Spaghetti Splatting
Differentiating Curvilinear Dirac Masses via Convolution

George Shramko[1] Shrisudhan Govindarajan[1,2] Rudy Lam[1] Yizheng Xie[1]
Daniel Rebain[2,3] David B. Lindell[4] Andrea Tagliasacchi[1,2,4]

[1]Simon Fraser University [2]Wayve Labs [3]University of British Columbia [4]University of Toronto



Figure 1. **Teaser** – We introduce a differentiable 3D representation which can be used to learn accurate radiance fields for novel view synthesis applications (left). We showcase the internal structure of our representation (right) by coloring each primitive with unique colors. [1]This shows that our primitives are a set of connected polylines (i.e. *spaghetti*), each obtained by convolving a polyline Dirac measure with a Gaussian or beta kernel. It is *splatting*, as each primitive is rendered by splatting its kernel onto the image.[2]

Abstract

In production systems, 3D geometry is represented by primitives such as points, lines, and triangles, as rendering hardware is optimized to generate images from them very efficiently. Learning such primitives from images by volume rendering is challenging, as they are Dirac masses defined at infinitesimal locations in space. EWA splatting, Poisson reconstruction, and 3D Gaussian Splatting overcome this by convolving Dirac masses with a low-frequency kernel, a recipe that generalizes along two axes: the kernel and the primitive. As points and triangles have already received significant attention, we investigate this recipe for curvilinear *primitives, and introduce Spaghetti Splatting. We make curves differentiable by convolving them with a kernel, and show that the convolution of a polyline converges to that of the continuous curve as its segments shrink. We approximate curves by polylines, and volume render each segment so that connected segments composite correctly. For fast training and inference, we linearize the camera projection per segment, which accommodates kernels beside Gaussians, such as Beta kernels. As a point is a curve of zero length, our representation is a strict superset of 3D Gaussian Splatting. Hence, it remains competitive with the state-of-the-art on general scenes, outperforms methods specialized to line-like geometry, natively supports anti-aliasing, and produces long connected strands that are directly amenable to physics simulation.*

[1]We thank Zhen (Colin) Li for help with capturing the pasta scene, and Francesco Di Sario and Maria Taktasheva for cooking, as well as eating the pasta.

[2]We provide details on how we perform the reconstruction for this teaser in the supplementary material.

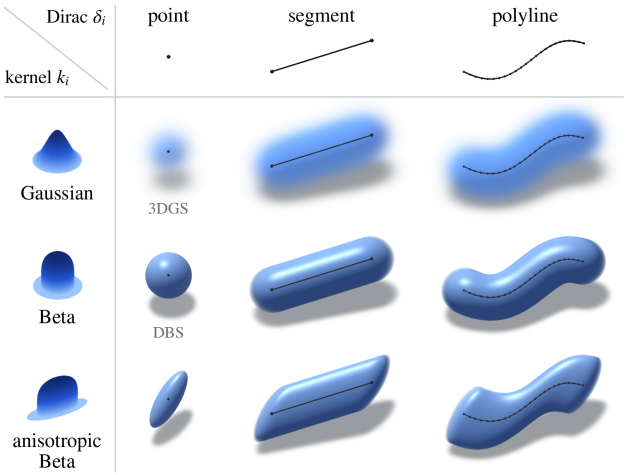


Figure 2. **Kernel ⊛ dirac** – Since Dirac masses and convolution kernels are independent degrees of freedom, we can modify either one to obtain a different differentiable primitive. The first column shows that convolving a point mass with a Gaussian or a Beta kernel yields the primitives of 3DGS [13] and DBS [16], respectively. Replacing the point mass with a line segment, while retaining either kernel, yields the primitives in the middle. Finally, we connect these segments into a polyline, which yields our *curvilinear* primitive. In the bottom row, we convolve the same Dirac masses with an *anisotropic* Beta kernel. For visualization, we shade the primitives with the normals of their convolution surfaces [4].

1. Introduction

In production graphics systems, 3D geometry is represented by points, lines, and triangles (*i.e.*, *primitives*), as rendering hardware is optimized to generate images from them very efficiently. Rather than constructing them by hand, we would like to *optimize* such primitives directly from a collection of images, by volume rendering [21] and gradient descent. However, this is challenging, as primitives are Dirac masses defined at *infinitesimal* locations in space, so there is nothing for a gradient to act on.

Research from about two decades ago showed us how to overcome this. EWA splatting [46] and Poisson reconstruction [12] both turn a Dirac-like representation into a volumetric one by *convolving* it with a low-frequency kernel. This makes sampling the representation onto a discrete grid of pixels well posed from the perspective of the Nyquist–Shannon theorem [41]. 3D Gaussian Splatting [13] uses this idea to differentiate through point-like primitives, which it convolves with anisotropic Gaussian kernels. *But could this idea be generalized?*

There are two potential axes of generalization, made apparent by the $\mathcal{L}_2$ objective that these methods optimize:

$$\mathcal{L}_2 = \sum_c \left\| \boldsymbol{I}_c - \mathcal{R}_c\Big(\sum_i \delta_i \circledast k_i \Big) \right\|_2, \qquad (1)$$

where $\boldsymbol{I}_c$ is the image taken from camera viewpoint c, $\mathcal{R}_c$ volume renders the scene from that same viewpoint, and δ_i is the Dirac measure of the i-th primitive, convolved with kernel k_i. One axis generalizes the *kernel*, as in Deformable Beta Splatting [16]. The other generalizes the *primitive*, with initial research on convexes [9], triangles [11], and, very recently, lines [34].[3] As points [13, 16, 41] and triangles [10, 11] have already received significant attention, we focus on differentiating through *curvilinear* primitives; see Figure 2.

