NURBS Splatting: A Unified Differentiable Rendering Framework for Vector Graphics

Jingye Qiu and Shizhe Zhou*

Hunan University, China
{anicoder, shizhe}@hnu.edu.cn

Abstract. Differentiable rendering of planar rational splines remains largely underexplored, despite their widespread use in vector graphics and design. Existing differentiable vector renderers primarily focus on Bézier curves and rely on analytic rasterization, which can suffer from gradient instability and limited flexibility. We propose NURBS Splatting, a unified framework that represents planar rational curves as continuous Gaussian fields. By sampling Gaussians along the curve parameter domain and inside closed regions, rendering is reformulated as a smooth accumulation process with stable gradients. Our method naturally supports long splines, rational weights, non-uniform knots, and closed-region filling. We demonstrate its effectiveness in calligraphy reconstruction, vectorization frameworks, and long-spline image abstraction, showing improved stability and reconstruction quality over existing approaches.

Keywords: Gaussian Splatting · NURBS · Differentiable Rendering

1 Introduction

Non-Uniform Rational B-Splines (NURBS) [32] are the standard curve and surface representation in CAD and manufacturing. Every major CAD kernel—SolidWorks, CATIA, Rhino, Siemens NX—and exchange formats such as STEP (ISO 10303) and IGES use NURBS, and isogeometric analysis [16] builds directly on them for finite-element simulation. Their advantage over polynomial B-splines is well established: rational weights and non-uniform knots let NURBS exactly represent conic sections—circles, ellipses, parabolic arcs—that polynomial splines can only approximate [10,32], while also providing per-control-point shape modulation for tighter local geometric control.

As pixel-based image generation models improve, differentiable rendering offers a promising way to convert these pixel outputs into physical, manufacturable vector formats. Because splines are compact and resolution-independent, they are highly suited for such tasks. Berio *et al*. [2], for example, showed that optimizing long smoothing splines can produce high-quality artistic abstractions that can be reproduced with robotic pen plotters. However, spline representations remain only partially explored in differentiable vector graphics. Current differentiable 2D renderers [23,25] are restricted to B'ezier curves, while Berio *et al*. [2]

* Corresponding author.

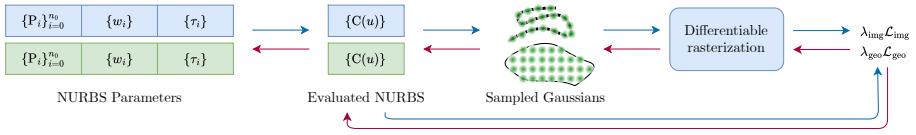

Fig. 1: We optimize NURBS parameters (control points $\mathbf{P}_i$, weights w_i, knot intervals τ_i) by sampling isotropic Gaussians along the curve and interior. These primitives are rendered via a differentiable tile-based rasterizer, enabling end-to-end optimization through image and geometric losses.

support B-splines but do not introduce rational weights or non-uniform knots. Consequently, differentiable renderers cannot faithfully represent shapes that CAD systems natively support—such as exact circular arcs and conic profiles—and remain disconnected from the NURBS-based modeling infrastructure that industry has relied on for decades.

We address this with **NURBS Splatting**. We adaptively sample isotropic Gaussians along NURBS curves—positions, widths, and analytical derivatives computed via the Cox–de Boor recursion—and inside closed regions via an SDF-modulated grid, then composite them through a tile-based Gaussian splatting rasterizer. The pipeline is fully differentiable: image-space losses drive gradients back to control points, rational weights, knot intervals, stroke widths, colors, and opacities. For filled closed curves, grid-step annealing progressively refines spatial resolution during optimization.

The main contributions of this work are as follows:

- The first differentiable renderer for NURBS curves in 2D image space, supporting joint optimization of rational weights and non-uniform knot vectors.
- A Gaussian splatting formulation for both open stroke contours and filled closed regions, combining adaptive arc-length sampling with SDF-modulated interior fill and grid-step annealing.
- Evaluation on calligraphy reconstruction, image vectorization, and neural image abstraction showing that rational splines improve both quality and speed over polynomial baselines.

2 Related Work

2.1 Differentiable Rendering

DiffVG [23] pioneered differentiable rasterization, which enabled gradient-based optimization of vector images. A variety of applications have since been explored, including image vectorization [4,14,28,40], text-guided drawing synthesis [12,37], text-to-SVG generation via score distillation [18, 44, 45], semantic typography [17], collage generation [41], animation generation [52], sketch refinement [35], *etc*. However, the DiffVG approach relies on per-pixel intersection tests, making high-resolution optimization slow and memory-intensive.

An alternative line of work uses Gaussian primitives for rendering. In 3D vision, Kerbl *et al*. [21] propose 3D Gaussian Splatting for novel-view synthesis. BézierGS [29] and SplineGS [48] further extend this paradigm with spline-based motion modeling. The idea has since been adapted to 2D domains. Huang *et al*. [15] introduce 2D Gaussian Splatting for radiance-field reconstruction, collapsing the 3D volume into oriented 2D Gaussian disks to enforce view-consistent geometry. Likewise, Zhang *et al*. [53] present GaussianImage, encoding images directly as collections of 2D Gaussians to achieve ultra-fast rendering and compression. In the context of vector graphics, Liu *et al*. [25] propose Bézier Splatting, which samples 2D Gaussian kernels along Bézier curves and rasterizes via a Gaussian-splatting pipeline. These works demonstrate that Gaussian splatting is a powerful differentiable primitive for both 3D scene reconstruction and 2D vector graphics rendering.

Notably, all the aforementioned methods employ either polynomial Bézier curves or polynomial B-splines as the underlying geometric primitives. To the best of our knowledge, no existing differentiable rendering framework supports rational spline primitives for 2D image-space optimization.

2.2 Non-Uniform Rational B-Splines

NURBS [32] generalize polynomial B-splines by introducing rational weights and non-uniform knot vectors, allowing exact representation of conic sections, analytic curves, and flexible free-form shapes. They have become the de facto industry standard for modeling curves and surfaces in CAD and graphics. While classical algorithms like the Cox–de Boor recursion [6, 7] evaluate and sample points for rendering, many modern professional tools employ GPU curve rasterization based on the implicit form of rational Bézier curves obtained from NURBS via knot insertion [26]. Beyond geometric modeling, NURBS have found broader scientific impact through isogeometric analysis [16], which unifies CAD geometry and finite-element analysis by using NURBS basis functions directly as the computational basis.

