

Bézier Splatting for Fast and Differentiable Vector Graphics Rendering

Xi Liu¹, Chaoyi Zhou¹, Nanxuan Zhao², Siyu Huang¹

¹Clemson University, ²Adobe Research



Figure 1: This work proposes *Bézier Splatting*, new differentiable vector graphics (VGs) representation that achieve an order-of-magnitude computational speedup (tested on a NVIDIA RTX 4090 GPU) in comparison with the state-of-the-art differentiable VGs rendering method DiffVG [14].

Abstract

Differentiable vector graphics (VGs) are widely used in image vectorization and vector synthesis, while existing representations are costly to optimize and struggle to achieve high-quality rendering results for high-resolution images. This work introduces a new differentiable VG representation, dubbed *Bézier Splatting*, that enables fast yet high-fidelity VG rasterization. *Bézier Splatting* samples 2D Gaussians along Bézier curves, which naturally provide positional gradients at object boundaries. Thanks to the efficient splatting-based differentiable rasterizer, *Bézier Splatting* achieves 30 \times and 150 \times faster per forward and backward rasterization step for open curves compared to DiffVG. Additionally, we introduce an adaptive pruning and densification strategy that dynamically adjusts the spatial distribution of curves to escape local minima, further improving VG quality. Furthermore, our new VG representation supports conversion to standard XML-based SVG format, enhancing interoperability with existing VG tools and pipelines. Experimental results show that *Bézier Splatting* significantly outperforms existing methods with better visual fidelity and significant optimization speedup. The project page is bezier-splatting.github.io.

1 Introduction

Vector graphics (VGs) represent images through parametric primitives such as points, curves, and shapes. Unlike raster images, they enable structured representations, lossless resizing, compact storage, and precise content editing, making them crucial for various applications such as user interfaces and animation.

Recently, differentiable VG rasterization gains significant attention, as it allows raster-based algorithms to edit or synthesize VGs through gradient-based optimization. DiffVG [14] is the first differentiable VG framework that leverages the anti-aliasing algorithm to differentiate the vector curves that are inherently discontinuous in pixel space. However, it suffers from slow training and low-fidelity rendering, particularly for high-resolution images. LIVE [17] further introduces a layer-wise coarse-to-fine strategy to improve quality and topology. However, it is highly computationally expensive, requiring 5 hours to vectorize a 2K-resolution image. Additionally, learning-based

methods such as Im2Vec [20] train neural networks to map pixels to VGs, but they are restricted to simple graphics and struggle with out-of-domain generalization. Towards scalable applications of VGs, these challenges highlight the need for a differentiable VG method with better efficiency, fidelity, and generalization capability.

This work presents *Bézier Splatting*, a new differentiable VG representation that optimizes Bézier curves through Gaussian splatting-based rasterization. We sample 2D Gaussian points along Bézier curves and their interior regions, then leverage the Gaussian Splatting framework [11] for efficient curve rasterization. Unlike DiffVG [14] which requires computationally intensive boundary sampling and gradient computation, 2D Gaussians inherently provide direct positional gradients at object boundaries through its differentiable Gaussian formulation, enabling over $150\times$ faster backward computation over DiffVG for open curves. We further introduce an adaptive pruning and densification strategy that adaptively removes redundant curves while adding new ones to necessary regions during optimization. It helps the optimization process escape the local minima of current spatial distributions of curves, formulating a “global receptive field” for further improving the VG optimization process. As shown in Fig. 1, Bézier Splatting outperforms the state-of-the-art differentiable VG rasterizer DiffVG [14] for both open and closed curves in terms of efficiency and rendering quality. We summarize the contribution of this work as follows.

- We propose a novel differentiable vector graphic representation, Bézier Splatting, which achieves $30\times$ faster forward and $150\times$ faster backward computation, while producing high-quality rendering results.
- We present an adaptive pruning and densification strategy to improve the optimization process of Bézier curves by escaping the local minima of the spatial distributions of curves.
- Extensive experiments demonstrate that Bézier Splatting outperforms existing differentiable VG rendering methods in efficiency and visual quality.

2 Related works

2.1 Vectorization and Rasterization

Image vectorization and vector graphic (VG) rasterization are important research topics in computer graphics and computer vision. DiffVG [14] introduces the first differentiable VG rasterization framework, laying a foundation for optimizing and generating vectorized representations through gradient-based methods. It expands the applicability of VGs to a broader range of tasks including image vectorization, text-to-VG generation, and painterly rendering. Based on DiffVG [14], LIVE [17] and O&R [6] further proposes a layer-wise path initialization strategy, vectorizing raster images into compact and semantically consistent VG representations while preserving image topology. Du *et al.* [3] use linear gradients to fill the regional colors, and Chen *et al.* [2] propose a specific implicit neural representation to model regional color distributions, enhancing the color representation within closed Bézier curves. However, these DiffVG-based VG rasterization approaches [14, 17, 3, 2] suffer from slow optimization, often requiring several hours to process a 2K-resolution image with 1,024 curves.

With the advancement of deep learning, another line of approaches directly learns deep neural networks for vector synthesis. Lopes *et al.* [15] combine image-based encoder and VG decoder to generate fonts. Img2Vec [20] integrates the encoder with recurrent neural networks (RNNs) by leveraging sequential modeling for structured vector synthesis. SVGFormer [1] further adopts a Transformer-based architecture [24] to improve the capacity for representing complex geometric structures. More recently, diffusion models [7] have been applied to text-to-VG synthesis [32, 10, 29, 28]. However, existing learning-based VG synthesis approaches are limited to generating simple graphics, and struggle with out-of-domain data. In this work, we propose a novel VG representation dubbed Bézier Splatting that achieves high-fidelity VG rendering within minutes of optimization for high-resolution images.