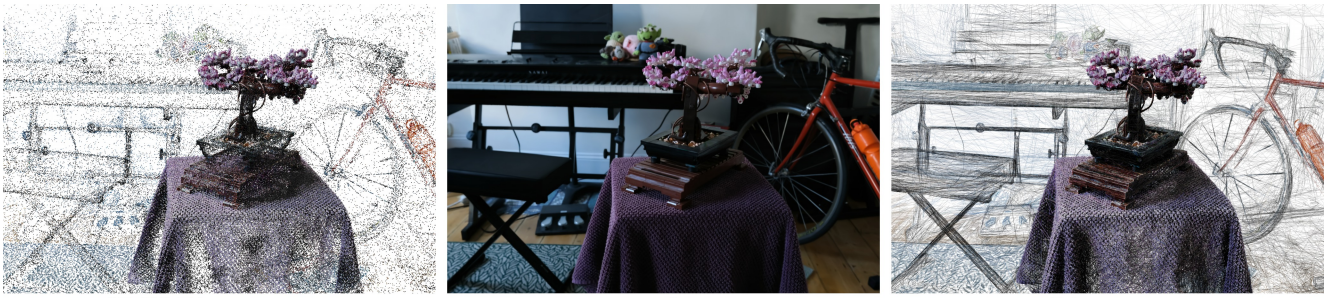
To this end, we approximate curves by polylines, and show that their convolution with a kernel converges to that of the continuous curve as the segments shrink. Whereas the Gaussian convolution of a polygon requires evaluating Owen's T function [7, 26], the ray integral of a Gaussian-convolved segment only involves the Gauss error function erf (*i.e.*, the anti-derivative of a Gaussian). By volume rendering the density of each segment [31], connected segments composite without overcounting at their joints. For fast training and inference, we follow EWA splatting [46] and linearize the camera projection per segment, which also accommodates Beta kernels [16].

Our representation is a strict superset of 3D Gaussian Splatting, as a point is nothing else than a curve of zero length. As such, it matches or outperforms the state-of-the-art on Mip-NeRF 360 [2], *i.e.*, Deformable Beta Splatting [16] for geometry, coupled with Spherical Voronoi [29] for modeling of view-dependent appearance. Spaghetti Splatting also outperforms methods specialized to line-like geometry: the concurrent work of Tojo et al. [34] on their Fuzzy benchmark, and Gaussian Haircut [43], which relies on hair-specific priors, on GeomHair [15]. It natively supports Mip-Splatting [41] anti-aliasing, so it remains stable across viewing distances. Finally, our representation consists of long connected strands, which makes it amenable to physics simulation, and closer to what an artist would produce.

We achieve these results through four key contributions:

- We note that the key ingredient behind 3D Gaussian Splatting and Poisson reconstruction, *i.e.*, convolving Dirac masses with a low-frequency kernel, provides a recipe that generalizes along both the kernel and the primitive.
- We investigate this recipe for *curvilinear* Dirac masses, and provide a convergent as well as efficient way to back-propagate through them.
- We generalize 3D Gaussian Splatting to a strictly larger family of representations, which remains competitive with the state-of-the-art on general scenes, while setting a new state-of-the-art for strand-based representations.
- We commit to fully release code to foster precise comparisons, and follow-up research.

[3] Note that the work by Tojo et al. [34] is concurrent to ours.



3DGS Dirac measure render our Dirac measure

Figure 3. **Inside our spaghetti** – We visualize the Dirac measure of Kheradmand et al. [14] (left), and of our Spaghetti Splatting (right) for the Bonsai scene from Mip-NeRF 360 [2]. (middle) We show one RGB render, not repeated, as it is visually identical for both representations. While 3DGS models thin strokes with scattered points (e.g. the bicycle spokes), our segments form continuous strokes that follow edges.

2. Related work

Neural Radiance Fields (NeRF) represents a scene as a continuous density and color field rendered by numerical integration along camera rays [21]. Later neural fields addressed aliasing and unbounded scenes [1–3], while multiresolution hash encodings substantially reduced their training cost [25]. 3DGS replaced the neural field with explicit anisotropic Gaussians [13], and a tile-based rasterizer derived from point-based volume rendering and EWA splatting [37, 46]. Subsequent methods have improved both its optimization and rendering. 3DGS-MCMC formulates population control as sampling [14], and StopThePop introduces per-pixel sorting for more consistent visibility [27]. Other methods parallelize the backward pass over Gaussians instead of pixels [19], integrate Gaussian density along each ray [33], render constant-density ellipsoids in closed form [18], or avoid the affine projection used by rasterization [22, 38]. Another line of work replaces the Gaussian kernel: Deformable Beta Splatting (DBS) learns the fall-off of a compact Beta kernel [16], while Student Splatting and Scooping uses heavy-tailed kernels with signed weights [45]. Mip-Splatting prevents aliasing across viewing distances with a three-dimensional smoothing filter based on the training views and a two-dimensional filter during rasterization [41]. Spherical Voronoi replaces spherical harmonics with a learned directional partition [29]. We build on several of these improvements and adopt 3DGS-MCMC population control, the Beta kernel of DBS, Mip-Splatting filtering, and Spherical Voronoi appearance modeling.

Beyond point primitives. Another line of work replaces point centers with extended geometric primitives. Triangle Splatting [11] and 3D Convex Splatting [9] optimize triangles and smooth convex bodies. Curve primitives have so far targeted sparse abstractions of a scene: 3Doodle optimizes 3D Bézier strokes and superquadric contours to render view-consistent sketches [6], while SketchSplat reconstructs parametric 3D edges by splatting Gaussians sampled along curves onto multi-view edge maps [40]. The concurrently developed opaque line primitives of Tojo et al. [34] are closely related to our work, as they also use line primitives directly for differentiable rendering. Their method rasterizes segments on a subpixel grid using the Bresenham algorithm and recovers high-quality geometry in object-centric scenes. That said, their line primitives are rendered at a constant screen-space width regardless of their depth from the camera, making the representation unsuitable for general scenes, as we show in our comparison on Mip-NeRF 360 (Fig. 7). We include evaluations of our method on the Fuzzy dataset introduced by their work.

Convolution surfaces. A classical computer graphics technique, Convolution Surfaces [4], has shown that one can represent many implicit shapes as an iso-surface of a convolution of a skeleton of curves and polygons with a smooth kernel. Recently, Xu et al. [39] demonstrated that one can efficiently render a Gaussian process implicit surface as an iso-surface of a sparse convolution of a kernel with randomly placed points. In both cases, the convolved field serves only to define an iso-surface. We instead use segment and polyline skeletons convolved with corresponding kernels as an explicit volume-rendering representation which we optimize from multi-view images. In this view, point-based splatting primitives are a special case of our segments with zero length, and our method maintains their fast rendering speed.

Hair reconstruction. Hair is a prominent example of geometry that is naturally described by curves. Neural Strands and Neural Haircut reconstruct strand geometry with learned priors [28, 30]. Gaussian Haircut and GaussianHair represent hair with Gaussians aligned to strand directions [17, 43]. Our reconstruction optimizes rooted polylines without a learned strand prior and passes the resulting curves directly to position-based simulation [23, 24]. We use the public strand models from GeomHair [15] as reference geometry [42].