Recent work has begun to integrate NURBS into differentiable and learning-based pipelines. Deva Prasad *et al*. [8] introduce NURBS-Diff, a differentiable NURBS layer that provides forward and backward passes through the NURBS evaluation process, enabling tasks such as curve and surface fitting, surface offsetting, *etc*. Worchel and Alexa [42] propose differentiable rendering of parametric geometry, including B-spline surfaces, by directly deriving screen-space gradients for parametric primitives. Tojo *et al*. [36] optimize B-spline curves via differentiable rendering for fabricable 3D wire art. On the generative side, SplineGen [54] leverages a generative model to fit B-spline surfaces to unorganized point clouds, NeuroNURBS [9] learns compact NURBS surface representations for 3D solids, and BrepGen [46] employs diffusion models to generate B-rep CAD models whose faces are NURBS surfaces. Despite this growing interest, existing differentiable and generative NURBS methods predominantly target 3D geometry fitting or CAD reconstruction rather than image-based rendering.

3 Method

Our pipeline is illustrated in Fig. 1. We first define NURBS curve primitives with learnable control points, rational weights, and knot vectors (Sec. 3.1). Isotropic Gaussians are then adaptively sampled along curve contours (Sec. 3.2) or within closed filled regions via an SDF-modulated grid (Sec. 3.3). These Gaussians are composited through a tile-based splatting rasterizer to produce a differentiable image (Sec. 3.4), and image-space losses drive gradients back to all curve parameters (Sec. 3.5). Although we focus on NURBS, our Gaussian sampling formulation is generic and readily extends to other parametric primitives.

3.1 NURBS Primitives

Each curve in our system is a NURBS curve of degree p (order k=p+1) with $n+1$ control points:

$$\mathbf{C}(u) = \frac{\sum_{i=0}^{n} N_{i,p}(u)\, w_i\, \mathbf{P}_i}{\sum_{i=0}^{n} N_{i,p}(u)\, w_i} = \frac{\mathbf{S}(u)}{W(u)}, \quad u \in [u_0, u_m], \tag{1}$$

where $\{\mathbf{P}_i \in \mathbb{R}^2\}$ are control points, $\{w_i \in \mathbb{R}^+\}$ are rational weights, and $\{N_{i,p}(u)\}$ are B-spline basis functions defined over a non-decreasing knot vector $\mathbf{U} = \{u_0, u_1, \ldots, u_m\}$. The basis functions $N_{i,p}(u)$ are defined by the Cox–de Boor recursion [6, 7], which guarantees compact support and C^{p-1} continuity.

The curve derivative, needed for arc-length estimation and regularization, follows by the quotient rule:

$$\mathbf{C}'(u) = \frac{W(u)\sum_{i=0}^{n} N'_{i,p}(u)\, w_i\, \mathbf{P}_i - \mathbf{S}(u)\sum_{i=0}^{n} N'_{i,p}(u)\, w_i}{W^2(u)}, \tag{2}$$

where the basis-function derivatives $N'_{i,p}$ are also obtainable via the Cox–de Boor recursion. The arc length used in Sec. 3.2 can be approximated by numerical quadrature:

$$L = \int_{u_p}^{u_{m-p}} \|\mathbf{C}'(u)\|_2\, du \approx \sum_{j=1}^{N_L} \|\mathbf{C}'(u_j)\|_2\, \Delta u. \tag{3}$$

Rather than optimizing control points and knots directly, we leverage key points $\{\mathbf{K}_i\}_{i=1}^{n_k}$ as the primary learnable parameters and derive control points and knot vectors from them.

For open curves, the control points coincide with the key points, and the knot vector is clamped at both endpoints with multiplicity k. Only the $n_k - k + 1$ interior knot intervals $\{\tau_j > 0\}$ are stored as learnable parameters; the full knot vector is reconstructed by prepending and appending p zero-intervals (to enforce endpoint multiplicity) and taking a cumulative sum, which inherently guarantees monotonicity.

For closed curves, the control polygon is formed by cyclically wrapping the last $\lceil p/2 \rceil$ and first $\lfloor p/2 \rfloor$ key points to produce $n_k + p$ control points. We store

n_k core knot intervals; the periodic knot vector is then built by wrapping the last p core intervals before and the first p after, followed by a cumulative sum.

Each curve also carries an RGB color $\mathbf{c} \in [0,1]^3$ and an opacity $o \in [0,1]$. We extend the control points to $\mathbb{R}^3$ by appending a third coordinate encoding local stroke width, so evaluating $\mathbf{C}(u)$ simultaneously yields position and width at parameter u.

3.2 Gaussian Sampling on Contours

Given the knot vector $\mathbf{U}$ and control points derived from the key points (Sec. 3.1), we adaptively place isotropic Gaussians along each curve with density proportional to arc length.

We first approximate the arc length L via Eq. (3) using N_L uniformly spaced parameter samples, then set the number of Gaussians as $M = \lceil \delta_c \cdot L \rceil$, where δ_c is a user-specified contour density. The M parameter values $\{u_j\}_{j=1}^{M}$ are distributed uniformly within the valid knot domain $[u_p, u_{m-p}]$.

To evaluate each sample efficiently, we exploit the compact support of B-spline basis functions: at parameter value u_j, only the $p+1$ basis functions whose support contains u_j are non-zero. The Gaussian center is therefore computed via a span-local formulation:

$$\boldsymbol{\mu}_j = \frac{\sum_{l=0}^{p} N_{i_j-p+l,p}(u_j)\, w_{i_j-p+l}\, \mathbf{P}_{i_j-p+l}}{\sum_{l=0}^{p} N_{i_j-p+l,p}(u_j)\, w_{i_j-p+l}}, \tag{4}$$

where i_j is the knot span index satisfying $u_{i_j} \le u_j < u_{i_j+1}$. This reduces evaluation cost from $O(Mn)$ to $O(Mp)$ per curve. The local stroke width s_j is read from the third coordinate of $\mathbf{C}(u_j)$; higher-order analytical derivatives, used for regularization (Sec. 3.5), are obtained through the same local recursion.

Each sample is assigned an isotropic Gaussian whose scale depends solely on the local stroke width:

$$\sigma_{j,x} = \sigma_{j,y} = s_j/\rho, \qquad \theta_j = 0, \qquad o_j = o, \tag{5}$$

where ρ is a global scale ratio controlling overlap between neighboring Gaussians and o is the curve-level opacity from Sec. 3.1. Unlike Bézier Splatting [25], which aligns anisotropic Gaussians to the curve tangent, this isotropic formulation eliminates rotation estimation and avoids visual artifacts—blurry edges at large stroke widths and spikes at high-curvature regions. Because M scales with arc length, Gaussian centers remain sufficiently dense that circular kernels provide smooth, uniform coverage without directional alignment.

