

Nexels: Neurally-Textured Surfels for Real-Time Novel View Synthesis with Sparse Primitives

Victor Rong^{*1,2}, Jan Held^{*3,4}, Victor Chu^{1,2}, Daniel Rebain⁵, Marc Van Droogenbroeck⁴, Kiriakos N. Kutulakos^{1,2}, Andrea Tagliasacchi^{†1,3}, and David B. Lindell^{†1,2}

¹University of Toronto ²Vector Institute ³Simon Fraser University

⁴University of Liège ⁵University of British Columbia

^{*},[†] Equal contribution

Abstract. Though Gaussian splatting has achieved impressive results in novel view synthesis, it requires millions of primitives to represent highly textured scenes — even when the underlying geometry is simple. We propose a representation that decouples geometry and appearance to achieve a compact and computationally efficient representation. Our method uses surfels for geometry and combines a global neural field with per-primitive colors for appearance. While existing approaches that texture primitives incur significant overhead in either memory or rendering speed, our method strikes a better balance by leveraging the neural field to implicitly compress texture information. We bound the computations required in the rendering pass by querying the neural field for a fixed number of primitives per pixel, enabling real-time rendering speeds. Our representation matches the perceptual quality of 3D Gaussian splatting while using $9.7\times$ fewer primitives and $5.5\times$ less memory on outdoor scenes, and $31\times$ fewer primitives and $3.7\times$ less memory on indoor scenes.

1 Introduction

In computer graphics, a scene’s geometry governs how light is blocked, while its appearance determines the color contributed to each rendered pixel. Point-based representations like 3D Gaussian splats (3DGS) *merge* these roles, since each primitive jointly encodes geometry and appearance [26]. This coupling stands in contrast to traditional surface-based methods, where coarse geometry (e.g., a quad) can still capture fine-grained appearance (e.g., a detailed painting) through texturing. As illustrated in Figure 1, Gaussian splats therefore require a large number of primitives to represent high-frequency appearance, leading to substantial memory usage.

A more compact representation becomes possible by *decoupling* geometry from appearance. When appearance is delegated to textures, mesh-based models can depict highly realistic scenes, as commonly seen in professionally produced games and films. Yet meshes have lagged behind in the task of novel view synthesis [20], where the objective is to generate new viewpoints from a collection of input images of a 3D scene. In the dense-pose setting we consider, the dominant

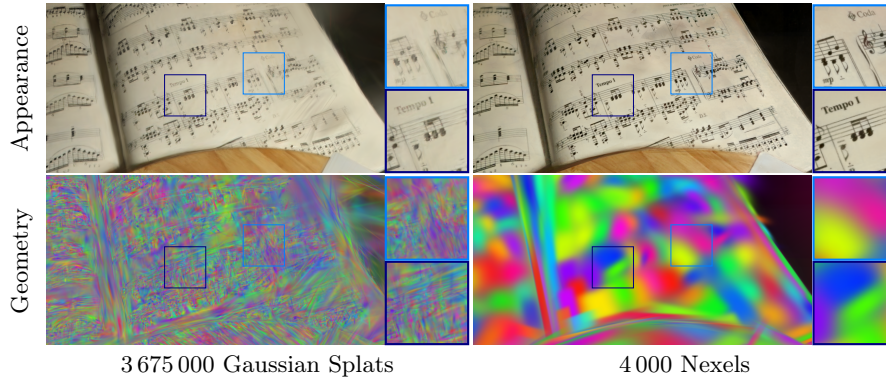


Fig. 1: Limitation of point-based representations. Our captured TABLE scene contains planar geometries with complex textures such as the sheet music above. An ideal representation would have very few parameters devoted to geometry, and much more devoted to appearance. Gaussian splatting needs millions of Gaussians to represent the fine details, each of which has a *fixed* ratio of geometry-to-appearance parameters.

strategy is *differentiable rendering*, which optimizes scene parameters (geometry, light, and materials) so that rendered views match the training images. Within this framework, meshes face difficulties due to their discrete connectivity, potential for self-intersections, and complications in computing silhouette gradients [30, 33, 36, 42].

To mitigate the need for an excessive number of primitives without resorting to the complexities of mesh optimization, recent work has proposed decoupling geometry and appearance in Gaussian splatting by introducing *textured splats*, most commonly via *per-primitive textures* [7, 43, 47, 55]. However, the memory footprint of such textures grows quadratically with resolution, which constrains their ability to represent high-frequency appearance in large-scale scenes. Zhang et al. [67] attempt to overcome this limitation by employing a neural field to provide appearance in a deferred manner: each primitive samples neural features from a multi-resolution hash grid, which are then alpha-composited and passed through a feed-forward network. While this design requires only a single network evaluation, hash-grid lookups are similarly expensive, and performing several of them for every ray-primitive intersection results in *slow rendering* (below 30 FPS), as well as *slow training*.

Although these methods take inspiration from textured meshes, they miss a crucial observation: in standard mesh rasterization with depth buffering, texture fetches are performed only for a *limited* set of *opaque* surface fragments [1, 25]. Motivated by this, we introduce a new representation that preserves the optimization advantages of point-based approaches while leveraging the lightweight appearance modeling of mesh-based methods – all without sacrificing real-time rendering performance.

Contributions. We introduce *nexels*, a novel representation composed of neurally-textured primitives that overcomes the coupling of geometry and appearance in Gaussian splatting. In particular:

- (i) For geometry, we introduce a differentiable *quad* indicator, enabling us to better reconstruct surfaces and sharp boundaries;
- (ii) For appearance, we learn a world-space neural field that provides view-dependent texture only for the most relevant primitives, capturing fine details efficiently while keeping computation low;
- (iii) We introduce a new dataset that highlights the limitations of point-based representations in accurately reconstructing regions with high-frequency textures;
- (iv) We achieve perceptual parity with 3DGS using $9.7\times$ fewer primitives on outdoor scenes and $31\times$ fewer primitives on indoor scenes;
- (v) We achieve better photometric quality than existing textured methods while rendering at real-time rates.

2 Related Work

Neural fields. A neural field implicitly represents a quantity over a region, such as 3D space, through neural network queries. In novel view synthesis, Mildenhall et al. [39] use neural fields to parameterize per-point radiance and density for volumetric rendering. These neural radiance fields (NeRFs) have slow rendering speed due to both the large number of samples needed for volumetric rendering as well as expensive per-sample calculations. To reduce the number of samples used for rendering volumes, early NeRF papers employ stratified sampling and auxiliary proposal networks to improve the sampling efficiency [2, 3, 39]. Later works use empty space skipping and other acceleration techniques to avoid unnecessary queries [32, 41, 51]. The number of queries can be further constrained using explicit representations such as shells and meshes [50, 59]. In parallel, follow-up methods reduce the computational cost of each query by moving much of the neural network’s representational power into spatial features backed by voxel grids [15, 54], hierarchical grids [38, 41, 56, 65], point sets [9, 11, 61], and other structures [6, 8]. Another line of work avoids the slow runtime of neural fields by only using them in an initial stage of training before baking in the appearance and geometry into a fast mesh representation [18, 45, 58, 63]. In contrast, we use surfels to obtain sample locations which avoids the added complication of mesh extraction while seamlessly integrating into the differentiable rendering scheme.

Gaussian splatting. Gaussian splatting completely forgoes the neural network and instead represents radiance fields as a set of explicit primitives which have decaying opacities according to a Gaussian distribution [26]. It leverages fast point-based rasterization schemes to achieve real-time performance. There has been a deluge of follow-up works, of which we will only focus on those that present new geometry or appearance representations. Several works find that using flattened 3D primitives also results in high-quality novel view synthesis

results [12, 24] and Ye et al. [64] use these to explicitly model opaque surfaces. Other non-Gaussian geometries have also been found to be effective: beta kernels, polynomial kernels, triangles, and more have been proposed [19, 21–23, 35, 40]. Deformable Beta Splatting (DBS) [35] is particularly effective at achieving higher rendering quality with fewer parameters. They achieve this reduction through a leaner set of parameters for view-dependent appearance, which is orthogonal to our own work. A number of works use neural fields to compress primitive information, but unlike our work, the splats have no spatial variation across them and cannot represent details smaller than themselves [10, 37, 46, 52, 60]. Other works combine 2DGS and 3DGS into a single hybrid representation, but face the same issue of no variation across the splats [34, 68].