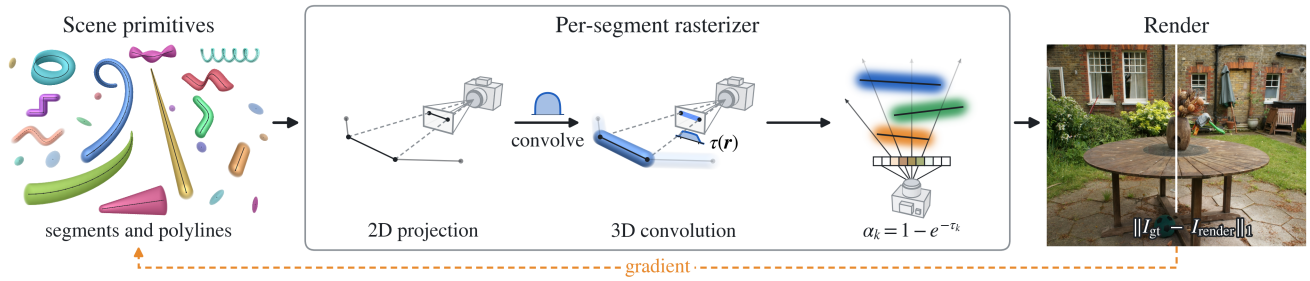


Figure 4. **Overview** – (Left) Our representation is a set of learnable polyline primitives convolved with Gaussian or Beta kernels. (Middle) Without loss of generality, we render each segment individually: we first project it onto the image plane, and then evaluate the integral of the convolution along the ray in screen space using the closed-form integrals of Section 3.3, which yields a per-segment screen-space rendering. The segments are then composited along each ray to obtain the accumulated color. (Right) We compute the L_1 loss between the rendered image and the ground-truth image, and backpropagate its gradient to the parameters of the primitives.

2.1. Volume rendering

Given a density field $\sigma(\cdot)$ and an emitter color field $\boldsymbol{c}(\cdot)$, the color along a ray $\boldsymbol{r}(t) = \boldsymbol{o}+t\boldsymbol{d}$ as per volume rendering [20] can be computed as:

$$\boldsymbol{C} = \int_0^\infty T(t)\,\sigma(\boldsymbol{r}(t))\,\boldsymbol{c}(\boldsymbol{r}(t))\,\mathrm{d}t, \tag{2}$$

where the transmittance term is defined as

$$T(t) = \exp\left(-\int_0^t \sigma(\boldsymbol{r}(s))\,\mathrm{d}s\right). \tag{3}$$

Splitting the ray into ordered intervals I_k from front to back, with a constant color $\boldsymbol{c}_k$ on each, turns (2) into front-to-back compositing,

$$\boldsymbol{C} = \sum_{k=1}^{K} T_k\,\alpha_k\,\boldsymbol{c}_k, \tag{4}$$

where α_k is the opacity of interval I_k and T_k is the fraction of light that survives the intervals in front of it, given by

$$\tau_k = \int_{I_k} \sigma(\boldsymbol{r}(t))\,\mathrm{d}t, \quad \alpha_k = 1 - e^{-\tau_k}, \quad T_k = \prod_{j<k}(1-\alpha_j).$$

3. Method

We begin by reviewing how a 3D Gaussian can be interpreted as the convolution of a Dirac mass with an anisotropic Gaussian kernel (Sec. 3.1). We then extend this formulation to Gaussian-convolved line segments (Sec. 3.2) and derive the corresponding closed-form rendering equation (Sec. 3.3). Finally, we describe fast rendering with EWA splatting (Sec. 3.4), anti-aliasing (Sec. 3.5), and optimization (Sec. 3.6).

3.1. 3D Gaussians as a convolution

One of the 3D Gaussian primitives $\mathcal{G}$ of Figure 3 (left) can be interpreted as the convolution of a Dirac mass placed at position $\boldsymbol{\mu}$ with a zero-mean Gaussian kernel $\mathcal{N}_{\boldsymbol{\Sigma}}=\mathcal{N}(\mathbf{0},\boldsymbol{\Sigma})$:

$$\mathcal{G}(\boldsymbol{x}) = \left[\delta(\cdot - \boldsymbol{\mu}) \circledast \mathcal{N}_{\boldsymbol{\Sigma}}\right](\boldsymbol{x}). \tag{5}$$

Here, $\boldsymbol{x}\in\mathbb{R}^3$ is a query point, δ is the three-dimensional Dirac distribution, $\boldsymbol{\Sigma}$ is the covariance of the kernel, and $\circledast$ denotes convolution. Hence, $\mathcal{G}$ is the Gaussian density with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$. This interpretation decouples the support of the primitive from the kernel, which determines how the primitive extends through 3D space. In the next section, we replace the point support with a segment and connect segments into polylines to obtain our Spaghetti Splatting curvilinear primitive; see Figure 3 (right).

3.2. Spaghetti Splatting

Following the interpretation in Section 3.1, we retain the kernel of 3DGS and replace the point support with a finite segment of learnable length. A segment is defined by its endpoints $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^3$. We write $\boldsymbol{v}=\boldsymbol{b}-\boldsymbol{a}$ and parameterize the segment as $\boldsymbol{x}(u)=\boldsymbol{a}+u\boldsymbol{v}$ with $u \in [0,1]$. The segment is then represented by the measure

$$m(\boldsymbol{x}) = \int_0^1 \delta\big(\boldsymbol{x} - \boldsymbol{x}(u)\big)\,\mathrm{d}u, \tag{6}$$

Our primitive $\mathcal{P}$ is the convolution of this segment measure with an anisotropic Gaussian kernel,

$$\mathcal{P}(\boldsymbol{x}) = \left[m \circledast \mathcal{N}_{\boldsymbol{\Sigma}}\right](\boldsymbol{x}). \tag{7}$$

When $\boldsymbol{a}=\boldsymbol{b}$, the above recovers the 3DGS point-kernel, hence Spaghetti Splatting is a strict generalization of 3DGS representations. The formulation in (7) extends naturally to other convolution kernels, such as the beta kernels of [16].