For open curves, we replicate the first and last Gaussian centers κ times, where κ is a fixed number of times proportional to δ_c, ensuring opaque, well-defined endpoints without additional parameters.

3.3 Gaussian Sampling in Regions

For closed curves marked as filled, we replace contour Gaussians with a dense set of interior Gaussian primitives whose opacity is modulated by a differentiable signed distance field (SDF).

We first sample $\lceil \delta_{\mathrm{b}} \cdot L \rceil$ boundary points $\{\mathbf{b}_j\}$ along the closed curve in pixel coordinates, where δ_{b} is a user-specified boundary density and L is the arc length from Sec. 3.2. These boundary points define the polyline against which the SDF is evaluated.

Next, we compute the axis-aligned bounding box of the boundary with symmetric padding proportional to its extent, clipped to the canvas. A uniform 2D grid at step size h is placed inside this padded box, yielding $Q = n_x \times n_y$ query points $\{\mathbf{q}_j\}$ in pixel coordinates. A larger h produces a coarser grid with fewer Gaussians, trading density for speed; we anneal h during optimization to accelerate early convergence while preserving final detail (Sec. 4.2).

For each query point $\mathbf{q}_j$, we generate an isotropic Gaussian primitive:

$$\boldsymbol{\mu}_j = \mathbf{q}_j, \qquad \theta_j = 0, \qquad \sigma_{j,x} = \sigma_{j,y} = \frac{\eta \cdot h}{\rho}, \tag{6}$$

where η is a coverage factor (we use η=1.5) that ensures sufficient overlap between neighboring Gaussians, and ρ is the global scale ratio from Sec. 3.2. The base opacity is the product of the curve-level opacity o and a sigmoid soft mask derived from the SDF:

$$o_j = o \cdot \mathrm{sigmoid}\Big(\frac{\gamma}{h} \cdot \mathrm{sdf}(\mathbf{q}_j)\Big), \tag{7}$$

where γ controls the sharpness of the boundary transition, and $\mathrm{sdf}(\mathbf{q}_j)$ returns positive values inside the region and negative values outside, computed with respect to the boundary polyline $\{\mathbf{b}_j\}$. Detailed algorithmic steps for computing the SDF are provided in the supplementary material.

Each Gaussian inherits its color from the parent curve. This pixel-space SDF formulation scales to large filled shapes while remaining fully differentiable with respect to boundary point positions.

3.4 Differentiable Splatting Rendering

Once Gaussian primitives are sampled from NURBS curves, we rasterize them using the tile-based Gaussian splatting rasterizer of GaussianImage [53].

The rasterizer sorts the 2D Gaussians by their stacking order (depth) and composites them front-to-back for each pixel. The rendered color $\mathcal{I}(\mathbf{x}_n)$ at pixel $\mathbf{x}_n$ is obtained by alpha blending the contributions of overlapping Gaussians:

$$\mathcal{I}(\mathbf{x}_n) = \sum_{i \in \mathcal{N}} \mathbf{c}_i \, \alpha_i(\mathbf{x}_n) \prod_{j=1}^{i-1} \big(1 - \alpha_j(\mathbf{x}_n)\big), \tag{8}$$

where $\mathcal{N}$ is the depth-ordered set of Gaussians covering pixel $\mathbf{x}_n$, $\mathbf{c}_i$ is the color of the i-th Gaussian, and $\alpha_i(\mathbf{x}_n)$ is its opacity at that location. Any residual transmittance is composited against a predefined background color.

Since all sampled Gaussians are isotropic with $\sigma_{i,x} = \sigma_{i,y} =: \sigma_i$ and $\theta_i = 0$ (Secs. 3.2 and 3.3), the covariance reduces to $\boldsymbol{\Sigma}_i = \sigma_i^2 \mathbf{I}$ and the per-pixel

Gaussian opacity simplifies to

$$\alpha_i(\mathbf{x}_n) = o_i \exp\!\left(-\frac{\|\mathbf{x}_n - \boldsymbol{\mu}_i\|^2}{2\,\sigma_i^2}\right), \tag{9}$$

where o_i is the base opacity and $\boldsymbol{\mu}_i$ is the Gaussian center. This formulation enables efficient backpropagation of gradients from the image loss through the Gaussian parameters to the NURBS control points and weights.

3.5 Optimization

The overall objective combines an image-space reconstruction loss with geometric regularization:

$$\mathcal{L} = \mathcal{L}_{\text{img}}(\mathcal{I}, \mathcal{I}^*) + \lambda_{\text{geo}}\,\mathcal{L}_{\text{geo}}, \tag{10}$$

where $\mathcal{I}$ and $\mathcal{I}^*$ are the rendered and target images, $\mathcal{L}_{\text{img}}$ is an image-based loss, and $\mathcal{L}_{\text{geo}}$ enforces regularity on the underlying NURBS curves. Gradients propagate from pixels through the Gaussian parameters to NURBS control points and weights via the differentiable pipeline above; the specific choice of $\mathcal{L}_{\text{img}}$ is application-dependent. Below we describe the two regularization terms shared across all applications; additional task-specific losses are introduced in Sec. 4.

Derivative Loss, the squared magnitude of the r-th derivative of a parametric curve, is a classical smoothing energy in curve fairing [33]: the second derivative relates to curvature, and the third captures its rate of change. We penalize this energy averaged over the parameter domain:

$$\mathcal{L}^r_{\text{deriv}}(\mathbf{C}) = \frac{1}{u_{\max} - u_{\min}} \int_{u_{\min}}^{u_{\max}} \left\| \frac{d^r \mathbf{C}(u)}{du^r} \right\|_2^2 du, \tag{11}$$

where r denotes the derivative order. We evaluate this integral using analytical derivatives obtained from Eq. (2) during contour sampling (Sec. 3.2), avoiding the numerical errors that arise from finite-difference approximations. The total loss averages $\mathcal{L}^r_{\text{deriv}}$ over all curves; following Berio *et al.* [2], we set $r = 3$.

Bounding Box Loss prevents curves from drifting outside the target region. We penalize points that violate the image bounding box:

$$\mathcal{L}_{\text{bbox}} = \sum_i \varphi(\mathbf{b}_{\min} - \mathbf{P}_i) + \varphi(\mathbf{P}_i - \mathbf{b}_{\max}), \tag{12}$$

where $\mathbf{b}_{\min}$ and $\mathbf{b}_{\max}$ are the bounding-box corners, $\mathbf{P}_i$ ranges over all sampled curve positions, and φ is a soft penalty (ReLU or Softplus) that activates only when a point lies outside the permitted region.

