requirements of the representation.

K Full NVS Results

We report additional model-memory budgets, all six Hexels configurations, compression results and per-scene performance on the same 34-scene benchmark. F stands for fixed primitives from SfM initialization whereas G stands for their densified counterparts.

K.1 Matched Model Memory

At the V2 memory budget, Hexels exceeds Triangle Splatting by 1.40 dB outdoors and 2.19 dB indoors on Mip-NeRF360 (Table 8). On DTU it renders at $1.74\times$ Triangle's mean FPS (Table 9).

Method	Mip-NeRF360 outdoor (5)						Mip-NeRF360 indoor (4)					
	Train min ↓	VRAM GiB ↓	FPS ↑	PSNR dB ↑	SSIM ↑	LPIPS VGG ↓	Train min ↓	VRAM GiB ↓	FPS ↑	PSNR dB ↑	SSIM ↑	LPIPS VGG ↓
3DGS	6.3	0.66	1112.3	16.72	0.313	0.722	11.4	2.44	376.6	27.13	0.823	0.412
3D Convex	24.3	0.88	178.0	19.04	0.416	0.638	54.6	1.86	49.3	25.04	0.826	0.404
BBSplat	28.0	1.41	109.0	19.24	0.374	0.688	52.2	8.00	44.4	25.19	0.779	0.443
Nexels	61.3	4.71	59.4	20.72	0.460	0.534	111.3	7.97	33.6	25.74	0.807	0.379
2DGS	16.5	1.92	245.1	16.11	0.328	0.736	25.1	5.93	115.4	22.13	0.762	0.489
Triangle Spl.	12.9	0.96	358.9	20.71	0.448	0.592	21.0	1.45	233.8	26.55	0.843	0.383
MeshSplatting	27.0	3.70	72.9	18.81	0.349	0.674	40.9	5.52	45.1	20.83	0.665	0.558
Hexels V2-F	17.8	0.63	495.8	22.11	0.562	0.497	32.8	0.87	228.1	28.74	0.897	0.290

Table 8: **Matched model memory on Mip-NeRF360.** Fixed-population Hexels V2-F defines mean budgets of 7.33/20.77 MB outdoors/indoors. Hexels V2 improves PSNR by 1.40/2.19 dB relative to Triangle Splatting, respectively.

Method	DTU (15)						Tanks and Temples (10)					
	Train min ↓	VRAM GiB ↓	FPS ↑	PSNR dB ↑	SSIM ↑	LPIPS VGG ↓	Train min ↓	VRAM GiB ↓	FPS ↑	PSNR dB ↑	SSIM ↑	LPIPS VGG ↓
3DGS	4.7	0.45	937.2	28.34	0.867	0.365	6.1	0.94	723.5	21.53	0.718	0.430
3D Convex	12.8	0.61	280.9	27.49	0.863	0.356	22.4	0.91	172.3	20.72	0.706	0.445
BBSplat	14.3	0.64	183.2	27.63	0.854	0.347	21.7	1.55	123.7	19.46	0.643	0.521
Nexels	28.8	2.26	154.6	28.83	0.887	0.277	69.9	2.98	93.6	22.02	0.729	0.383
2DGS	8.6	0.70	426.3	22.88	0.786	0.450	11.1	1.61	249.1	19.37	0.675	0.481
Triangle Spl.	7.7	0.61	513.9	26.65	0.840	0.364	10.1	0.80	459.9	22.04	0.764	0.368
MeshSplatting	13.3	1.85	128.7	21.44	0.685	0.499	17.2	2.17	117.5	16.66	0.581	0.553
Hexels V2-F	8.5	0.56	895.4	29.57	0.895	0.298	14.4	0.60	393.2	23.49	0.819	0.287

Table 9: **Matched model memory on DTU and Tanks and Temples.** Fixed-population Hexels V2-F defines mean budgets of 3.94/21.12 MB. Hexels V2 improves PSNR by 2.92/1.44 dB relative to Triangle Splatting, respectively.