A recent line of work applies textures to Gaussian splats to increase the appearance capacity of the representation. This has been done through per-primitive image textures [7, 43, 47, 55] and neural fields [62, 67]. Despite promising results, there are three key issues which hinder these methods, summarized in Tab. 1. First, the majority of these works initialize from a trained dense Gaussian splatting model, which limits their ability to optimize a representation with fewer primitives. Furthermore, the texturing in all prior methods is applied for every ray-primitive interaction. For BBSplat and Textured Gaussians, whose alpha textures further increase overdraw, renders of scene-level reconstructions typically take over fifty milliseconds. Finally, an ideal texture should be able to capture multiscale details within a reasonable amount of memory. Works which use per-primitive image textures are limited by their resolution, which scales poorly with memory. Only NeST-Splatting fully overcomes the issue by using a neural field, but this results in much slower renders. Our representation uses a neural field at a fixed number of samples per-pixel, leading to real-time renders.

Method	Scene	Adapt.	Speed	Mem.
Texture-GS [62]	×	×	✓	×
GStex [47]	✓	×	✓	×
Textured Gaussians [7]	✓	×	×	×
BBSplat [55]	✓	✓	×	×
NeST-Splatting [67]	✓	×	×	✓
GS-Texturing [43]	✓	✓	✓	×
Nexels	✓	✓	✓	✓

Table 1: Prior textured works. We summarize the limitations of recent representations with textured primitives. The criteria are *scene-level reconstructions*, *adaptive density control with textures*, *real-time rendering speeds (30+ FPS)*, and *memory-efficient texture*.

3 Preliminaries

Our work builds on both Gaussian splatting and neural field methods. In particular, we make use of the differentiable surfel rasterization of 2D Gaussian splatting (2DGS) [24] and the neural field architecture of Instant-NGP [41].

2D Gaussian Splatting [24]. Each surfel in 2D Gaussian splatting (2DGS) is a 2D Gaussian in 3D space parameterized by its opacity $\alpha \in \mathbb{R}$, rotation matrix $\mathbf{R} = [\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3] \in \mathbb{SO}_3$, mean $\boldsymbol{\mu} \in \mathbb{R}^3$, and scale $\boldsymbol{\sigma} \in \mathbb{R}^2$. The first two axes of the rotation, $\mathbf{v}_1$ and $\mathbf{v}_2$, determine the axes of the Gaussian, while $\mathbf{v}_3$ functions as the surfel normal. Additionally, each primitive has a view-dependent radiance

$\mathbf{c}^\odot \in \mathbb{R}^{3S}$ parameterized by channel-wise spherical harmonics coefficients, where S is the number of coefficients for a predetermined degree.

Following 3DGS, the surfels are preprocessed and sorted by the depth of their centers. For a ray $\mathbf{r} := \mathbf{o} + t\mathbf{d}$ from the camera, the intersection in primitive space (u, v) can be computed differentiably from its geometry parameters [24, 53]. This is used to compute its alpha value, $\alpha := \mathbf{o} \exp(-(u^2 + v^2)/2)$. The primitives are alpha-composited in a tile-based rasterizer. The final color is

$$\mathbf{c}(\mathbf{r}) := \sum_i T_i \alpha_i \mathbf{c}_i^\odot(\mathbf{r}), \quad T_i := \prod_{j=1}^{i-1} (1 - \alpha_j). \quad (1)$$

Instant-NGP [41]. Instant-NGP is a neural field architecture composed of a multiresolution hash-grid $\mathcal{H}$ and a tiny multi-layer perceptron (MLP) network $\mathcal{G}$. The multiresolution hash-grid has L levels, each defining a hash-table $\mathcal{H}_\ell$ at given scale s_ℓ . In our implementation, each hash-table has exactly T entries. Finally, it has a feature dimension of F . Altogether it has $L \times T \times F$ parameters.

Given an input coordinate $x \in \mathbb{R}^3$, an Instant-NGP $\mathcal{F}$ can be configured to return a vector whose size is the MLP output dimension. First, a hash $\mathbf{h}_\ell \in \{0, 1, \dots, T-1\}$ is calculated for each hash-grid level ℓ based on $s_\ell \cdot x$. This hash is indexed into the grid at that level to retrieve a feature vector. These are concatenated to obtain $\mathcal{H}(x) := (\mathcal{H}_{0, \mathbf{h}_0}, \dots, \mathcal{H}_{L-1, \mathbf{h}_{L-1}}) \in \mathbb{R}^{L \times F}$. Finally, the MLP is applied to yield $\mathcal{F}(x) := \mathcal{G}(\mathcal{H}(x))$.

4 Nexels

This section provides an overview of the main components of our method. Section 4.1 introduces the geometry and appearance representations, while Section 4.2 describes the rendering process. Finally, Section 4.3 outlines the optimization framework, including adaptive density control and the loss terms used during training. For notational convenience, we present our method for a single pixel p with ray $\mathbf{r}$ and use view-*independent* radiance in equations.

4.1 Representation

Geometry. We represent the geometry of nexels using a set of N surfels defined by a kernel governing their opacities. Following 2DGS, the i^{th} primitive has a mean $\mu_i \in \mathbb{R}^3$, rotation $\mathbf{R}_i \in \text{SO}_3$, scale $\sigma_i \in \mathbb{R}^2$, and opacity $\mathbf{o}_i \in \mathbb{R}$. In addition, we have a gamma parameter $\gamma_i \in \mathbb{R}^2$ that controls how rect-like the primitive is.

The opacity at an intersection (u, v) in primitive space is defined by the generalized Gaussians,

$$\alpha := \mathbf{o}_i \exp\left(-\frac{|u|^{2\gamma_{i,1}}}{2}\right) \exp\left(-\frac{|v|^{2\gamma_{i,2}}}{2}\right). \quad (2)$$

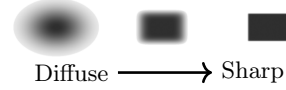


Fig. 2: Kernel. We use a generalized Gaussian kernel in order to model near-opaque primitives. Values below 0.1 are set to 0 for visualization purposes. (Left) For $\gamma=1$, our nexels correspond to a Gaussian distribution. (Right) As $\gamma \rightarrow \infty$, they converge toward the indicator of a quad.

Figure 2 shows the nixel shape for different values of γ . As $\gamma \rightarrow 1$, the kernel converges to a standard Gaussian. As $\gamma \rightarrow \infty$ it converges toward a quad with a sharp transition. This kernel is well suited for modeling opaque and flat surfaces: for large σ and γ , the function remains close to 1 over most of the support, unlike a conventional Gaussian which has a long tail. Our kernel behaves like a Gaussian where smoothness is needed, while still being able to produce hard edges in regions with sharp transitions. We constrain $\gamma \geq 1$, which guarantees that our kernel is continuous with class C^∞ . Unlike prior work which propose similar kernels [19, 35], our primitive can model hard straight edges. This is because we take the outer product of two generalized Gaussians rather than evaluate the generalized Gaussian of the distance. As γ_1 and γ_2 grow, the kernel approaches a rectangle rather than an ellipse as in prior work [19, 35].

Appearance. Each surfel has two modes of appearance: non-textured and textured. During rendering, we select which to use on a per-pixel basis. The non-textured appearance of surfel i is the same as 2DGS [24], $\mathbf{c}_i^\odot \in \mathbb{R}^{3S}$.

The textured appearance of each primitive is represented using a shared neural field $\mathcal{F}$ backed by an Instant-NGP architecture [41]. The field takes as input any 3D position $x \in \mathbb{R}^3$ and outputs a view-dependent radiance $\mathcal{F}(x)$ as a set of spherical harmonics coefficients. The texture at a given point on any of the surfels can be computed by querying the neural field at that position.

