Learning View-Dependent Splatting Kernels

HUAKENG DING* and ZHANPENG LIU*, State Key Lab of CAD and CG, Zhejiang University, China
FAN PEI, State Key Lab of CAD and CG, Zhejiang University, China
KUN ZHOU, State Key Lab of CAD and CG, Zhejiang University, China and Hangzhou Research Institute of Holographic and AI Technology, China
HONGZHI WU, State Key Lab of CAD and CG, Zhejiang University, China

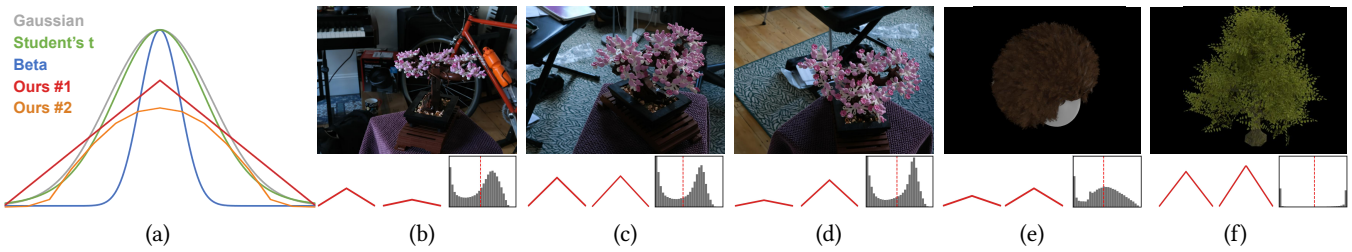

Fig. 1. We present a differentiable framework to automatically learn view-dependent 2D kernels in a splatting-based pipeline to improve reconstruction quality and representation efficiency for novel 3D view synthesis. We show 1D profile comparisons between our learned kernels and existing ones in (a). For each set of 4 images from (b) to (f), the top one is a view in a scene; the left 2 images in the bottom are the 1D profiles of two of our learned kernels; and the right image in the bottom shows a histogram of parameter d_1 (Eq. (6)) of our kernels corresponding to the view above it, with the red dotted line indicating the mean (the range of the horizontal axis is [0,1]). Our learned primitives vary across views from (b) to (d), and are adaptive to different scenes (b-d), (e) and (f).

We present a differentiable framework to automatically learn view-dependent 2D kernels in a splatting-based pipeline to improve reconstruction quality and representation efficiency for novel 3D view synthesis. Our volumetric primitive is defined as a bounding ellipsoid and a 3D-kernel latent vector. We first learn a projection network to output a 2D-kernel latent, taking the attributes of the ellipsoid and the 3D-kernel latent as input. Next, the result is sent to a decoder to produce a radially symmetric 2D kernel in terms of Mahalanobis distance, bounded by the projected ellipsoid. The neural networks along with per-primitive attributes are jointly optimized. The effectiveness of our approach is demonstrated on standard benchmarks, comparing favorably against state-of-the-art techniques on both analytical and learned kernels. Finally, we extend the idea to learn general 2D kernels for 2D splatting as well as image representation.

CCS Concepts: • **Computing methodologies** → **Rasterization**; **Shape modeling**.

ACM Reference Format:
Huakeng Ding, Zhanpeng Liu, Fan Pei, Kun Zhou, and Hongzhi Wu. 2026. Learning View-Dependent Splatting Kernels. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers (SIGGRAPH Conference Papers '26), July 19–23, 2026, Los Angeles, CA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3799902.3811064

*Equal contributions. Corresponding authors: {kunzhou,hwu}@acm.org

Authors' Contact Information: Huakeng Ding; Zhanpeng Liu, State Key Lab of CAD and CG, Zhejiang University, Hangzhou, Zhejiang, China; Fan Pei, State Key Lab of CAD and CG, Zhejiang University, China; Kun Zhou, State Key Lab of CAD and CG, Zhejiang University, China and Hangzhou Research Institute of Holographic and AI Technology, China; Hongzhi Wu, State Key Lab of CAD and CG, Zhejiang University, China.

SIGGRAPH Conference Papers '26, Los Angeles, CA, USA

ACM ISBN 979-8-4007-2554-8/2026/07
https://doi.org/10.1145/3799902.3811064

1 Introduction

Since its debut in [Kerbl et al. 2023], 3D Gaussian Splatting (3DGS) has achieved tremendous success in image-based representation and rendering. First, a standard 3D Gaussian *kernel* is scaled, rotated, and translated to produce a semi-transparent anisotropic *shape*. This forms a basic primitive, a collection of which are used to model an input scene. Next, this 3D shape is approximately projected to the 2D screen (i.e., splatting), and blended according to a sorted order via efficient tile-based differentiable rendering. All related parameters are optimized with respect to multi-view input images.

The splatting kernel is a crucial factor in the reconstruction quality and representation efficiency of any related pipeline. Consider a simple 2D example: it is far more efficient to represent a rectangle with a set of squares than Gaussians. Researchers devote considerable efforts to this topic, by *manually* replacing the Gaussians in [Kerbl et al. 2023] with various analytical kernels [Chen et al. 2024; Hamdi et al. 2024; Liu et al. 2025b; Zhu et al. 2025]. This series of work naturally leads to two fundamental questions: (1) *what are optimal splatting kernels?* and (2) *how can we automatically discover them?*

Recent work explores *automatically* learning alternative volumetric primitives from data [Held et al. 2025c]. However, closed-form equations do not exist for computing line integrals over general 3D primitives, a step in splatting. So a "slicing" approximation is adopted to avoid the expensive numerical integration. On the other hand, exact integral can be avoided/computed, by restricting the learned primitive to be a planar shape [Huang et al. 2025], or represented by a specific type of network [Zhou et al. 2025]. In both cases, the expressiveness of the representation is limited. It is yet

unclear how to systematically learn general splatting kernels in a data-driven fashion.

In this paper, we present a differentiable framework to automatically learn view-dependent 2D splatting kernels, as an implicit representation of volumetric primitives. Our approach avoids the computation of line integral, by explicitly modeling the integral results. We also introduce learned view-dependency to our kernels to improve reconstruction quality. Specifically, we start with a bounding ellipsoid as in vanilla 3DGS. Next, a neural 3D-to-2D projection is performed by a bounding-ellipsoid-aware multi-layer perceptron (MLP), which takes as input a latent vector that represents a 3D kernel, and outputs another one that represents a 2D kernel after the projection. Afterward, a decoder transforms the resulting latent into a radially symmetric 2D kernel in terms of Mahalanobis distance, as the final splatting shape. This shape is passed on to the rest of a standard splatting pipeline for further processing (e.g., coloring, alpha-blending).

The effectiveness of our framework is demonstrated on 4 standard benchmarks. In both 3D/2D splatting, our approach is the best or second-best in all reconstruction quality metrics across all benchmarks, compared with state-of-the-art techniques. Moreover, our volumetric primitives also demonstrate superior representation efficiency, as well as good scalability with respect to the number of primitives. We also apply our planar primitives to learn to efficiently represent 2D images. Our code is publicly available at https://optkernel.github.io.

2 Related Work

This section focuses on research on improving the shapes of splatting primitives in the context of novel view synthesis. We divide existing work based on whether its primitives are volumetric or planar. We do not cover other aspects of improvements, including but not limited to density control [Kheradmand et al. 2024; Park et al. 2025], appearance [Bi et al. 2024; Wang et al. 2024; Yang et al. 2024], and anti-aliasing [Yu et al. 2024a], as they are orthogonal to our focus. Interested readers are directed to excellent surveys [Bao et al. 2024; Chen and Wang 2024; Fei et al. 2025] for a broader view of the topic.