At the V3 budget, the PSNR gains over Triangle are 0.54/0.49 dB on outdoor/indoor Mip-NeRF360 (Table 10) and 2.71/0.82 dB on DTU/T&T (Table 11). Indoor 3DGS remains 0.24 dB ahead of Hexels. Across these V2/V3 budgets, Hexels exceeds MeshSplatting by 3.00–8.14 dB across the four dataset panels, with higher SSIM and lower LPIPS.

Method	Mip-NeRF360 outdoor (5)						Mip-NeRF360 indoor (4)					
	Train min ↓	VRAM GiB ↓	FPS ↑	PSNR dB ↑	SSIM ↑	LPIPS VGG ↓	Train min ↓	VRAM GiB ↓	FPS ↑	PSNR dB ↑	SSIM ↑	LPIPS VGG ↓
3DGS	7.1	0.88	830.8	18.68	0.383	0.675	13.5	2.26	318.8	29.51	0.886	0.326
3D Convex	26.7	1.07	116.3	20.31	0.470	0.581	70.7	2.72	35.1	28.29	0.877	0.329
BBSplat	29.8	2.05	92.4	19.92	0.405	0.648	58.9	12.99	35.0	27.03	0.834	0.367
Nexels	63.5	5.11	54.7	21.88	0.542	0.454	113.1	8.23	31.6	27.84	0.865	0.307
2DGS	16.9	2.23	220.9	16.69	0.354	0.718	29.1	8.28	83.5	28.39	0.864	0.358
Triangle Spl.	13.8	1.03	337.2	21.73	0.520	0.518	24.8	1.74	195.8	28.78	0.886	0.312
MeshSplatting	26.3	3.76	71.7	19.27	0.378	0.648	41.9	5.54	42.5	22.16	0.706	0.528
Hexels V3-F	17.6	0.66	485.3	22.27	0.565	0.497	32.7	0.99	217.5	29.27	0.904	0.284

Table 10: **Matched model memory on Mip-NeRF360.** Fixed-population Hexels V3-F defines mean budgets of 16.42/46.56 MB outdoors/indoors. Hexels V3 improves PSNR by 0.54/0.49 dB relative to Triangle Splatting, respectively.

Method	DTU (15)						Tanks and Temples (10)					
	Train min ↓	VRAM GiB ↓	FPS ↑	PSNR dB ↑	SSIM ↑	LPIPS VGG ↓	Train min ↓	VRAM GiB ↓	FPS ↑	PSNR dB ↑	SSIM ↑	LPIPS VGG ↓
3DGS	5.1	0.57	823.3	29.45	0.895	0.320	7.8	1.31	561.6	23.07	0.781	0.352
3D Convex	15.2	0.71	214.8	28.19	0.886	0.309	28.5	1.25	118.8	22.19	0.755	0.380
BBSplat	14.7	0.74	182.8	28.82	0.891	0.285	24.5	2.55	99.1	20.87	0.692	0.451
Nexels	29.1	2.19	151.7	29.15	0.899	0.247	73.0	3.29	82.2	22.90	0.775	0.321
2DGS	9.4	0.97	359.4	26.61	0.860	0.367	12.7	2.39	186.3	21.63	0.749	0.385
Triangle Spl.	8.5	0.65	482.6	27.23	0.866	0.326	12.0	1.04	393.6	22.94	0.806	0.303
MeshSplatting	13.3	1.86	126.3	22.59	0.703	0.486	17.7	2.25	113.2	17.24	0.609	0.527
Hexels V3-F	8.5	0.58	792.5	29.94	0.899	0.294	14.6	0.71	371.1	23.76	0.824	0.282

Table 11: **Matched model memory on DTU and Tanks and Temples.** Fixed-population Hexels V3-F defines mean budgets of 8.83/47.34 MB. Hexels V3 improves PSNR by 2.71/0.82 dB relative to Triangle Splatting, respectively.

Table 12 lists the model-memory budgets for each scene.