2.2 Gaussian Splatting

3D Gaussian Splatting [11, 37] emerges as a promising approach for novel view synthesis (NVS) and attracts significant attention from the community. Its explicit 3D Gaussian-based volumetric representation enables high-fidelity 3D reconstruction, while the differentiable tile-based rasterization pipeline ensures real-time and high-quality rendering. It has been applied to various domains and

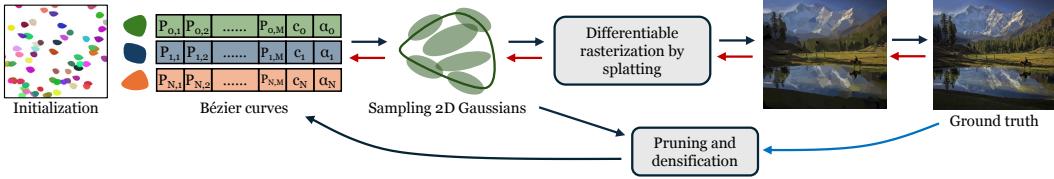


Figure 2: An illustration of the algorithm flow of Bézier Splatting. It begins by randomly initializing Bézier curves and uniformly sampling Gaussians points along them. These Gaussians are then rasterized into the image, enabling gradient-based computation to optimize parameters of both Bézier curves and Gaussians. Curves with negligible opacity or extremely small shapes are removed, while new curves are adaptively added into areas with high reconstruction error, ensuring curves are placed in areas requiring finer details. → forward, ← backpropagation, ↙ error map.

tasks, such as 4D modeling [16, 27] and 3D scene generation [30, 22]. Several works further improve the Gaussian primitives for better representation quality. SuGaR [4] proposes to approximate 3D Gaussians with 2D Gaussians for enhanced surface reconstruction. 2D Gaussian Splatting [8] directly adopts 2D Gaussians for 3D reconstruction for simplified optimization and improved geometric fidelity. TetSphere splatting[5] further employs tetrahedral meshes as the geometric primitives to achieve high-quality geometry. Particularly for 2D image representation, GaussianImage [34] adopts 2D Gaussians [8] for efficient image representation, achieving a compact and expressive alternative to rasters or implicit representations [21, 18]. Image-GS [35] further enhances it through a content-adaptive compression approach. This work proposes to integrate Gaussian splatting with vector representations for differentiable VG rasterization. By sampling 2D Gaussians on Bézier curves and rasterizing through the efficient Gaussian splatting pipeline, the proposed Bézier Splatting representation enables fast yet high-quality VG rendering, even for high-resolution images with complex structures.

3 Method

3.1 Overall

Given a raster image, our goal is to efficiently vectorize it into a VG representation that closely resembles the input while preserving the details. Existing methods, including DiffVG [14] and its following work [17, 2], incur substantial computational costs due to the pixel color accumulation computation. Specifically, DiffVG first constructs a bounding volume hierarchy (BVH) tree to determine the curves that intersect with each individual pixel, then solves equations to precisely determine whether a pixel lies within a region and to compute the inward or outward gradients at boundary points.

To overcome this inefficiency, this work proposes a novel VG representation, Bézier Splatting, which is inspired by the high computational efficiency and expressive fitting capacity of Gaussian splatting [34] for rasterization. Bézier Splatting *samples 2D Gaussian points along Bézier curves and their interior regions*, then *leverages the Gaussian splatting method for efficient rasterization*, enjoying the following advantages:

- This design significantly accelerates the forward and backward pass of Bézier curve rasterization, achieving an order-of-magnitude speedup without specialized optimization techniques;
- The 2D Gaussian representation inherently provides direct position gradients for object boundaries, eliminating the need for additional computations such as boundary sampling and gradient derivation via the Reynolds transport theorem in DiffVG [14];
- It supports richer texture representation by allowing properties such as spatially varying opacity and width, facilitating complex effects such as linear-gradient color transitions within a single curve.

To further improve the fidelity and expressiveness of Bézier Splatting, we introduce a pruning and densification approach (Sec. 3.5), which dynamically removes redundant Bézier curves while adaptively adding necessary curves in regions that have high reconstruction error. Fig. 2 illustrates the algorithm flow of Bézier Splatting. More details are discussed in the following sections.

3.2 Primitives of Bézier Splatting

Bézier curves. We adopt Bézier curves as the parametric primitives of VGs. The representation includes N Bézier curves with a degree of M , as:

$$\mathcal{B}_i(t) = \sum_{j=0}^M B_j^M(t) P_j^{(i)}, \quad t \in [0, 1], \quad i \in \{1, \dots, N\}, \quad (1)$$

where t is a normalized position on the curve, $P_j^{(i)}$ represents the j -th control point of the i -th Bézier curve, and $B_j^M(t)$ is the Bernstein polynomial of degree M , given by:

$$B_j^M(t) = \binom{M}{j} (1-t)^{M-j} t^j. \quad (2)$$

Each Bézier curve $\mathcal{B}_i(t)$ is associated with an RGB color parameter $c_i \in \mathbb{R}^3$, and an opacity parameter $o_i \in [0, 1]$ that defines the transparency of the curve.

To formulate an open curve, we follow DiffVG [14] to adopt three sequentially connected Bézier curves with two control points on each. An open curve requires an additional width parameter to define the stroke thickness. For a closed curve, we adopt two connected Bézier curves, enabling a more efficient sampling in enclosed regions. For closed curves, color filling is applied. The connected Bézier curves in either an open or closed curve share the color parameters, but they have separate opacity parameters to better model the opacity changes along the curves or within closed areas for enriching the texture representation capacity.

2D Gaussians on curves. Our Bézier Splatting novelly associates 2D Gaussians with each Bézier curve. The standard formulation of 2DGS [8] parameterizes each 2D Gaussian by position, color, rotation, scale, opacity, and depth. However, to ensure a compact and differentiable representation, these parameters of Gaussians in our Bézier Splatting are inherited from the corresponding control points. We discuss more details of the sampling of Gaussians, rasterization process, and backward computation in the following sections.