2.1 Volumetric Primitives

2.1.1 Analytical Kernels. The majority of related work falls into this category, which replaces 3D Gaussians in vanilla 3DGS with alternative, explicitly defined volumetric primitives or implicitly defined ones as view-independent 2D kernels (after projection).

For the former, the 3D density distribution of a primitive is analytically defined. Notable examples include generalized exponential [Hamdi et al. 2024], Student's t [Zhu et al. 2025], and deformable Beta kernels [Liu et al. 2025b]. In addition, these kernels come with analytical formulas for computing splatting results, leading to high rendering performance. While reconstruction quality and representation efficiency are constantly improving, related approaches are limited in terms of expressiveness: their shape variations are entirely determined by hand-crafted, analytical equations, and thus cannot fully adapt to input scenes in a data-driven fashion.

On the other hand, existing work implicitly models volumetric primitives as view-independent 2D splatting kernels, such as linear [Chen et al. 2024], overlapping Gaussians [Liu et al. 2025a], B-spline [Thomas and Seelamantula 2025], or even Gaussian multiplied with a 2D alpha texture [Chao et al. 2025]. However, no 3D consistency among 2D kernels across different views is imposed, resulting in suboptimal reconstructions. In comparison, our learned view-dependency of kernels serves as a form of data-driven consistency, leading to improved reconstructions.

It is worth mentioning that most of the aforementioned kernels are *radially symmetric*, the reason of which will be elaborated in Sec. 5.3.2. Also note that while the *shape* of a popular primitive [Hamdi et al. 2024; Kerbl et al. 2023; Liu et al. 2025b; Zhu et al. 2025] is view-dependent, its *kernel* is view-independent as a function of Mahalanobis distance (e.g., according to Eq. (2) for Gaussians). In comparison, our kernel is view dependent, which brings in extra degrees of freedom, as illustrated in Fig. 3.

2.1.2 Learned Representations. Few papers explore learning optimal splatting primitives directly from data. Held et al. [2025c] optimize the vertices of a convex shape as a primitive, and simplify the line integral for splatting as a constant value, which gets further softened near the boundary. Zhou et al. [2025] essentially splat neural fields as primitives. They are represented as a *specific type* of shallow MLPs, so that their exact line integrals can be derived as closed-form equations.

Unlike the above work, we do not explicitly model the density distribution of a 3D primitive. Instead, the view-dependent projection of a primitive is modeled, which avoids the challenging line integral computation over a general 3D density field. Moreover, our approach is not tied to any specific type of neural networks.

2.2 Planar Primitives

2.2.1 Analytical Kernels. Starting with the pioneering work of 2DGS [Huang et al. 2024], researchers improve the original 2D surfels with alternative analytical 2D kernels, such as Hermite polynomials [Yu et al. 2024b], Gabor [Watanabe et al. 2025], and even triangles [Held et al. 2025a,b]. Similar to related work on volumetric primitives, existing approaches here are limited in expressiveness: the splatting shape variations are determined by analytical equations and cannot fully adapt to input scenes.

2.2.2 Learned Representations. Svitov et al. [2024] multiply a learnable 2D RGBA texture with a 2D Gaussian primitive to introduce spatial variations with a shared uv paramterization. This can be viewed as a refined splatting shape with the details from the alpha channel of the texture. Note that research which augments a primitive with an RGB texture does not change its shape [Papantonakis et al. 2025; Rong et al. 2024; Xu et al. 2024; Zhang et al. 2025], and thus is orthogonal to our work. Recently, Huang et al. [2025] propose a learnable 2D radial kernel with a number of control points corresponding to different angles.

In comparison, our work is a unified approach for both volumetric and planar primitives, while the methods above are limited to handle the latter. Moreover, the view-dependency of our kernels achieves higher quality with existing work, where the 2D kernels

(prior to affine transformations) are view independent. Note that our planar primitives (Sec. 5.4) are asymmetric and can be viewed as a generalized version of [Huang et al. 2025], as Eq. (8) is an MLP (i.e., we do not assume parametric curves as in their paper).

3 Preliminaries

This section briefly describes the key terms in 3D/2D Gaussian splatting that are most related to the definitions of our kernels in the subsequent text.

For 3DGS [Kerbl et al. 2023], the attributes of a 3D Gaussian primitive include its mean $\boldsymbol{\mu}_{3D}$ and covariance $\Sigma_{3D} = \mathbf{RSS}^{\mathbf{T}}\mathbf{R}^{\mathbf{T}}$, where $\mathbf{S}$ is a scaling matrix and $\mathbf{R}$ is a rotation one. The projection of a Gaussian onto an image plane is approximated via EWA splatting [Zwicker et al. 2001]. The resulting 2D covariance matrix Σ_{2D} is computed as:

$$\Sigma_{2D} = \mathbf{JW}\Sigma_{3D}\mathbf{W}^T\mathbf{J}^T, \tag{1}$$

where $\mathbf{W}$ is the viewing transformation, and $\mathbf{J}$ is the Jacobian of affine approximation of projection. We denote the projected mean as $\boldsymbol{\mu}_{2D}$. The screen-space shape of a projected primitive can be defined as:

$$G(\mathbf{x}) = \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_{2D})^T\Sigma_{2D}^{-1}(\mathbf{x}-\boldsymbol{\mu}_{2D})\right) = \exp(-\frac{1}{2}r^2), \tag{2}$$

Here $\mathbf{x}$ are the coordinates of a pixel, and r^2 is the squared Mahalanobis distance. The latter can be viewed as a transform from an anisotropic Gaussian distribution to a standard isotropic normal distribution, which *solely depends* on r^2.

For 2DGS [Huang et al. 2024], its primitives are oriented disks. The screen-space shape of a projected primitive is defined as:

$$G'(\mathbf{x}) = \exp\left(-\frac{1}{2}(u^2+v^2)\right), \tag{3}$$

where (u, v) are the local 2D coordinates of the intersection of a camera ray corresponding to $\mathbf{x}$ against a primitive on its tangent plane. This native (u, v) parameterization in 2DGS makes it convenient to define spatial variations on a primitive (e.g., general 2D kernel or 2D texture).

4 Overview

The geometric attributes of our primitive consist of a 3D bounding ellipsoid, similar to the truncated Gaussian in vanilla 3DGS, and a latent vector that represents a 3D kernel (Sec. 5.1). For each primitive, our pipeline first projects the ellipsoid to the image plane as a 2D bounding ellipse. Next, we project our 3D kernel to 2D, by transforming the 3D-kernel latent into one that represents a 2D kernel, via a projection MLP that is also aware of attributes of the ellipsoid (Sec. 5.2). The result is then sent to a decoder to output a radially symmetric 2D kernel in terms of Mahalanobis distance, bounded by the 2D ellipse, as the final splatting shape (Sec. 5.3). This shape is passed on to the rest of any standard splatting-based pipeline for further processing (e.g., coloring, alpha-blending). Please refer to Fig. 2 for an illustration.

5 Our Approach

5.1 Volumetric Primitives

To define its geometry, each of our volumetric primitives includes a 3D bounding ellipsoid and a 3D-kernel latent: the attributes for the ellipsoid consist of its center $\boldsymbol{\mu}_{3D}$, a 3D vector of scaling factors $\mathbf{s}$ (to represent the scaling matrix $\boldsymbol{S}$), and a 4D quaternion q (to represent the rotation matrix $\boldsymbol{R}$); the latent $\mathbf{z}_{3D}$ is a vector (whose dimension is 5 in most of our experiments), which stores the parameters of an implicit 3D kernel.

We employ a bounding ellipsoid for the following reasons. First, it directly models scaling/rotation/translation transformations to the kernel; our MLPs can focus on learning "normalized" shapes, prior to applying affine transformations. We believe this is a good balance between quality and performance. A more aggressive approach that directly learns shapes would probably require larger (slower) MLPs to handle the extra complexity. Second, the ellipsoid serves as an efficient geometric proxy for sorting and culling. Its similarity with truncated Gaussians in vanilla 3DGS also allows us to reuse much of that pipeline (Sec. 3).