Scene	V1 (MB)	V2 (MB)	V3 (MB)
Mip-NeRF360			
bicycle	4.342	6.296	14.111
bonsai	16.529	23.967	53.719
counter	12.461	18.069	40.499
flowers	3.068	4.448	9.970
garden	11.101	16.097	36.079
kitchen	19.309	27.999	62.755
room	9.010	13.065	29.283
stump	2.564	3.718	8.333
treehill	4.189	6.074	13.614
DTU NVS			
scan105	3.545	5.141	11.522
scan106	3.707	5.375	12.047
scan110	2.042	2.960	6.635
scan114	3.442	4.992	11.188
scan118	2.270	3.292	7.379
scan122	1.741	2.524	5.658
scan24	2.496	3.620	8.113
scan37	4.065	5.894	13.210
scan40	4.692	6.803	15.247
scan55	3.742	5.426	12.161
scan63	1.421	2.061	4.618
scan65	1.809	2.624	5.880
scan69	1.858	2.694	6.037
scan83	1.281	1.858	4.164
scan97	2.658	3.854	8.639
Tanks and Temples			
Barn	16.220	23.519	52.714
Caterpillar	15.024	21.785	48.829
Courthouse	39.629	57.461	128.793
Francis	8.545	12.391	27.773
Horse	5.384	7.807	17.500
Ignatius	12.424	18.015	40.380
Lighthouse	12.828	18.600	41.690
Meetingroom	9.868	14.309	32.072
Train	14.734	21.365	47.887
Truck	10.993	15.940	35.728

Table 12: **Model-memory budgets for all 34 scenes.** V1/V2/V3 require $80N$/$116N$/$260N$ bytes, where N is the SfM point count.

K.2 Fixed and Densified Hexels

Densifying V3 improves PSNR by 1.37 dB outdoors and 0.78 dB indoors on Mip-NeRF360 (Table 13), increasing deployment size from 5.6 to 108.8 MB and 15.6 to 43.0 MB, respectively. On DTU, densified V3 has 0.19 dB lower PSNR than fixed V3 but 0.026 lower LPIPS; on T&T it gains 0.76 dB (Table 14). Fixed models therefore offer a particularly compact option for DTU, while growth improves quality on the larger scenes.

Method	Mip-NeRF360 outdoor (5)							
	Train min ↓	VRAM GiB ↓	FPS ↑	PSNR dB ↑	SSIM ↑	LPIPS VGG ↓	Deploy. MB ↓	Prims. K
Hexels V1-F	17.6	0.60	503.4	21.98	0.557	0.501	4.27	63.2
Hexels V1-G	28.8	2.44	141.1	23.43	0.667	0.349	85.94	1278.6
Hexels V2-F	17.8	0.63	495.8	22.11	0.562	0.497	6.19	63.2
Hexels V2-G	29.3	3.84	125.8	23.39	0.663	0.358	112.12	1151.6
Hexels V3-F	17.6	0.66	485.3	22.27	0.565	0.497	5.65	63.2
Hexels V3-G	33.6	17.27	95.1	23.63	0.674	0.347	108.78	1254.9

Method	Mip-NeRF360 indoor (4)							
	Train min ↓	VRAM GiB ↓	FPS ↑	PSNR dB ↑	SSIM ↑	LPIPS VGG ↓	Deploy. MB ↓	Prims. K
Hexels V1-F	32.4	0.83	230.1	28.36	0.891	0.297	12.05	179.1
Hexels V1-G	43.1	1.32	114.9	29.31	0.903	0.270	36.23	539.4
Hexels V2-F	32.8	0.87	228.1	28.74	0.897	0.290	17.47	179.1
Hexels V2-G	41.7	1.40	121.8	29.27	0.904	0.271	46.45	476.9
Hexels V3-F	32.7	0.99	217.5	29.27	0.904	0.284	15.65	179.1
Hexels V3-G	42.4	9.81	107.3	30.05	0.912	0.261	42.96	495.2

Table 13: **Fixed and densified Hexels on Mip-NeRF360.** F/G denote fixed/grown populations.