3.3 Sampling Gaussians on Bézier Curves

This work introduces a fast differentiable VG rasterizer based on Gaussian splatting, allowing gradients from raster images to be backpropagated to the 2D Gaussians, then further backpropagated to the Bézier curves through a differentiable sampling strategy, resulting in a highly efficient optimization of the Bézier curves.

Specifically, for each Bézier curve $\mathcal{B}_i(t)$, we uniformly sample K points along it based on Eq. 1. The sampled point set \mathbf{b}_i is:

$$\mathbf{b}_i = [\mathcal{B}_i(t_0), \mathcal{B}_i(t_1), \dots, \mathcal{B}_i(t_{K-1})], \quad (3)$$

where t_k is uniformly sampled from $[0, 1]$.

Sampling 2D Gaussians on open curves. We represent an open curve by using 3 sequential Bézier curves $\mathcal{B}_i(t)$ with a degree of 4 to form a single continuous stroke, by following the same setting as DiffVG [14]. The stroke consists of 10 control points, as the end point of each Bézier curve serves as the start point of the next. To ensure that the final rendering result generates a continuous stroke with consistent width and color, we calculate the x-direction scale of a Gaussian point by the distance between neighboring points by following Eq. 7, while the y-direction scale is a learnable parameter that represents the stroke width.

For both closed and open curves, the depth d of a Gaussian is assigned based on the area of curves, ensuring smaller curves are not occluded by larger ones. It prevents them from being ignored during optimization, as the gradient would not count Gaussians when the accumulated opacity exceeds 1.

Sampling 2D Gaussians on closed curves. Achieving accurate color filling is non-trivial for closed curves. A straightforward approach involves uniformly sampling a large number of points, identifying intersecting curves, solving equations to determine which curves cover them, and then computing scaling factors based on the sampled points. However, this process is computationally expensive, making it inefficient for curve rasterization and optimization.

To address this limitation, we propose a new structure named *paired Bézier curve structure*, which supports two groups of Bézier curves with equal cardinality and arbitrary degrees, forming a closed

region. This structure enables efficient and flexible sampling within the enclosed region between the two curve groups. Specifically, the two Bézier curves $\mathcal{B}_1(t)$ and $\mathcal{B}_{R+1}(t)$ share the same start and end points, forming the inside curves and boundaries of a closed region. A total of R intermediate Bézier curves are generated by linearly interpolating the corresponding control points $P_j^{(0)}$ and $P_j^{(R+1)}$ as:

$$P_j^{(k)} = (1 - t_k)P_j^{(0)} + t_k P_j^{(R+1)}, \quad k = 1, \dots, R, \quad (4)$$

where $t_k \in [0, 1]$ are sampled from a normalized cumulative distribution function (CDF). The interpolated control points $\{P_j^{(k)}\}$ define intermediate curves that form a dense strip between the two boundaries. Note that this paired structure can naturally extend to cases where each curve is composed of multiple connected Bézier segments, provided that all segments have the same number of control points, similar to the representation used in DiffVG [14].

where $t_k \in [0, 1]$ are sampled from a normalized cumulative distribution function (CDF). $P_j^{(k)}$ is interpolated new control points. This non-uniform sampling ensures that points near the boundary curves have small scales, mitigating the influence of interior Gaussians to the exterior of closed area. The interpolated Bézier curves are:

$$\mathcal{B}_k^{\text{interp}}(t) = \sum_{j=0}^M B_j^M(t) P_j^{(k)}, \quad k = 1, \dots, R. \quad (5)$$

Then, 2D Gaussians are sampled on these interpolated curves, by following the same procedure as open curves (Eq. 3), resulting in a structured and efficient point sampling within the enclosed region. Furthermore, to mitigate artifacts caused by non-convex curve shapes during interpolation, we incorporate the Xing loss from LIVE [17], which enforces convexity constraints on curve shapes, to improve the stability of the interpolation process. The full set of sampled points on the i -th Bézier curve is:

$$\mathbf{X} = [\mathbf{b}_0, \mathbf{b}_1, \dots, \mathbf{b}_{R+1}] \in \mathbb{R}^{(R+2) \times K \times 2}. \quad (6)$$

Let $\mathbf{X}_{r,k}$ denote the 2D position of the k -th sampled point on the r -th interpolated curve. To construct anisotropic 2D Gaussian primitives aligned with the local geometry, we define the spatial scales as follows. The *x-direction* of each 2D Gaussian is defined to align with the local tangent direction of the curve (i.e., along the curve), while the *y-direction* is perpendicular to it (i.e., across adjacent curves).

The scale $\sigma_x(r, k)$ is computed from the Euclidean distance between consecutive points along the curve, and $\sigma_y(r, k)$ is computed from the distance between corresponding points on adjacent curves:

$$\begin{aligned} \sigma_x(r, k) &= |\mathbf{X}_{r,k+1} - \mathbf{X}_{r,k}|_2 / \rho, \\ \sigma_y(r, k) &= |\mathbf{X}_{r+1,k} - \mathbf{X}_{r,k}|_2 / \rho. \end{aligned} \quad (7)$$

Here, ρ is a global constant that controls the overall density and overlap of the Gaussians. The rotation $\theta_{r,k}$ of each Gaussian is defined by the angle of the local tangent vector, estimated from neighboring points as

$$\theta_{r,k} = \text{atan2}(y_{r,k+1} - y_{r,k-1}, x_{r,k+1} - x_{r,k-1}). \quad (8)$$

For boundary points, the rotation is set to align with the nearest available neighbor.

