

Deformable Triangle Splatting: Flexible Primitives for Real-Time Radiance Field Rendering

Oriol Jiménez-Ayguadé  and Antonio Agudo 

Institut de Robòtica i Informàtica Industrial, CSIC-UPC, Barcelona, Spain
{ojimenez,aagudo}@iri.upc.edu

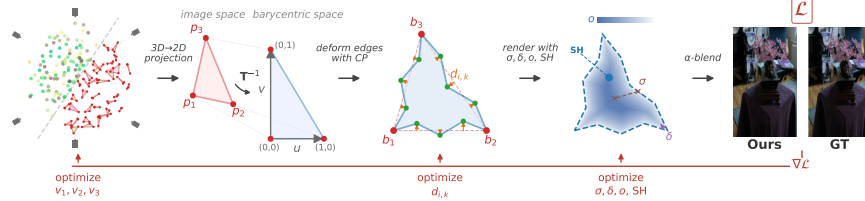


Fig. 1: Method overview. From left to right: A sparse SfM point cloud initializes the primitives; Each triangle is first projected from 3D to image space ($\mathbf{p}_i$), then mapped via $\mathbf{T}^{-1}$ to a canonical barycentric coordinate frame, ensuring view-independent deformations; K learnable Control Points (CP) per edge (green dots) displace the boundary by $d_{i,k}$ along edge normals (orange arrows), producing a deformable piecewise-linear contour; The deformed polygon is rendered with a smooth distance-based falloff controlled by σ , corner smoothness δ , per-primitive opacity o , and Spherical-Harmonic (SH) view-dependent color; Finally, primitives are composed via front-to-back α -blending; gradients of the loss $\mathcal{L}$ optimize vertex positions $\mathbf{v}_i$, displacements $d_{i,k}$, and rendering parameters ($\sigma, \delta, o, \text{SH}$).

Abstract. Recent radiance field methods represent scenes with 2D primitives that offer surface alignment and efficient rasterization, from Gaussian disks to triangles, yet all rely on convex boundaries: curved and concave structures demand excessive primitives. We introduce Deformable Triangle Splatting, which augments each triangle with K control points per edge, each parameterized by a single learnable scalar displacement that shifts the boundary inward or outward, enabling non-convex shape representation while preserving the three base vertices that define the 3D plane. To render these non-convex primitives differentiably, we design a rasterization pipeline in the triangle’s barycentric coordinate space, ensuring view-consistent rendering. A winding number test determines whether each pixel lies inside the deformed primitive, and a window function controlled by two learnable parameters, sharpness and corner smoothness, together with a per-primitive scalar opacity, produces the smooth opacity transition from interior to boundary. Validation is done in a variety of real-world scenes, outperforming recent works based on non-volumetric primitives in terms of visual quality and versatility while still achieving competitive rendering efficiency.

Project page: orioljim1.github.io/detrjs

Keywords: Differentiable Rendering · Radiance Fields · Non-Convex Primitives · Deformable Splatting.

1 Introduction

The field of novel view synthesis has been transformed by two major paradigms: Neural Radiance Fields (NeRF) [34] and 3D Gaussian Splatting (3DGS) [21]. While NeRF [34] established the potential of continuous volumetric fields for high-fidelity reconstruction, its reliance on expensive ray marching motivated acceleration techniques such as multi-resolution hash encodings [35]. 3DGS [21] subsequently advanced the field by employing explicit anisotropic Gaussians, enabling real-time rendering without sacrificing visual quality. In spite of these advances, a fundamental challenge remains: accurately representing complex scene geometry with a unified primitive. Volumetric primitives like Gaussians inherently struggle with sharp boundaries and flat surfaces due to their smooth fall-off. Other works have explored alternative primitives such as 2D Gaussian Splatting (2DGS) [17] which flattens primitives for better surface alignment. Meanwhile 3D Convex Splatting (3DCS) [15] and Universal Beta Splatting (UBS) [27] employ smooth convex hulls and Beta kernels, respectively, to capture hard edges and dense volumes. However, these methods rely on fixed or convex shapes, limiting their ability to model non-convex local geometry without excessive primitive proliferation. More recently, the most standard primitive of computer graphics and simulation was rescued in Triangle Splatting (TS) [14], where a soup of triangles was directly optimized, bridging differentiable rendering and traditional mesh pipelines. The use of triangle primitives is a key factor in the topic as GPUs feature dedicated hardware pipelines for ultra-efficient triangle rendering and processing. However, this representation still imposes a rigid and fixed topology based on three vertices, leading to inefficient over-tessellation when representing curved or non-convex structures.

Unlike previous work, our approach augments each triangle with K learnable control points per edge, producing a soup of non-uniform primitives capable of representing a wide variety of convex and non-convex shapes. In this sense, the proposed primitive can be interpreted as a versatile representation for 3D content, as multiple specialized primitives can be learned directly from data using the same underlying structure. To achieve that, each control point is associated with a scalar displacement that shifts the boundary inward or outward along the edge normal in the primitive’s intrinsic barycentric coordinate space. This allows a single primitive to represent non-convex shapes such as arrows, crescents, or any shape with concavities; without modifying the three base vertices that define its 3D plane. Moreover, our formulation based on barycentric coordinates ensures that all per-pixel deformation computations remain strictly view-independent, unlike several existing methods that rely on a dependent, as shown later.

To render these deformed primitives differentiably, we introduce three complementary mechanisms. First, a ray-casting winding number test classifies pixels as inside or outside the deformed polygon, handling non-convex and self-intersecting boundaries robustly. Second, a polynomial smooth minimum over the point-to-segment distances to all boundary edges produces a C^1 -continuous distance field whose corner rounding is governed by a learnable parameter. Third, a power-law window function controls an opacity falloff from the primitive’s

center to its boundary. Together, these ensure that gradients flow from the photometric loss through the opacity to the boundary control points. We provide an efficient CUDA implementation that is extensively evaluated on challenging scenes, outperforming recent works in terms of visual fidelity and versatility while still obtaining competitive rendering speed.

2 Related Work

We next review the most related work dealing with neural radiance fields and primitive-based differentiable rendering.

Neural Radiance Fields. In recent years, NeRF [34] has emerged as a leading approach for image-based 3D reconstruction and novel view synthesis. This work introduced a continuous implicit scene representation parameterized by multilayer perceptrons, demonstrating high-fidelity view synthesis from sparse input images where the camera locations are obtained by Structure-from-Motion (SfM) [44]. Since then, NeRF-based methods have progressed rapidly [35, 36, 49, 51, 52], improving robustness through anti-aliasing techniques [2, 3], modeling dynamic scenes [10, 38, 39, 43], applying piecewise linear models for volume density [48], quantifying the uncertainty [33, 45] and considering few-shot generalization [4, 8]. A major limitation is their high computational cost, resulting in slow training and rendering. To mitigate that, subsequent work has explored multi-resolution grid encodings [5, 25, 35], and even baking strategies for real-time rendering [7, 41]. Despite these developments, existing methods remain largely reliant on implicit field queries rather than explicit rasterization pipelines.