For a primitive i , we define $t_i^*(\mathbf{r})$ as the intersection depth and $x_i^*(\mathbf{r}) := \mathbf{o} + t^* \mathbf{d}$ as the world-space intersection. Naively, the textured appearance for pixel p would be $\mathcal{F}(x_i^*(\mathbf{r}))$. Using point samples for differentiable rendering leads to inaccuracies in the fine levels of the hash-grid. Following the down-weighting analysis of Barron et al. [4], we multiply the hash-grid features of level ℓ by

$$\Delta_{i,\ell}(\mathbf{r}) := 1 - \exp\left(-\frac{1}{2\pi} \left(\frac{f}{s_\ell t_i^*(\mathbf{r})}\right)^2\right), \quad (3)$$

where f is the focal length. This requires the same amount of computation as the naive sampling and removes noticeable aliasing artifacts. Ultimately, we define the filtered texture radiance along $\mathbf{r}$ for primitive i as

$$\mathbf{c}_i^\oplus(\mathbf{r}) := \mathcal{G}(\Delta_i(\mathbf{r}) \cdot \mathcal{H}(x_i^*(\mathbf{r}))). \quad (4)$$

Note that the only primitive information needed to compute $\mathbf{c}_i^\oplus(\mathbf{r})$ is $t_i^*(\mathbf{r})$.

4.2 Rendering

Our rendering step is composed of *two passes* over the image pixels. The first pass computes the initial render using only the non-textured appearance and collects information on which primitives should be textured. The second pass applies the neural texture at the collected ray-primitive intersections and updates the initial render. Figure 3 provides an overview of the different rendering passes.

Collection Pass. In the first pass, we compute the alpha α_i and transmittance values T_i across the M surfels which intersect a pixel’s ray. We use Eq. (1) to compute the initial render $\mathbf{N}$ for each pixel, assuming no primitive is textured.

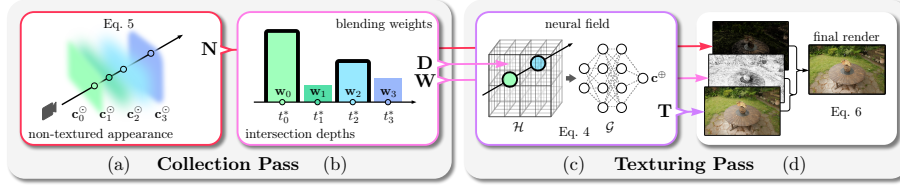


Fig. 3: Rendering pipeline. (a) We iterate over the surfels and determine the weights at each intersection while also returning a non-textured render. (b) We collect the K intersections which have the largest weights. (c) We pass their K positions into a neural field to obtain textured appearance. (d) We composite the textures with the non-textured render to get the final output.

When rendering a single pixel, we limit the number of primitives which are textured to a hyperparameter $K \ll M$. We determine the subset of primitives which are textured based on their blending weights which we denote as $\mathbf{w}_i := \alpha_i T_i$. For a given pixel, the K primitives with the highest weights are selected. Inspired by fragment buffer techniques [5], we maintain a per-pixel buffer of size K . Each buffer entry contains a primitive id, its weight, and its depth, totalling to $3K$ 32-bit registers. Initially, the buffer is empty. When iterating through the M primitives and computing the weights, we update the buffer to store the running primitives with the highest weights. The final id, depth, and weight buffers are written into $H \times W \times K$ images $\mathbf{I}$, $\mathbf{D}$, and $\mathbf{W}$, respectively. These are returned along with the non-textured render $\mathbf{N}$. Additionally, for the K primitives in $\mathbf{I}$, we remove their non-textured colour from the render $\mathbf{N}$. The final value of the non-textured render at pixel p is

$$\mathbf{N}_p := \sum_{i=0}^{M-1} [i \notin \mathbf{I}_p] \mathbf{w}_i \mathbf{c}_i^\odot. \quad (5)$$

Texturing Pass. The texturing pass first computes world-space intersection positions from the depth-map buffer $\mathbf{D}$, yielding $x \in \mathbb{R}^{H \times W \times K \times 3}$. Following Eq. (4), we query the neural field to compute the filtered texture $\mathbf{T} \in \mathbb{R}^{H \times W \times K \times 3}$ where $\mathbf{T}_{p,j} := \mathbf{c}_{\mathbf{I}_{p,j}}^\oplus(\mathbf{r})$ for pixel p with corresponding ray $\mathbf{r}$. Extending Eq. (1) to include the choice between texturing and not texturing, we have

$$\sum_{i=0}^{M-1} T_i \alpha_i \mathbf{c}_i(\mathbf{r}) = \sum_{i=0}^{M-1} \mathbf{w}_i ([i \notin \mathbf{I}_p] \mathbf{c}_i^\odot + [i \in \mathbf{I}_p] \mathbf{c}_i^\oplus(\mathbf{r})) = \mathbf{N}_p + \sum_{j=0}^{K-1} \mathbf{W}_{p,j} \mathbf{T}_{p,j}. \quad (6)$$

Hence, we compute the final render from $\mathbf{N}$, $\mathbf{W}$, and $\mathbf{T}$.

4.3 Optimization

We follow the scheme of 3DGS [26] and optimize against training views for 30,000 iterations with regularizations. We interleave density control operations.

Adaptive Density Control. We introduce a hyperparameter P to control the number of primitives. Similar to BBSplat [55], we sample the point cloud output of COLMAP for initialization [49]. Specifically, we use farthest point sampling to reduce the initial point cloud to $0.5P$ points [31]. We initialize the scale at each point based on its distance to its predecessors during farthest point sampling.

We also prune unnecessary primitives and add new ones in regions where the current geometry is insufficient. We perform a densification and pruning step every 100 iterations from iteration 500 to 25,000. We select 5% of primitives to split evenly along their longest axis. The selection is stochastic with probabilities defined using blended errors similar to the method of Rota Bulò et al. [48]. We cap the number of primitives added in the splitting by P . Afterward, we prune all primitives whose opacity is below 0.005.

Losses. At each iteration, we render the nexels from a given training view to produce a predicted image. Following Kerbl et al. [26], we compute a photometric loss between the prediction and ground truth using a mixture of an L_1 loss $\mathcal{L}_1$ and D-SSIM loss $\mathcal{L}_{\text{D-SSIM}}$ [26], $\mathcal{L}_{\text{image}} := (1 - \lambda_{\text{D-SSIM}})\mathcal{L}_1 + \lambda_{\text{D-SSIM}}\mathcal{L}_{\text{D-SSIM}}$. We also include a texture loss, $\mathcal{L}_{\text{texture}}$, so that the non-pre-multiplied texture fits to the ground truth image I^{GT} :

$$\frac{1}{3HW} \sum_{y=0}^{H-1} \sum_{x=0}^{W-1} \sum_{c=0}^2 \left| I_{y,x,c}^{\text{GT}} - \frac{\sum_{j=0}^{K-1} \mathbf{w}_{y,x,j} \mathbf{T}_{y,x,j,c}}{\sum_{j=0}^{K-1} \mathbf{w}_{y,x,j}} \right|. \quad (7)$$

To penalize non-textured rendering, we minimize the total blending weights of the non-textured appearance,

$$\mathcal{L}_{\text{alpha}} := \frac{1}{HW} \sum_{y=0}^{H-1} \sum_{x=0}^{W-1} \left(1 - \sum_{j=0}^{K-1} \mathbf{w}_{y,x,j} \right). \quad (8)$$

We include an L_1 loss on the opacity [27] $\mathcal{L}_{\text{opacity}} := \frac{1}{N} \sum_{i=0}^{N-1} \mathbf{o}_i$, and a grid weight regularization term, $\mathcal{L}_{\text{grid}} := \sum_{\ell=0}^{L-1} \sum_{i=0}^{T-1} \sum_{j=0}^{F-1} s_i^{-3} \mathcal{H}_{\ell,i,j}$, which encourages zero-mean grid values that justify the down-weighting done for grid anti-aliasing [4]. The final loss is

$$\mathcal{L} := \mathcal{L}_{\text{image}} + \lambda_{\text{alpha}} \mathcal{L}_{\text{alpha}} + \lambda_{\text{texture}} \mathcal{L}_{\text{texture}} + \lambda_{\text{opacity}} \mathcal{L}_{\text{opacity}} + \lambda_{\text{grid}} \mathcal{L}_{\text{grid}}.$$

We set $\lambda_{\text{alpha}} = 0.005$, $\lambda_{\text{texture}} = 0.5$, $\lambda_{\text{opacity}} = 0.01$, $\lambda_{\text{grid}} = 0.01$, and $\lambda_{\text{D-SSIM}} = 0.2$ for all experiments.