4 Applications

We demonstrate the versatility of NURBS Splatting on three representative vector-graphics tasks: calligraphy reconstruction (Sec. 4.1), layer-wise image vectorization (Sec. 4.2), and neural image abstraction (Sec. 4.3).

4.1 Calligraphy Reconstruction

In calligraphy, stroke trajectories and their varying widths encode both style and meaning. Recovering these as editable vector strokes from raster images is useful for font design, digital archiving, and robotic handwriting reproduction.

Most calligraphy generation methods produce raster images via GANs [43, 49] or diffusion models [24], discarding stroke structure entirely. Methods that output vector outlines as Bézier sequences [34, 38, 39] describe glyphs as closed regions, not as strokes with varying width. Classical stroke extraction [5, 27] and the segmentation method of StrokeStyles [1] do model individual strokes but lack end-to-end differentiable optimization in image space.

Our framework represents each stroke as a NURBS curve with a per-control-point width channel. For initialization, we use the line vectorization method of Magne and Sorkine-Hornung [30]: a medial axis is extracted from the binarized image via the Voronoi diagram method [3], a CNN classifies intersections to produce ordered sample points along the axis, and NURBS curves are fit to these points with constant initial width.

The curves are then optimized end-to-end against the raster target with MSE loss and geometric regularization:

$$\mathcal{L} = \lambda_{\text{MSE}}\mathcal{L}_{\text{MSE}} + \lambda_{\text{deriv}}\mathcal{L}^3_{\text{deriv}} + \lambda_{\text{bbox}}\mathcal{L}_{\text{bbox}}. \tag{13}$$

Control point positions, NURBS weights, knot intervals, and per-control-point widths are updated jointly, recovering both stroke trajectories and thickness variations. Results are presented in Section 5.2.

4.2 Layer-wise Image Vectorization

Ma *et al.* [28] propose LIVE, a layer-wise vectorization approach that greedily adds closed Bézier paths one at a time, each initialized at the region of highest reconstruction error and optimized via DiffVG [23]. A distance-weighted loss modulates per-pixel gradients based on proximity to the newly added shape boundary, concentrating optimization effort where it matters most. We adapt LIVE to our framework, replacing the DiffVG rendering backend with NURBS Splatting while introducing a *grid-step annealing* strategy for filled shapes.

To accelerate convergence without sacrificing fidelity, we anneal the fill grid step h from a coarse value ($h_{\max}$=4) to a fine one ($h_{\min}$=1) using a cosine schedule between iteration fractions t_b=0.1 and t_e=0.8:

$$h(t) = h_{\max} + (h_{\min} - h_{\max})\,\frac{1+\cos(\pi(1-\phi))}{2}, \quad t/T \in (t_b, t_e), \tag{14}$$

where $\phi = (t/T - t_b)/(t_e - t_b)$ and h is clamped outside the range. The coarse grid reduces the number of primitives in the early stages, thereby stabilizing large-scale deformations (Fig. 2).

The overall loss for each round combines a pixel-wise reconstruction term with geometric regularization:

$$\mathcal{L} = \mathcal{L}_{\text{img}} + \lambda_{\text{deriv}}\mathcal{L}^3_{\text{deriv}} + \lambda_{\text{xing}}\mathcal{L}_{\text{xing}}, \tag{15}$$

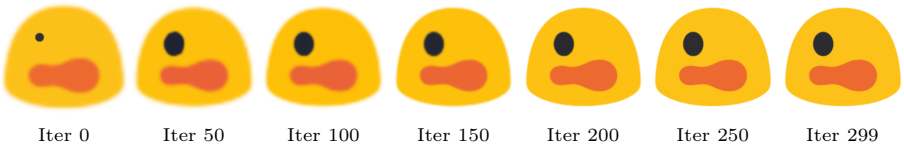

Fig. 2: Visualization of grid-step annealing. The grid resolution progressively increases, allowing coarse-to-fine optimization.

Fig. 3: Single-stroke area-filling results. Letters are stylized with different style images; digits use coverage loss only (no stylization).

where $\mathcal{L}_{\text{img}}$ is the distance-weighted MSE loss, $\mathcal{L}^3_{\text{deriv}}$ is the third-order derivative smoothing loss, and $\mathcal{L}_{\text{xing}}$ is a self-crossing penalty adapted from LIVE that penalizes control-point configurations whose consecutive segments reverse winding direction. Results are presented in Section 5.3.

4.3 Neural Image Abstraction

Berio *et al*. [2] apply long smoothing B-splines to generate abstracted vector images from raster inputs. Because NURBS generalize polynomial B-splines, these applications transfer naturally to our framework while gaining rational-weight flexibility. We demonstrate two representative tasks: single-stroke area filling and diffusion-guided image abstraction.

The *Single-Stroke Area Filling* task aims to produce a single NURBS curve that fills a colored shape given its bitmap image. We initialize a long NURBS curve from a TSP route [19] and optimize it with an MSE coverage loss between the rendered and target images. Lowering the target opacity reduces curve density in the interior. To enable semantic stylization, we add a term $\mathcal{L}_{\text{style}}$ based on the patch-wise directional CLIP loss of Kwon and Ye [22], which aligns encoded features of the rendered curves with those of a reference style image. Geometric regularization via $\mathcal{L}^3_{\text{deriv}}$ and $\mathcal{L}_{\text{bbox}}$ enforces smoothness and prevents control-point drift. Representative results appear in Fig. 3.

For the *Single-Stroke Image Abstraction* task, we couple our renderer with a diffusion-based Score Distillation Sampling (SDS) pipeline. Following Berio *et al*. [2], the pipeline uses ControlNet [50] with Canny edge detection and IP-Adapter [47], guided by the text prompt "A black and white ink drawing." The SDS gradient with respect to NURBS parameters $\boldsymbol{\psi}$ is

$$\nabla_{\boldsymbol{\psi}} \mathcal{L}_{\text{SDS}}(\boldsymbol{\psi}) = \mathbb{E}_t \left[w(t) \left(\hat{\boldsymbol{\epsilon}}(x_t, t, y) - \boldsymbol{\epsilon} \right) \frac{\partial g(\boldsymbol{\psi})}{\partial \boldsymbol{\psi}} \right], \tag{16}$$