3.4 Differentiable Rasterization by Splatting

Once all Gaussians on Bézier curves are sampled, the rasterization process follows the Gaussian splatting pipeline [11], which is very fast in both forward and backward computation. Different from GaussianImage [34], we use α -blending for pixel rendering instead of the accumulation-based blending. Accumulation-based blending calculates the pixel value based on all overlapping Gaussians, such that it conflicts with the rendering principles of VGs, where occlusion plays a crucial role in defining vector structures. α -blending ensures proper occlusion handling, as it allows foreground elements to contribute more to the final pixel value. The rendering of a pixel is:

$$C_n = \sum_{i \in M} \mathbf{c}_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j). \quad (9)$$

C_n is the n -th pixel color, \mathbf{c}_i is the color of corresponding Gaussians, α_i is computed by the projected 2D covariance Σ_i :

$$\alpha_i = o_i \exp^{-\sigma_i}, \quad \sigma_i = \frac{1}{2} \mathbf{d}_n^T \Sigma_i^{-1} \mathbf{d}_n. \quad (10)$$

where d_n is the distance between the pixel and Gaussian center, and Σ_i can be modeled by θ_i , σ_x^i and σ_y^i as:

$$\Sigma_i = (\mathbf{R}_i \mathbf{S}_i)(\mathbf{R}_i \mathbf{S}_i)^T. \quad (11)$$

$$\mathbf{R}_i = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i) \end{bmatrix}, \quad \mathbf{S}_i = \begin{bmatrix} \sigma_x^i & 0 \\ 0 & \sigma_y^i \end{bmatrix}. \quad (12)$$

Discussion on efficiency. The rasterization process of Bézier Splatting is very efficient, since we directly sample 2D Gaussians from Bézier curves in a differentiable manner, then splat them to the 2D plane. The rasterization pipeline remains end-to-end differentiable while being highly optimized for parallel computation and large-scale matrix operations. As a result, Bézier Splatting is a highly efficient and fully differentiable VG representation, which ensures both fast rendering and gradient-based optimization. It not only preserves the flexibility of VGs but also allows seamless integration into deep learning frameworks, making it well-suited for tasks requiring high-quality and editable vector representations.

3.5 Optimization

Training objective. Given a raster image $\mathcal{I} \in \mathbb{R}^{H \times W \times 3}$, the goal of image vectorization is to vectorize the image into Bézier curves while ensuring a high-fidelity reconstruction. We first randomly generate a set of Bézier curves to lay on the canvas, then employ the differentiable rasterizer discussed in Sec. 3.4 to render a raster image $\hat{\mathcal{I}} \in \mathbb{R}^{H \times W \times 3}$. The Bézier curves can then be optimized through any gradient-based loss functions. In this work, we formulate the optimization objective as minimizing the loss between \mathcal{I} and $\hat{\mathcal{I}}$ while enforcing the curves to be convex. Therefore, we only adopt an L_2 loss and a Xing loss L_{Xing} [17], as:

$$L = \lambda_1 \|\hat{\mathcal{I}} - \mathcal{I}\|_2^2 + \lambda_2 L_{\text{Xing}} \quad (13)$$

where λ_1 and λ_2 are hyperparameters that trade off the two loss functions.

Adaptive curve pruning and densification. The gradients of 2D Gaussians are influenced by local pixels only. Therefore, it is hard for 2D Gaussians to dynamically reallocate to regions requiring finer-grained details. The Gaussians would be trapped in local minima, leading to redundant Gaussians that are optimized as either low opacity or excessively large size, resulting in artifacts in rendering results.

In standard 3D Gaussian Splatting pipeline [11], Gaussians with low opacity or excessive size are pruned, whereas those with high gradient responses are split into two. However, this strategy is not directly applicable to Bézier curves. Note that in volumetric representations, large Gaussians are usually unnecessary for modeling any particular structure. In contrast, VG representations often encompass large uniform regions such as backgrounds or areas with homogeneous colors (e.g., walls). Consequently, the size-based pruning strategy would remove critical structures in VG representations. Similarly, splitting a Bézier curve into two can introduce significant randomness, as high-gradient regions do not always indicate poor reconstruction in VGs. Since VGs assume a uniform color within each enclosed region, complex textures naturally produce high gradients. This does not imply the region should be split, as it may disrupt the semantic consistency of the VG representation.

To address this issue, this work introduces a new pruning and densification strategy to dynamically adjust the density of Bézier curves throughout the optimization process. For pruning redundant Bézier curves, we apply three criteria to ensure a stable and precise optimization process. First, we remove curves with opacity below a dynamic threshold that gradually decreases as optimization progresses, ensuring that weakly contributing curves are eliminated while preserving essential structures. Second, we remove curves with an area below a predefined threshold, as they contribute minimally to the final representation. To remove visually insignificant or noisy Bézier curves, we apply an opacity-based filtering strategy: for per-segment opacities, we discard curves whose middle segment is significantly fainter than both ends—often indicating that a single curve improperly spans a region better represented by multiple curves; otherwise, we discard curves with overall opacity below a threshold (e.g., 0.2) to eliminate globally low-visibility curves. Third, we remove curves that exhibit high color similarity with surrounding curves and have significant overlap, as they provide little

Table 1: Computational speed for rendering a $2,040 \times 1,344$ image with 2,048 curves.

	Open curve			Closed curve		
	DiffVG [14]	Bézier Splatting	Speedup	DiffVG [14]	Bézier Splatting	Speedup
Forward pass	141.3ms	4.5ms	$31.4 \times$	85.2ms	14.1ms	$6.0 \times$
Backward pass	701.3ms	4.7ms	$149.2 \times$	448.3ms	24.58ms	$18.2 \times$