3.3. Closed-form volume rendering

We now describe how our primitive defines a volumetric density field, and, without loss of generality, derive the closed-form volume rendering integral for the *Gaussian* kernel. We parameterize each primitive with a volumetric density that follows the support of its kernel. For a point $\boldsymbol{x}$, the density is

$$\sigma(\boldsymbol{x}) = \rho\left[m \circledast \mathcal{N}_{\boldsymbol{\Sigma}}\right](\boldsymbol{x}). \tag{8}$$

Expanding the convolution into an integral over a point gives

$$\sigma(\boldsymbol{x}) = \rho \int m(\boldsymbol{y})\, \mathcal{N}_{\boldsymbol{\Sigma}}(\boldsymbol{x}-\boldsymbol{y})\, \mathrm{d}\boldsymbol{y}. \tag{9}$$

Substituting the segment measure of (6) and swapping the order of integration yields

$$\sigma(\boldsymbol{x}) = \rho \int_0^1 \left[\int_{\boldsymbol{y}} \delta\big(\boldsymbol{y} - \boldsymbol{x}(u)\big)\, \mathcal{N}_{\boldsymbol{\Sigma}}(\boldsymbol{x}-\boldsymbol{y})\, \mathrm{d}\boldsymbol{y} \right] \mathrm{d}u. \tag{10}$$

The inner integral over $\boldsymbol{y}$ collapses by the *sifting* property of the Dirac delta, which results in

$$\sigma(\boldsymbol{x}) = \rho \int_0^1 \mathcal{N}_{\boldsymbol{\Sigma}}\big(\boldsymbol{x} - \boldsymbol{x}(u)\big)\, \mathrm{d}u. \tag{11}$$

Intuitively, (11) states that the density at $\boldsymbol{x}$ is the kernel response accumulated over every point of the segment, which is precisely what convolving a segment with a Gaussian kernel should produce. We now turn from the density to the opacity that a primitive contributes along a camera ray. Following [31], the opacity along a ray $\boldsymbol{r}(t) = \boldsymbol{o} + t\boldsymbol{d}$ is

$$\alpha(\boldsymbol{r}) = 1 - \exp\big(-\tau(\boldsymbol{r})\big),\ \tau(\boldsymbol{r}) = \int_{-\infty}^{\infty} \sigma\big(\boldsymbol{r}(t)\big)\, \mathrm{d}t. \tag{12}$$

Substituting the density from (11), evaluated at the points $\boldsymbol{r}(t)$ along the ray, we can write the $\tau(r)$ as

$$\tau(\boldsymbol{r}) = \int_{-\infty}^{\infty} \rho \int_0^1 \mathcal{N}_{\boldsymbol{\Sigma}}\big(\boldsymbol{r}(t) - \boldsymbol{x}(u)\big)\, \mathrm{d}u\, \mathrm{d}t. \tag{13}$$

Since $\mathcal{N}_{\boldsymbol{\Sigma}}$ is non-negative, Tonelli's theorem allows us to swap the order of integration, which gives

$$\tau(\boldsymbol{r}) = \rho \int_0^1 \int_{-\infty}^{\infty} \mathcal{N}_{\boldsymbol{\Sigma}}\big(\boldsymbol{r}(t) - \boldsymbol{x}(u)\big)\, \mathrm{d}t\, \mathrm{d}u. \tag{14}$$

The inner integral is the integral of a Gaussian along a ray, which is itself a one-dimensional Gaussian in the segment coordinate u. Denoting this inner integral by $f(u)$, we simplify (14) to

$$\begin{aligned} \tau(\boldsymbol{r}) &= \int_0^1 \rho\, f(u)\, \mathrm{d}u, \\ f(u) &= \int_{-\infty}^{\infty} \mathcal{N}_{\boldsymbol{\Sigma}}\big(\boldsymbol{r}(t) - \boldsymbol{x}(u)\big)\, \mathrm{d}t. \end{aligned} \tag{15}$$

Let us define the mean of the one-dimensional Gaussian $f(u)$ to be $\bar{u}$ and its variance to be $1/(2\kappa^2)$, then (15) simplifies:

$$f(u) = f(\bar{u})\, e^{-\kappa^2 (u-\bar{u})^2}. \tag{16}$$

For a unit ray direction $\boldsymbol{d}$, these quantities are

$$\kappa^2 = \|\boldsymbol{v}\|^2_{\boldsymbol{Q}_\perp}, \qquad \bar{u} = \frac{\boldsymbol{v}^\top \boldsymbol{Q}_\perp (\boldsymbol{o} - \boldsymbol{a})}{\kappa^2}, \tag{17}$$

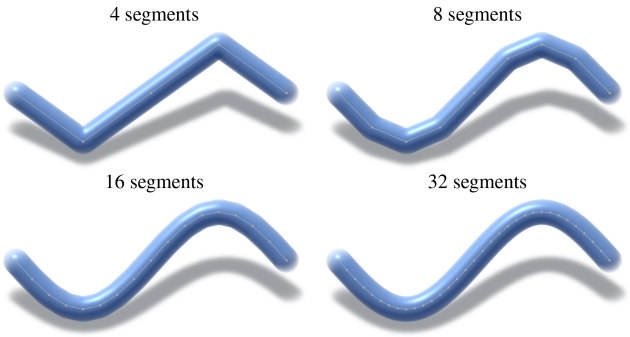


Figure 5. **Convergence** – A curve rendered as a polyline of 4, 8, 16, and 32 Beta segments rasterized independently converges to the smooth curve without knots at the joints.

where $\boldsymbol{Q}{=}\frac{1}{2}\boldsymbol{\Sigma}^{-1}$ and $\boldsymbol{Q}_\perp{=}\boldsymbol{Q} - \boldsymbol{Q}\boldsymbol{d}\boldsymbol{d}^\top\boldsymbol{Q}/(\boldsymbol{d}^\top\boldsymbol{Q}\boldsymbol{d})$ is the matrix $\boldsymbol{Q}$ of the marginal of $\mathcal{N}_{\boldsymbol{\Sigma}}$ along the camera ray. Substituting (16) into (15) reduces the optical depth to a single integral,

$$\begin{aligned} \tau(\boldsymbol{r}) &= \rho\, f(\bar{u}) \int_0^1 e^{-\kappa^2 (u-\bar{u})^2}\, \mathrm{d}u, \\ f(\bar{u}) &= \sqrt{\frac{\pi}{\boldsymbol{d}^\top \boldsymbol{Q}\boldsymbol{d}}}\, e^{-\|\boldsymbol{o} - \boldsymbol{x}(\bar{u})\|^2_{\boldsymbol{Q}_\perp}}. \end{aligned} \tag{18}$$

This integral has a closed form through the antiderivative of a Gaussian, $\int e^{-\kappa^2(u-\bar{u})^2}\, \mathrm{d}u = \frac{\sqrt{\pi}}{2\kappa} \operatorname{erf}\big(\kappa(u-\bar{u})\big)$. Evaluating it at the limits and using that erf is odd gives

$$\tau(\boldsymbol{r}) = \frac{\rho\, f(\bar{u})\sqrt{\pi}}{2\kappa} \Big[\operatorname{erf}\big(\kappa\bar{u}\big) + \operatorname{erf}\big(\kappa(1-\bar{u})\big) \Big]. \tag{19}$$

Substituting (19) into (12) yields the opacity of the primitive along the ray in closed form. The same pipeline also supports other kernel profiles, such as the compact Beta kernel of DBS [16]. The full derivation of the screen-space expressions is given in the supplementary material.