The learnable appearance of each primitive is similarly defined as in 3DGS, which includes an overall opacity and the parameters of a color model [Kerbl et al. 2023; Liu et al. 2025b].

5.2 Neural Projection

Below we describe how to project our volumetric primitive to an image plane, for a given camera specification. First, for the 3D bounding ellipsoid, we directly project it with Eq. (1) to obtain a 2D bounding ellipse.

Next, similar to how Eq. (1) projects a 3D ellipsoid to a 2D ellipse, we learn a global projection MLP Φ_{proj} to transform the implicit 3D kernel (i.e., $\mathbf{z}_{3D}$) to a latent vector $\mathbf{z}_{2D}$ (whose dimension is 5 in most experiments), which represents a 2D kernel after the projection:

$$\mathbf{z}_{2D} = \Phi_{\text{proj}}\left(\mathbf{z}_{3D}, \boldsymbol{\mu}_{3D}^{\text{cam}}, \mathbf{s}, \mathbf{R}^{\text{cam}}\right). \tag{4}$$

A straightforward solution might take the 3D geometry (represented as $\mathbf{z}_{3D}$ in our case) and a camera specification only as input, as in the majority of related work. However, we discover through experiments (Tab. 3) that adding other factors as input to Φ_{proj} brings quality improvement while maintaining generalization ability.

Specifically, we add the camera-space coordinates of the ellipsoidal center, $\boldsymbol{\mu}_{3D}^{\text{cam}}$, to provide spatial awareness for the MLP. Also, the scaling factors, $\mathbf{s}$, add scale awareness. Finally, we send in $\mathbf{R}^{\text{cam}}$, the rotation matrix of our primitive in the camera space, because it is a compact representation of the orientation of the bounding ellipsoid/primitive in the camera space, efficiently encoding the 3D view information. By default, Φ_{proj} is implemented as a lightweight, 4-layer MLP, with 64 neurons per hidden layer and leaky-ReLU activation after each layer except for the last. The global MLP is jointly trained with the kernel decoder, which will be introduced in the next subsection, with respect to all input images.

5.3 Kernel Decoder

5.3.1 Definition. Our kernel decoder, Φ_{dec}, decodes a radially symmetric 2D kernel from a latent $\mathbf{z}_{2D}$. Specifically, for a given pixel, we first compute its squared Mahalanobis distance r^2 with respect to

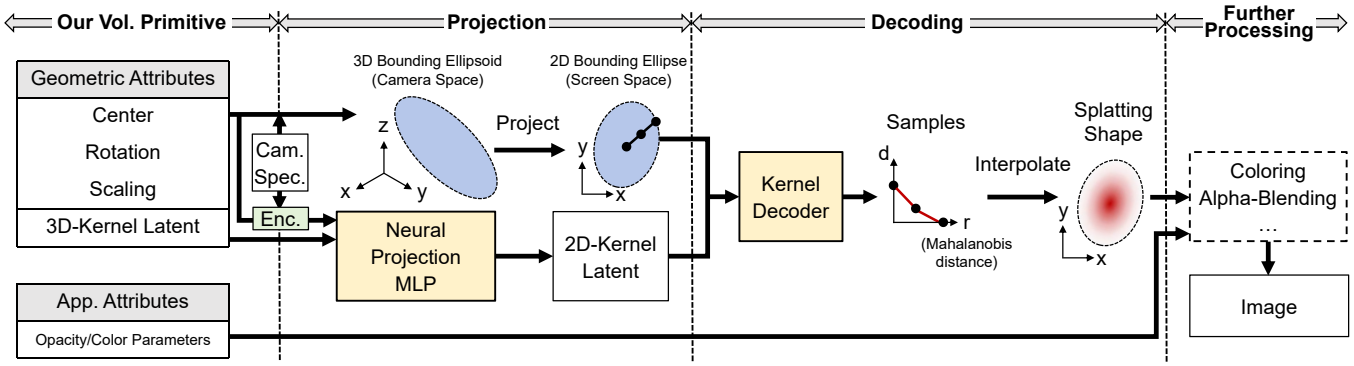

Fig. 2. Our pipeline for splatting volumetric primitives. For each primitive, we first project the bounding ellipsoid to the image plane as a 2D bounding ellipse. Next, we project our 3D kernel to 2D, by transforming the 3D-kernel latent into one that represents a 2D kernel, via a projection MLP that is also aware of attributes of the ellipsoid. The result is then sent to a decoder to output a radially symmetric 2D kernel in terms of Mahalanobis distance, bounded by the 2D ellipse, as the final splatting shape. To accelerate the process, we pre-sample the 1D profile of the 2D kernel and draw the splat by linearly interpolating the samples. Finally, the shape is passed on to the rest of a standard splatting-based pipeline for further processing (e.g., coloring, alpha-blending). Vol. = volumetric, Enc. = encoding, and Cam. Spec. = camera specification.

the projected primitive center $\boldsymbol{\mu}_{2D}$ according to Eq. (2), based on the attributes from the corresponding 2D bounding ellipse. Next, we replace the analytical shape function in related work (e.g., Gaussian [Kerbl et al. 2023] or Beta [Liu et al. 2025b]) with a global MLP Φ_{dec}. It takes r^2 and $\mathbf{z}_{2D}$ as input, and outputs an opacity value d as the result:

$$d = \Phi_{\text{dec}}(r^2, \mathbf{z}_{2D}). \tag{5}$$

This MLP essentially describes a 1D profile of a radially symmetric 2D splatting kernel, for r^2 in the range of [0,1]. Our default implementation of Φ_{dec} is a lightweight, 3-layer MLP, with 4 neurons per hidden layer and leaky-ReLU activation after each layer except for the last. The final layer is a sigmoid function to bound the output.

5.3.2 Justification of Radial Symmetry. The reason we choose to output a radially symmetric 2D kernel, rather than a general one, is due to the theoretical discontinuity in mapping an arbitrary 3D view condition to a 2D frame (on which a general 2D kernel can be defined). Mathematically, this is known as the hairy ball theorem [Hatcher 2002]. In our pilot study, we apply a smoothing trick to alleviate this discontinuity. But it results in a rapid rotation of the 2D kernel around certain view directions, which is disturbing during animation. On the other hand, by restricting our output to a radially symmetric 2D kernel, we avoid the aforementioned mapping in the first place, as the 2D frame is no longer needed. Note that this challenge is also mentioned in existing work on learnable kernels (cf. Sec. 3.4 of [Huang et al. 2025]).

5.3.3 Acceleration. Directly executing Φ_{dec} for every pixel within our bounding ellipse could be computationally expensive. To improve rendering performance, we pre-sample the 1D profile that Φ_{dec} represents, and perform interpolation during rendering, substantially reducing the runtime cost of calling Φ_{dec}. This essentially decouples the number of pixels within our primitive from the number of calls to Φ_{dec}.

Specifically, for each primitive, we place k uniform samples of r to cover the range of [0,1], which are denoted as $\{r_i\}_{1\le i\le k}$. Next, we run Φ_{dec} only at these samples, and store the results as $\{d_i\}_{1\le i\le k}$:

$$d_i = \Phi_{\text{dec}}(r_i^2, \mathbf{z}_{2D}). \tag{6}$$

Finally, during rasterization, for any r computed from a particular pixel, we find its closet samples such that $r_c \le r \le r_{c+1}$, and efficiently compute the final result via linear interpolation:

$$d = (1-\lambda)d_c + \lambda d_{c+1}, \text{ with } \lambda = \frac{r - r_c}{r_{c+1} - r_c}. \tag{7}$$

Note that our approach is not tied to any type of interpolation. More sophisticated interpolation can also be plugged in, to bring higher-order smoothness at the cost of more computational budget. We use $k = 2$ in most experiments, as it suffices to produce results with high quality, according to Sec. 6.2.