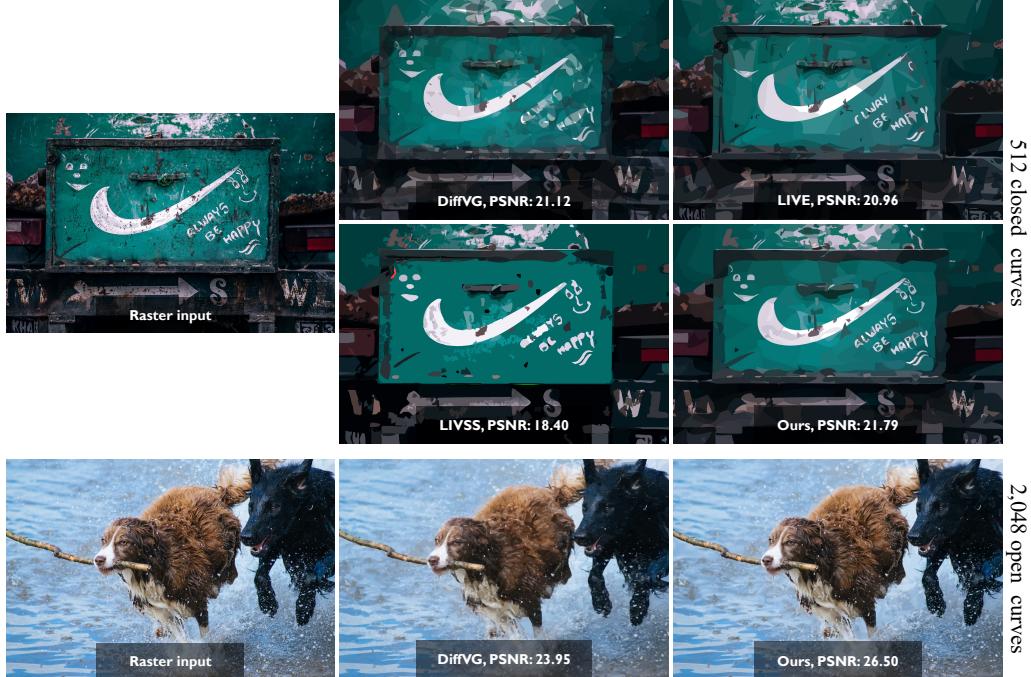


Figure 3: A qualitative comparison of our method and the state-of-the-art differentiable VG rasterization methods, including DiffVG [14], LIVE [17], and LIVSS [26].

additional information and can be pruned without affecting the final representation. By removing such curves, we allow the optimization to introduce more appropriate replacements, leading to a more accurate and visually coherent vectorized representation. For curve densification, we adopt an error-driven curve allocation strategy inspired by LIVE [17]. Specifically, we compute connected error regions, rank them by area, and add new Bézier curves into the highest-error regions. This adaptive redistribution mechanism ensures that curves are allocated to where they would contribute the most to the reconstruction fidelity. The pruning and densification strategy enables a “global receptive field” for redistributing curve density, effectively preserving visual details for high-fidelity rendering while maintaining a compact vector representation.

4 Experiments

4.1 Experimental Setups

Implementation details. We implement Bézier Splatting in PyTorch [19] and optimize it by using the Adam optimizer [12] with a StepLR learning rate scheduler. The learning rate is initialized at 0.01 for color, 2e-4 for Bézier curve control points, and 0.1 for opacity. For the pruning and densification strategy, the opacity threshold is set to 0.02, and the overlap threshold based on Axis-Aligned Bounding Boxes (AABB) is set to 0.9. Following the approach in LIVE [17], new curves are initialized in a circular pattern, and the number of added curves matches the number of removed ones to maintain a constant total curve count. Open curves are optimized for 15,000 iterations, and closed curves are optimized for 10,000 iterations. Pruning and densification are applied every 400 steps until the last but 1,000 steps, after which they are halted to stabilize the representation.

Datasets. We comprehensively evaluate our method across different image domains. We use the publicly available DIV2K [23] dataset for evaluating natural images. Due to the high computational cost of baseline method LIVE [17], we uniformly subsample the DIV2K dataset by selecting one out of every four images, resulting in a final evaluation set of 200 images from the original 800-image

Table 2: Quantitative evaluation and optimization efficiency of differentiable VG methods on the DIV2K dataset [23].

	Method	256 curves				512 curves				1024 curves			
		SSIM \uparrow	PSNR \uparrow	LPIPS \downarrow	Opt. \downarrow	SSIM \uparrow	PSNR \uparrow	LPIPS \downarrow	Opt. \downarrow	SSIM \uparrow	PSNR \uparrow	LPIPS \downarrow	Opt. \downarrow
Open	DiffVG [14]	0.552	19.83	0.563	18.9min	0.587	21.47	0.537	22.0min	0.616	22.62	0.517	30.6min
	Ours	0.600	22.17	0.540	3.4min	0.646	23.79	0.498	3.3min	0.699	25.45	0.448	3.2min
Closed	DiffVG [14]	0.578	20.69	0.548	16.4min	0.601	21.82	0.531	18.5min	0.631	22.95	0.509	25.1min
	LIVE [17]	0.576	20.09	0.543	2.6h	0.611	21.70	0.521	4.2h	0.648	23.11	0.495	5.1h
	LIVSS [26]	0.586	17.71	0.542	39.2min	0.630	18.71	0.530	54.3min	0.678	19.83	0.517	1.4h
	Ours	0.580	20.74	0.546	7.8min	0.607	22.11	0.528	8.3min	0.639	23.45	0.507	8.6min



Figure 4: Our Bézier Splatting achieves high-quality image vectorization results for various types of images including artworks, cartoons, and natural images. Curve type and count are indicated at the bottom right of each sample.

dataset. Additionally, we test our method on non-photorealistic images, including the artwork images from Clipart1K dataset [9] and cartoon images from Danbooregions dataset [31], to demonstrate its effectiveness for diverse types of images, as shown in Fig. 4.

4.2 Method Comparison

Table 1 reports the forward and backward runtime for processing 2,048 curves on an image with a resolution of $2,040 \times 1,344$. Compared to DiffVG [14], our method significantly accelerates VG rasterization by $31.4 \times$ faster per forward step and $149.2 \times$ faster per backward step for open curves. For closed curves, our color filling strategy requires sampling 20 additional Bézier curves per closed curve, while our method remains highly efficient, achieving $6 \times$ faster forward and $18.2 \times$ faster backward computation.