For polylines, we treat each segment independently: we evaluate the volume integral and the resulting opacity separately for every segment, and then composite the segments along the ray as individual primitives; see Figure 5.

3.4. Efficient rendering

Evaluating the closed form for every primitive–pixel pair requires extensive computation of ray-dependent quantities, which is too slow for splatting-style training and rendering. We therefore compute EWA [46] at the midpoint once per frame and scale it along the segment with inverse depth, which reduces the per-pixel evaluation to a few screen-space quantities. This allows the kernel to vary in screen space, keeping the 3D shape consistent. The yellow segment in Figure 4 shows the resulting foreshortening. We additionally show analysis of our EWA computation in the supplementary material.

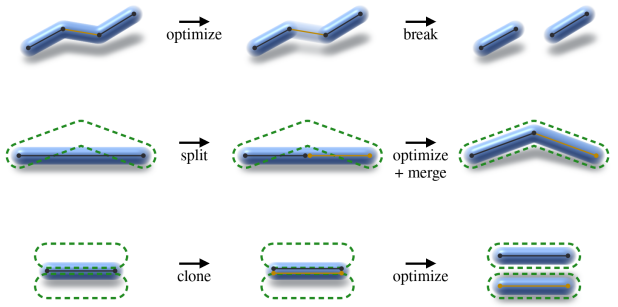


Figure 6. **Densification** – We take inspiration from the densification strategy of 3DGS MCMC [14], but adapt it to our primitives. When the opacity of a segment falls below a threshold, MCMC relocation moves it, potentially breaking up the polyline it was in. The relocation target is then sampled in two ways. Long segments are split into two, short segments are cloned.

3.5. Anti-aliasing

To avoid aliasing, we follow Mip-Splatting [41] and apply a 3D as well as a 2D Gaussian filter to our primitives. The 3D filter band-limits each primitive to the maximal sampling rate of the training views, so that it cannot encode detail the training images cannot resolve. The 2D filter convolves each projected primitive with a sub-pixel screen-space Gaussian, which prevents primitives that project to less than a pixel from aliasing or flickering. Under the EWA linearization (Sec. 3.4), both filters simply add to the covariance of the projected kernel along the entire segment, so the closed form still applies. We thus pre-filter each primitive in closed form, with a single sample per pixel. In contrast, the concurrent work of Tojo et al. [34] *does not* support spatial convolutions and must therefore resort to multisample anti-aliasing with four samples per pixel.

3.6. Optimization

We optimize our representation with the objective

$$\mathcal{L} = \mathcal{L}_{\text{recon}} + \lambda_1 \mathcal{L}_{\text{MCMC}} + \lambda_2 \mathcal{L}_{\text{connect}}, \qquad (20)$$

where $\mathcal{L}_{\text{recon}}$ is the standard photometric loss, a weighted combination of an L_1 and a D-SSIM term, between the rendered image $\hat{\boldsymbol{I}}$ and the ground-truth image $\boldsymbol{I}$. The $\mathcal{L}_{\text{MCMC}}$ term is the opacity and width regularization of [14]. During optimization, we merge polylines whose endpoints fall within a predefined distance threshold. To encourage this, $\mathcal{L}_{\text{connect}}$ actively pulls nearby polyline endpoints together by minimizing the Euclidean distance between them. We describe this attraction loss in more detail in the supplementary material.

Densification. As in other per-scene optimization methods, we adaptively resample our primitives to allocate representation capacity where it is needed. Following 3DGS-MCMC [14], we prune segments and sample their relocation targets based on opacity. We augment the MCMC relocation to additionally split long segments into two halves (Fig. 6). Notably, subdividing a segment leaves the rendering in image space exactly unchanged when both children inherit the density of the parent; see the supplementary material.

4. Experiments

We evaluate our primitive through extensive comparisons with a range of baselines on two tasks: novel view synthesis of general scenes (Sec. 4.1) to demonstrate how this representation is a generalization of 3DGS, and reconstruction of hair strands from images (Sec. 4.2) to demonstrate its effectiveness its in-domain setting.

4.1. General scenes

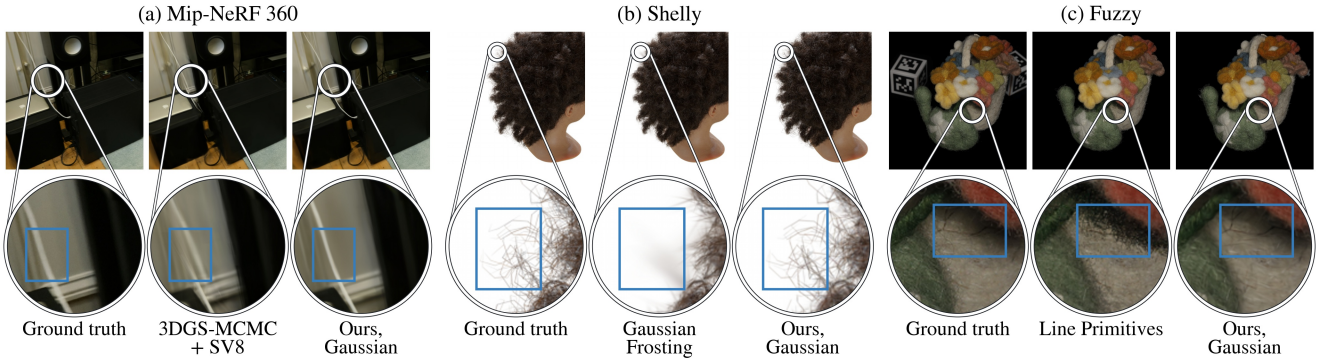
We evaluate our method on 23 scenes from three diverse datasets – Mip-NeRF 360, Shelly and Fuzzy dataset. Mip-NeRF 360 [2] is the standard benchmark for unbounded scenes, with five outdoor and four indoor scenes. Shelly [36] contains six synthetic objects with fur, wool, and thin leaves. The very recently introduced Fuzzy dataset [34] contains eight captured objects with fibrous surfaces, such as cactus spines, fur, and brush bristles. We follow the standard evaluation protocol of each dataset. On Mip-NeRF 360, we downsample the indoor scenes by a factor of two and the outdoor scenes by a factor of four, and we hold out every eighth image for testing [13]. On Fuzzy, we downsample the images by a factor of four and likewise hold out every eighth view for testing. On Shelly, we train and evaluate on images of 1920×1080 pixels with a white background. The released Shelly train/test split mistakenly contains test cameras that also appear in the training set. We therefore remove from the training set all images that coincide with a test image.