Method	DTU (15)							
	Train min ↓	VRAM GiB ↓	FPS ↑	PSNR dB ↑	SSIM ↑	LPIPS VGG ↓	Deploy. MB ↓	Prims. K
Hexels V1-F	8.4	0.55	881.9	28.94	0.890	0.303	2.30	34.0
Hexels V1-G	10.8	0.61	448.2	29.31	0.897	0.278	6.56	97.5
Hexels V2-F	8.5	0.56	895.4	29.57	0.895	0.298	3.33	34.0
Hexels V2-G	10.3	0.64	527.2	29.14	0.891	0.282	8.90	91.1
Hexels V3-F	8.5	0.58	792.5	29.94	0.899	0.294	3.11	34.0
Hexels V3-G	10.6	1.21	487.3	29.75	0.902	0.268	8.85	100.4

Method	Tanks and Temples (10)							
	Train min ↓	VRAM GiB ↓	FPS ↑	PSNR dB ↑	SSIM ↑	LPIPS VGG ↓	Deploy. MB ↓	Prims. K
Hexels V1-F	13.3	0.55	444.1	23.05	0.811	0.295	12.27	182.1
Hexels V1-G	20.8	1.38	197.9	23.95	0.829	0.261	51.37	761.6
Hexels V2-F	14.4	0.60	393.2	23.49	0.819	0.287	17.77	182.1
Hexels V2-G	22.7	1.96	167.3	24.27	0.834	0.257	70.13	718.3
Hexels V3-F	14.6	0.71	371.1	23.76	0.824	0.282	15.94	182.1
Hexels V3-G	24.3	12.06	142.7	24.52	0.839	0.249	64.30	739.1

Table 14: **Fixed and densified Hexels on DTU and Tanks and Temples.** F/G denote fixed/grown populations.

K.3 DEPLOYMENT COMPRESSION

We evaluate lossless compression and appearance product quantization (PQ1024) on all six variants across 34 scenes. Lossless compression reorders attribute bytes and applies DEFLATE, preserving the FP32 parameters. PQ1024 replaces RGB appearance vectors with 1,024-entry codebooks: one per SH coefficient for V3, and corresponding codebooks for the spatial and directional gradients of V1/V2. Geometry and base color remain lossless. Parameters are decoded to FP32 before rendering, without retraining.

We use lossless compression for V1/V2 and PQ1024 for V3. V2 quantization reduces PSNR by 0.55–0.61 dB, whereas V3-grown PQ1024 changes mean PSNR/SSIM/LPIPS by -0.01674 dB/-0.000259/$+0.000849$ (Table 15). The largest dataset-mean PSNR loss for V3 grown is 0.0303 dB on Mip-NeRF360. Its mean deployment size is 43.87 MB, 60.09% below lossless compression; resident rendering memory is unchanged. The comparison tables use the selected decoded models for both quality and deployment size. Baselines retain their original deployment formats.

Encoding time includes codebook fitting; decoding measures cached reading and CPU reconstruction, followed by upload into an allocated model. These timings exclude viewer startup.

Variant	Codec	Encode s	Decode ms	Upload ms	FPS	ΔPSNR dB	ΔSSIM	ΔLPIPS VGG	Deploy. MB	Saved %
V1 fixed	**Lossless**	0.5	34.5	4.3	620.8	0.0000	0.00000	0.00000	6.67	24.2
V1 fixed	PQ1024	1.8	33.1	3.1	619.1	-0.0338	-0.00057	0.00175	4.97	43.3
V1 grown	**Lossless**	2.8	163.2	8.5	290.2	0.0000	0.00000	0.00000	34.90	24.4
V1 grown	PQ1024	4.0	152.3	8.1	289.3	-0.0325	-0.00058	0.00168	25.91	43.8
V2 fixed	**Lossless**	0.8	47.6	3.4	610.4	0.0000	0.00000	0.00000	9.66	21.8
V2 fixed	PQ1024	2.4	42.0	3.1	599.9	-0.6061	-0.00767	0.02537	5.38	55.9
V2 grown	**Lossless**	3.7	209.3	10.4	314.6	0.0000	0.00000	0.00000	46.50	21.9
V2 grown	PQ1024	5.4	186.3	12.2	309.0	-0.5450	-0.00777	0.02449	25.69	56.7
V3 fixed	Lossless	1.7	103.1	4.6	565.1	0.0000	0.00000	0.00000	21.59	18.8
V3 fixed	**PQ1024**	4.7	77.4	5.4	555.7	-0.0200	-0.00032	0.00085	8.73	66.5
V3 grown	Lossless	8.4	464.2	21.0	284.5	0.0000	0.00000	0.00000	109.92	18.9
V3 grown	**PQ1024**	11.7	363.1	22.8	283.5	-0.0167	-0.00026	0.00085	43.87	67.4