Differentiable Rendering based on primitives. To bridge the gap between visual fidelity and rendering speed, recent approaches have shifted towards explicit primitive-based representations. While points [20], meshes [29], and voxels [11], had been used, the proposal of 3DGS [21] revolutionized this direction by representing scenes with millions of anisotropic 3D Gaussians projected via a tile-based rasterizer, enabling real-time rendering at 1080p and beyond. Based on that work, other approaches focus on enhancing shape representation [12], improving memory efficiency [26, 30], proposing better densification and pruning strategies [22] or handling dynamic models [1, 9, 16, 50]. Moreover, another line of approaches extended this with more flexible kernel functions [6, 13, 18, 28], linear [31] and quadratic [53] primitives as well as texture billboards [42, 46]. 3DCS [15] utilized smooth convex hulls to capture hard edges and dense volumes. However, the volumetric nature of 3D Gaussians poses inherent limitations for surface modeling: their smooth falloff produces *fuzzy* geometry lacking well-defined surfaces, problematic for downstream tasks requiring accurate normals such as relighting or material editing. 2DGS [17] addressed this by flattening 3D Gaussians into 2D oriented disks, enforcing stronger surface alignment. TS [14] exploited surface triangular primitives, leveraging its ubiquity in graphics to ensure mesh compatibility and demonstrating state-of-the-art results among non-volumetric methods. However, this method still produces tessellation artifacts in some challenging geometries as well as low sharpness.

We overcome most of the limitations of previous methods with an approach that introduces deformable triangular primitives. To this end, our approach retains the triangle as the base primitive for its geometric interpretability, but augments it with learnable boundary deformations to increase the versatility and adaptability of our primitives to capture a wide spectrum of real-world scenarios. Unlike 2DGS [17], 3DCS [15], or TS [14], which rely on fixed shapes (disks, convex hulls, or rigid triangles, respectively), our primitive is adaptable according to the data, resulting in a soup of non-uniform primitives that correctly represents both convex and non-convex shapes by means of a unified formulation while reducing the number of primitives required to cover complex geometries. Moreover, we propose a view-independent rendering pipeline based on projecting primitives onto barycentric coordinates. We are not aware of any other work jointly offering all these characteristics.

3 Our Method

We present **DEformable TRIangle Splatting** (DETRIS), a differentiable rendering method that augments triangle-based primitives with learnable boundary deformations. Conceptually, our approach follows the deformable template paradigm introduced by [47], where each primitive begins as a reference triangle whose boundary deforms under photometric supervision, rather than relying on internal energy terms based on classical deformable models. Our approach enables the representation of complex, non-convex shapes while maintaining the efficiency of explicit surface-based rendering [14, 17, 21, 28]. Figure 1 provides an overview of our approach. In this section, we first describe our deformable primitive model and its differentiable rasterization pipeline, followed by our adaptive densification strategy and the optimization procedure.

3.1 Differentiable Rendering of Deformable Primitives

Deformable K-Point Primitives. Each primitive is defined by three freely optimizable 3D vertices $\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3\} \subset \mathbb{R}^3$ that determine a triangular base, as in TS [14]. After perspective projection, these vertices map to 2D image-space positions $\mathbf{p}_i \in \mathbb{R}^2$, which in turn define a *barycentric coordinate system* $\mathbf{b} = (u, v)$ with $\mathbf{p}_1 \mapsto (0, 0)$, $\mathbf{p}_2 \mapsto (1, 0)$, $\mathbf{p}_3 \mapsto (0, 1)$. The forward affine map from barycentric to image coordinates is $\mathbf{p} = \mathbf{p}_1 + \mathbf{T}\mathbf{b}$, where $\mathbf{T} = [\mathbf{p}_2 - \mathbf{p}_1, \mathbf{p}_3 - \mathbf{p}_1] \in \mathbb{R}^{2 \times 2}$. During rasterization, each pixel location $\mathbf{p}$ is mapped to barycentric coordinates via the per-primitive inverse $\mathbf{b} = \mathbf{T}^{-1}(\mathbf{p} - \mathbf{p}_1)$, which we precompute once per triangle before the per-pixel render pass.

Every boundary deformation is performed in this intrinsic barycentric space, making it *view-independent*. Each of the three edges is subdivided by K control points. For edge $i \in \{1, 2, 3\}$ connecting barycentric vertices $\mathbf{b}_i^{\text{start}}$ and $\mathbf{b}_i^{\text{end}}$, the k -th control point with $k \in \{1, \dots, K\}$ is placed at:

$$\mathbf{b}_{i,k} = \mathbf{b}_i^{\text{start}} + \frac{k}{K+1} \mathbf{e}_i + d_{i,k} \hat{\mathbf{n}}_i, \quad (1)$$

where $\mathbf{e}_i = \mathbf{b}_i^{\text{end}} - \mathbf{b}_i^{\text{start}}$ is the barycentric edge vector, $d_{i,k} \in \mathbb{R}$ is a learnable scalar displacement, and $\hat{\mathbf{n}}_i$ is the unit outward normal to edge i in barycentric space, computed as the normalized perpendicular to $\mathbf{e}_i$ oriented away from the opposite vertex. The original triangle vertices are not displaced in this step, they are instead freely optimized on the 3D space. This produces a piecewise-linear boundary with a total of $M = 3(K+1)$ vertices. A positive $d_{i,k}$ expands the boundary outward (beyond the original triangle edge), while a negative value contracts it inward, allowing concavities. Since the displacements are in barycentric units, they are scale-invariant with respect to the triangle’s image-space size.

Inside/Outside Classification via Winding Number. For convex shapes, a simple half-plane intersection test suffices for rasterization [32]. However, once CPs deform the boundary into non-convex shapes, this test is no longer valid. To handle this, we employ a non-zero winding rule [19], implemented via ray casting. For a query point $\mathbf{x}$ in barycentric coordinates, we cast a horizontal ray and count signed intersections with the deformed polygon edges as:

$$\omega(\mathbf{x}) = \sum_{m=1}^M \psi_m(\mathbf{x}), \quad \psi_m = \begin{cases} +1 & \text{if } y_m \leq y_{\mathbf{x}} < y_{m+1} \wedge (\mathbf{p}_m - \mathbf{x}) \times (\mathbf{p}_{m+1} - \mathbf{x}) > 0 \\ -1 & \text{if } y_{m+1} \leq y_{\mathbf{x}} < y_m \wedge (\mathbf{p}_m - \mathbf{x}) \times (\mathbf{p}_{m+1} - \mathbf{x}) < 0 \\ 0 & \text{otherwise} \end{cases}$$

where $\mathbf{p}_m$ are the vertices of the deformed polygon in barycentric space, y_m their vertical coordinates, and $\times$ denotes a 2D cross product. A pixel is classified as *inside* if $\omega(\mathbf{x}) \neq 0$. For simple (non-self-intersecting) polygons, this is equivalent to the even-odd ray casting rule; for self-intersecting edges, it correctly handles overlapping regions. Pixels with $\omega(\mathbf{x}) = 0$ are skipped, contributing no opacity.