5 Results

Our method is implemented in Python and CUDA on top of the INRIA 3DGS codebase [26]. We use PyTorch for performing parameter activations and composing the render passes with autograd and use the Adam optimizer for all parameters [28, 44]. We implement custom CUDA kernels for our multiresolution

	Mip-NeRF360					Tanks & Temples					Custom				
	PSNR	SSIM	LPIPS	Memory	#Prim.	PSNR	SSIM	LPIPS	Memory	#Prim.	PSNR	SSIM	LPIPS	Memory	#Prim.
3DGS [26]	27.21	0.815	0.214	640MB	2.7M	23.80	0.853	0.169	348MB	1.5M	26.65	0.858	0.224	1630MB	7.0M
2DGS [24]	27.04	0.805	0.252	480MB	2.0M	23.15	0.831	0.212	200MB	0.8M	25.98	0.831	0.267	450MB	1.8M
DBS [35]	28.10	0.829	0.192	340MB	3.1M	24.82	0.871	0.144	220MB	2.0M	26.72	0.844	0.273	110MB	1.0M
Instant-NGP [41]	25.12	0.666	0.370	50MB	-	20.94	0.696	0.335	50MB	-	23.14	0.701	0.424	40MB	-
Mip-NeRF360 [3]	27.68	0.792	0.237	40MB	-	24.03	0.849	0.184	40MB	-	26.43	0.814	0.281	40MB	-
ZipNeRF [4]	28.53	0.829	0.187	310MB	-	25.67	0.892	0.120	310MB	-	26.78	0.848	0.224	310MB	-
NeST-Splat [67]	26.68	0.795	0.212	440MB	1.0M	23.02	0.824	0.181	260MB	0.5M	25.40	0.824	0.246	900MB	3.2M
BBSplat [55]	26.98	0.783	0.231	260MB	0.4M	23.83	0.854	0.147	260MB	0.4M	26.57	0.844	0.225	260MB	0.4M
GS-Texturing [43]	26.77	0.783	0.241	310MB	0.2M	23.40	0.833	0.203	100MB	0.1M	26.46	0.851	0.245	430MB	0.2M
Ours	27.35	0.802	0.201	160MB	0.4M	23.55	0.841	0.155	160MB	0.4M	27.04	0.851	0.222	160MB	0.4M

Table 2: Novel view synthesis. We compare nexels with 400K primitives to baselines at their original default settings.

hash-grid operations and rasterization passes. We use the NVIDIA tinycudann library for the fully-fused MLP pass [41]. For all scenes, we use a texture limit of just $K = 2$. The MLP has two hidden layers without bias terms, each with a width of 64, and ReLU activations. We use degree 3 spherical harmonics coefficients for both appearance modes, corresponding to $S = 16$ and an MLP output dimension of 48. For the hash-grid, $L = 16$ and $F = 2$. Unless otherwise stated, we use $T = 2^{20}$. The main results are presented in Section 5.2 with additional results in the supplementary Section S4.2.

5.1 Datasets and Metrics

We perform our evaluation over fifteen real-world scenes across three datasets: all nine scenes from Mip-NeRF360 [3], the TRAIN and TRUCK scenes from the Tanks & Temples dataset [29], and four scenes which we collect involving high amounts of texture detail. Existing datasets seldom have close-up views despite these being important for photorealism. Fine appearance details within these scenes barely show up in their metrics. Our scenes contain high-frequency appearance, such as text and patterns, observed at a range of distances, including close-up shots. We describe the capture and preprocessing details in the supplementary. The full dataset will be released publicly.

We compare against several methods which are or are competitive to the state-of-the-art for novel view synthesis. These consist of point-based representations [24, 26, 35], neural field representations [3, 4, 41], and textured representations [43, 55, 67]. We refer to the supplementary for experimental details for our baselines. In addition, we consider Gaussian splat variants which can produce sparser point sets than the original 3DGS [14, 27].

5.2 Evaluation

We evaluate photometric quality with the standard PSNR, SSIM, and LPIPS metrics used in novel view synthesis and differentiable rendering [39, 66]. The LPIPS metric across all baselines is computed with the same normalization as the original 3DGS paper [26, 48]. We primarily show LPIPS throughout the paper as we find that it best corresponds to the reconstruction quality of fine texture

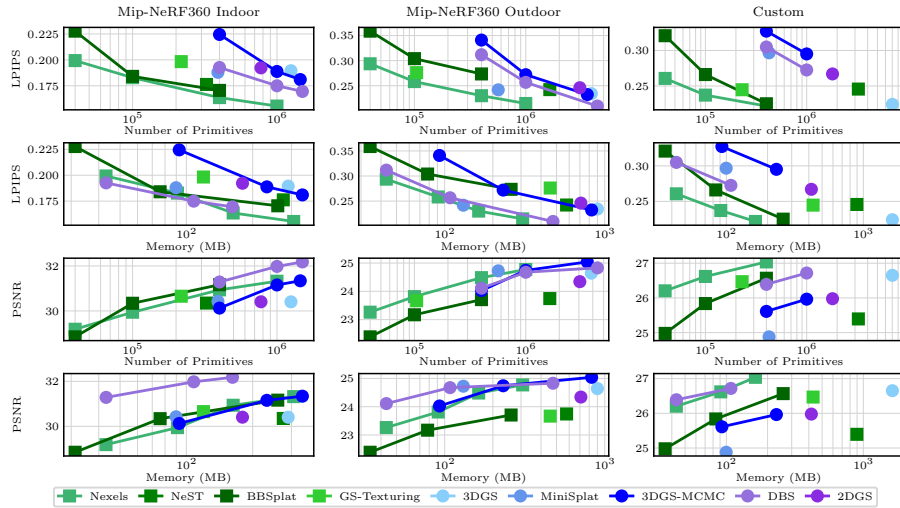


Fig. 4: Quality versus representation size. We evaluate the LPIPS $\downarrow$ (top two rows) and PSNR $\uparrow$ (bottom two rows) across multiple settings for several methods and display separate graphs averaged over the four Mip-NeRF360 indoor scenes (left column), five Mip-NeRF360 outdoor scenes (middle column), and the four highly textured scenes of our custom dataset (right column).

details, which is a weakness of Gaussian splatting. It also generally corresponds better to human preference [66]. Timings for all methods except Mip-NeRF360 and ZipNeRF are performed on a single NVIDIA RTX 6000 Ada GPU using the CUDA implementation of each method’s render pass. For the Mip-NeRF360 and ZipNeRF models, we use a single NVIDIA RTX Pro 6000 Blackwell GPU. We measure the memory needed to perform each method’s forward pass. The memory of BBSplat is computed after their texture compression step [55]. For all other methods, we do not include quantization or other compression steps when computing memory. For methods which use Instant-NGP, including our own, the hash-grid parameters are 16-bit floats as the forward pass operates over half datatypes.

Novel-view synthesis. We compare all baselines at each of their original default settings in Table 2. On the Mip-NeRF360 dataset, nexels is competitive with point-based representations with only 400K primitives, achieving better PSNR than all primitive-based methods except Deformable Beta Splatting (DBS [35]), which has over $7\times$ the number of primitives and twice the memory. On our custom dataset, we achieve the highest PSNR and LPIPS and comes second in SSIM to 3DGS at $10\times$ our memory. We achieve better scores than NeST-Splatting, BBSplat, and GS-Texturing on the Mip-NeRF360 and custom datasets.