Table 15: **Deployment compression across 34 scenes.** Quality changes are relative to uncompressed models; bold codecs denote the selected formats.

L PER-SCENE NVS RESULTS

Tables 16, 17 and 18 report V3 densified on all nine Mip-NeRF360 scenes, fifteen DTU scans and ten T&T scenes.

Scene	Train min	VRAM peak GiB	FPS	PSNR dB	SSIM	LPIPS VGG	Deploy. MB	Primitives K
bicycle	32.4	18.55	88.8	24.00	0.693	0.331	112.06	1293.9
flowers	35.1	19.08	96.1	20.35	0.539	0.431	113.87	1312.7
garden	40.6	20.49	66.7	26.91	0.840	0.170	156.53	1808.7
stump	26.6	12.67	124.3	24.73	0.690	0.360	68.43	787.7
treehill	33.2	15.56	99.5	22.19	0.609	0.444	93.02	1071.7
room	31.4	7.42	140.9	29.82	0.900	0.322	36.04	414.5
counter	44.4	7.62	104.5	28.56	0.900	0.275	35.69	410.1
kitchen	51.0	9.42	86.5	30.49	0.917	0.175	41.16	475.3
bonsai	42.8	14.78	97.4	31.34	0.932	0.272	58.95	680.7
Mean	37.5	13.95	100.5	26.49	0.780	0.309	79.53	917.3

Table 16: **Per-scene V3 densified results on Mip-NeRF360.** The final row reports the mean across scenes.

Scene	Train min	VRAM peak GiB	FPS	PSNR dB	SSIM	LPIPS VGG	Deploy. MB	Primitives K
scan24	12.7	0.95	326.0	26.67	0.911	0.166	9.93	112.9
scan37	13.4	2.93	280.0	25.47	0.896	0.193	19.26	220.9
scan40	14.3	4.04	245.8	26.73	0.905	0.195	23.24	268.1
scan55	12.2	3.10	285.4	29.08	0.882	0.267	19.99	230.4
scan63	10.4	0.60	583.7	30.02	0.938	0.166	4.06	44.7
scan65	9.5	0.67	506.5	29.98	0.886	0.280	7.07	79.5
scan69	9.7	0.65	479.7	27.02	0.885	0.241	6.84	76.8
scan83	9.4	0.62	574.9	29.54	0.889	0.334	3.76	41.3
scan97	11.1	0.68	412.2	27.71	0.875	0.287	7.18	80.8
scan105	10.7	0.66	446.4	30.15	0.885	0.274	6.18	69.4
scan106	9.1	0.69	561.7	33.17	0.916	0.323	6.46	73.0
scan110	8.3	0.62	704.2	31.97	0.913	0.343	3.71	40.6
scan114	9.8	0.69	478.7	30.39	0.909	0.289	7.21	81.5
scan118	8.9	0.62	706.9	33.95	0.920	0.326	4.11	45.6
scan122	8.9	0.60	716.7	34.35	0.919	0.337	3.71	40.8
Mean	10.6	1.21	487.3	29.75	0.902	0.268	8.85	100.4

Table 17: **Per-scene V3 densified results on DTU.** The final row reports the mean across scenes.