Distance Field with Polynomial Smooth Minimum. For each interior pixel, we compute an Euclidean distance from $\mathbf{x}$ to every boundary segment $(\mathbf{p}_m, \mathbf{p}_{m+1})$ as:

$$D_m(\mathbf{x}) = \left\| \mathbf{x} - \left[\mathbf{p}_m + \text{clamp} \left(\frac{(\mathbf{x} - \mathbf{p}_m) \cdot \mathbf{e}_m}{\|\mathbf{e}_m\|^2}, 0, 1 \right) \mathbf{e}_m \right] \right\|, \quad (2)$$

where $\mathbf{e}_m = \mathbf{p}_{m+1} - \mathbf{p}_m$. This is the standard point-to-line-segment distance, computed in barycentric coordinates.

3DCS [15] introduced a smoothness parameter δ that controls corner rounding via a LogSumExp approximation to the max function over half-plane distances. We adopt the same conceptual role for δ —controlling how sharp or rounded the corners of the primitive appear—but replace the LogSumExp formulation with a **polynomial smooth minimum** [40] over segment distances. This choice has three advantages: 1) the polynomial smooth minimum is C^1 -continuous and produces bounded, localized corner rounding (the smoothing only affects regions within distance g of a corner), whereas LogSumExp produces global smoothing [40]; 2) it requires only additions, multiplications, and a **max**—no transcendental functions—making it substantially cheaper to evaluate

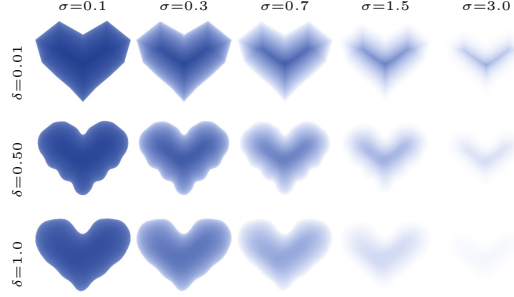


Fig. 2: σ and δ interpretation. A deformable heart primitive with $K=3$ is rendered with increasing falloff exponent σ and corner smoothness δ . Higher σ concentrates opacity near the medial axis; higher δ rounds the sharp corners into smooth curves.

per pixel on GPU than the $\exp/\log$ calls required by LogSumExp; and 3) it operates directly on the minimum of unsigned distances rather than the maximum of signed half-plane distances, which is more natural for our piecewise-linear boundary with M parts. Specifically, we apply a sequential pairwise fold as:

$$\text{smin}(a, b; g) = \min(a, b) - \frac{g}{4} h^2, \quad \text{where } h = \max\left(1 - \frac{|b - a|}{g}, 0\right), \quad (3)$$

with smoothing radius $g = \delta r_{\text{bary}}$, where δ is a learnable per-primitive parameter (activated via softplus to ensure positivity) and $r_{\text{bary}} = \frac{2-\sqrt{2}}{2} \approx 0.293$ is the inradius of the unit barycentric triangle. The final smoothed distance is obtained by sequentially folding all segment distances:

$$S_1 = D_1, \quad S_m = \text{smin}(S_m, D_m; g), \quad m = \{2, \dots, M\}, \quad S(\mathbf{x}) = S_M. \quad (4)$$

When δ is small, $g \rightarrow 0$ and the smooth minimum reduces to the hard minimum, yielding sharp corners. In contrast, for large values, corners become visibly rounded. Figure 2 visualizes the effect of δ and σ on a rendered primitive.

Power-Law Window Function. The smoothed distance $S(\mathbf{x})$ is converted to opacity via a power-law window, following TS [14]. Unlike the sigmoid-based indicator of 3DCS [15], whose tail spreads indefinitely as σ decreases, causing every primitive to contribute opacity to every pixel, the power-law window is exactly zero outside the primitive boundary. The key adaptation for our deformable boundary is the normalization radius R : TS [14] uses the incenter distance of the rigid triangle, which is no longer valid when the boundary deforms. We instead compute R as the *polygon inradius* of the deformed boundary via the shoelace formula [37]:

$$\phi(\mathbf{x}) = \min\left(1, \frac{S(\mathbf{x})}{R}\right), \quad R = \frac{2A}{P}, \quad (5)$$

where $A = \frac{1}{2} |\sum_m (\mathbf{p}_m \times \mathbf{p}_{m+1})|$ is the polygon area and $P = \sum_m \|\mathbf{p}_{m+1} - \mathbf{p}_m\|$ its perimeter, both in barycentric coordinates. For a convex polygon, R equals the

true inradius; for a non-convex one it is an upper bound, so $\phi \leq 1$ everywhere in the interior. For an undeformed triangle, R recovers r_{bary} , and the computation requires only $\mathcal{O}(M)$ operations during preprocessing. The per-pixel opacity is then:

$$\alpha(\mathbf{x}) = \min(0.99, o \cdot \phi(\mathbf{x})^\sigma), \quad (6)$$

where $o \in [0, 1]$ is a per-primitive learnable base opacity and $\sigma > 0$ a learnable falloff exponent (activated via an exponential function): large σ concentrates opacity near the center, while small σ yields a nearly uniform profile.

Why Barycentric Space? A natural alternative, which we initially implemented and evaluated, is to define displacements and distance computations directly in image space. However, this formulation has a fundamental flaw: **the learned primitive shape becomes view-dependent**. Because the same 3D triangle projects differently under each camera, all per-pixel rendering computations (segment distances, smooth-minimum field, opacity falloff) are evaluated in pixel units that change with viewpoint, even when the displacements are fixed. As a result, a primitive that appears sharp from one viewpoint may look excessively rounded from another, and multi-view supervision produces conflicting gradients for the same displacement parameters. By performing all deformations in the canonical barycentric frame, we decouple intrinsic shape from image projection: all boundary tests are evaluated in a fixed reference frame, and both $d_{i,k}$ and δ carry exactly the same geometric meaning in every training view, eliminating the multi-view gradient conflict. Figure 3-right shows that a deformed arrow primitive maintains a consistent shape across viewpoints under our formulation, while the image-space version visibly distorts.

View-dependence of the image-space window. Even the base window function of TS [14], $\phi(\mathbf{x}) = (\hat{d}/\hat{d}_{\text{inc}})^\sigma$ with distances in pixel coordinates, is affected. Although this ratio cancels uniform scaling, perspective projection is *projective*, not affine: vertices at different depths undergo non-uniform foreshortening, so $\hat{d}/\hat{d}_{\text{inc}}$ is not preserved across viewpoints. Figure 3-left confirms this: rendering an undeformed triangle under progressive rotation, the discrepancy grows to $\max |\Delta| = 76$ at 45° and 181 at 60° , concentrated on the most foreshortened edges, while the edge aligned with the rotation axis shows near-zero error, consistent with the analysis. Our barycentric formulation is exactly view-independent by construction.