Visual quality under primitive and memory budgets. In Figure 4, we plot the perceptual quality (LPIPS) and PSNR against the number of primitives and

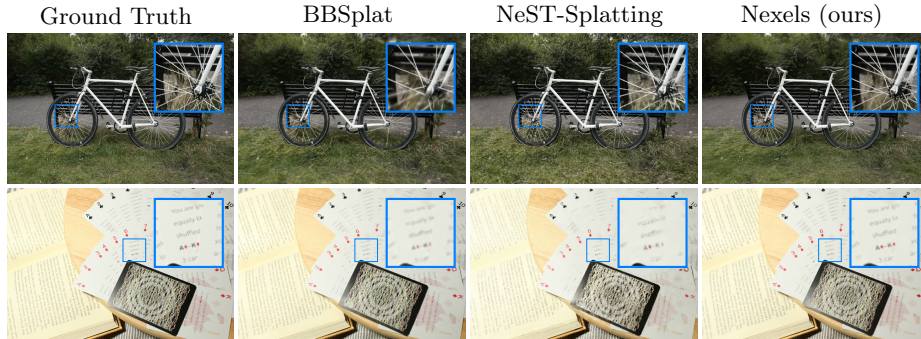


Fig. 5: Comparisons at low number of primitives. We show the BICYCLE and TABLE scenes reconstructed with 40K primitives across different textured representations. Only nexels is able to maintain a high quality in this extreme setting.

memory across three scene groups. For baselines which can specify a number of primitives, we include data points corresponding to appropriate subsets of {40K, 100K, 400K, 1M} primitives. Details are included in the supplementary.

Our representation exceeds all other models in perceptual quality when equalized by number of primitives. For example, 400K nexels obtain an LPIPS score of 0.164 across Mip-NeRF360 indoor scenes, while BBSplat and DBS obtain LPIPS scores of 0.171 and 0.192, respectively. Conversely, to reach a given LPIPS, nexels requires far fewer primitives than other methods. The best result of 3DGS on the Mip-NeRF360 outdoor scenes is 0.234 LPIPS with 3.9 million Gaussians. We achieve 0.230 LPIPS with just 400K nexels. Similarly, 3DGS obtains 0.190 LPIPS on indoor scenes with 1.2 million Gaussians. We reach the same score

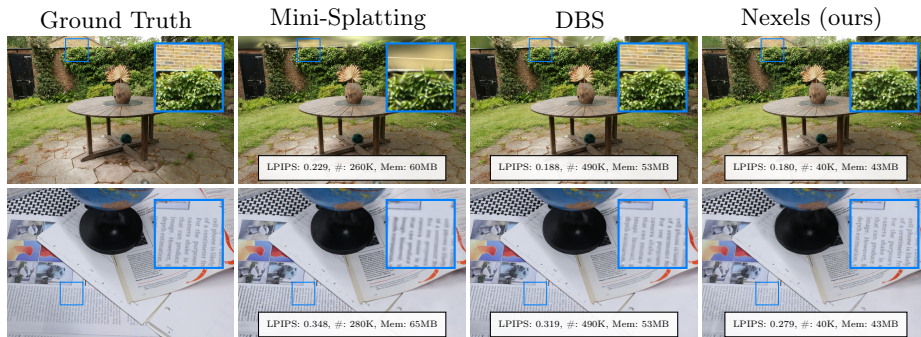


Fig. 6: Comparisons at low memory. We adjust the settings of baseline methods designed to support low memory representations to get a model around 50MB in size. We show the average LPIPS↓ across test views, number of primitives, and total memory for the GARDEN and TRIPOD scenes. With less memory and far fewer primitives, nexels is able to capture fine appearance details and sparsely-observed background structures which point-based methods struggle with. We use $T = 2^{19}$ for the hash grid here.

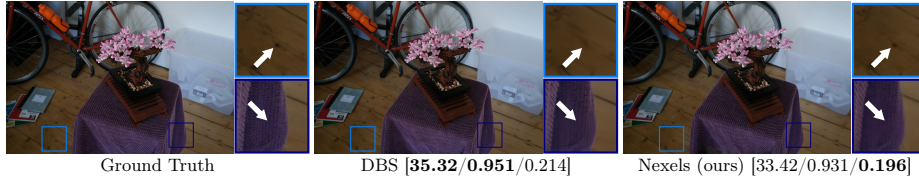


Fig. 7: Quality metrics. We show the PSNR $\uparrow$ /SSIM $\uparrow$ /LPIPS $\downarrow$ for a single test view, evaluated with a DBS and nexels model, each with 43MB. Though the DBS model achieves a better PSNR and SSIM, it doesn’t capture perceptually-similar content as well as the nexels model, as highlighted by the zoom-ins.

at 40K nexels. Our method is also competitive when measured against memory. At 40MB on Mip-NeRF360 outdoor scenes, we achieve 0.294 LPIPS while DBS achieves 0.312 LPIPS. When equalized at any amount of memory, nexels is always among the top three in perceptual quality for all three sets of scenes. We include qualitative results echoing these trends in Figures 5 and 6.

For the custom scenes with high amounts of texture, nexels is clearly better than non-textured representations. We achieve an LPIPS of 0.240 with just 40K nexels, outperforming DBS with 2.5 million splats, which obtains 0.259 LPIPS. BBSplat comes near us at 400K primitives but struggles at low primitive counts.

Our representation is less strong when evaluated based on PSNR as PSNR tends to under-penalize blurry reconstructions of fine details, as shown in Figure 7. From the test view shown, DBS at 400K splats and 43MB achieves +1.9 better PSNR over our nexels at 40K primitives and 43MB, despite reconstructing the fine details less accurately. On the other hand, LPIPS is more discriminative towards fine details as it is a learnt perceptual metric.

Training time and rendering speed. Table 3 compares training time and rendering speed for different textured and neural field works. NeST-Splatting averages five hours of training and only renders at 23 FPS across Mip-NeRF360 scenes. This is due to its texturing, where hash-grid lookups are done for each ray-primitive intersection, totaling thousands of random accesses per ray. BBSplat has a faster training time, but their use of alpha textures results in high overdraw, leading to a rendering speed of 20 FPS. In contrast, nexels trains 4.2 and 1.9 times faster than NeST-Splatting and BBSplat, respectively, and render at 50 FPS. Among the three real-time textured or neural field methods: Instant-NGP, GS-Texturing, and nexels; nexels performs the best in terms of photometric results. Though

Method	FPS	Train.
Instant-NGP	53	0.1h
Mip-NeRF360	0.1*	23h*
ZipNeRF	0.3*	3.3h*
NeST-Splat	23	4.9h
BBSplat	20	2.2h
GS-Texturing	149	0.9h
Nexels	50	1.1h

Table 3: Timing comparisons.

We show the average FPS of test-views across Mip-NeRF360 scenes as well as training times for the neural field and textured representations. The * results use a more powerful NVIDIA Blackwell GPU.

	24	37	40	55	63	65	69	83	97	105	106	110	114	118	122	Avg.
3DGS*	2.14	1.53	2.08	1.68	3.49	2.21	1.43	2.07	2.22	1.75	1.79	2.55	1.53	1.52	1.50	1.96
2DGS*	0.48	0.91	0.39	0.39	1.01	0.83	0.81	1.36	1.27	0.76	0.70	1.40	0.40	0.76	0.52	0.80
Nexels	0.87	0.94	0.91	0.51	1.69	1.28	1.03	1.01	1.82	0.69	0.61	2.04	1.34	0.42	0.54	1.05
+ $\mathcal{L}_n$	0.49	0.75	0.29	0.35	1.32	0.64	0.88	1.37	1.45	0.57	0.54	2.32	0.46	0.58	0.59	0.84

Table 4: Geometric accuracy. Chamfer distance ($\downarrow$) across the DTU scenes. Starred results (*) are taken from the 2DGS paper [24].

Mip-NeRF360 and ZipNeRF generally achieve better quality, as shown in Table 2, this requires extremely slow renders. Nexels is able to leverage the quality of neural fields while remaining fast by using a subset of the primitives to provide quadrature points, essentially acting as an acceleration structure.

Geometric accuracy. We evaluate nexels on the fifteen DTU scenes used by 2DGS [24]. For each scene, we train with a limit of 200K primitives for 30K iterations on all images of the dataset. Following 2DGS, we apply TSDF to rendered depth maps to extract a mesh and compute its Chamfer distance to the ground truth. As shown in Table 4, nexels achieves much better accuracy than 3DGS and outperforms 2DGS on four of the fifteen scenes, despite not using any geometric losses. Since nexels uses surfels with well-defined depths, we also evaluated nexels with the 2DGS normal loss included during training with a weight of 0.05 ($+\mathcal{L}_n$). This gives competitive performance to 2DGS.