Training. We train for 30k iterations on all datasets, except for the outdoor scenes of Mip-NeRF 360, which, following Spherical Voronoi (SV) [29], we train for 20k iterations. MCMC relocation [14] stops 5k iterations before the end of training, and polyline merging (Sec. 3.6) runs only after relocation stops. We cap the number of primitives at 500k on Shelly and Fuzzy, and at the per-scene budgets of Spherical Voronoi on Mip-NeRF 360. All our models use the Spherical Voronoi appearance model with eight sites (SV8) [29].

Baselines. We compare with two point-based methods that use the SV8 [29] appearance model: 3DGS-MCMC [14] + SV8 and Deformable Beta Splatting (DBS) [16] + SV8. We also compare with Gaussian Frosting [8] and with the concurrent Line Primitives by Tojo et al. [34], which optimizes opaque line primitives.

Metrics. On the test views, we report PSNR, SSIM [35], and LPIPS [44] (with a VGG backbone). We also report the FPS at test resolution, measured on an RTX 4090 GPU.

Comparisons – Figure 7. Our method with the Beta kernel



Method	Mip-NeRF 360 [2] PSNR ↑	SSIM ↑	LPIPS ↓	FPS ↑	Shelly [36] PSNR ↑	SSIM ↑	LPIPS ↓	FPS ↑	Fuzzy [34] PSNR ↑	SSIM ↑	LPIPS ↓	FPS ↑
Line Primitives [34]	8.66	0.350	0.571	**287**	33.75	0.932	0.068	**731**	33.64	0.939	0.055	**858**
Gaussian Frosting [8]	26.93	0.804	0.209	59	36.33	0.955	**0.047**	121	34.29	0.948	0.043	124
3DGS-MCMC + SV8 [14, 29]	28.39	0.834	0.190	198	37.10	0.959	0.050	464	35.00	0.956	0.041	422
DBS + SV8 [16, 29]	28.71	0.837	**0.189**	125	37.22	0.959	0.049	297	34.98	0.956	**0.040**	283
Ours, Gaussian kernel	28.56	0.835	0.200	177	**37.38**	**0.962**	0.048	452	**35.03**	**0.957**	**0.040**	415
Ours, Beta kernel	**28.78**	**0.838**	0.193	129	37.12	0.958	0.049	284	34.90	0.955	**0.040**	257

Figure 7. **Comparison on Mip-NeRF 360, Shelly, and Fuzzy** – (a, b) Unlike point-based primitives [8, 14, 29], our method can reconstruct thin elongated geometry effectively. (c) We can also model wide texture-less areas, like the dark shadow on the figure, with wide kernels, unlike Line Primitives [34]. We mark the three best values in each column with gold, silver, and bronze medals.



Method	PSNR ↑ / SSIM ↑ / LPIPS ↓	F1 at 2 / 3 / 4 mm, ↑
Gaussian Haircut [43]	29.56 / 0.923 / 0.064	46.9 / 63.5 / 72.7
Ours, Gaussian Kernel	**32.25 / 0.940 / 0.043**	**61.2 / 80.3 / 88.9**

Figure 8. **Comparison with Gaussian Haircut [43]** – averaged over two datasets: GeomHair [15] (selected 8 hairstyles) and the curly hair model of Yuksel [42]. Top: strands of the curly hair model in a held-out view, one random color per strand. Our strands closely match the ground-truth strands, whereas those of Gaussian Haircut are poorly structured.

outperforms all baselines on Mip-NeRF 360, and our method with the Gaussian kernel outperforms all baselines on Shelly and Fuzzy; see Figure 7. Our method also matches the frame rate of 3DGS-MCMC across all three datasets. Since Line Primitives [34] rasterizes thin lines of *constant* width, it cannot model real-world scenes. We provide qualitative video comparisons with the baselines on the project webpage in the supplementary material.

4.2. Hair reconstruction

For Hair reconstruction, we evaluate our method on a subset of GeomHair [15]. GeomHair provides 400 strand-level hairstyles reconstructed from head scans, we select eight hairstyles that cover a wide range of hair length, curvature, and arrangement and place each hairstyle on the head mesh of GeomHair and render 100 views at 1200×800 pixels from cameras around the head, path-traced in Blender with the hair shading model of Chiang et al. [5]. We hold out every eighth view, which leaves 88 training and 12 test views. We additionally evaluate on the challenging curly hair model of Yuksel [42], which we render and split in the same way. We report each metric as the average across the two datasets and show per-haircut breakdown in the supplementary material.

Training and baseline. We train for 30k iterations, following the same protocol as for the general scenes. We compare against Gaussian Haircut [43], which relies on hair-specific priors. By default, they prune all Gaussians outside a sphere fitted to the hair, which removes the chin and neck on short hairstyles. We enlarge this sphere to enclose the head mesh, so that both methods render the entire head.

Metrics. We report PSNR, SSIM, and LPIPS on the test views, rendered over a white background. To evaluate the strand geometry, we compare our strands with the ground-truth strands of each hairstyle after resampling both at 0.5 mm. Following Takimoto et al. [32], a point in one

strand set is considered correct if the other set contains a point within a distance threshold whose tangent line differs by less than an angle threshold. We report the F1 score based on this criterion at the thresholds 2 mm and 20°, 3 mm and 30°, and 4 mm and 40° [30].

Comparisons. We show that our method outperforms Gaussian Haircut on all four metrics, with a 2.7 dB higher PSNR and a 14.3-point higher F1 score at 2 mm. We show all the haircuts used in the supplementary material. Like for the general scenes, we also provide qualitative video comparisons with the baselines on our project webpage.

4.3. Ablation study

We ablate the main design choices of our method to measure the effect of each one. Specifically, we test our densification strategies (midpoint splitting and MCMC-style segment cloning), polyline merging, and attraction regularization. Table 1 summarizes the quantitative effect of each choice.