3.2 Adaptive Pruning and Densification

We follow the pruning and densification strategy of TS [14], adapted for our novel deformable primitives. Next, we introduce the most important details.

Pruning. During rasterization, we track the maximum blending weight $T \cdot \alpha$ (importance score) for each primitive, with T the transmittance. We prune primitives whose importance score falls below a threshold τ_{prune} , as well as those not rendered in at least two training views with more than one pixel of coverage.

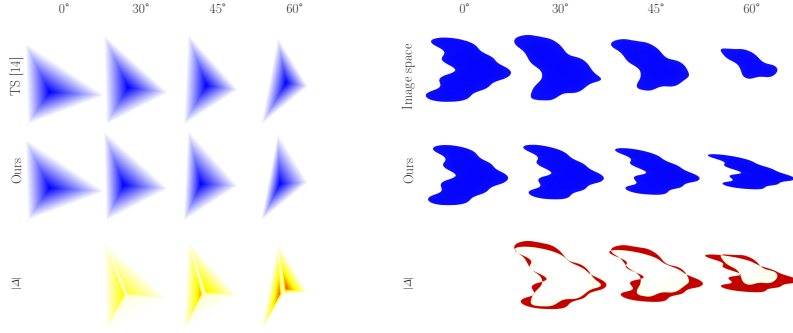


Fig. 3: Image-space vs. barycentric formulation under viewpoint change. The same primitive is rendered from three X-axis rotations; both formulations start from a pixel-identical frontal view. **Left:** Window function ϕ^σ effect. The image-space ratio $\hat{d}/\hat{d}_{\text{inc}}$ [14] diverges from our barycentric window as perspective foreshortening increases (max $|\Delta| = 181$ at 60°). **Right:** Boundary deformation δ : a deformed arrow with matched parameters. The image-space formulation distorts under oblique views while our barycentric shape is view-independent. Bottom rows show per-pixel absolute difference with the smallest and largest errors represented in white and red, respectively.

Densification. We sample primitives for densification from a probability distribution based on learned opacity and inverse sharpness ($1/\sigma$), following the spirit of probabilistic densification [22]. This preferentially targets solid primitives for splitting. In particular, for primitives larger than an image-space threshold τ_{split} , we apply **midpoint subdivision**, splitting each triangle into four children by connecting edge midpoints. A key challenge is preserving the deformed boundary across this topological change. We **interpolate the parent’s control point displacements** onto the child edges where each parent edge’s displacement profile is sampled at the appropriate half to produce the child’s control points. Crucially, child edges that traverse in the opposite direction to the parent edge have their displacement sequences **reversed**, ensuring geometric continuity and preventing visual artifacts. For primitives smaller than τ_{split} , we **clone** the primitive and add random in-plane noise.

3.3 Optimization

The training loss combines photometric l_1 and D-SSIM terms [21], an opacity regularizer $\mathcal{L}_{\text{op}} = \overline{|\text{sigmoid}(\hat{o})|}$ encouraging transparency, and distortion $\mathcal{L}_d$ and normal consistency $\mathcal{L}_n$ losses [17]. During the densification phase (iterations $< T_{\text{dense}}$), we additionally apply a size regularizer $\mathcal{L}_s$ that penalizes the inverse square root of the triangle area, discouraging degenerate (collapsed) primitives [14]. A curvature term $\mathcal{L}_{\text{curv}} = \|\mathbf{d}\|^2$ penalizes large control point displacements, encouraging smooth boundaries and preventing overfitting. Our loss is:

$$\mathcal{L} = (1-\lambda_1)\mathcal{L}_1 + \lambda_1\mathcal{L}_{\text{D-SSIM}} + \lambda_2\mathcal{L}_{\text{op}} + \lambda_3\mathcal{L}_d + \lambda_4\mathcal{L}_n + \lambda_5\mathcal{L}_s + \lambda_6\mathcal{L}_{\text{curv}}, \quad (7)$$

where λ_{1-6} represent the weight coefficients.

Gradient Management and Initialization. Under joint optimization, vertex position gradients dominate the control point gradients because they directly affect projection, tile assignment, and barycentric coordinates. To mitigate this, we delay the control point learning rate to zero for the first $T_{\text{delay}} = 3000$ iterations, allowing vertices to settle into reasonable positions before enabling boundary deformation. After this warm-up, control points train jointly with vertices using the standard optimizer, a simple delay that is sufficient for effective control point utilization, as validated later. For initialization, starting from a sparse SfM point cloud [44], we create one triangle per 3D point. Let $\mathbf{q} \in \mathbb{R}^3$ be a point and r the distance to its nearest neighbor. Three vertices are sampled from the unit sphere, scaled by r , randomly rotated, and offset by $\mathbf{q}$. All control point displacements $d_{i,k}$ are set to zero, so primitives begin as standard triangles.

4 Experimental Evaluation

In this section, we present our experimental results on Mip-NeRF 360 [2] and Tanks & Temples [24] datasets. We provide both quantitative and qualitative evaluation and compare our approach against state-of-the-art solutions, including the implicit methods Mip-NeRF 360 [2], Zip-NeRF [3] and Instant-NGP [35]; volumetric primitives such as 3DGS [21], 3DGS-MCMC [22], Deformable Beta Splatting [28] and 3D Convex Splatting [15]; as well as non-volumetric primitives such as BBSplat [46], 2DGS [17] and Triangle Splatting [14]. In our case, we train for 30k iterations with $K=3$ control points per edge until otherwise stated, using an Adam optimizer [23]. To make a fair comparison, the maximum total number of primitives and the rest of hyperparameters are the same as in [14], the only additions are the learning rate for CPs, δ , and λ_6 . which can all be seen in the supplementary. For quantitative analysis, the metrics LPIPS, PSNR, and SSIM are reported [34].

4.1 Control Points Deformation Analysis

First of all, we evaluate the effect of the K control points in our approach. When this value is increased, it provides greater boundary flexibility but also enhances per-primitive parameter count and, therefore, the corresponding rasterization cost (more segments means additional distance tests per pixel). To validate that, Table 1 reports quality metrics together with training cost, rendering speed, and model size as a function of K on the Bonsai and Garden scenes included in Mip-NeRF 360 [2]. In general, performance degrades as K decreases, but both training time and model size grow approximately linearly with K . As the results always remain within reasonable bounds, it is not necessary to adjust this parameter precisely. In practice, we set $K=3$ in our experiments, as this value provides a good trade off between visual quality and rendering speed.