5.3 Ablations

The results in Table 5 ablate individual components of our method to demonstrate their importance. We evaluate each individual experiment across the Mip-NeRF360 scenes trained with 400K primitives. First, we experiment with different hash table sizes. We find that $T = 2^{20}$ provides the best quality overall at a reasonable memory budget. Next, we verify that the main component, the neural texture, is useful. We find that all visual metrics, especially LPIPS, worsen without the neural texture. Qualitatively, the non-textured renders lose back-

T	PSNR	SSIM	LPIPS	Mem.		PSNR	SSIM	LPIPS	Mem.	FPS
2^{19}	26.99	0.794	0.217	130MB	No texture	27.20	0.787	0.261	100MB	85
2^{20}	27.35	0.802	0.201	160MB	No gamma	27.10	0.797	0.210	160MB	48
2^{21}	27.30	0.801	0.198	230MB	No prim. SH	27.00	0.797	0.205	90MB	47
2^{22}	27.12	0.796	0.202	360MB	No down.	27.03	0.797	0.211	160MB	49
					No alpha reg.	26.92	0.795	0.215	160MB	47
					Full	27.35	0.802	0.201	160MB	50

Table 5: Ablations. Left: Across different hash-grid table sizes, we find that 2^{20} performs the best overall. Right: We show how dropping individual elements of nexels affects the photometric quality, memory, and FPS of the representation averaged across all Mip-NeRF360 scenes.

ground details. Removing the γ parameter and using 2D Gaussians as the kernel results in slight degradations overall as a Gaussian kernel leads to lower texture weights. Dropping the higher-order per-primitive spherical harmonics coefficients and only relying on the neural field for view-dependent effects is a possibility if memory is the main priority, as it removes 45 parameters for each primitive. The increase in LPIPS is very small, although there is a trade-off in PSNR. Omitting the hash-grid downweighting factor in Eq. (3) leads to noticeable aliasing artifacts in test views, as the fine levels overfit at the intersections seen from training poses. Finally, removing the alpha regularization term results in worse performance as the top- K texture weights are lower.

We additionally find that ensuring all primitives contribute to the render, either through textured or non-textured appearance, is crucial for optimization as it allows for denser gradients during back-propagation. This is important for early training iterations where all primitives move towards the underlying scene geometry, regardless of whether the primitive

was textured. On the other hand, the neural texture does not need to be rendered at multiple overlapping primitives to guide its optimization. Thanks to the volumetric nature of the neural field, its gradient flow is the same whether it is rendered by several coinciding surfels or by a single surfel with the same total weight.

We validate this in Table 6. In the first experiment (during train & eval), we remove the non-textured appearance in both training and evaluation and only rely on the weight-normalized textured appearance. In the second experiment (during eval), we use the full render during training, but drop the non-textured appearance in evaluation. Even though both experiments only use the textured appearance, the latter performs better. Thus, an unbounded number of primitives should receive gradient during optimization.

Drop non-textured app. PSNR SSIM LPIPS			
during train & eval	17.95	0.562	0.392
during eval	24.71	0.733	0.257

Table 6: Optimization analysis. We drop the non-textured appearance at different stages to show the importance of an unbounded number of primitives providing appearance during optimization.

5.4 Limitations

The speed of the Instant-NGP model is reliant on tensor core operations, which are only available on high-end GPUs. While nexels is real-time on the machinery we use, real-time rendering on mobile GPUs or low-end laptop GPUs is currently not possible. In addition, nexels produce noise in unseen regions, which is likely less appealing for end users than the blurry artifacts of Gaussian splats. Our work also does not support the full complexity of the imaging process, including motion blur and depth-of-field blur. These effects are particularly relevant for differentiable rendering of fine details, as even small deviations from the expected pinhole camera model can damage reconstructions. Finally, if the primitives optimize to incorrect positions due to local minima, as sometimes occurs in the background at low numbers of primitives, the texture does not help.

6 Conclusion

We present the first representation for novel view synthesis that removes the reliance on dense primitives while achieving real-time rendering at high quality. Our results show that far fewer primitives are needed to model scene-level datasets than recent research would suggest. Future work could incorporate level-of-detail structures to model city-scale scenes. Sparse surfaces also work well for ray-tracing. Another direction would be to integrate existing neural field [17, 57] and Gaussian compression schemes [13, 16] on top of our representation. Finally, although the top- K strategy was presented specifically for texturing appearance, our method provides a general technique for combining neural field and per-primitive information which could be used in other applications. In conclusion, nexels break new ground in the exploration between volumetric and surface representations, capturing high-frequency appearance at a low memory and compute cost.

Acknowledgments. JH acknowledges support from the F.R.S.-FNRS. DBL and KNK acknowledge support from NSERC under the RGPIN program. DBL also acknowledges support from the Canadian Foundation for Innovation and the Ontario Research Fund.

References

1. Akenine-Möller, T., Haines, E., Hoffman, N., Pesce, A., Iwanicki, M., Hillaire, S.: Real-Time Rendering. AK Peters/CRC Press, Boca Raton, Florida, USA, 4 edn. (2018)
2. Barron, J.T., Mildenhall, B., Tancik, M., Hedman, P., Martin-Brualla, R., Srinivasan, P.P.: Mip-NeRF: A multiscale representation for anti-aliasing neural radiance fields. In: IEEE/CVF Int. Conf. Comput. Vis. (ICCV). pp. 5835–5844. Inst. Electr. Electron. Eng. (IEEE), Montréal, Can. (Oct 2021)
3. Barron, J.T., Mildenhall, B., Verbin, D., Srinivasan, P.P., Hedman, P.: Mip-NeRF 360: Unbounded anti-aliased neural radiance fields. In: IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR). pp. 5460–5469. Inst. Electr. Electron. Eng. (IEEE), New Orleans, LA, USA (Jun 2022)
4. Barron, J.T., Mildenhall, B., Verbin, D., Srinivasan, P.P., Hedman, P.: Zip-NeRF: Anti-aliased grid-based neural radiance fields. In: IEEE/CVF Int. Conf. Comput. Vis. (ICCV). pp. 19640–19648. Inst. Electr. Electron. Eng. (IEEE), Paris, Fr. (Oct 2023)
5. Bavoil, L., Callahan, S.P., Lefohn, A., Comba, J.L.D., Silva, C.T.: Multi-fragment effects on the GPU using the k-buffer. In: ACM Symp. Interact. 3D Graph. Games. pp. 97–104. ACM, Seattle, WA, USA (Apr 2007)
6. Cao, A., Johnson, J.: HexPlane: A fast representation for dynamic scenes. In: IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR). pp. 130–141. IEEE, Vancouver, Can. (Jun 2023)
7. Chao, B., Tseng, H.Y., Porzi, L., Gao, C., Li, T., Li, Q., Saraf, A., Huang, J.B., Kopf, J., Wetzstein, G., Kim, C.: Textured Gaussians for enhanced 3D scene appearance modeling. In: IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR). pp. 8964–8974. IEEE, Nashville, TN, USA (Jun 2025)