5.4 Extension to Planar Primitives

In addition to volumetric primitives, our idea can also be applied to learn general 2D kernels for planar primitives in the context of 2D splatting [Huang et al. 2024]. To reduce repetitive text, below we primarily describe the differences in defining/processing our planar primitives versus volumetric ones.

First, to define the geometry, each of our planar primitives includes a bounding ellipse, and a 10D latent code $\mathbf{z}_{2D}^{\text{prior}}$ that represents a general 2D kernel *prior* to projection. Next, similar to Sec. 5.2, we learn a projection MLP with the same architecture as Φ_{proj} in Sec. 5.2, to transform $\mathbf{z}_{2D}^{\text{prior}}$ to another 10D latent $\mathbf{z}_{2D}^{\text{after}}$, which implicitly represents a general 2D kernel after projection. Finally, for each pixel within the projected bounding ellipse, a kernel decoder takes $\mathbf{z}_{2D}^{\text{after}}$ and local 2D coordinates as input, and outputs an opacity value d:

$$d = \Phi_{\text{dec}}^{\text{planar}}(u, v, \mathbf{z}_{2D}^{\text{after}}). \tag{8}$$

Here (u, v) are the 2D coordinates of the intersection of a camera ray corresponding to a pixel against our planar primitive on its tangent plane. The architecture of $\Phi_{\text{dec}}^{\text{planar}}$ is almost the same as its counterpart Φ_{dec} for volumetric primitives, with the number

of neurons per hidden layer doubled to 8, to cope with the extra complexity of a general 2D kernel.

5.5 Implementation Details

We pre-train neural projection and kernel decoder prior to 3D reconstruction. To start, the 5D latent code $\mathbf{z}_{3D}$ is initialized to all zeros, while all weights in Φ_{proj} and Φ_{dec} are initialized with [He et al. 2015]. We then jointly optimize Φ_{proj} and Φ_{dec}. To synthesize training data, we randomly sample $\boldsymbol{\mu}_{3D}^{\text{cam}}$, $\mathbf{s}$, and $\mathbf{R}^{\text{cam}}$, and then randomly sample r to obtain target profile sample according to Eq. (6). In the volumetric case, we employ a 1D cosine function as the target profile (i.e., $d = \cos(\frac{\pi}{2}r^2)$); for the planar case, 2D Gaussian surfels are used. We find that pre-training results in higher reconstruction quality than without it. Please see Tab. 5 for a detailed comparison.

For training, we employ a two-stage strategy to improve stability, as our primitives have higher degrees of freedom compared with standard 3D/2DGS. After pre-training, in the first stage of training, we freeze neural projection and kernel decoder for the first 2000 iterations for a scene whose initial point cloud comes from SfM/random points, respectively. In the second stage, all parameters are optimized jointly with respect to the loss function defined in [Liu et al. 2025b], which is an image loss plus the regularization terms for opacity and scale. Note that we also adopt the kernel-agnostic MCMC density control strategy from the same paper, as it is effective and not tied to specific types of kernels. The learning rate for neural projection/kernel decoder decays exponentially from 1.6×10^{-4} to 1.6×10^{-6}. For other parameters, the learning rates are set according to [Kheradmand et al. 2024]. We use the Adam optimizer to train 30K iterations in each experiment.

6 Results and Discussions

We conduct most experiments on a workstation with dual AMD EPYC 7763 CPUs, 768GB RAM, and an RTX 4090 GPU. The only exception is the reconstruction experiments with planar primitives on Drjohnson and Treehill scenes, where an RTX PRO 6000 is used due to the memory requirement.

We test on all 21 scenes from 4 standard benchmarks, including 9 scenes from Mip-NeRF 360 [Barron et al. 2022], 2 from Tanks & Temples [Knapitsch et al. 2017], 2 from Deep Blending [Hedman et al. 2018], and 8 from NeRF Synthetic [Mildenhall et al. 2020]. All scenes are physically captured images, with the exception of NeRF Synthetic. For quantitative assessments of reconstruction quality, we compute PSNR, SSIM, and LPIPS averaged over all test images. Our average training time is 53 minutes.

6.1 Comparisons

6.1.1 Reconstruction Quality. We compare our volumetric primitives with state-of-the-art 3D splatting techniques, including (1) analytical kernels: 3DGS [Kerbl et al. 2023], 3DGS-MCMC [Kheradmand et al. 2024], SSS [Zhu et al. 2025] and DBS [Liu et al. 2025b], and (2) learned representations: SplatNet [Zhou et al. 2025] and 3DCS [Held et al. 2025c]. For 2D splatting, we compare our planar primitives with 2DGS [Huang et al. 2024] as well as learned representations: BBSplat [Svitov et al. 2024] and DRK [Huang et al. 2025].

Tab. 1 shows quantitative comparison results. Note that for our volumetric/planar primitives, two variants with different color models, spherical harmonics (SH) [Kerbl et al. 2023] or spherical Beta (SB) [Liu et al. 2025b], are included. In both 3D/2D splatting, our approach is the *best or second-best* in all quality metrics across all benchmarks. Please also see the supplemental material for a detailed, per-scene breakdown. For qualitative comparisons between some top performing approaches, please refer to Fig. 7 as well as the accompanying video.

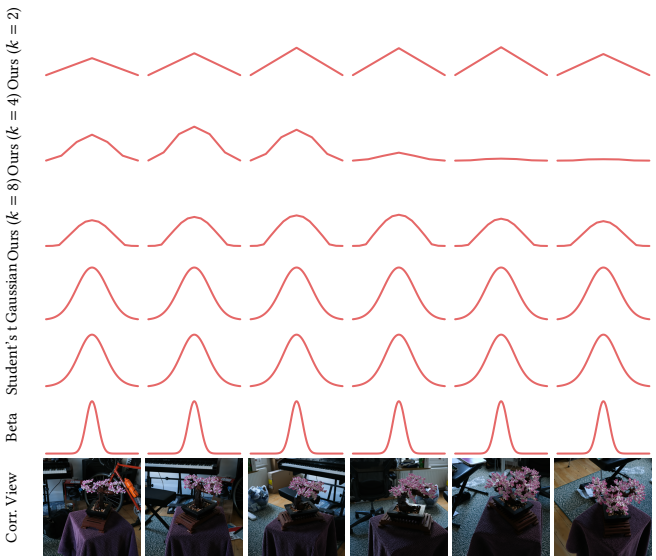

Fig. 3. Visualization of 1D profiles of different radially symmetric kernels. From the top row to bottom, our kernels (k=2, 4, 8), Gaussian [Kerbl et al. 2023], Student's t [Zhu et al. 2025] and Beta kernel [Liu et al. 2025b], and the corresponding views. All profiles in a column correspond to the same view shown at the bottom row. Note that our kernels vary with the view, while the kernels in existing work are view-independent.

6.1.2 Visualization. We first visualize the 1D profiles of individual radially symmetric kernels in different approaches, including variants of ours with different k, vanilla 3DGS, SSS and DBS in Fig. 3. Here one key difference is that our kernels are *view-dependent* by learning, while the ones (prior to any affine transformations described in Sec. 3) in existing work are view-independent.

In Fig. 5, we visualize the statistics of our learned kernels across views and scenes. The histogram of $d_1 = \Phi_{\text{dec}}(0, \mathbf{z}_{2D})$ of each kernel, as defined in Eq. (6), not only changes with the view due to our learned view dependency, but it also varies across different scenes, demonstrating our ability to *adapt* the geometry of kernels to input data.