Midpoint splits. We assess the importance of midpoint splitting by removing it from the densification strategy and relying on segment cloning alone. This slightly degrades performance, which shows that midpoint splitting contributes to training.

Segment cloning. We assess segment cloning, following [14], by disabling it and relying on midpoint splits alone. This leads to a significant drop in reconstruction quality. Without cloning, primitives cannot be effectively redistributed to regions that require short segments.

Polyline merging. Next, we ablate the polyline merging strategy by disabling it during training. This has no effect on the reconstruction metrics, since the representation can still model a connected polyline using many independent segments. We can see this as the connected components become much smaller without merging, which leads to substantially less structural cohesiveness.

Attraction regularization. We assess the attraction loss by disabling it during training. This also has no direct effect on reconstruction quality. However, it weakens polyline merging: without the attraction loss, strands are not encouraged to form a cohesive unit, which reduces the number of connected segments in a polyline.

Midpoint split	Segment cloning	Polyline merging	Attraction loss	PSNR ↑	Mean connected component size
✗	✓	✓	✓	33.13	3.26
✓	✗	✓	✓	32.54	4.00
✓	✓	✗	✓	33.59	1.00
✓	✓	✓	✗	33.57	2.77
✓	✓	✓	✓	33.54	3.50

Table 1. **Ablations** – Averaged over Fuzzy, Shelly, and Mip-NeRF 360; each row removes one component (✗). The size of a connected component is its number of segments; a segment without a joint has size 1.

5. Conclusion

We introduced a splatting representation formed by convolving finite segments with anisotropic kernels, with anisotropic point primitives recovered in the zero-length limit. For Gaussian kernels, the contribution of a segment along a camera ray has a closed form, and the corresponding compositing rule allows connected segments to be rendered independently without changing the result under subdivision. We also derived a screen-space approximation that supports compact profiles and renders at speeds comparable to point splatting. The experiments show competitive image reconstruction and demonstrate that the same primitives can produce connected curves suitable for shading and simulation, while broader claims about automatically recovered topology require further evaluation.

Limitations and future work. While our curvilinear primitives demonstrate a new capability, our representation also opens up many directions for future research. Because high-frequency color requires high-frequency geometry, reproducing fine texture currently requires many short segments. Combining the curvilinear primitives with learnable texture maps could reduce the number of primitives and improve the recovered geometry. We have also not explored interpolating the kernel between the segment endpoints, which would extend our family of primitives to shapes such as cones, triangles, and trapezoids, and could further generalize the representation to surfaces. Real hair reconstruction also remains challenging, and integrating learned hair priors more carefully is a promising direction. Finally, it would be interesting to predict our representation with feedforward models, and to exploit the recovered curves to reverse engineer physical properties of the captured objects.

Acknowledgments. George Shramko was supported in part by Natural Sciences and Engineering Research Council of Canada (NSERC) CGRS-M scholarship. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant [2023-05617], NSERC Collaborative Research and Development Grant, the SFU Visual Computing Research Chair, Google Research, Digital Research Alliance of Canada, and Advanced Research Computing at the University of British Columbia. We thank George Kopanas, Bernhard Kerbl, and Peter Hedman for their valuable feedback on early versions of the manuscript. Their comments and suggestions helped us improve both the presentation and the clarity of this work. We thank Dany Bittel for providing the bumblebee capture data, and Will Eastcott for showing a 3DGS reconstruction of it to the world.

References

[1] Jonathan T. Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P. Srinivasan. Mip-NeRF: A multiscale representation for anti-aliasing neural radiance fields. In *ICCV*, 2021. 3

[2] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-NeRF 360: Unbounded anti-aliased neural radiance fields. In *CVPR*, 2022. 2, 3, 6, 7

[3] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Zip-NeRF: Anti-aliased grid-based neural radiance fields. In *ICCV*, 2023. 3

[4] Jules Bloomenthal and Ken Shoemake. Convolution surfaces. In *SIGGRAPH*, 1991. 2, 3

[5] Matt Jen-Yuan Chiang, Benedikt Bitterli, Chuck Tappan, and Brent Burley. A practical and controllable hair and fur model for production path tracing. *Computer Graphics Forum*, 35 (2):275–283, 2016. 7

[6] Changwoon Choi, Jaeah Lee, Jaesik Park, and Young Min Kim. 3Doodle: Compact abstraction of objects with 3D strokes. *ACM Transactions on Graphics*, 43(4), 2024. 3

[7] A. R. DiDonato, M. P. Jarnagin, Jr., and R. K. Hageman. Computation of the integral of the bivariate normal distribution over convex polygons. *SIAM Journal on Scientific and Statistical Computing*, 1(2):179–186, 1980. 2

[8] Antoine Guédon and Vincent Lepetit. Gaussian frosting: Editable complex radiance fields with real-time rendering. In *European Conference on Computer Vision (ECCV)*, pages 413–430, 2024. 6, 7

[9] Jan Held, Renaud Vandeghen, Abdullah Hamdi, Adrien Deliège, Anthony Cioppa, Silvio Giancola, Andrea Vedaldi, Bernard Ghanem, and Marc Van Droogenbroeck. 3D convex splatting: Radiance field rendering with 3D smooth convexes. In *CVPR*, 2025. 2, 3

[10] Jan Held, Sanghyun Son, Renaud Vandeghen, Daniel Rebain, Matheus Gadelha, Yi Zhou, Anthony Cioppa, Ming C. Lin, Marc Van Droogenbroeck, and Andrea Tagliasacchi. MeshSplatting: Differentiable Rendering with Opaque Meshes. In *Computer Vision and Pattern Recognition (CVPR)*, 2026. oral. 2

[11] Jan Held, Renaud Vandeghen, Adrien Deliège, Abdullah Hamdi, Silvio Giancola, Anthony Cioppa, Andrea Vedaldi, Bernard Ghanem, Andrea Tagliasacchi, and Marc Van Droogenbroeck. Triangle splatting for real-time radiance field rendering. In *3DV*, 2026. 2, 3

[12] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Symposium on Geometry Processing (SGP)*. The Eurographics Association, 2006. 2

[13] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3D gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), 2023. 2, 3, 6

[14] Shakiba Kheradmand, Daniel Rebain, Gopal Sharma, Weiwei Sun, Yang-Che Tseng, Hossam Isack, Abhishek Kar, Andrea Tagliasacchi, and Kwang Moo Yi. 3D gaussian splatting as Markov chain Monte Carlo. In *NeurIPS*, 2024. 3, 6, 7, 8