8. Chen, A., Xu, Z., Geiger, A., Yu, J., Su, H.: TensorRF: Tensorial radiance fields. In: Eur. Conf. Comput. Vis. (ECCV). Lect. Notes Comput. Sci., vol. 13692, pp. 333–350. Springer Nat. Switz., Tel Aviv, Israël (2022)
9. Chen, H., Miller, B., Gkioulekas, I.: 3D reconstruction with fast dipole sums. *ACM Trans. Graph.* **43**(6), 1–19 (Nov 2024)
10. Chen, Y., Wu, Q., Lin, W., Harandi, M., Cai, J.: HAC: Hash-grid assisted context for 3D Gaussian splatting compression. In: Eur. Conf. Comput. Vis. (ECCV). Lect. Notes Comput. Sci., vol. 15065, pp. 422–438. Springer Nat. Switz. (Sept 2024)
11. Chen, Z., Li, Z., Song, L., Chen, L., Yu, J., Yuan, J., Xu, Y.: NeuRBF: A neural fields representation with adaptive radial basis functions. In: IEEE/CVF Int. Conf. Comput. Vis. (ICCV). pp. 4159–4171. IEEE, Paris, Fr. (Oct 2023)
12. Dai, P., Xu, J., Xie, W., Liu, X., Wang, H., Xu, W.: High-quality surface reconstruction using Gaussian surfels. In: ACM SIGGRAPH Conf. Pap. vol. 35, pp. 1–11. ACM, Denver, CO, USA (Jul 2024)
13. Fan, Z., Wang, K., Wen, K., Zhu, Z., Xu, D., Wang, Z.: LightGaussian: Unbounded 3D Gaussian compression with 15x reduction and 200+ FPS. In: Adv. Neural Inf. Process. Syst. (NeurIPS). vol. 37, pp. 140138–140158. Curran Assoc. Inc., Vancouver, Can. (Dec 2024)
14. Fang, G., Wang, B.: Mini-splatting: Representing scenes with a constrained number of Gaussians. In: Eur. Conf. Comput. Vis. (ECCV). Lect. Notes Comput. Sci., vol. 15135, pp. 165–181. Springer Nat. Switz. (2024)
15. Fridovich-Keil, S., Yu, A., Tancik, M., Chen, Q., Recht, B., Kanazawa, A.: Plenoxels: Radiance fields without neural networks. In: IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR). pp. 5491–5500. Inst. Electr. Electron. Eng. (IEEE), New Orleans, LA, USA (Jun 2022)
16. Girish, S., Gupta, K., Shrivastava, A.: EAGLES: Efficient accelerated 3D Gaussians with lightweight encodings. In: Eur. Conf. Comput. Vis. (ECCV). Lect. Notes Comput. Sci., vol. 15121, pp. 54–71. Springer Nat. Switz. (Nov 2024)
17. Govindarajan, S., Sambugaro, Z., Shabanov, A., Takikawa, T., Rebain, D., Sun, W., Conci, N., Yi, K.M., Tagliasacchi, A.: Lagrangian hashing for compressed neural field representations. In: Eur. Conf. Comput. Vis. (ECCV). Lect. Notes Comput. Sci., vol. 15085, pp. 183–199. Springer Nat. Switz. (Nov 2024)
18. Guo, Y.C., Cao, Y.P., Wang, C., He, Y., Shan, Y., Zhang, S.H.: VMesh: Hybrid volume-mesh representation for efficient view synthesis. In: ACM SIGGRAPH Asia Conf. Pap. pp. 1–11. ACM, Sydney, New South Wales, Aust. (Dec 2023)
19. 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: IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR). pp. 19812–19822. Inst. Electr. Electron. Eng. (IEEE), Seattle, WA, USA (Jun 2024)
20. Held, J., Son, S., Vandeghen, R., Rebain, D., Gadelha, M., Zhou, Y., Cioppa, A., Lin, M.C., Van Droogenbroeck, M., Tagliasacchi, A.: MeshSplatting: Differentiable rendering with opaque meshes. In: IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR). IEEE, Denver, CO, USA (Jun 2026)
21. Held, J., Vandeghen, R., Deliège, A., Hamdi, Abdullah Rebain, D., Giancola, S., Cioppa, A., Vedaldi, A., Ghanem, B., Tagliasacchi, A., Van Droogenbroeck, M.: Triangle splatting for real-time radiance field rendering. In: Int. Conf. 3D Vis. (3DV). pp. 1–10. Vancouver, Can. (Mar 2026)
22. Held, J., Vandeghen, R., Hamdi, A., Deliège, A., Cioppa, A., Giancola, S., Vedaldi, A., Ghanem, B., Van Droogenbroeck, M.: 3D convex splatting: Radiance field rendering with 3D smooth convexes. In: IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR). pp. 21360–21369. IEEE, Nashville, TN, USA (Jun 2025)

23. Held, J., Vandeghen, R., Son, S., Rebain, D., Gadelha, M., Zhou, Y., Lin, M.C., Van Droogenbroeck, M., Tagliasacchi, A.: Triangle splatting+: Differentiable rendering with opaque triangles. arXiv **abs/2509.25122** (2025)
24. Huang, B., Yu, Z., Chen, A., Geiger, A., Gao, S.: 2D Gaussian splatting for geometrically accurate radiance fields. In: ACM SIGGRAPH Conf. Pap. vol. 35, pp. 1–11. ACM, Denver, CO, USA (Jul 2024)
25. Hughes, J., van Dam, A., McGuire, M., Sklar, D., Foley, J.D., Feiner, S., Akeley, K.: Computer Graphics: Principles and Practice. Addison-Wesley, 3 edn. (2014)
26. Kerbl, B., Kopanas, G., Leimkuehler, T., Drettakis, G.: 3D Gaussian splatting for real-time radiance field rendering. ACM Trans. Graph. **42**(4), 1–14 (Jul 2023)
27. 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: Adv. Neural Inf. Process. Syst. (NeurIPS). vol. 37, pp. 80965–80986. Curran Assoc. Inc., Vancouver, Can. (Dec 2024)
28. Kingma, D., Ba, J.: Adam: A method for stochastic optimization. arXiv **abs/1412.6980** (Dec 2014)
29. Knapitsch, A., Park, J., Zhou, Q.Y., Koltun, V.: Tanks and temples: benchmarking large-scale scene reconstruction. ACM Trans. Graph. **36**(4), 1–13 (Jul 2017)
30. Laine, S., Hellsten, J., Karras, T., Seol, Y., Lehtinen, J., Aila, T.: Modular primitives for high-performance differentiable rendering. ACM Trans. Graph. **39**(6), 1–14 (Nov 2020)
31. Li, J., Zhou, J., Xiong, Y., Chen, X., Chakrabarti, C.: An adjustable farthest point sampling method for approximately-sorted point cloud data. In: IEEE Work. Signal Process. Syst. (sips). pp. 1–6. IEEE, Rennes, France (Nov 2022)
32. Li, R., Gao, H., Tancik, M., Kanazawa, A.: NerfAcc: Efficient sampling accelerates NeRFs. In: IEEE/CVF Int. Conf. Comput. Vis. (ICCV). pp. 18491–18500. IEEE, Paris, Fr. (Oct 2023)
33. Li, T.M., Aittala, M., Durand, F., Lehtinen, J.: Differentiable Monte Carlo ray tracing through edge sampling. ACM Trans. Graph. **37**(6), 1–11 (Dec 2018)
34. Lin, J., Gu, J., Fan, L., Wu, B., Lou, Y., Chen, R., Liu, L., Ye, J.: HybridGS: Decoupling transients and statics with 2D and 3D Gaussian splatting. In: Proceedings of the Computer Vision and Pattern Recognition Conference. pp. 788–797 (2025)
35. Liu, R., Sun, D., Chen, M., Wang, Y., Feng, A.: Deformable Beta splatting. In: ACM SIGGRAPH Conf. Pap. pp. 1–11. ACM, Vancouver, Can. (Aug 2025)
36. Liu, S., Chen, W., Li, T., Li, H.: Soft rasterizer: A differentiable renderer for image-based 3D reasoning. In: IEEE Int. Conf. Comput. Vis. (ICCV). pp. 7707–7716. Inst. Electr. Electron. Eng. (IEEE), Seoul, South Korea (Oct 2019)
37. Lu, T., Yu, M., Xu, L., Xiangli, Y., Wang, L., Lin, D., Dai, B.: Scaffold-GS: Structured 3D Gaussians for view-adaptive rendering. In: IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR). pp. 20654–20664. IEEE, Seattle, WA, USA (Jun 2024)
38. Martel, J.N.P., Lindell, D.B., Lin, C.Z., Chan, E.R., Monteiro, M., Wetzstein, G.: ACORN: Adaptive coordinate networks for neural scene representation. ACM Trans. Graph. **40**(4), 1–13 (Jul 2021)
39. Mildenhall, B., Srinivasan, P.P., Tancik, M., Barron, J.T., Ramamoorthi, R., Ng, R.: NeRF: Representing scenes as neural radiance fields for view synthesis. Commun. ACM **65**(1), 99–106 (Dec 2021)
40. Moenne-Loccoz, N., Mirzaei, A., Perel, O., de Lutio, R., Martinez Esturo, J., State, G., Fidler, S., Sharp, N., Gojcic, Z.: 3D Gaussian ray tracing: Fast tracing of particle scenes. ACM Trans. Graph. **43**(6), 1–19 (Nov 2024)