Table 1: Quantitative results in terms of K per edge on Bonsai (left) and Garden scenes (right) [2]. $K=0$ disables control-point deformation (our rasterizer only), isolating the gain due to the learned deformations.

Bonsai (Indoor)							Garden (Outdoor)						
K	PSNR $\uparrow$	SSIM $\uparrow$	LPIPS $\downarrow$	Time	FPS $\uparrow$	Size	K	PSNR $\uparrow$	SSIM $\uparrow$	LPIPS $\downarrow$	Time	FPS $\uparrow$	Size
0	31.58	0.944	0.163	43 min	63	824 MB	0	27.08	0.860	0.107	56 min	49	1384 MB
1	32.30	0.948	0.165	45 min	65	816 MB	1	27.35	0.867	0.101	52 min	46	1460 MB
2	32.39	0.948	0.163	51 min	55	832 MB	2	27.40	0.868	0.098	61 min	38	1508 MB
3	32.44	0.948	0.162	59 min	47	863 MB	3	27.41	0.868	0.098	71 min	33	1542 MB
4	32.51	0.948	0.162	68 min	40	881 MB	4	27.42	0.868	0.098	83 min	27	1577 MB
5	32.52	0.948	0.159	79 min	33	911 MB	5	27.42	0.868	0.098	96 min	23	1605 MB

4.2 Novel View Synthesis Results

We now report our results in a quantitative manner on Table 2. As can be seen, our approach obtains competitive solutions by achieving among non-volumetric primitives the best LPIPS on all dataset splits, the best indoor PSNR and SSIM on Mip-NeRF 360 [2], as well as the best T&T [24] LPIPS. It is worth noting that our method outperforms TS [14] on all average metrics across the three dataset splits. On a per-scene basis, we only fall behind on two outdoor scenes (Stump and Treehill), while matching or exceeding TS [14] on the remaining nine (see the supplementary material for full per-scene results). The consistent LPIPS improvement across *all* outdoor scenes (0.206 vs. 0.217) confirms that the perceptual quality of our renders is higher even where PSNR is slightly lower.

Figure 4 compares rendered test views on four scenes spanning indoor and outdoor settings. Our deformable primitives better resolve fine spoke geometry (Bicycle), sharper object edges (Counter), and crisper foliage (Stump, Treehill), whereas TS [14] exhibits tessellation artifacts and 3DGS [21] produces soft edges due to its Gaussian falloff. These gains are consistent with the per-scene LPIPS improvements provided in the supplementary.

Table 2: Quantitative comparison on Mip-NeRF 360 [2] and Tanks & Temples [24] datasets. The table includes implicit methods [2, 3, 35], those based on volumetric [15, 21, 22, 28] and non-volumetric [14, 17, 46] primitives. Average columns report Mip-NeRF 360 [2] for all scenes. Train (in minutes) and Mem (in MB) are measured on the same benchmark. **Bold** indicates best among non-volumetric methods. All results reported in [14]; $\dagger$ results reported in [28]; * measured on an RTX 5090.

Method	Outdoor Mip-NeRF 360			Indoor Mip-NeRF 360			Average Mip-NeRF 360				Tanks & Temples			
	LPIPS $\downarrow$	PSNR $\uparrow$	SSIM $\uparrow$	LPIPS $\downarrow$	PSNR $\uparrow$	SSIM $\uparrow$	LPIPS $\downarrow$	FPS $\uparrow$	Train $\downarrow$	Mem $\downarrow$	LPIPS $\downarrow$	PSNR $\uparrow$	SSIM $\uparrow$	FPS $\uparrow$
Implicit														
Instant-NGP [35]	—	—	—	—	—	—	0.331	9.4	—	—	0.305	21.92	0.745	14.4
Mip-NeRF 360 [2]	0.283	24.47	0.691	0.179	31.72	0.917	0.237	0.06	—	—	0.257	22.22	0.759	0.14
Zip-NeRF [3]	0.207	25.58	0.750	0.167	32.25	0.926	0.189	0.18	300	569	—	—	—	—
Vol.														
3DGS [21]	0.234	24.64	0.731	0.189	30.41	0.920	0.214	134	42	734	0.183	23.14	0.841	154
3DGS-MCMC [22] $\dagger$	0.210	25.51	0.760	0.208	31.08	0.917	0.210	82	—	—	0.190	24.29	0.860	129
DBS [28]	0.246	25.10	0.751	0.220	32.29	0.936	0.234	123	—	—	0.140	24.85	0.870	150
3DCS [15]	0.238	24.07	0.700	0.166	31.33	0.927	0.207	25	87	666	0.156	23.94	0.851	33
Non-Vol.														
BBSplat [46]	0.281	23.55	0.669	0.178	30.62	0.921	0.236	25	96	175	0.172	25.12	0.868	66
2DGS [17]	0.246	24.34	0.717	0.195	30.40	0.916	0.252	64	29	484	0.212	23.13	0.831	122
Triangle Splatting [14]	<u>0.217</u>	24.27	<u>0.722</u>	<u>0.160</u>	<u>30.80</u>	<u>0.928</u>	<u>0.191</u>	97	39/25*	795/813*	<u>0.143</u>	23.14	0.857	165
DETRIS (Ours)	0.206	<u>24.28</u>	0.725	0.155	31.13	0.930	0.183	43	55*	946*	0.133	<u>23.31</u>	<u>0.860</u>	79

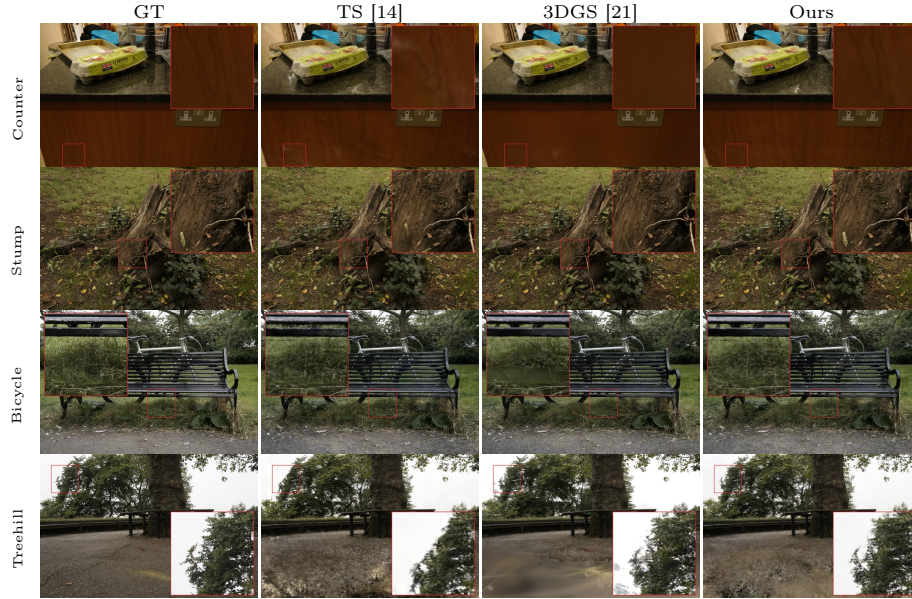


Fig. 4: Qualitative comparison on real scenes. Each row shows a test view from a different scene. Each column shows the ground truth, TS [14], 3DGS [21], and ours.