6.1.3 Representation Efficiency. We compare the representation efficiency between our approach and state-of-the-art work of 3DGS, 3DGS-MCMC, SSS and DBS in Tab. 2. Specifically, following the protocol detailed in the supplemental material of [Liu et al. 2025b], we disable all view-dependent color components to focus on the impact of the geometry of kernels. Next, we perform comparisons with

Table 1. Quantitative comparisons between our approach and state-of-the-art techniques on 4 standard datasets: Mip-NeRF360 [Barron et al. 2022], Tanks & Temples [Knapitsch et al. 2017], Deep Blending [Hedman et al. 2018] and NeRF Synthetic [Mildenhall et al. 2020]. We highlight the best, second-best, and third-best results in red, orange, and yellow, respectively. Baseline results are obtained from the original papers whenever available; otherwise, we run the official implementation and list the results in *italics*. SH = spherical harmonics, SB = spherical Beta.

		Mip-NeRF360			Tanks & Temples			Deep Blending			NeRF Synthetic		
		PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
2D Splatting	2DGS [Huang et al. 2024]	27.04	0.805	0.223	*23.13*	*0.831*	*0.212*	*29.50*	*0.902*	*0.257*	33.07	*0.967*	*0.035*
	BBSplat [Svitov et al. 2024]	*26.88*	*0.794*	*0.253*	*23.59*	*0.851*	*0.150*	*29.28*	*0.902*	*0.245*	*32.00*	*0.956*	*0.044*
	DRK [Huang et al. 2025]	26.76	0.787	0.236	*22.49*	*0.867*	*0.243*	*29.42*	*0.922*	*0.322*	*33.22*	*0.968*	*0.043*
	Ours (2D+SH)	28.42	0.824	0.219	24.50	0.857	0.161	30.18	0.911	0.248	34.57	0.973	0.026
	Ours (2D+SB)	28.44	0.821	0.225	24.59	0.857	0.163	30.26	0.910	0.250	34.49	0.972	0.027
3D Splatting	3DGS [Kerbl et al. 2023]	27.20	0.815	0.214	23.15	0.840	0.183	29.41	0.903	0.243	33.31	0.969	0.037
	3DGS-MCMC [Kheradmand et al. 2024]	28.29	0.840	0.210	24.29	0.860	0.190	29.67	0.895	0.320	33.80	0.970	0.040
	3DCS [Held et al. 2025c]	27.29	0.802	0.207	23.95	0.851	0.157	29.81	0.902	0.237	*31.68*	*0.958*	*0.048*
	SplatNet [Zhou et al. 2025]	27.21	0.791	0.216	23.59	0.846	0.162	29.20	0.892	0.264	33.34	0.967	0.032
	SSS [Zhu et al. 2025]	*28.25*	*0.838*	*0.171*	24.87	0.873	0.138	30.07	0.907	0.247	*34.29*	*0.971*	*0.029*
	DBS [Liu et al. 2025b]	28.60	0.844	0.182	24.79	0.868	0.148	30.10	0.910	0.240	34.64	0.973	0.028
	Ours (3D,w/o SH) for ablation	28.53	0.835	0.192	25.24	0.872	0.148	30.48	0.912	0.234	34.37	0.972	0.028
	Ours (3D+SH)	28.73	0.842	0.185	25.42	0.877	0.141	30.52	0.913	0.233	34.72	0.973	0.027
	Ours (3D+SB)	28.82	0.840	0.189	25.35	0.874	0.145	30.54	0.913	0.237	34.58	0.973	0.027

Table 2. Comparisons on representation efficiency and runtime performance. We compare our approach with 3DGS [Kerbl et al. 2023], 3DGS-MCMC [Kheradmand et al. 2024], SSS [Zhu et al. 2025] and DBS [Liu et al. 2025b] on Tanks & Temples dataset, with 3 different memory footprint for primitives. Quantitative measures, including reconstruction quality(PSNR), primitive count and rendering speed(FPS) are reported. Note that our memory footprint does not change with k. #Prim. = number of primitives.

Memory	~25MB			~50MB			~100MB		
	PSNR↑	#Prim.	FPS↑	PSNR↑	#Prim.	FPS↑	PSNR↑	#Prim.	FPS↑
3DGS	23.12	446K	295	23.36	893K	232	23.38	1.79M	198
3DGS-MCMC	22.83	446K	301	23.40	893K	235	23.68	1.79M	165
SSS	23.90	391K	130	24.20	781K	104	24.38	1.56M	80
DBS	23.57	417K	319	23.79	833K	249	24.24	1.67M	178
Ours (k=2)	22.80	328K	123	24.18	657K	80	25.22	1.32M	42
Ours (k=4)	24.08	328K	102	24.66	657K	67	25.15	1.32M	36
Ours (k=8)	24.19	328K	96	24.70	657K	55	25.04	1.32M	28

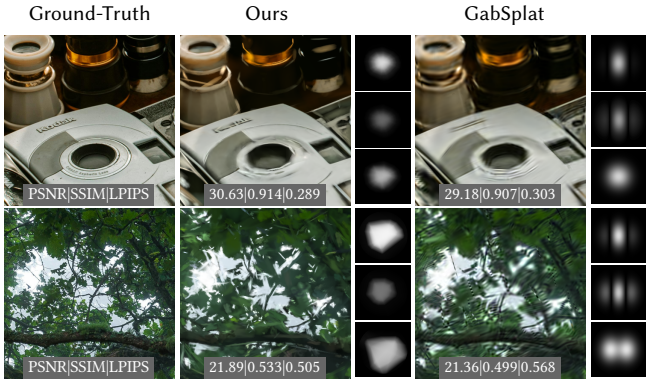

Fig. 4. Comparisons between our approach and GabSplat [Wurster et al. 2024] on efficiently representing 2D images, using a similar memory footprint for primitives. We show close-up views alongside with visualization of 2D kernel samples for each approach. Quantitative errors in PSNR, SSIM and LPIPS are reported at the bottom of each related image. Note that unlike in 3D splatting, our learned kernels here are general 2D ones and not restricted to being radially symmetric.

a similar memory footprint for primitives from different methods. Our approach results in higher quality compared to the baselines, demonstrating improved representation efficiency. Moreover, our representation is scalable with respect to the size of available memory footprint. Please also refer to Fig. 8 for qualitative comparisons.

6.1.4 Extension to Image Representation. We further extend our learned 2D kernels to efficient representation of generic 2D images in Fig. 4. Our approach compares favorably against a state-of-the-art method that uses hand-crafted Gabor kernels [Wurster et al. 2024], with a similar memory footprint for primitives. Qualitative and quantitative results, along with visualization of primitives are shown in the figure.

6.2 Ablations

We first evaluate the impact of different sets of input to the projection MLP Φ_{proj} in Tab. 3. Our current choice of input achieves the best overall quality, by making the MLP spatial-, scale- and view-aware.

Next, Tab. 4 evaluates the impact of various factors (the dimension of $\mathbf{z}_{3D}/\mathbf{z}_{2D}$, and the width of $\Phi_{\text{proj}}/\Phi_{\text{enc}}$) over the reconstruction quality and memory footprint for primitives. Our default choice of these factors strikes a good balance between quality and memory footprint.