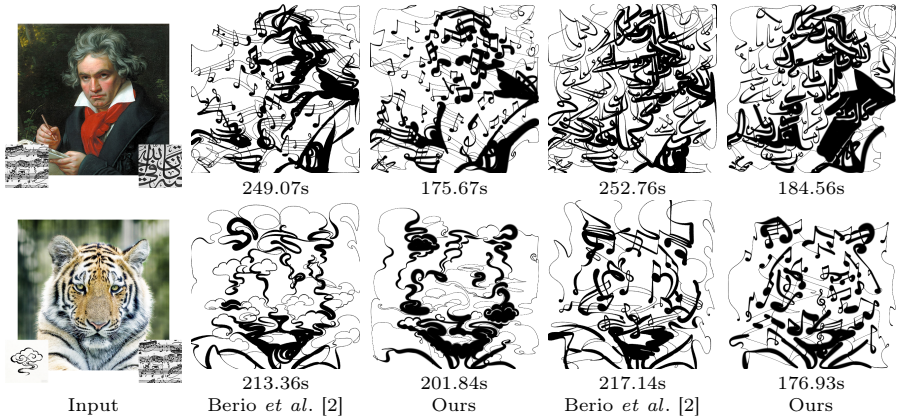

Fig. 4: Image abstraction with semantic stylization. Style images are inset in the bottom corners of the inputs. Our method matches stylization quality while running 1.27× faster (single NVIDIA RTX 4090, 512×512, 300 iterations).

where $\hat{\boldsymbol{\epsilon}}(x_t, t, y)$ is the predicted denoising direction of the frozen diffusion model for a latent x_t at time step t, $\boldsymbol{\epsilon}$ is the sampled noise, and $w(t)$ is a weighting function dependent on t. The final loss combines SDS with geometric regularization $\mathcal{L}^3_{\text{deriv}}$ and $\mathcal{L}_{\text{bbox}}$. As shown in Fig. 4, our method achieves 1.27× faster optimization than Berio *et al.* [2] at comparable stylization quality—even though differentiable rendering is not the primary bottleneck of the SDS pipeline and NURBS evaluation is more complex than polynomial B-spline evaluation.

5 Experiments

We implement our NURBS Splatting framework in PyTorch. We show general comparisons on rendering efficiency, quality, and editability in Sec. 5.1; experiments for our calligraphy reconstruction application are detailed in Sec. 5.2; experiments for LIVE are in Sec. 5.3. All experiments are run on a single NVIDIA RTX 4090. Code and dataset are available at `https://github.com/AnicoderAndy/nurbs-splatting`.

5.1 Efficiency, Quality, and Editability

Our method shows improved efficiency compared to Berio *et al.* [2] when the number of control points scales, as it avoids matrix multiplication to convert the splines. However, the superiority is not seen when rendering with simple geometric primitives, comparing our method with Berio *et al.* [2] and Bézier Splatting (BS) [25]. Details are provided in the supplementary material.

As for rendering quality, we qualitatively compare our method with BS which uses anisotropic Gaussian sampling on three diagnostic cases (Fig. 5). For thick

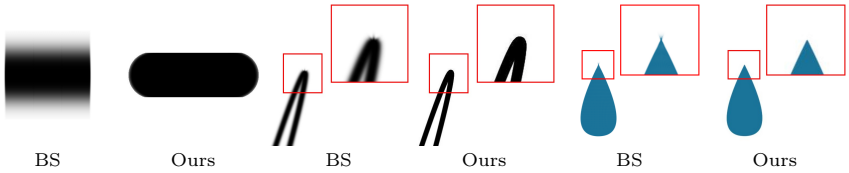

Fig. 5: Rendering comparison on three diagnostic cases. Bézier Splatting (BS) shows blurred edges on thick strokes (left pair), spike artifacts near high-curvature regions (middle pair, local crop), and spikes at filled-region junctions (right pair). Bézier splines are equivalently converted to NURBS for fair geometric comparison.

Table 1: Quantitative comparison of calligraphy reconstruction quality and runtime. The baseline is Berio *et al.* [2]. Best and second-best results are highlighted in **bold** and underline, respectively.

Method	MSE↓	PSNR↑	SSIM↑	Hausdorff↓	F1↑	Opt. Time (s)
Baseline	0.0057	24.72	<u>0.9813</u>	15.26	0.9642	10.65
Ours	**0.0038**	**26.32**	0.9793	**10.69**	<u>0.9741</u>	8.73
Ours (δ_c=30)	<u>0.0039</u>	<u>26.24</u>	**0.9824**	11.07	**0.9743**	9.34
Ours w/o weights	0.0046	25.63	0.9777	11.70	0.9701	8.98
Ours w/o knots	0.0040	26.07	0.9789	<u>10.98</u>	0.9731	<u>5.63</u>
Ours w/o w & k	0.0042	25.92	0.9785	11.34	0.9721	**5.12**

strokes, BS exhibits severely blurred edges. In the high-curvature case, BS produces spike artifacts due to the discontinuity and instability of Gaussian rotation evaluation via `atan2`. In the filled-region case, BS generates spikes at region junctions caused by incorrect curve interpolation at the shared endpoints of paired Bézier curves. Our method resolves all three artifacts through purely isotropic Gaussian sampling, consistently yielding sharp boundaries and smooth geometry.

A major advantage of vector graphics is their editability. The control points of optimized NURBS curves from our method provide a clean structure allowing manual adjustment. We present a visualization in the supplementary material.

5.2 Calligraphy Reconstruction

We construct a calligraphy benchmark of 192 characters: all 92 Japanese Kana and the first 100 characters from General Standard Chinese Characters [31]. Each character is rasterized from existing fonts [11, 20] to 512×512 images.

Since, to the best of our knowledge, no existing method performs stroke-level reconstruction of calligraphy, we implement a baseline based on Berio *et al.* [2], which optimizes uniform B-splines. Both methods use identical initialization via medial-axis extraction [30]. We evaluate reconstruction quality using standard pixel-level measures (MSE, PSNR, SSIM) and geometry-aware metrics (symmetric Hausdorff distance and F1 score on binarized stroke masks).

Fig. 6: Qualitative comparison of calligraphy reconstruction. Top row: Japanese characters. Bottom row: Chinese characters. Our method better preserves stroke trajectories and width variations compared to the baseline [2].

Table 1 shows the quantitative results averaged over 191 characters. At the contour density δ_c=18, our full method reduces MSE by 33%, improves PSNR by +1.6 dB, and lowers the Hausdorff distance by 30%, while running 1.2× faster. The speed gain stems from the avoidance of the matrix multiplication required to convert B-splines into Bézier curves. Gains are most pronounced on Japanese Kana (+2.10 dB PSNR), where frequent sharp bends and hooks benefit from non-uniform knot spacing that concentrates basis-function support in high-curvature regions. Figure 6 presents qualitative comparisons on representative characters, showing that our method more faithfully preserves stroke trajectories and width variation than the baseline.

The default configuration drops slightly in SSIM because isotropic Gaussians produce softer boundary transitions than the analytic edge rendering of DiffVG. Increasing the sampling density to δ_c=30 sharpens these boundaries, recovering the best SSIM (**0.9824**) and achieving the highest F1 (**0.9743**) at only a modest increase in optimization time.