Scene	Train min	VRAM peak GiB	FPS	PSNR dB	SSIM	LPIPS VGG	Deploy. MB	Primitives K
Train	23.5	10.11	144.2	21.43	0.794	0.260	49.37	568.2
Truck	28.8	21.09	99.0	24.22	0.846	0.212	106.43	1224.1
Francis	22.1	9.13	167.3	28.33	0.899	0.332	41.00	469.9
Horse	26.4	16.56	114.9	24.81	0.877	0.208	81.41	931.8
Lighthouse	23.7	7.28	165.0	23.15	0.848	0.264	35.54	407.7
Barn	21.5	8.39	181.8	28.35	0.867	0.215	40.60	467.8
Caterpillar	21.6	13.20	149.9	23.79	0.798	0.264	58.56	672.9
Courthouse	25.4	3.33	159.6	22.86	0.800	0.271	43.33	500.0
Ignatius	29.7	23.35	83.2	22.39	0.783	0.237	142.05	1636.3
Meetingroom	20.3	8.18	162.2	25.89	0.879	0.231	44.69	512.6
Mean	24.3	12.06	142.7	24.52	0.839	0.249	64.30	739.1

Table 18: **Per-scene V3 densified results on Tanks and Temples.** The final row reports the mean across scenes.

M ADDITIONAL QUALITATIVE COMPARISONS

Figures 11–13 compare V3 densified with native 2DGS, Triangle Splatting and MeshSplatting on three additional scenes per dataset. The enlarged regions show differences in object boundaries and fine texture.