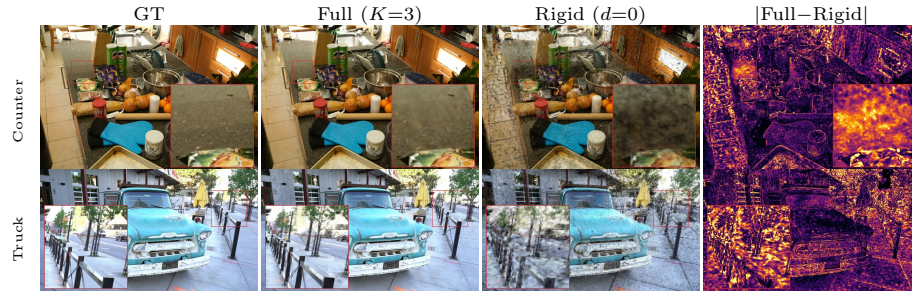


Fig. 5: Deformation contribution maps. For each scene we render the trained model with learned control-point displacements active (*Full*) and with all displacements and corner smoothness δ zeroed (*Rigid*, $d=0$), keeping all other parameters unchanged. The rightmost column shows the per-pixel absolute difference $|Full-Rigid|$ (inferno colormap, percentile-normalized), isolating the visual contribution of the learned boundary deformations from vertex positions, opacities, and colors. Red insets highlight the patch of highest deformation activity. The effect concentrates at fine boundary structures—thin object edges in Counter (top), cabin edges and wheel wells in Truck (bottom)—the same regions that drive our LPIPS improvement over TS [14].

To understand *where* the learned deformations contribute, Figure 5 compares each model rendered with and without control-point displacements (see caption for more details). The deformation signal concentrates at fine boundary struc-

tures, such as thin object edges in Counter and cabin edges and wheel wells in Truck, precisely where LPIPS improves most over TS [14]. Quantitatively, zeroing the control points increases the mean absolute error vs. ground truth by +0.052 on Bonsai and +0.029 on Bicycle (shown in the supplementary), confirming that the learned deformations reduce reconstruction error at boundaries.

Table 2 also reports rendering speed, training time, and model size. The additional per-pixel distance computations for M boundary segments reduce FPS compared to TS [14], though our throughput remains comparable to 3DCS [15] and BBSplat [46]; hierarchical culling or approximate distance fields could close this gap. Training takes roughly $2\times$ longer than TS [14] on the same hardware (NVIDIA RTX 5090), a trade-off we consider justified by the consistent quality gains. Regarding model size, each deformable primitive carries $3K+1=10$ extra floats (9 CP displacements + 1 corner smoothness δ), a +17% per-primitive overhead. Since the optimizer converges to a comparable primitive count, the net storage increase comes almost entirely from these extra parameters rather than primitive count.

4.3 Shape Expressiveness

To illustrate the representational advantage of our deformable primitives, Fig. 6 compares our method against TS [14] and 3DGS [21] on synthetic 2D shape-fitting tasks. Each method optimizes from the same initial configuration to match a target shape. With a single primitive, our deformable triangles closely approximate circles, rectangles, Gaussians, and non-convex shapes such as a 3-clover, arrow, Pac-Man, and heart. In contrast, TS [14]’s rigid boundary limits it to crude triangular approximations even with three primitives ($3\times$ more parameters), while 3DGS [21] fundamentally cannot produce sharp edges even with eight ($8\times$), yielding only soft blobs. Our method handles both convexity and concavity within a single deformable primitive.

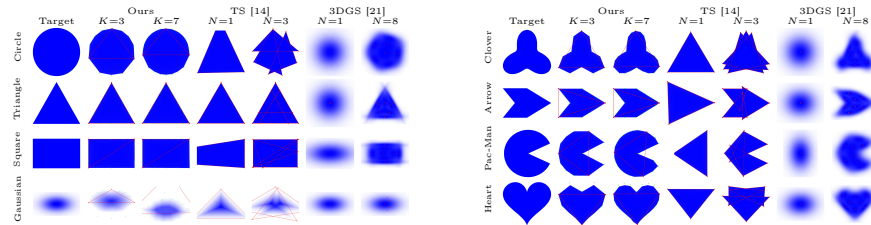


Fig. 6: Single-primitive shape fitting. In both sides, the same information is displayed. Each row represents a target shape and columns compare our method ($K=3$ and $K=7$ control points per edge) with TS [14] and 3DGS [21]. N denotes the number of primitives, being $N=1$ in our case. Red dots and lines show triangle vertices and edges; green dots mark the learned control points along edges; for 3DGS [21], red dots indicate Gaussian centers. **Left:** Convex shapes. **Right:** Non-convex shapes.

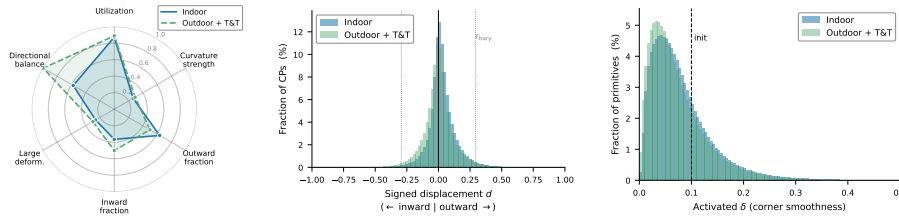


Fig. 7: Learned parameter analysis (Indoor [2] vs. Outdoor [2] + T&T [24]). **Left:** Radar profile of six CP application metrics, each normalized to $[0, 1]$. **Middle:** Signed CP displacement histogram ($d < 0$: inward, $d > 0$: outward); dashed lines mark $\pm r_{\text{bary}}$. Indoor scenes show a clear outward skew; outdoor ones are nearly symmetric. **Right:** Distribution of the corner-smoothness parameter δ (softplus-activated); the dashed line marks the initialization ($\delta=0.1$). Most primitives converge to sharper corners.

4.4 Ablation Study

To validate our design, we first analyze learned control-point statistics, then ablate key components.