Table 2 quantitatively evaluates the quality of differentiable VG representations by three commonly used metrics, MS-SSIM [25], PSNR, and LPIPS [33]. Our method demonstrates higher optimization efficiency and rendering fidelity for both open and closed curves with different curve numbers.

Fig. 3 visually compares DiffVG [14], LIVE [3], [26], and our Bézier Splatting under 512 closed curves and 2048 open curves. Compared to DiffVG [14], our method captures significantly more fine-grained textures. Compared to LIVE [17], our method effectively enhances rendering quality, resulting in higher-fidelity results. This improvement stems from our method’s ability to globally optimize all curves, rather than LIVE’s layer-by-layer path-adding strategy. Compared to LIVSS [26], our method demonstrates superior ability in preserving fine structural details, especially in text regions. Due to its heavy reliance on layer-wise semantic simplification, LIVSS struggles to optimize regions where semantic information is ambiguous or difficult to extract. Moreover, by optimizing the entire VG simultaneously, our approach ensures a cohesive and natural appearance, avoiding the accumulation of errors and inconsistencies introduced by layer-wise updates.

As shown in Fig. 4, Bézier Splatting consistently renders high-quality VGs across various image domains from photorealistic natural images, watercolor paintings, to cartoon images. Unlike learning-based image vectorization methods [15, 20, 1], which can not easily generalize to out-of-domain data, our method can flexibly handle different types of images.

4.3 Adaptability with Layer-wised Image Vectorization

Existing works such as LIVE [17], SGLIVE [36] and LIVSS [26] have demonstrated that layer-wise vectorization strategies can significantly improve the topology and compositionality of DiffVG-generated vector graphics [14]. Since the proposed Bézier Splatting is a fast and differentiable VGs



Figure 5: As a differentiable VGs renderer, Bézier Splatting can integrate with topology-aware strategies such as layer-wise vectorization [17] to further improve compositionality and detail.

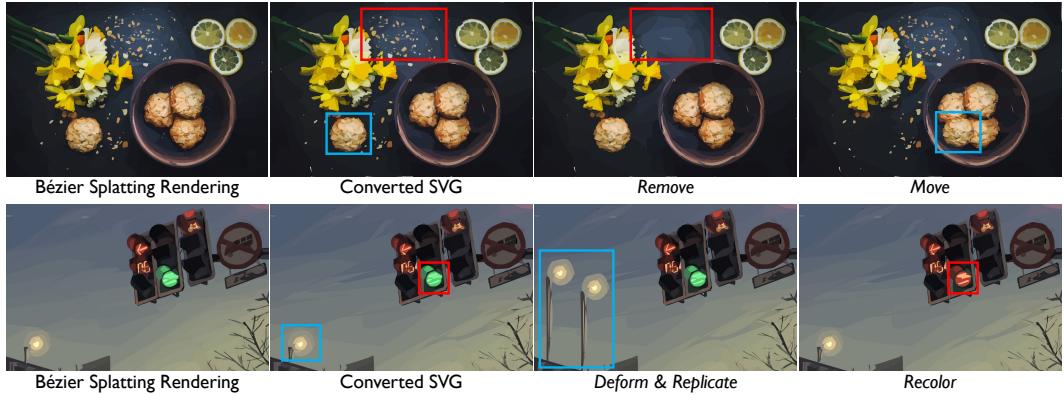


Figure 6: Our Bézier Splatting representation is compatible with the standard SVG XML format and supports flexible vector editing operations.

renderer alternative to DiffVG, is naturally compatible with these topology-aware strategies and can leverage them to further enhance the structural quality of the optimized vector outputs. As shown in Fig. 5, we incorporate the layer-wise vectorization approach from LIVE [17] into the Bézier Splatting optimization process. This integration yields improved compositional integrity and finer details for regions with complex patterns such as the butterfly’s wings.

4.4 Editability and Compatibility with SVG XML Format

Our Bézier Splatting representation is fully compatible with the standard SVG XML format, ensuring the same editability as standard SVGs and seamless integration with existing SVG tools. As shown in Fig. 6, a high-fidelity SVG can be obtained via a simple conversion algorithm (see Appendix for details). Once converted to SVG, the vectorized output is fully editable at the primitive level, allowing users to freely deform, remove, move, replicate, and recolor individual elements.

5 Conclusion

This work has presented Bézier Splatting, a novel differentiable vector graphics (VGs) representation that leverages Gaussian splatting for efficient Bézier curve optimization. Our method achieves $30\times$ faster forward computation and $150\times$ faster backward computation in rasterization compared to existing methods, while also delivering high rendering fidelity. Additionally, our adaptive pruning and densification strategy improves optimization by dynamically adjusting curve placement during optimization. Extensive experiments have demonstrated that Bézier Splatting outperforms existing differentiable VG methods in both training efficiency and visual quality, making it a promising solution for scalable applications of VGs.

Limitation and future work. The closed curves in Bézier Splatting are enforced to have convex shapes using Xing loss [17] to prevent false interpolation results, which may slightly reduce the capacity of VG representations. Additionally, the closed curves require sampling more Gaussian points to represent area boundaries precisely, leading to slower computation compared to open curves. An interesting future direction is to leverage the efficiency and differentiability of our approach for high-quality VG synthesis applications such as text-to-VG generation or text-based VG editing.