Moreover, in the last 3 rows of Tab. 2, we evaluate the impact of k (Sec. 5.3.3) over reconstruction quality with different memory footprints for primitives. With a small footprint, increasing k results in higher reconstruction quality, as the expressiveness of learned kernels is increased. With a large footprint, this trend is not observed. We believe the reason is that the average screen-space size for each

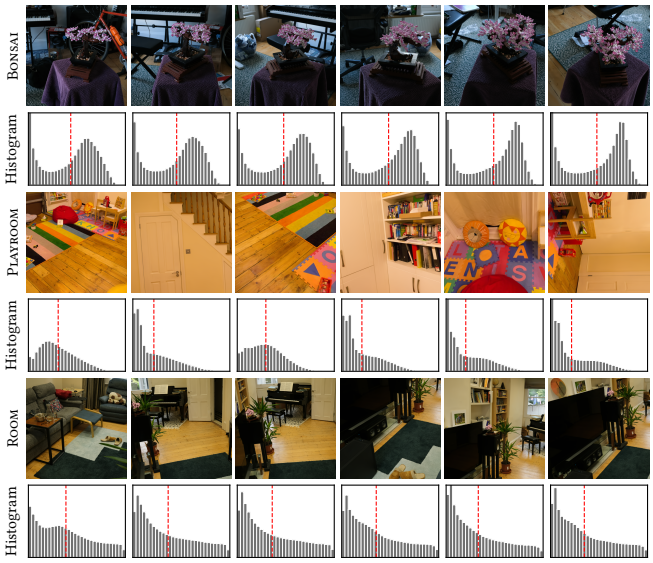

Fig. 5. Distributions of our kernels across different scenes and views. Each column shows a different view. We show a histogram of d_1 (Eq. (6)) of our learned kernels corresponding to the image above it, with the red dotted line indicating the mean; the range of the horizontal axis is [0,1].

Table 3. Ablation on the input to Φ_{proj} on two datasets. Quantitative errors in PSNR, SSIM and LPIPS are reported. Note that the last row corresponds to our current choice of the input.

Input to Φ_{proj}	Tanks & Temples			Deep Blending		
	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
$\mathbf{z}_{3D}$	24.92	0.873	0.144	29.96	0.906	0.241
$\mathbf{z}_{3D}$, $\boldsymbol{\omega}_o$	24.97	0.874	0.144	30.09	0.911	0.235
$\boldsymbol{\mu}_{3D}^{\text{cam}}$, $\mathbf{s}$, $\mathbf{R}^{\text{cam}}$	24.82	0.871	0.144	30.49	0.913	0.235
$\mathbf{z}_{3D}$, $\mathbf{s}$, $\mathbf{R}^{\text{cam}}$	25.37	0.876	0.143	30.06	0.914	0.239
$\mathbf{z}_{3D}$, $\boldsymbol{\mu}_{3D}^{\text{cam}}$, $\mathbf{R}^{\text{cam}}$	25.45	0.876	0.143	30.18	0.911	0.239
$\mathbf{z}_{3D}$, $\boldsymbol{\mu}_{3D}^{\text{cam}}$, $\mathbf{s}$	24.53	0.832	0.175	30.25	0.914	0.245
$\mathbf{z}_{3D}$, $\boldsymbol{\mu}_{3D}^{\text{cam}}$, $\mathbf{s}$, $\mathbf{R}^{\text{cam}}$	25.42	0.877	0.141	30.52	0.913	0.233

Table 4. Ablations on the dimension of $\mathbf{z}_{3D}/\mathbf{z}_{2D}$ and the width of $\Phi_{\text{proj}}/\Phi_{\text{enc}}$ over the reconstruction quality and memory footprint on two datasets. Quantitative measures in PSNR and MB are listed. Mem. = memory footprint.

Variants	Tanks & Temples		Deep Blending	
	PSNR↑	Mem.(MB)↓	PSNR↑	Mem.(MB)↓
$\dim(z_{3D})$=1	25.10	360.04	30.19	660.04
$\dim(z_{3D})$=10	25.52	414.04	30.12	759.04
$\dim(z_{2D})$=1	24.94	384.04	29.37	704.04
$\dim(z_{2D})$=10	25.46	384.04	29.97	704.04
width(Φ_{proj})=32	25.14	384.01	30.13	704.01
width(Φ_{proj})=128	25.50	384.15	30.33	704.15
width(Φ_{dec})=8	25.47	384.04	30.48	704.04
width(Φ_{dec})=2	24.75	384.04	30.41	704.04
Ours	25.42	384.04	30.47	704.04

splat reduces with the increase in primitive count (i.e., memory footprint); simple conical-hat shaped kernels ($k = 2$) suffice to produce satisfactory results, while achieving faster rendering speed compared with using a larger k. Note that our experiments are consistent with the main conclusion in [Celarek et al. 2025].

We also investigate the impact of different target shapes for pre-training our kernel decoder Φ_{dec} in Tab. 5. According to the table, pre-training outperforms no pre-training; we select the cosine shape for our pipeline (Sec. 5.5), as it leads to the best results. Note that the learning rate is multiplied by 10 in no pre-training experiments.

Table 5. Impact of different target shapes for pre-training Φ_{dec}. Reconstruction errors on the Tanks & Temples dataset in PSNR are listed in the table.

	no pre-train	Gaussian	polynomial	linear	cosine
$k = 2$	23.18	25.24	25.37	25.22	25.42
$k = 4$	23.72	25.04	25.37	25.26	25.34
$k = 8$	18.54	25.11	25.20	25.15	25.42

In Tab. 1, we replace the view-dependent SH-based color model with a single constant color for each primitive, denoted as Ours (3D,w/o SH), leading to lower-quality results. Color view-dependency and geometric one should not be viewed as equivalent as they work in different channels (i.e., RGB vs. alpha channel).

Finally, we evaluate in Fig. 6 the distributions of learned volumetric primitives for two types of scenes, hair and vegetation. For each scene, we generate 100 training images, by randomly sampling a view the upper hemisphere, which points toward the center of the scene, and rendering the corresponding 3D scene with Blender. Strong correlations between primitive distributions and the scene type are shown in the figure. For each type of scene, the overall distributions of learned primitives look quite similar. This suggests interesting future work to further compress these similar primitives for more efficient representations. In other words, our framework could serve as a tool to *automatically* generate novel, specialized splatting primitives (instead of general ones) that are tailored to efficiently represent certain types of scenes.

7 Limitations and Future Work

Our work is subject to a number of limitations. First, as shown in Tab. 2, our rendering speed is lower than that of state-of-the-art techniques, even though in many cases we can achieve real-time performance (>50 fps). The reason is that Φ_{proj} and Φ_{dec} are primarily designed for reconstruction quality and representation efficiency; the current running time breakdown among Φ_{proj}, Φ_{dec}, and rasterization is roughly 9:2:1. It will be promising to distill a high-performance version of the MLPs via various acceleration approaches (i.e., weight pruning, switching to FP16 precision, etc), or even derive equivalent analytical equations via, e.g., symbolic regression. Second, we do not take anti-aliasing into consideration. It seems possible to combine our approach with related techniques like [Yu et al. 2024a]. In addition, no advanced encoding methods (e.g., positional/frequency) are adopted for input to $\Phi_{\text{proj}}/\Phi_{\text{dec}}$. We are interested in exploring different encodings to further improve the performance, especially for 2D splatting.

We hope that our work could inspire more research into automatic design of graphics representations (e.g., developing efficient,

Fig. 6. Distributions of our kernels on different types of scenes. From the top row to bottom, vegetation and hair scenes. Each inset shows the corresponding histogram of d_1 (Eq. (6)) of our learned kernels, with the red dotted line indicating the mean; the range of the horizontal axis is [0,1].

specialized representations for particular types of scenes), or even the processing pipeline [Zeng et al. 2025]. It is also intriguing to analyze what exactly is learned in the view dependency, and compare it with rigid Euclidean 3D consistency. Our learned consistency might be helpful in other tasks, such as regularizing generative 3D modeling. In addition, it is desirable to compute quantitative metrics over the temporal domain to systemically analyze the reconstruction quality in animation [Liang et al. 2023]. Finally, we would like to combine with state-of-the-art work on relighting [Bi et al. 2024], to support efficient image synthesis with novel view and lighting conditions.