Comparing only uniform polynomial B-splines—the Baseline against our "w/o w & k" variant—isolates the effect of the rendering backend: splatting yields a 2.1× speedup (5.12 s vs. 10.65 s) while simultaneously improving PSNR by +1.2 dB and Hausdorff by 26%. This confirms that the Gaussian rasterization pipeline is consistently beneficial even without NURBS-specific parameters.

Ablation studies (bottom three rows of Tab. 1) isolate the individual contributions of learnable weights and knots. Removing rational weights ("w/o weights") incurs the largest single-component degradation (−0.69 dB PSNR, +1.01 px Hausdorff relative to the full model), indicating that per-control-point weighting is especially valuable for modulating local curvature. Removing knots ("w/o knots") has a smaller effect on reconstruction quality but provides the largest time saving (5.63 s). Jointly dropping both ("w/o w & k") still comfortably outperforms the baseline on all metrics except SSIM, while reducing optimization time to roughly half that of the baseline (5.12 s)—offering a strong

Table 2: Quantitative comparison on layer-wise image vectorization (100 Noto Emoji images at 480×480). Best and second-best results are in **bold** and underline.

Method	MSE↓	SSIM↑	LPIPS↓	Time (s)↓
Baseline	0.0025	0.9633	0.0576	590
Ours	<u>0.0017</u>	<u>0.9803</u>	<u>0.0339</u>	<u>410</u>
Ours w/o step	**0.0015**	**0.9839**	**0.0334**	527
Ours w/o w & k	0.0018	0.9795	0.0348	**354**

quality–speed trade-off for applications where rational parameterization is unnecessary. Importantly, the benefits of weights and knots are complementary: the full model improves +0.25 dB over the next-best single-ablation variant ("w/o knots"), confirming that rational weights and non-uniform knots address distinct geometric limitations.

5.3 Layer-wise Image Vectorization

Since the original dataset used in LIVE [28] is not publicly available, we construct a new evaluation benchmark using Noto Emoji v1 [13], following their protocol. We select 100 representative emojis and rasterize them at a resolution of 480×480. We compare our method against the original LIVE [28], which uses DiffVG [23] as its rendering backend.

In each round, both methods initialize a new closed path with 12 key points as a circle and optimize for 300 iterations; 10 rounds are run in total, yielding 10 paths per image. We report standard reconstruction metrics: MSE, SSIM, and LPIPS [51], and measure the total optimization time per image.

Table 2 summarizes quantitative results averaged over 100 emojis. All our variants substantially outperform the LIVE baseline on every metric while also requiring less optimization time.

Our full method with the grid-step annealing strategy reduces MSE by 32%, improves SSIM by +0.017, and lowers LPIPS by 41%, while achieving a 1.4× speedup. Disabling step annealing ("w/o step") further improves reconstruction quality at the cost of longer optimization (527 s), yet remains 11% faster than LIVE, indicating that our optimization framework remains computationally efficient even without the annealing strategy. Removing rational weights and knots ("w/o w & k") still outperforms the baseline on all metrics—reducing MSE by 28%, improving SSIM by +0.016, and lowering LPIPS by 40%—while achieving the fastest optimization at 354 s (1.7× speedup), albeit with a smaller quality margin than the full model.

Qualitatively, Fig. 7 demonstrates that our method is more robust to complex topologies, particularly for thin interleaving structures in facial expressions, where the DiffVG-based LIVE baseline often exhibits artifacts such as missing details and geometric distortions.

Fig. 7: Qualitative comparison of vector reconstruction. Our method faithfully reconstructs complex facial expressions and fine details where the DiffVG-based baseline tends to fail.

6 Conclusion

We presented NURBS Splatting, the first differentiable rendering framework capable of optimizing rational B-splines in 2D image space. By representing curves and filled regions as continuous Gaussian fields, our method unifies vector graphics rendering with gradient-based optimization, allowing joint refinement of control points, rational weights, and non-uniform knot vectors. Experiments on calligraphy reconstruction and image vectorization demonstrate consistent improvements over polynomial baselines, achieving higher reconstruction fidelity. Crucially, the ability to optimize rational weights introduces geometric flexibility beyond prior differentiable vector graphics methods, narrowing the gap between learning-based reconstruction and CAD-standard representations.

Our current region-filling strategy relies on a heuristic SDF-based grid, which provides limited advantages over scanline methods for regular geometries. In addition, although splatting enables efficient rendering, the number of Gaussian primitives can grow with curve complexity, potentially increasing memory usage for highly detailed illustrations compared to implicit representations. Future work will focus on extending the framework to 3D NURBS surfaces to support native CAD geometry, developing a more efficient analytic region-filling algorithm, and exploring integration with generative models to synthesize editable vector designs directly from text or images.

Acknowledgements

We thank the anonymous reviewers for their valuable feedback and suggestions. This work was funded by the National Natural Science Foundation of China, No. 62076090; the Huxiang Youth Talent Support Program, Hunan Province, China, No. 2020RC3014; and the Natural Science Foundation of Hunan Province, China, No. 2022JJ30173.