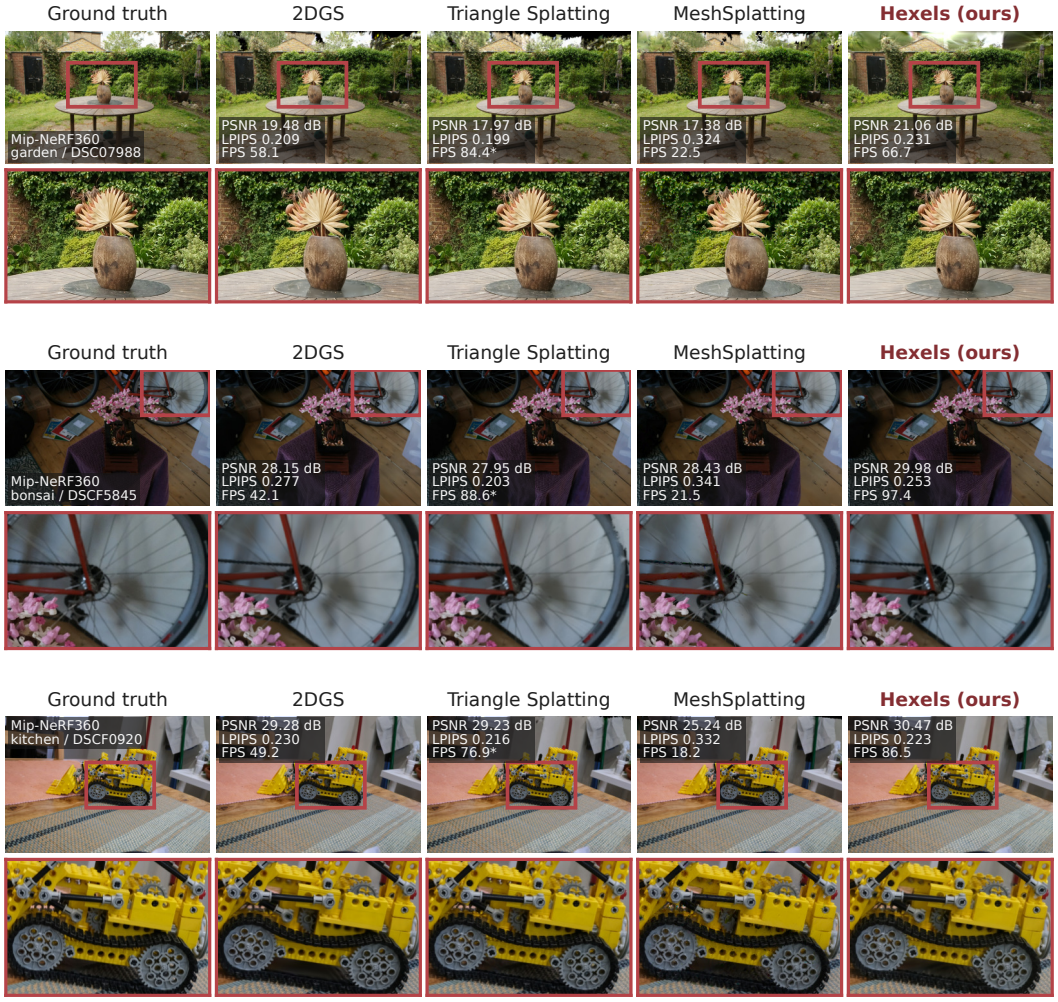


Figure 11: **Additional Mip-NeRF360 comparisons.** Hexels V3 densified and native baselines on garden, bonsai and kitchen, with shared zoom regions.