Acknowledgments

Fig. 1 & 6 use assets by khaloui with the extended use license, and by MagicCGIStudios with the standard license. This work is partially supported by NSF China (62332015, 62227806, & 62421003), the XPLORER PRIZE, Information Technology Center, State Key Lab of CAD&CG, Zhejiang University, and a gift from Adobe.

References

Yanqi Bao, Tianyu Ding, Jing Huo, Yaoli Liu, Yuxin Li, Wenbin Li, Yang Gao, and Jiebo Luo. 2024. 3D Gaussian Splatting: Survey, Technologies, Challenges, and Opportunities. arXiv:2407.17418 [cs.CV] https://arxiv.org/abs/2407.17418

Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. 2022. Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields. *CVPR* (2022).

Zoubin Bi, Yixin Zeng, Chong Zeng, Fan Pei, Xiang Feng, Kun Zhou, and Hongzhi Wu. 2024. GS^3: Efficient Relighting with Triple Gaussian Splatting. In *SIGGRAPH Asia 2024 Conference Papers*.

Adam Celarek, George Kopanas, George Drettakis, Michael Wimmer, and Bernhard Kerbl. 2025. Does 3D Gaussian Splatting Need Accurate Volumetric Rendering? arXiv:2502.19318 [cs.GR] https://arxiv.org/abs/2502.19318

Brian Chao, Hung-Yu Tseng, Lorenzo Porzi, Chen Gao, Tuotuo Li, Qinbo Li, Ayush Saraf, Jia-Bin Huang, Johannes Kopf, Gordon Wetzstein, and Changil Kim. 2025. Textured Gaussians for Enhanced 3D Scene Appearance Modeling. In *CVPR*.

Guikun Chen and Wenguan Wang. 2024. A Survey on 3D Gaussian Splatting. *Comput. Surveys* (2024). https://api.semanticscholar.org/CorpusID:266844057

Haodong Chen, Runnan Chen, Qiang Qu, Zhaoqing Wang, Tongliang Liu, Xiaoming Chen, and Yuk Ying Chung. 2024. Beyond Gaussians: Fast and High-Fidelity 3D Splatting with Linear Kernels. arXiv:2411.12440 [cs.CV] https://arxiv.org/abs/2411.12440

Ben Fei, Jingyi Xu, Rui Zhang, Qingyuan Zhou, Weidong Yang, and Ying He. 2025. 3D Gaussian Splatting as a New Era: A Survey. *IEEE Transactions on Visualization and Computer Graphics* 31, 8 (Aug. 2025), 4429–4449. doi:10.1109/tvcg.2024.3397828

Abdullah Hamdi, Luke Melas-Kyriazi, Jinjie Mai, Guocheng Qian, Ruoshi Liu, Carl Vondrick, Bernard Ghanem, and Andrea Vedaldi. 2024. GES: Generalized Exponential Splatting for Efficient Radiance Field Rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 19812–19822.

Allen Hatcher. 2002. *Algebraic Topology*. Cambridge University Press.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 1026–1034.

Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. 2018. Deep Blending for Free-viewpoint Image-based Rendering. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)* 37, 6 (2018), 257:1–257:15.

Jan Held, Sanghyun Son, Renaud Vandeghen, Daniel Rebain, Matheus Gadelha, Yi Zhou, Anthony Cioppa, Ming C. Lin, Marc Van Droogenbroeck, and Andrea Tagliasacchi. 2025a. MeshSplatting: Differentiable Rendering with Opaque Meshes. arXiv:2512.06818 [cs.CV] https://arxiv.org/abs/2512.06818

Jan Held, Renaud Vandeghen, Adrien Deliege, Abdullah Hamdi, Anthony Cioppa, Silvio Giancola, Andrea Vedaldi, Bernard Ghanem, Andrea Tagliasacchi, and Marc Van Droogenbroeck. 2025b. Triangle Splatting for Real-Time Radiance Field Rendering. *arXiv* (2025).

Jan Held, Renaud Vandeghen, Abdullah Hamdi, Adrien Deliege, Anthony Cioppa, Silvio Giancola, Andrea Vedaldi, Bernard Ghanem, and Marc Van Droogenbroeck. 2025c. 3D Convex Splatting: Radiance Field Rendering with 3D Smooth Convexes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger, and Shenghua Gao. 2024. 2D Gaussian Splatting for Geometrically Accurate Radiance Fields. In *SIGGRAPH 2024 Conference Papers*. Association for Computing Machinery. doi:10.1145/3641519.3657428

Yi-Hua Huang, Ming-Xian Lin, Yang-Tian Sun, Ziyi Yang, Xiaoyang Lyu, Yan-Pei Cao, and Xiaojuan Qi. 2025. Deformable Radial Kernel Splatting. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2025).

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 2023. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Transactions on Graphics* 42, 4 (July 2023). https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/

Shakiba Kheradmand, Daniel Rebain, Gopal Sharma, Weiwei Sun, Yang-Che Tseng, Hossam Isack, Abhishek Kar, Andrea Tagliasacchi, and Kwang Moo Yi. 2024. 3D Gaussian Splatting as Markov Chain Monte Carlo. In *Advances in Neural Information Processing Systems (NeurIPS)*. Spotlight Presentation.

Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. 2017. Tanks and Temples: Benchmarking Large-Scale Scene Reconstruction. *ACM Transactions on Graphics* 36, 4 (2017).

Hanxue Liang, Tianhao Wu, Param Hanji, Francesco Banterle, Hongyun Gao, Rafal Mantiuk, and Cengiz Oztireli. 2023. Perceptual Quality Assessment of NeRF and Neural View Synthesis Methods for Front-Facing Views. arXiv:2303.15206 [cs.CV] https://arxiv.org/abs/2303.15206

Rong Liu, Dylan Sun, Meida Chen, Yue Wang, and Andrew Feng. 2025b. Deformable Beta Splatting. arXiv:2501.18630 [cs.CV] https://arxiv.org/abs/2501.18630

Xiaofeng Liu, Guanchen Meng, Chongyang Feng, Risheng Liu, Zhongxuan Luo, and Xin Fan. 2025a. TNT-GS: Truncated and Tailored Gaussian Splatting. In *Proceedings of the 33rd ACM International Conference on Multimedia* (Dublin, Ireland) *(MM '25)*. Association for Computing Machinery, New York, NY, USA, 573–581. doi:10.1145/3746027.3755573

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2020. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *ECCV*.

Panagiotis Papantonakis, Georgios Kopanas, Frédo Durand, and George Drettakis. 2025. Content-Aware Texturing for Gaussian Splatting. *ArXiv* abs/2512.02621 (2025). https://api.semanticscholar.org/CorpusID:280026323

Hyunwoo Park, Gun Ryu, and Wonjun Kim. 2025. DropGaussian: Structural Regularization for Sparse-view Gaussian Splatting. arXiv:2504.00773 [cs.CV] https://arxiv.org/abs/2504.00773

Victor Rong, Jingxiang Chen, Sherwin Bahmani, Kiriakos N Kutulakos, and David B Lindell. 2024. GStex: Per-Primitive Texturing of 2D Gaussian Splatting for Decoupled Appearance and Geometry Modeling. *arXiv preprint arXiv:2409.12954* (2024).

David Svitov, Pietro Morerio, Lourdes Agapito, and Alessio Del Bue. 2024. BillBoard Splatting (BBSplat): Learnable Textured Primitives for Novel View Synthesis. *arXiv preprint arXiv:2411.08508* (2024).