References

1. Berio, D., Leymarie, F.F., Asente, P., Echevarria, J.: StrokeStyles: Stroke-based Segmentation and Stylization of Fonts. ACM Trans. Graph. **41**(3), 28:1–28:21 (Apr 2022). https://doi.org/10.1145/3505246
2. Berio, D., Stroh, M., Calinon, S., Fol Leymarie, F., Deussen, O., Shamir, A.: Neural Image Abstraction using Long Smoothing B-Splines. ACM Trans. Graph. **44**(6), 225:1–225:11 (Dec 2025). https://doi.org/10.1145/3763345
3. Brandt, J.W., Algazi, V.R.: Continuous skeleton computation by Voronoi diagram. CVGIP: Image Understanding **55**(3), 329–338 (May 1992). https://doi.org/10.1016/1049-9660(92)90030-7
4. Chakraborty, S., Batra, V., Phogat, A., Jain, V., Ranawat, J.S., Dhingra, S., Wampler, K., Fisher, M., Lukáč, M.: Image Vectorization via Gradient Reconstruction. Comput. Graph. Forum **44**(2) (May 2025). https://doi.org/10.1111/cgf.70055
5. Chen, X., Lian, Z., Tang, Y., Xiao, J.: An automatic stroke extraction method using manifold learning. In: Peytavie, A., Bosch, C. (eds.) Proceedings of the European Association for Computer Graphics: Short Papers. pp. 65–68. EG '17, Eurographics Association, Goslar, DEU (Apr 2017). https://doi.org/10.2312/egsh.20171016
6. Cox, M.G.: The Numerical Evaluation of B-Splines. IMA Journal of Applied Mathematics **10**(2), 134–149 (Oct 1972). https://doi.org/10.1093/imamat/10.2.134
7. de Boor, C.: On calculating with B-splines. Journal of Approximation Theory **6**(1), 50–62 (Jul 1972). https://doi.org/10.1016/0021-9045(72)90080-9
8. Deva Prasad, A., Balu, A., Shah, H., Sarkar, S., Hegde, C., Krishnamurthy, A.: NURBS-Diff: A Differentiable Programming Module for NURBS. Computer-Aided Design **146**, 103199 (May 2022). https://doi.org/10.1016/j.cad.2022.103199
9. Fan, J., Gholami, B., Bäck, T., Wang, H.: NeuroNURBS: Learning Efficient Surface Representations for 3D Solids (Nov 2024). https://doi.org/10.48550/arXiv.2411.10848
10. Farin, G.: Curves and Surfaces for CAGD: A Practical Guide. Morgan Kaufmann (Sep 2007). https://doi.org/10.1016/B978-1-55860-737-8.X5000-5
11. Founder Type Foundry: Founder Ouyang Xun Regular Script Font. https://www.foundertype.com/index.php/FontInfo/index/id/6583.html, accessed: 2026-02-28
12. Frans, K., Soros, L., Witkowski, O.: CLIPDraw: Exploring text-to-drawing synthesis through language-image encoders. In: Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., Oh, A. (eds.) Advances in Neural Information Processing Systems. vol. 35, pp. 5207–5218. Curran Associates, Inc. (2022)
13. Google Fonts: Noto Emoji. https://github.com/googlefonts/noto-emoji, accessed: 2026-03-05
14. Hirschorn, O., Jevnisek, A., Avidan, S.: Optimize & Reduce: A Top-Down Approach for Image Vectorization. Proceedings of the AAAI Conference on Artificial Intelligence **38**(3), 2148–2156 (Mar 2024). https://doi.org/10.1609/aaai.v38i3.27987
15. Huang, B., Yu, Z., Chen, A., Geiger, A., Gao, S.: 2D Gaussian Splatting for Geometrically Accurate Radiance Fields. In: ACM SIGGRAPH 2024 Conference Papers. pp. 1–11. SIGGRAPH '24, Association for Computing Machinery, New York, NY, USA (Jul 2024). https://doi.org/10.1145/3641519.3657428
16. Hughes, T.J.R., Cottrell, J.A., Bazilevs, Y.: Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. Computer Methods in

Applied Mechanics and Engineering **194**(39), 4135–4195 (Oct 2005). https://doi.org/10.1016/j.cma.2004.10.008
17. Iluz, S., Vinker, Y., Hertz, A., Berio, D., Cohen-Or, D., Shamir, A.: Word-As-Image for Semantic Typography. ACM Trans. Graph. **42**(4), 151:1–151:11 (Jul 2023). https://doi.org/10.1145/3592123
18. Jain, A., Xie, A., Abbeel, P.: VectorFusion: Text-to-SVG by Abstracting Pixel-Based Diffusion Models. In: 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 1911–1920 (Jun 2023). https://doi.org/10.1109/CVPR52729.2023.00190
19. Kaplan, C.S., Bosch, R.: TSP Art. In: Sarhangi, R., Moody, R.V. (eds.) Renaissance Banff: Mathematics, Music, Art, Culture. pp. 301–308. Bridges Conference, Southwestern College, Winfield, Kansas (2005)
20. Keikan & Midnight Garden: Yoppa Fude Font. https://karu-k.booth.pm/items/2985842, accessed: 2026-02-28
21. Kerbl, B., Kopanas, G., Leimkuehler, T., Drettakis, G.: 3D Gaussian Splatting for Real-Time Radiance Field Rendering. ACM Trans. Graph. **42**(4), 139:1–139:14 (Aug 2023). https://doi.org/10.1145/3592433
22. Kwon, G., Ye, J.C.: CLIPstyler: Image Style Transfer with a Single Text Condition. In: 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 18041–18050 (Jun 2022). https://doi.org/10.1109/CVPR52688.2022.01753
23. Li, T.M., Lukáč, M., Gharbi, M., Ragan-Kelley, J.: Differentiable vector graphics rasterization for editing and learning. ACM Trans. Graph. **39**(6), 193:1–193:15 (Dec 2020). https://doi.org/10.1145/3414685.3417871
24. Liao, Q., Xia, G., Wang, Z.: Calliffusion: Chinese Calligraphy Generation and Style Transfer with Diffusion Modeling (May 2023). https://doi.org/10.48550/arXiv.2305.19124
25. Liu, X., Zhou, C., Zhao, N., Huang, S.: Bézier splatting for fast and differentiable vector graphics rendering. In: Belgrave, D., Zhang, C., Lin, H., Pascanu, R., Koniusz, P., Ghassemi, M., Chen, N. (eds.) Advances in Neural Information Processing Systems. vol. 38, pp. 51528–51559. Curran Associates, Inc. (2025)
26. Loop, C., Blinn, J.: Resolution independent curve rendering using programmable graphics hardware. ACM Trans. Graph. **24**(3), 1000–1009 (Jul 2005). https://doi.org/10.1145/1073204.1073303
27. Ma, X., Pan, Z., Zhang, F.: The automatic generation of chinese outline font based on stroke extraction. Journal of Computer Science and Technology **10**(1), 42–52 (1995). https://doi.org/10.1007/BF02939521
28. Ma, X., Zhou, Y., Xu, X., Sun, B., Filev, V., Orlov, N., Fu, Y., Shi, H.: Towards Layer-wise Image Vectorization. In: 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 16293–16302. IEEE, New Orleans, LA, USA (Jun 2022). https://doi.org/10.1109/CVPR52688.2022.01583
29. Ma, Z., Jiang, J., Chen, Y., Zhang, L.: BézierGS: Dynamic Urban Scene Reconstruction with Bézier Curve Gaussian Splatting. In: 2025 IEEE/CVF International Conference on Computer Vision (ICCV). pp. 25519–25528 (Oct 2025). https://doi.org/10.1109/ICCV51701.2025.02367
30. Magne, T., Sorkine-Hornung, O.: Single-Line Drawing Vectorization. Comput. Graph. Forum **44**(7), e70228 (2025). https://doi.org/10.1111/cgf.70228
31. Ministry of Education of the People's Republic of China: Table of General Standard Chinese Characters (2013)
32. Piegl, L., Tiller, W.: The NURBS Book (2nd Ed.). Springer-Verlag, Berlin, Heidelberg (1997)