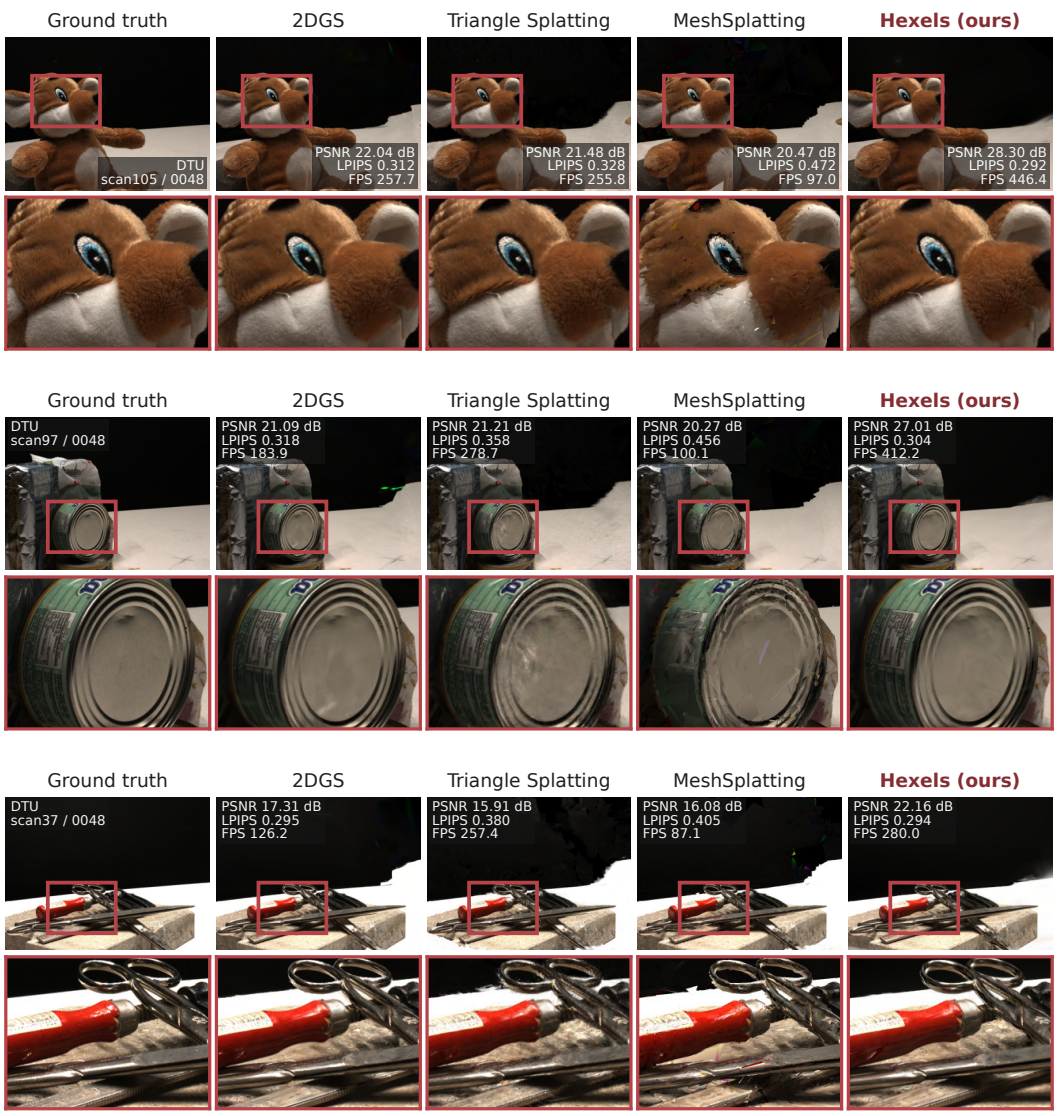


Figure 12: **Additional DTU comparisons.** Hexels V3 densified and native baselines on scans 105, 97 and 37. Red boxes mark enlarged details.

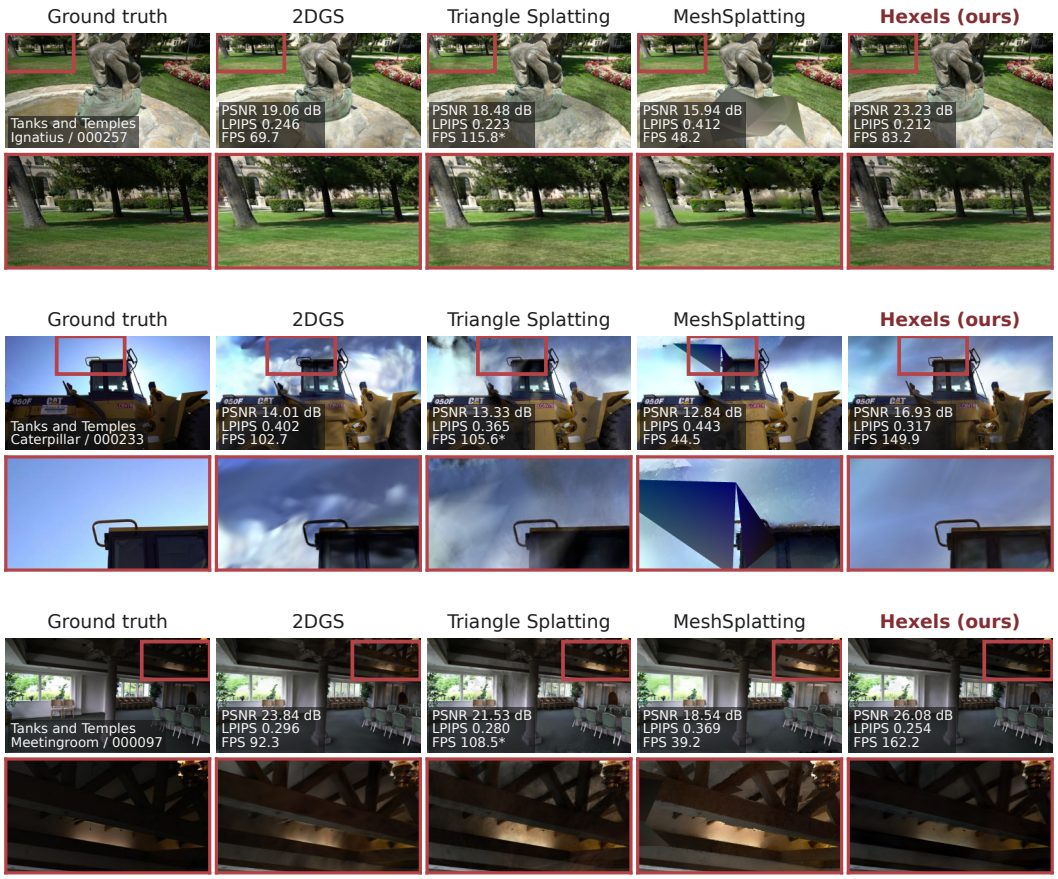


Figure 13: **Additional Tanks and Temples comparisons.** Hexels V3 densified and native baselines on Ignatius, Caterpillar and Meetingroom, with shared zoom regions.