Nygel George Thomas and Chandra Sekhar Seelamantula. 2025. SplineSplat: Representing Radiance Fields Using B-Splines. *Proceedings of the SIGGRAPH Asia 2025 Technical Communications* (2025). https://api.semanticscholar.org/CorpusID:283713219

Zhiru Wang, Shiyun Xie, Chengwei Pan, and Guoping Wang. 2024. SpecGaussian with Latent Features: A High-quality Modeling of the View-dependent Appearance for

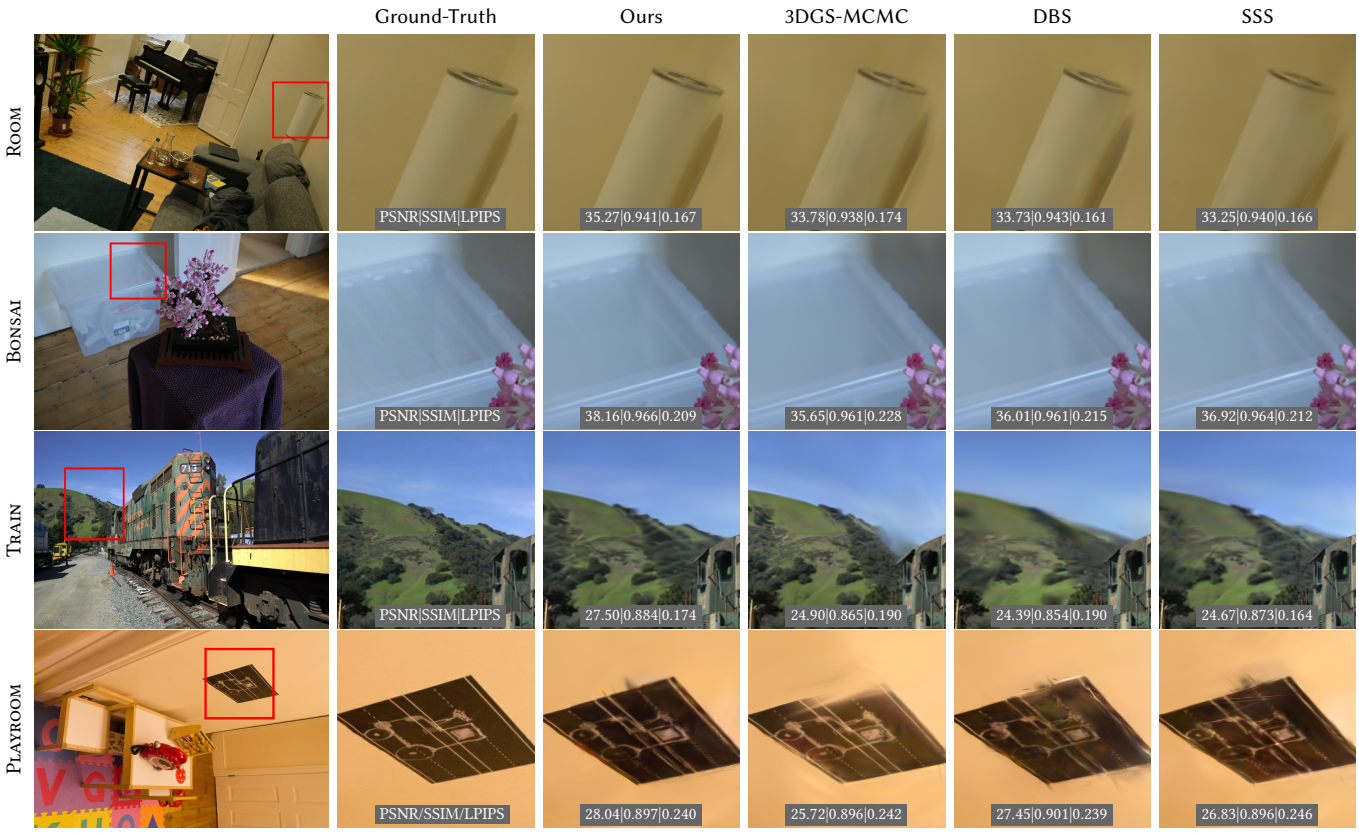

Fig. 7. Qualitative comparisons with state-of-the-art techniques. We compare our approach with 3DGS-MCMC [Kheradmand et al. 2024], SSS [Zhu et al. 2025] and DBS [Liu et al. 2025b]. Quantitative errors in PSNR, SSIM and LPIPS are also reported at the bottom of each related image.

3D Gaussian Splatting. In *Proceedings of the 32nd ACM International Conference on Multimedia* (Melbourne VIC, Australia) *(MM '24)*. Association for Computing Machinery, New York, NY, USA, 6270–6278. doi:10.1145/3664647.3681059

Haato Watanabe, Kenji Tojo, and Nobuyuki Umetani. 2025. 3D Gabor Splatting: Reconstruction of High-frequency Surface Texture using Gabor Noise. In *Proceedings of the Eurographics 2025 Short Papers*. Eurographics Association.

Skylar Wurster, Ran Zhang, and Changxi Zheng. 2024. Gabor Splatting for High-Quality Gigapixel Image Representations. In *ACM SIGGRAPH 2024 Posters* (Denver, CO, USA) *(SIGGRAPH '24)*. Association for Computing Machinery, New York, NY, USA, Article 66, 2 pages. doi:10.1145/3641234.3671081

Rui Xu, Wenyue Chen, Jiepeng Wang, Yuan Liu, Peng Wang, Lin Gao, Shiqing Xin, Taku Komura, Xin Li, and Wenping Wang. 2024. SuperGaussians: Enhancing Gaussian Splatting Using Primitives with Spatially Varying Colors. arXiv:2411.18966 [cs.CV] https://arxiv.org/abs/2411.18966

Ziyi Yang, Xinyu Gao, Yangtian Sun, Yihua Huang, Xiaoyang Lyu, Wen Zhou, Shaohui Jiao, Xiaojuan Qi, and Xiaogang Jin. 2024. Spec-gaussian: Anisotropic view-dependent appearance for 3d gaussian splatting. *arXiv preprint arXiv:2402.15870* (2024).

Ruihan Yu, Tianyu Huang, Jingwang Ling, and Feng Xu. 2024b. 2DGH: 2D Gaussian-Hermite Splatting for High-quality Rendering and Better Geometry Reconstruction. arXiv:2408.16982 [cs.CV] https://arxiv.org/abs/2408.16982

Zehao Yu, Anpei Chen, Binbin Huang, Torsten Sattler, and Andreas Geiger. 2024a. Mip-Splatting: Alias-free 3D Gaussian Splatting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 19447–19456.

Chong Zeng, Yue Dong, Pieter Peers, Hongzhi Wu, and Xin Tong. 2025. RenderFormer: Transformer-based Neural Rendering of Triangle Meshes with Global Illumination. In *ACM SIGGRAPH 2025 Conference Papers*.

Xin Zhang, Anpei Chen, Jincheng Xiong, Pinxuan Dai, Yujun Shen, and Weiwei Xu. 2025. Neural Shell Texture Splatting: More Details and Fewer Primitives. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

Xilong Zhou, Bao-Huy Nguyen, Loïc Magne, Vladislav Golyanik, Thomas Leimkühler, and Christian Theobalt. 2025. Splat the Net: Radiance Fields with Splattable Neural Primitives. *arXiv preprint arXiv:2510.08491* (2025).

Jialin Zhu, Jiangbei Yue, Feixiang He, and He Wang. 2025. 3D Student Splatting and Scooping. In *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR)*. 21045–21054.

M. Zwicker, H. Pfister, J. van Baar, and M. Gross. 2001. EWA volume splatting. In *Proceedings Visualization, 2001. VIS '01.* 29–538. doi:10.1109/VISUAL.2001.964490

Fig. 8. Qualitative comparisons with state-of-the-art techniques under different memory footprint for primitives. We compare our approach (k=2, 4, 8) with 3DGS-MCMC [Kheradmand et al. 2024], SSS [Zhu et al. 2025] and DBS [Liu et al. 2025b]. Quantitative errors in PSNR, SSIM and LPIPS are reported at the bottom of each related image.