References

- [1] Defu Cao, Zhaowen Wang, Jose Echevarria, and Yan Liu. Svgformer: Representation learning for continuous vector graphics using transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10093–10102, 2023. [2](#), [8](#)
- [2] Ye Chen, Bingbing Ni, Jinfan Liu, Xiaoyang Huang, and Xuanhong Chen. Towards high-fidelity artistic image vectorization via texture-encapsulated shape parameterization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15877–15886, 2024. [2](#), [3](#)
- [3] Zheng-Jun Du, Liang-Fu Kang, Jianchao Tan, Yotam Gingold, and Kun Xu. Image vectorization and editing via linear gradient layer decomposition. *ACM Transactions on Graphics (TOG)*, 42(4), 2023. [2](#), [8](#)
- [4] Antoine Guédon and Vincent Lepetit. Sugar: Surface-aligned gaussian splatting for efficient 3d mesh reconstruction and high-quality mesh rendering. *CVPR*, 2024. [3](#)
- [5] Minghao Guo, Bohan Wang, Kaiming He, and Wojciech Matusik. Tetsphere splatting: Representing high-quality geometry with lagrangian volumetric meshes. *arXiv preprint arXiv:2405.20283*, 2024. [3](#)
- [6] Or Hirschorn, Amir Jevnisek, and Shai Avidan. Optimize & reduce: a top-down approach for image vectorization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 2148–2156, 2024. [2](#)
- [7] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020. [2](#)
- [8] Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger, and Shenghua Gao. 2d gaussian splatting for geometrically accurate radiance fields. In *SIGGRAPH 2024 Conference Papers*. Association for Computing Machinery, 2024. [3](#), [4](#)
- [9] Naoto Inoue, Ryosuke Furuta, Toshihiko Yamasaki, and Kiyoharu Aizawa. Cross-domain weakly-supervised object detection through progressive domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5001–5009, 2018. [8](#)
- [10] Ajay Jain, Amber Xie, and Pieter Abbeel. Vectorfusion: Text-to-svg by abstracting pixel-based diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1911–1920, 2023. [2](#)
- [11] 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](#), [5](#), [6](#)
- [12] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015. [7](#)
- [13] Kodak. Kodak lossless true color image suite, 1999. Accessed: 2025-03-07. [15](#), [22](#)
- [14] Tzu-Mao Li, Michal Lukáč, Gharbi Michaël, and Jonathan Ragan-Kelley. Differentiable vector graphics rasterization for editing and learning. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, 39(6):193:1–193:15, 2020. [1](#), [2](#), [3](#), [4](#), [5](#), [7](#), [8](#)
- [15] Raphael Gontijo Lopes, David Ha, Douglas Eck, and Jonathon Shlens. A learned representation for scalable vector graphics. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7930–7939, 2019. [2](#), [8](#)
- [16] Jonathon Luiten, Georgios Kopanas, Bastian Leibe, and Deva Ramanan. Dynamic 3d gaussians: Tracking by persistent dynamic view synthesis. In *3DV*, 2024. [3](#)
- [17] Xu Ma, Yuqian Zhou, Xingqian Xu, Bin Sun, Valerii Filev, Nikita Orlov, Yun Fu, and Humphrey Shi. Towards layer-wise image vectorization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2022. [1](#), [2](#), [3](#), [5](#), [6](#), [7](#), [8](#), [9](#), [13](#), [15](#)

- [18] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM transactions on graphics (TOG)*, 41(4):1–15, 2022. 3
- [19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019. 7
- [20] Pradyumna Reddy, Michael Gharbi, Michal Lukac, and Niloy J Mitra. Im2vec: Synthesizing vector graphics without vector supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7342–7351, 2021. 2, 8
- [21] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in neural information processing systems*, 33:7462–7473, 2020. 3
- [22] Jiaxiang Tang, Jiawei Ren, Hang Zhou, Ziwei Liu, and Gang Zeng. Dreamgaussian: Generative gaussian splatting for efficient 3d content creation. In *The Twelfth International Conference on Learning Representations*, 2024. 3
- [23] Radu Timofte, Eirikur Agustsson, Luc Van Gool, Ming-Hsuan Yang, Lei Zhang, Bee Lim, et al. Ntire 2017 challenge on single image super-resolution: Methods and results. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2017. 7, 8, 13, 15, 16, 17, 18, 19, 20, 21
- [24] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 2
- [25] Zhou Wang, Eero P. Simoncelli, and Alan C. Bovik. Multi-scale structural similarity for image quality assessment. *Conference Record of the Asilomar Conference on Signals, Systems and Computers*, 2:1398–1402, 2003. Conference Record of the Thirty-Seventh Asilomar Conference on Signals, Systems and Computers ; Conference date: 09-11-2003 Through 12-11-2003. 8
- [26] Zhenyu Wang, Jianxi Huang, Zhida Sun, Yuanhao Gong, Daniel Cohen-Or, and Min Lu. Layered image vectorization via semantic simplification. *arXiv preprint arXiv:2406.05404*, 2024. 7, 8
- [27] Guanjun Wu, Taoran Yi, Jiemin Fang, Lingxi Xie, Xiaopeng Zhang, Wei Wei, Wenyu Liu, Qi Tian, and Xinggang Wang. 4d gaussian splatting for real-time dynamic scene rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 20310–20320, 2024. 3
- [28] Ximing Xing, Chuang Wang, Haitao Zhou, Jing Zhang, Qian Yu, and Dong Xu. Diffsketcher: Text guided vector sketch synthesis through latent diffusion models. *Advances in Neural Information Processing Systems*, 36:15869–15889, 2023. 2
- [29] Ximing Xing, Haitao Zhou, Chuang Wang, Jing Zhang, Dong Xu, and Qian Yu. Svdreamer: Text guided svg generation with diffusion model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4546–4555, 2024. 2
- [30] Taoran Yi, Jiemin Fang, Junjie Wang, Guanjun Wu, Lingxi Xie, Xiaopeng Zhang, Wenyu Liu, Qi Tian, and Xinggang Wang. Gaussiandreamer: Fast generation from text to 3d gaussians by bridging 2d and 3d diffusion models. In *CVPR*, 2024. 3
- [31] Lvmin Zhang, Yi JI, and Chunping Liu. Danbooregion: An illustration region dataset. In *European Conference on Computer Vision (ECCV)*, 2020. 8, 15, 23
- [32] Peiying Zhang, Nanxuan Zhao, and Jing Liao. Text-to-vector generation with neural path representation. *ACM Transactions on Graphics (TOG)*, 43(4):1–13, 2024. 2
- [33] 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. 8