41. Müller, T., Evans, A., Schied, C., Keller, A.: Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.* **41**(4), 1–15 (Jul 2022)
42. Nicolet, B., Jacobson, A., Jakob, W.: Large steps in inverse rendering of geometry. *ACM Trans. Graph.* **40**(6), 1–13 (Dec 2021)
43. Papantonakis, P., Kopanas, G., Durand, F., Drettakis, G.: Content-aware texturing for Gaussian splatting. In: *Eurographics Symp. Render.* pp. 1–12. Eurographics Assoc., Copenhagen, Denmark (Jun 2025)
44. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: Pytorch: An imperative style, high-performance deep learning library. In: *Adv. Neural Inf. Process. Syst. (NeurIPS)*. vol. 32, pp. 8026–8037. Curran Assoc. Inc., Vancouver, Can. (Dec 2019)
45. Reiser, C., Garbin, S., Srinivasan, P., Verbin, D., Szeliski, R., Mildenhall, B., Barron, J., Hedman, P., Geiger, A.: Binary opacity grids: Capturing fine geometric detail for mesh-based view synthesis. *ACM Trans. Graph.* **43**(4), 1–14 (Jul 2024)
46. Ren, K., Jiang, L., Lu, T., Yu, M., Xu, L., Ni, Z., Dai, B.: Octree-GS: Towards consistent real-time rendering with LOD-structured 3D Gaussians. *IEEE Trans. Pattern Anal. Mach. Intell.* pp. 1–15 (2025)
47. Rong, V., Chen, J., Bahmani, S., Kutulakos, K.N., Lindell, D.B.: GStex: Per-primitive texturing of 2D Gaussian splatting for decoupled appearance and geometry modeling. In: *IEEE/CVF Winter Conf. Appl. Comput. Vis. (WACV)*. pp. 3508–3518. IEEE, Tucson, AZ, USA (Feb 2025)
48. Rota Bulò, S., Porzi, L., Kotschieder, P.: Revising densification in Gaussian splatting. In: *Eur. Conf. Comput. Vis. (ECCV)*. *Lect. Notes Comput. Sci.*, vol. 15121, pp. 347–362. Springer Nat. Switz. (Nov 2024)
49. Schonberger, J.L., Frahm, J.M.: Structure-from-motion revisited. In: *IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*. pp. 4104–4113. Inst. Electr. Electron. Eng. (IEEE), Las Vegas, NV, USA (Jun 2016)
50. Sharma, G., Rebain, D., Yi, K.M., Tagliasacchi, A.: Volumetric rendering with baked quadrature fields. In: *Eur. Conf. Comput. Vis. (ECCV)*. *Lect. Notes Comput. Sci.*, vol. 15121, pp. 275–292. Springer Nat. Switz. (Nov 2024)
51. Sharp, N., Jacobson, A.: Spelunking the deep. *ACM Trans. Graph.* **41**(4), 1–16 (Jul 2022)
52. Shin, S., Park, J., Cho, S.: Locality-aware Gaussian compression for fast and high-quality rendering. *arXiv [abs/2501.05757](https://arxiv.org/abs/2501.05757)* (2025)
53. Sigg, C., Weyrich, T., Botsch, M., Gross, M.: GPU-based ray-casting of quadratic surfaces. In: *Symposium on Point-Based Graphics*. Eurographics Assoc., Boston, MA, USA (2006)
54. Sun, C., Sun, M., Chen, H.T.: Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In: *IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*. pp. 5449–5459. Inst. Electr. Electron. Eng. (IEEE), New Orleans, LA, USA (Jun 2022)
55. Svitov, D., Morerio, P., Agapito, L., Del Bue, A.: Billboard splatting (BBSplat): Learnable textured primitives for novel view synthesis. In: *IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*. pp. 25029–25039. IEEE, Honolulu, HI, USA (Oct 2025)
56. Takikawa, T., Litalien, J., Yin, K., Kreis, K., Loop, C., Nowrouzezahrai, D., Jacobson, A., McGuire, M., Fidler, S.: Neural geometric level of detail: Real-time rendering with implicit 3D shapes. In: *IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*. pp. 11353–11362. IEEE, Nashville, TN, USA (Jun 2021)
57. Takikawa, T., Müller, T., Nimier-David, M., Evans, A., Fidler, S., Jacobson, A., Keller, A.: Compact neural graphics primitives with learned hash probing. In:

- ACM SIGGRAPH Asia Conf. Pap. pp. 1–10. ACM, Sydney, New South Wales, Aust. (Dec 2023)
58. Wang, Z., Shen, T., Gao, J., Huang, S., Munkberg, J., Hasselgren, J., Gojcic, Z., Chen, W., Fidler, S.: Neural fields meet explicit geometric representations for inverse rendering of urban scenes. In: IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR). pp. 8370–8380. IEEE, Vancouver, Can. (Jun 2023)
 59. Wang, Z., Shen, T., Nimier-David, M., Sharp, N., Gao, J., Keller, A., Fidler, S., Müller, T., Gojcic, Z.: Adaptive shells for efficient neural radiance field rendering. *ACM Trans. Graph.* **42**(6), 1–15 (Dec 2023)
 60. Wang, Z., Xu, D.: HyRF: Hybrid radiance fields for memory-efficient and high-quality novel view synthesis. In: Adv. Neural Inf. Process. Syst. (NeurIPS). vol. 39, pp. 1–24. Curran Assoc. Inc., San Diego, CA, USA (Dec 2025)
 61. Xu, Q., Xu, Z., Philip, J., Bi, S., Shu, Z., Sunkavalli, K., Neumann, U.: Point-NeRF: Point-based neural radiance fields. In: IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR). pp. 5428–5438. IEEE, New Orleans, LA, USA (Jun 2022)
 62. Xu, T.X., Hu, W., Lai, Y.K., Shan, Y., Zhang, S.H.: Texture-GS: Disentangling the geometry and texture for 3D Gaussian splatting editing. In: Eur. Conf. Comput. Vis. (ECCV). Lect. Notes Comput. Sci., vol. 15083, pp. 37–53. Springer Nat. Switz. (Oct 2024)
 63. Yariv, L., Hedman, P., Reiser, C., Verbin, D., Srinivasan, P.P., Szeliski, R., Barron, J.T., Mildenhall, B.: BakedSDF: Meshing neural SDFs for real-time view synthesis. In: ACM SIGGRAPH Conf. Proc. pp. 1–9. ACM, Los Angeles, CA, USA (Jul 2023)
 64. Ye, K., Shao, T., Zhou, K.: When Gaussian meets surfel: Ultra-fast high-fidelity radiance field rendering. *ACM Trans. Graph.* **44**(4), 1–15 (Jul 2025)
 65. Yu, A., Li, R., Tancik, M., Li, H., Ng, R., Kanazawa, A.: PlenOctrees for real-time rendering of neural radiance fields. In: IEEE/CVF Int. Conf. Comput. Vis. (ICCV). pp. 5732–5741. IEEE, Montréal, Can. (Oct 2021)
 66. Zhang, R., Isola, P., Efros, A.A., Shechtman, E., Wang, O.: The unreasonable effectiveness of deep features as a perceptual metric. In: IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR). pp. 586–595. IEEE, Salt Lake City, UT, USA (Jun 2018)
 67. Zhang, X., Chen, A., Xiong, J., Dai, P., Shen, Y., Xu, W.: Neural shell texture splatting: More details and fewer primitives. In: IEEE/CVF Int. Conf. Comput. Vis. (ICCV). pp. 25229–25238. IEEE, Honolulu, HI, USA (Oct 2025)
 68. Zhang, Y., Sun, G., Chen, C.: EGGS: Exchangeable 2D/3D Gaussian splatting for geometry-appearance balanced novel view synthesis. *Advances in Neural Information Processing Systems* **38**, 147665–147691 (2026)