Control-point utilization. To verify that the learned control points are not degenerate, we analyze the displacement statistics across all 323M control points from the 11 evaluated scenes at convergence. Each displacement d is a signed scalar in barycentric coordinates: positive values push the boundary outward (away from the opposite vertex), negative ones pull it inward. We normalize by the inradius r_{bary} of the unit barycentric triangle; a displacement of r_{bary} moves a control point from the edge midpoint to the centroid. Across all scenes, 88.5% of control points carry meaningful deformation ($|d| > 0.01$), with a mean magnitude of $|d| = 0.083$ (28% of r_{bary}); 29.0% of all control points undergo large deformations ($|d| > 0.1$). Figure 7 compares indoor and outdoor scene categories. Indoor scenes exhibit a strong outward bias (63.5% outward vs. 36.5% inward among active CPs), consistent with primitives expanding to cover smooth walls and flat surfaces. Outdoor [2] and T&T [24] scenes are nearly perfectly balanced (50%/50%), reflecting the need for both convex and concave boundary adjustments to capture complex vegetation and fine structures.

Corner smoothness. The learned δ has a global mean of 0.075 (median 0.061), with the majority of primitives falling below the 0.1 initialization (see Figure 7-right), indicating that the optimizer favors crisper corners while keeping the parameter well-distributed rather than collapsing it to zero.

Component ablation. Table 3 evaluates three key components averaged over all scenes. Among the three factors, edge deformation has the largest impact: removing it entirely (*w/o Def.*) produces the biggest drop in PSNR (-0.09 dB) and the largest LPIPS increase ($+0.006$), confirming that the learnable control points are the primary driver of reconstruction improvement (baseline wins on 9 of 11 scenes). Disabling the curvature regularizer *w/o* $\mathcal{L}_{\text{curv}}$, ($\lambda_6=0$ in Eq. (7)) has a minor effect on average metrics, with the baseline winning on 7 of 11 scenes. However, analyzing the learned control points reveals that without $\mathcal{L}_{\text{curv}}$, the fraction of CPs with large deformations ($|d| > 0.1$) rises from 29.0% to 36.0% and

the mean displacement magnitude increases from 0.083 to 0.102 (+23%). This confirms that the regularizer acts as a control mechanism that prevents unconstrained boundary growth without significantly affecting reconstruction quality. Finally, removing the control point warm-up delay (*w/o Delay*, $T_{\text{delay}}=0$) reduces PSNR by 0.04 dB, with the baseline winning on 8 of 11 scenes, the largest single-scene drop being +0.23 dB on Kitchen, confirming that letting vertex positions settle before activating boundary deformation improves convergence.

Table 3: Ablation study averaged over all scenes. **Bold** means best per metric.

Configuration	PSNR↑	SSIM↑	LPIPS↓
Ours (baseline)	26.588	0.824	0.174
w/o Deformation	26.500	0.822	0.180
w/o $\mathcal{L}_{\text{curv}}$	26.549	0.823	0.175
w/o Delay	26.545	0.823	0.175

5 Conclusion

We have presented DETRIS, a method that augments triangle primitives’ capabilities with K learnable control points per edge to represent shapes as a soup of non-uniform primitives. By performing all per-pixel deformations and distance computations in a canonical barycentric space, our formulation ensures these boundaries remain strictly view-independent within a fully differentiable rendering pipeline. Our approach also introduces a ray-casting winding number test for robust inside/outside classification, a polynomial smooth-minimum distance field with controllable corner rounding, and a power-law window function for opacity falloff. A simple learning rate delay suffices for the effective use of control points without requiring explicit gradient scheduling. All our claims have been experimentally validated on real-world scenes, providing competitive visual fidelity and rendering speed among non-volumetric methods. Furthermore, our method demonstrates better adaptability and versatility by learning specialized primitive shapes directly from data, as opposed to current methods that rely on fixed geometric shapes, which limits their expressiveness or drastically increases the number of primitives required to model a scene. Future work will explore multi-layer primitives for handling semi-transparent and thin structures.

Acknowledgements

This work has been supported by the project GRAVATAR PID2023-151184OB-I00 funded by MCIU/AEI/10.13039/501100011033 and by ERDF, UE.

References

1. Bae, J., Kim, S., Yun, Y., Lee, H., Bang, G., Uh, Y.: Per-gaussian embedding-based deformation for deformable 3D gaussian splatting. In: ECCV (2024)
2. Barron, J.T., Mildenhall, B., Verbin, D., Srinivasan, P.P., Hedman, P.: Mip-NeRF 360: Unbounded anti-aliased neural radiance fields. In: CVPR (2022)
3. Barron, J.T., Mildenhall, B., Verbin, D., Srinivasan, P.P., Hedman, P.: Zip-NeRF: Anti-aliased grid-based neural radiance fields. In: ICCV (2023)
4. Chan, E., Lin, C., Chan, M., Nagano, K., Pan, B., de Mello, S., Gallo, O., Guibas, L., Tremblay, J., Khamis, S., Karras, T., Wetzstein, G.: Efficient geometry-aware 3D generative adversarial networks. In: CVPR (2022)
5. Chen, A., Xu, Z., Geiger, A., Yu, J., Su, H.: Tensorf: Tensorial radiance fields. In: ECCV (2022)
6. Chen, H., Chen, R., Qu, Q., Wang, Z., Liu, T., Chen, X., Chung, Y.: Beyond gaussians: Fast and high-fidelity 3D splatting with linear kernels. arXiv preprint arXiv:2411.12440 (2024)
7. Chen, Z., Funkhouser, T., Hedman, P., Tagliasacchi, A.: MobileNeRF: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures. In: CVPR (2023)
8. Du, Y., Smith, C., Tewari, A., Sitzmann, V.: Learning to render novel views from wide-baseline stereo pairs. In: CVPR (2023)
9. Fan, C.D., Chang, C.W., Liu, Y.R., Lee, J.Y., Huang, J.L., Tseng, Y.C., Liu, Y.L.: SpectroMotion: Dynamic 3D reconstruction of specular scenes. In: CVPR (2025)
10. Fridovich-Keil, S., Meanti, G., Warburg, F., Recht, B., Kanazawa, A.: K-planes: Explicit radiance fields in space, time, and appearance. In: CVPR (2023)
11. Fridovich-Keil, S., Yu, A., Tancik, M., Chen, Q., Recht, B., Kanazawa, A.: Plenoxels: Radiance fields without neural networks. In: CVPR (2022)
12. Guedón, A., Lepetit, V.: Gaussianfrothing: Editable complex radiance fields with real-time rendering. In: ECCV (2024)
13. Hamdi, A., Melas-Kyriazi, L., Mai, J., Qian, G., Liu, R., Vondrick, C., Ghanem, B., Vedaldi, A.: GES: Generalized exponential splatting for efficient radiance field rendering. In: CVPR (2024)
14. Held, J., Vandeghen, R., Deliege, A., Hamdi, A., Giancola, S., Cioppa, A., Vedaldi, A., Ghanem, B., Tagliasacchi, A., Droogenbroeck, M.V.: Triangle splatting for real-time radiance field rendering. In: 3DV (2026)
15. Held, J., Vandeghen, R., Hamdi, A., Deliege, A., Cioppa, A., Giancola, S., Vedaldi, A., Ghanem, B., Droogenbroeck, M.V.: 3D convex splatting: Radiance field rendering with 3D smooth convexes. In: CVPR (2025)
16. Hu, S., Hu, T., Liu, Z.: Articulated gaussian splatting from monocular human videos. In: CVPR (2024)
17. Huang, B., Yu, Z., Chen, A., Geiger, A., Gao, S.: 2D gaussian splatting for geometrically accurate radiance fields. SIGGRAPH (2024)
18. Huang, Y.H., Lin, M.X., Sun, Y.T., Yang, Z., Lyu, X., Cao, Y.P., Qi, X.: Deformable radial kernel splatting. In: CVPR (2025)
19. Jacobson, A., Kavan, L., Sorkine-Hornung, O.: Robust inside-outside segmentation using generalized winding numbers. SIGGRAPH (2013)
20. Kato, H., Ushiku, Y., Harada, T.: Neural 3D mesh renderer. In: CVPR (2018)
21. Kerbl, B., Kopanas, G., Leimkühler, T., Drettakis, G.: 3D gaussian splatting for real-time radiance field rendering. TOG (2023)