[15] Rachmadio Noval Lazuardi, Artem Sevastopolsky, Egor Zakharov, Matthias Niessner, and Vanessa Sklyarova. GeomHair: Reconstruction of hair strands from colorless 3D scans, 2026. 2, 3, 7

[16] Rong Liu, Dylan Sun, Meida Chen, Yue Wang, and Andrew Feng. Deformable beta splatting. In *SIGGRAPH Conference Papers*, 2025. 2, 3, 4, 5, 6, 7

[17] Haimin Luo, Min Ouyang, Zijun Zhao, Suyi Jiang, Longwen Zhang, Qixuan Zhang, Wei Yang, Lan Xu, and Jingyi Yu. GaussianHair: Hair modeling and rendering with light-aware gaussians. *arXiv preprint arXiv:2402.10483*, 2024. 3

[18] Alexander Mai, Peter Hedman, George Kopanas, Dor Verbin, David Futschik, Qiangeng Xu, Falko Kuester, Jonathan T. Barron, and Yinda Zhang. EVER: Exact volumetric ellipsoid rendering for real-time view synthesis. In *ICCV*, 2025. 3

[19] Saswat Subhajyoti Mallick, Rahul Goel, Bernhard Kerbl, Markus Steinberger, Francisco Vicente Carrasco, and Fernando De La Torre. Taming 3DGS: High-quality radiance fields with limited resources. In *SIGGRAPH Asia 2024 Conference Papers*, pages 1–11. ACM, 2024. 3

[20] Nelson Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99–108, 1995. 4

[21] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020. 2, 3

[22] Nicolas Moenne-Loccoz, Ashkan Mirzaei, Or Perel, Riccardo de Lutio, Janick Martinez Esturo, Gavriel State, Sanja Fidler, Nicholas Sharp, and Zan Gojcic. 3D gaussian ray tracing: Fast tracing of particle scenes. *ACM Transactions on Graphics*, 43(6), 2024. 3

[23] Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. Position based dynamics. *Journal of Visual Communication and Image Representation*, 18(2):109–118, 2007. 3

[24] Matthias Müller, Tae-Yong Kim, and Nuttapong Chentanez. Fast simulation of inextensible hair and fur. In *Workshop on Virtual Reality Interaction and Physical Simulation (VRIPHYS)*, 2012. 3

[25] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics*, 41 (4), 2022. 3

[26] Donald B. Owen. Tables for computing bivariate normal probabilities. *The Annals of Mathematical Statistics*, 27(4): 1075–1090, 1956. 2

[27] Lukas Radl, Michael Steiner, Mathias Parger, Alexander Weinrauch, Bernhard Kerbl, and Markus Steinberger. StopThePop: Sorted gaussian splatting for view-consistent real-time rendering. *ACM Transactions on Graphics*, 43(4), 2024. 3

[28] Radu Alexandru Rosu, Shunsuke Saito, Ziyan Wang, Chenglei Wu, Sven Behnke, and Giljoo Nam. Neural strands: Learning hair geometry and appearance from multi-view images. In *ECCV*, 2022. 3

[29] Francesco Di Sario, Daniel Rebain, Dor Verbin, Marco Grangetto, and Andrea Tagliasacchi. Spherical Voronoi: Directional appearance as a differentiable partition of the sphere. In *CVPR*, 2026. 2, 3, 6, 7

[30] Vanessa Sklyarova, Jenya Chelishev, Andreea Dogaru, Igor Medvedev, Victor Lempitsky, and Egor Zakharov. Neural haircut: Prior-guided strand-based hair reconstruction. In *ICCV*, 2023. 3, 8

[31] Andrea Tagliasacchi and Ben Mildenhall. Volume rendering digest (for nerf), 2022. 2, 5

[32] Yusuke Takimoto, Hikari Takehara, Hiroyuki Sato, Zihao Zhu, and Bo Zheng. Dr.Hair: Reconstructing scalp-connected hair strands without pre-training via differentiable rendering of line segments. In *CVPR*, 2024. 7

[33] Chinmay Talegaonkar, Yash Belhe, Ravi Ramamoorthi, and Nicholas Antipa. Volumetrically consistent 3D gaussian rasterization. In *CVPR*, 2025. 3

[34] Kenji Tojo, Ariel Shamir, Nobuyuki Umetani, and Bernd Bickel. Inverse rendering for modeling with line primitives. *ACM Transactions on Graphics*, 45(6), 2026. SIGGRAPH Asia 2026. 2, 3, 6, 7

[35] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004. 6

[36] Zian Wang, Tianchang Shen, Merlin Nimier-David, Nicholas Sharp, Jun Gao, Alexander Keller, Sanja Fidler, Thomas Müller, and Zan Gojcic. Adaptive shells for efficient neural radiance field rendering. *ACM Transactions on Graphics*, 42 (6), 2023. 6, 7

[37] Lee Westover. Footprint evaluation for volume rendering. In *SIGGRAPH*, 1990. 3

[38] Qi Wu, Janick Martinez Esturo, Ashkan Mirzaei, Nicolas Moenne-Loccoz, and Zan Gojcic. 3DGUT: Enabling distorted cameras and secondary rays in gaussian splatting. In *CVPR*, 2025. 3

[39] Kehan Xu, Benedikt Bitterli, Eugene d'Eon, and Wojciech Jarosz. Practical Gaussian process implicit surfaces with sparse convolutions. *ACM Transactions on Graphics*, 44(6), 2025. 3

[40] Haiyang Ying and Matthias Zwicker. SketchSplat: 3D edge reconstruction via differentiable multi-view sketch splatting. In *ICCV*, pages 25649–25659, 2025. 3

[41] Zehao Yu, Anpei Chen, Binbin Huang, Torsten Sattler, and Andreas Geiger. Mip-Splatting: Alias-free 3D gaussian splatting. In *CVPR*, 2024. 2, 3, 6

[42] Cem Yuksel. HAIR model files. http://www.cemyuksel.com/research/hairmodels/, 2026. 3, 7

[43] Egor Zakharov, Vanessa Sklyarova, Michael J. Black, Giljoo Nam, Justus Thies, and Otmar Hilliges. Human hair reconstruction with strand-aligned 3D gaussians. In *ECCV*, 2024. 2, 3, 7

[44] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018. 6

[45] Jialin Zhu, Jiangbei Yue, Feixiang He, and He Wang. 3D student splatting and scooping. In *CVPR*, 2025. 3

[46] Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, and Markus Gross. EWA splatting. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):223–238, 2002. 2, 3, 5