33. Pottmann, H.: Smooth curves under tension. Computer-Aided Design **22**(4), 241–245 (May 1990). https://doi.org/10.1016/0010-4485(90)90053-F
34. Thamizharasan, V., Liu, D., Agarwal, S., Fisher, M., Gharbi, M., Wang, O., Jacobson, A., Kalogerakis, E.: VecFusion: Vector Font Generation with Diffusion. In: 2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 7943–7952 (Jun 2024). https://doi.org/10.1109/CVPR52733.2024.00759
35. Tian, Y., Liu, M., Jiang, H., Tu, Y., Su, D.: SketchRefiner: Text-Guided Sketch Refinement Through Latent Diffusion Models. IEEE Transactions on Visualization and Computer Graphics **31**(12), 10711–10722 (Dec 2025). https://doi.org/10.1109/TVCG.2025.3613388
36. Tojo, K., Shamir, A., Bickel, B., Umetani, N.: Fabricable 3D Wire Art. In: ACM SIGGRAPH 2024 Conference Papers. pp. 1–11. SIGGRAPH '24, Association for Computing Machinery, New York, NY, USA (Jul 2024). https://doi.org/10.1145/3641519.3657453
37. Vinker, Y., Pajouheshgar, E., Bo, J.Y., Bachmann, R.C., Bermano, A.H., Cohen-Or, D., Zamir, A., Shamir, A.: CLIPasso: Semantically-aware object sketching. ACM Trans. Graph. **41**(4), 86:1–86:11 (Jul 2022). https://doi.org/10.1145/3528223.3530068
38. Wang, Y., Lian, Z.: DeepVecFont: Synthesizing high-quality vector fonts via dual-modality learning. ACM Trans. Graph. **40**(6), 265:1–265:15 (Dec 2021). https://doi.org/10.1145/3478513.3480488
39. Wang, Y., Wang, Y., Yu, L., Zhu, Y., Lian, Z.: DeepVecFont-v2: Exploiting Transformers to Synthesize Vector Fonts with Higher Quality. In: 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 18320–18328 (Jun 2023). https://doi.org/10.1109/CVPR52729.2023.01757
40. Wang, Z., Huang, J., Sun, Z., Gong, Y., Cohen-Or, D., Lu, M.: Layered Image Vectorization via Semantic Simplification. In: 2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 7728–7738 (Jun 2025). https://doi.org/10.1109/CVPR52734.2025.00724
41. Wang, Z., Lu, M.: Image-Space Collage and Packing with Differentiable Rendering. In: Proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers. Association for Computing Machinery, New York, NY, USA (2025). https://doi.org/10.1145/3721238.3730690
42. Worchel, M., Alexa, M.: Differentiable Rendering of Parametric Geometry. ACM Trans. Graph. **42**(6), 232:1–232:18 (Dec 2023). https://doi.org/10.1145/3618387
43. Wu, S.J., Yang, C.Y., Hsu, J.Y.j.: CalliGAN: Style and Structure-aware Chinese Calligraphy Character Generator (May 2020). https://doi.org/10.48550/arXiv.2005.12500
44. Xing, X., Wang, C., Zhou, H., Zhang, J., Yu, Q., Xu, D.: DiffSketcher: Text Guided Vector Sketch Synthesis through Latent Diffusion Models. In: Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., Levine, S. (eds.) Advances in Neural Information Processing Systems. vol. 36, pp. 15869–15889. Curran Associates, Inc. (2023)
45. Xing, X., Zhou, H., Wang, C., Zhang, J., Xu, D., Yu, Q.: SVGDreamer: Text Guided SVG Generation with Diffusion Model. In: 2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 4546–4555 (Jun 2024). https://doi.org/10.1109/CVPR52733.2024.00435

46. Xu, X., Lambourne, J., Jayaraman, P., Wang, Z., Willis, K., Furukawa, Y.: BrepGen: A B-rep Generative Diffusion Model with Structured Latent Geometry. ACM Trans. Graph. **43**(4), 119:1–119:14 (Jul 2024). https://doi.org/10.1145/3658129
47. Ye, H., Zhang, J., Liu, S., Han, X., Yang, W.: IP-Adapter: Text Compatible Image Prompt Adapter for Text-to-Image Diffusion Models (Aug 2023). https://doi.org/10.48550/arXiv.2308.06721
48. Yoon, J., Han, S., Oh, J., Lee, M.: SplineGS: Learning Smooth Trajectories in Gaussian Splatting for Dynamic Scene Reconstruction. In: The Thirteenth International Conference on Learning Representations (2025), https://openreview.net/forum?id=tMG6btjBfd, accessed: 2026-06-26
49. Zeng, J., Chen, Q., Liu, Y., Wang, M., Yao, Y.: StrokeGAN: Reducing Mode Collapse in Chinese Font Generation via Stroke Encoding. Proceedings of the AAAI Conference on Artificial Intelligence **35**(4), 3270–3277 (May 2021). https://doi.org/10.1609/aaai.v35i4.16438
50. Zhang, L., Rao, A., Agrawala, M.: Adding Conditional Control to Text-to-Image Diffusion Models. In: 2023 IEEE/CVF International Conference on Computer Vision (ICCV). pp. 3813–3824 (Oct 2023). https://doi.org/10.1109/ICCV51070.2023.00355
51. Zhang, R., Isola, P., Efros, A.A., Shechtman, E., Wang, O.: The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 586–595 (Jun 2018). https://doi.org/10.1109/CVPR.2018.00068
52. Zhang, X., Liu, Z., Liu, J., Li, X., Qi, M.: SketchDancing: A Text-Driven Framework for Vector Sketch Animation Generation. In: Proceedings of the 3rd International Workshop on Multimodal and Responsible Affective Computing. pp. 128–136. MRAC '25, Association for Computing Machinery, New York, NY, USA (Oct 2025). https://doi.org/10.1145/3746270.3760235
53. Zhang, X., Ge, X., Xu, T., He, D., Wang, Y., Qin, H., Lu, G., Geng, J., Zhang, J.: GaussianImage: 1000 FPS Image Representation and Compression by 2D Gaussian Splatting. In: Leonardis, A., Ricci, E., Roth, S., Russakovsky, O., Sattler, T., Varol, G. (eds.) Computer Vision – ECCV 2024. pp. 327–345. Springer Nature Switzerland, Cham (2025). https://doi.org/10.1007/978-3-031-72673-6_18
54. Zou, Q., Zhu, L., Wu, J., Yang, Z.: SplineGen: Approximating unorganized points through generative AI. Computer-Aided Design **178**, 103809 (Jan 2025). https://doi.org/10.1016/j.cad.2024.103809