22. Kheradmand, S., Rebain, D., Sharma, G., Sun, W., Tseng, J., Isack, H., Kar, A., Tagliasacchi, A., Yi, K.M.: 3D gaussian splatting as markov chain monte carlo. In: NeurIPS (2024)
23. Kingma, D., Ba, J.: Adam: A method for stochastic optimization. In: ICLR (2015)
24. Knapitsch, A., Park, J., Zhou, Q.Y., Koltun, V.: Tanks and temples: Benchmarking large-scale scene reconstruction. TOG (2017)
25. Kulhanek, J., Sattler, T.: Tetra-NeRF: Representing neural radiance fields using tetrahedra. In: ICCV (2023)
26. Lee, J.C., Rho, D., Sun, X., Ko, J.H., Park, E.: Compact 3D gaussian representation for radiance field. In: CVPR (2024)
27. Liu, R., Gao, Z., Planche, B., Chen, M., Nguyen, V.N., Zheng, M., Choudhuri, A., Chen, T., Wang, Y., Feng, A., Wu, Z.: Universal beta splatting. ICLR (2026)
28. Liu, R., Sun, D., Chen, M., Wang, Y., Feng, A.: Deformable beta splatting. In: SIGGRAPH (2025)
29. Liu, S., Chen, W., Li, T., Li, H.: Soft rasterizer: A differentiable renderer for image-based 3D reasoning. In: ICCV (2019)
30. Lu, T., Yu, M., Xu, L., Xiangli, Y., Wang, L., Lin, D., Dai, B.: Scaffold-GS: Structured 3D gaussians for view-adaptive rendering. In: CVPR (2024)
31. von Lützw, N., Nießner, M.: Linprim: Linear primitives for differentiable volumetric rendering. In: NeurIPS (2025)
32. Marschner, S., Shirley, P.: Fundamentals of Computer Graphics. A K Peters/CRC Press (2021)
33. Martin-Brualla, R., Radwan, N., Sajjadi, M., Barron, J., Dosovitskiy, A., Duckworth, D.: NeRF in the wild: Neural radiance fields for unconstrained photo collections. In: CVPR (2021)
34. Mildenhall, B., Srinivasan, P.P., Tancik, M., Barron, J.T., Ramamoorthi, R., Ng, R.: NeRF: Representing scenes as neural radiance fields for view synthesis. In: ECCV (2020)
35. Müller, T., Evans, A., Schied, C., Keller, A.: Instant neural graphics primitives with a multiresolution hash encoding. SIGGRAPH (2022)
36. Niemeyer, M., Barron, J., Mildenhall, B., Sajjadi, M., Geiger, A., Radwan, N.: RegNeRF: Regularizing neural radiance fields for view synthesis from sparse inputs. In: CVPR (2022)
37. O'Rourke, J.: Computational geometry in C. Cambridge university press (1998)
38. de Paco, S., Agudo, A.: 4DPV: 4D pet from videos by coarse-to-fine non-rigid radiance fields. In: ACCV (2024)
39. Park, K., Sinha, U., Barron, J.T., Bouaziz, S., Goldman, D.B., Seitz, S.M., Martin-Brualla, R.: Nerfies: Deformable neural radiance fields (2021)
40. Quilez, I.: Smooth minimum. <https://iquilezles.org/articles/smin/> (2013), accessed: 2025-01-15
41. Reiser, C., Szeliski, R., Verbin, D., Srinivasan, P., Mildenhall, B., Geiger, A., Barron, J., Hedman, P.: MERF: Memory-efficient radiance fields for real-time view synthesis in unbounded scenes. TOG (2023)
42. Rong, V., Chen, J., Bahmani, S., Kutulakos, K., Lindell, D.B.: Gstex: Per-primitive texturing of 2D gaussian splatting for decoupled appearance and geometry modeling. In: WACV (2025)
43. Salort-Benejam, L., Agudo, A.: Nerfscopy: Neural radiance fields for in-vivo time-varying tissues from endoscopy. In: ISBI (2026)
44. Schönberger, J.L., Frahm, J.M.: Structure-from-motion revisited. In: CVPR (2016)
45. Shen, J., Agudo, A., Moreno-Noguer, F., Ruiz, A.: Conditional-flow NeRF: Accurate 3D modelling with reliable uncertainty quantification. In: ECCV (2022)

46. Svitov, D., Morerio, P., Agapito, L., Del Bue, A.: Billboard splatting (BBSplat): Learnable textured primitives for novel view synthesis. In: ICCV (2025)
47. Terzopoulos, D., Platt, J., Barr, A., Fleischer, K.: Elastically deformable models. In: SIGGRAPH (1987)
48. Uy, M.A., Nakayama, G.K., Yang, G., Thomas, R.K., Guibas, L., Li, K.: NeRF revisited: Fixing quadrature instability in volume rendering. In: NeurIPS (2023)
49. Verbin, D., Hedman, P., Mildenhall, B., Zickler, T., Barron, J., Srinivasan, P.: Ref-NeRF: Structured view-dependent appearance for neural radiance fields. In: CVPR (2022)
50. Wu, R., Zhang, Z., Chen, M., Yan, Z., Zuo, W.: Deblur4dgs: 4D gaussian splatting from blurry monocular video. In: AAAI (2026)
51. Xu, Q., Xu, Z., Philip, J., Bi, S., Shu, Z., Sunkavalli, K., Neumann, U.: Point-NeRF: Point-based neural radiance fields. In: CVPR (2022)
52. Zhang, X., Bi, S., Sunkavalli, K., Su, H., Xu, Z.: NeRFusion: Fusing radiance fields for large-scale scene reconstruction. In: CVPR (2022)
53. Zhang, Z., Huang, B., Jiang, H., Zhou, L., Xiang, X., Shen, S.: Quadratic gaussian splatting: High quality surface reconstruction with second-order geometric primitives. In: ICCV (2025)