- [34] Xinjie Zhang, Xingtong Ge, Tongda Xu, Dailan He, Yan Wang, Hongwei Qin, Guo Lu, Jing Geng, and Jun Zhang. Gaussianimage: 1000 fps image representation and compression by 2d gaussian splatting. In *European Conference on Computer Vision*, 2024. [3](#) [5](#)
- [35] Yunxiang Zhang, Alexandr Kuznetsov, Akshay Jindal, Kenneth Chen, Anton Sochenov, Anton Kaplanyan, and Qi Sun. Image-gs: Content-adaptive image representation via 2d gaussians, 2024. [3](#)
- [36] Hengyu Zhou, Hui Zhang, and Bin Wang. Segmentation-guided layer-wise image vectorization with gradient fills. In *European Conference on Computer Vision*, pages 165–180. Springer, 2024. [8](#)
- [37] Matthias Zwicker, Hanspeter Pfister, Jeroen Van Baar, and Markus Gross. Ewa volume splatting. In *Visualization, 2001. VIS 01. Proceedings*, pages 29–538. IEEE, 2001. [2](#)

A Ablation Study

The effectiveness of adaptive pruning and densification and layer-wise vectorization strategy. We conduct an ablation study to evaluate the effectiveness of our adaptive pruning and densification strategy. Quantitative results are shown in Table 3. To further investigate the flexibility of our framework, we incorporate a layer-wise curve addition strategy inspired by LIVE [17], where curves are progressively added during training. Although this strategy achieves slightly better reconstruction metrics, it doubles the training time compared to our adaptive pruning and densification approach, making it less favorable for time-sensitive applications. Fig. 7 shows our result with layer-wise image vectorization strategy.

Table 3: An ablation study on adaptive pruning and densification and layer-wise training strategy, evaluated on 512 closed curves with DIV2K dataset [23].

Method	$SSIM^{\uparrow}$	$PSNR^{\uparrow}$	$LPIPS^{\downarrow}$	Opt^{\downarrow}
No Strategy	0.590	21.10	0.530	8.3 min
– w/ Prune&Densify	0.607	22.11	0.528	8.3 min
– w/ Layer-wise	0.613	22.21	0.521	16.2 min



Figure 7: Our Bézier Splatting is fully compatible with the layer-wise vectorization strategies [17].

Comparing different numbers of curves. We progressively increase the number of curves for vectorizing a watercolor image (Fig. 8) with numerous small spots. As the number of curves grows, our approach first reconstructs the foreground object, the bird, then gradually refines the smaller spots. This demonstrates that our method prioritizes the primary structure before optimizing finer details, effectively distributing curves to balance global structure and local texture representation, thanks to our adaptive pruning and densification strategy.

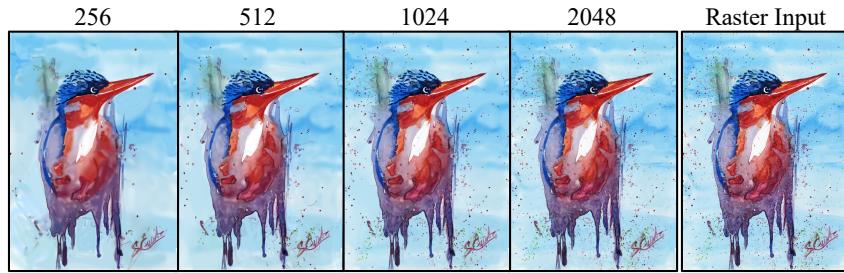


Figure 8: A comparison of different curve numbers by Bézier Splatting.

A systematic evaluation of computation time. To systematically evaluate the computational efficiency of our framework, we report detailed timing statistics across varying configurations of interpolation curve numbers and curve resolutions, as summarized in Table 4. The total forward time includes two major parts: the Gaussian sampling and the Gaussian splatting. Increasing the number of interpolated curves within the closed Bézier curves leads to a near-linear growth in both forward and backward runtimes, demonstrating good computational scalability. When fixing the interpolation

number (*e.g.*, at 40) and varying the number of Bézier curves, the forward time increases moderately, reflecting the additional cost of handling finer spatial detail. The splatting stage dominates the forward time at higher number of curves due to the increased number of sampled points per region, while the Gaussian sampling time is the primary bottleneck at lower number of curves. This is attributed to overheads from sequential memory allocation and data transfer operations, which are not effectively parallelized on the GPU, suggesting potential for further optimization.

Table 4: A systematic evaluation of computation time. We test the forward and backward time (ms) under varying numbers of sampled curves within one closed Bézier curve (Inter. #), and the total number of Bézier curves (Curve #).

Inter. #	Curve #	Forward w/ gradient	Sample Gaussians	Splatting	Backward	Render (FPS)
20	2048	10.66	4.12	6.11	21.64	103.45
40	2048	14.79	5.11	9.32	25.06	68.30
80	2048	27.01	8.22	17.74	33.40	37.90
40	256	7.20	3.81	3.09	16.72	150.30
40	512	7.70	3.88	3.67	17.42	144.30
40	1024	10.39	4.06	5.96	19.96	114.30
40	2048	14.79	5.11	9.32	25.06	68.30

B Convert Bézier Splatting to standard SVG Format

As described in Algorithm 1, we convert our optimized Bézier Splatting representation into a standard SVG file for compatibility with downstream vector editing tools. Each curve is represented by a set of $3k + 1$ control points $\mathcal{C}_i \in \mathbb{R}^{(3k+1) \times 2}$, corresponding to k continuous cubic Bézier segments. These control points are normalized to the range $[-1, 1]$ and are first transformed into pixel coordinates according to the canvas size (W, H) .

For each curve i , we initialize the SVG path string d with a $[\text{M } P_i^0]$ command. We then iteratively construct each cubic segment using three internal control points: $(p_1, p_2, p_3) = (P_i^{3j+1}, P_i^{3j+2}, P_i^{3j+3})$, and append the segment as a path command $[\text{C } p_1, p_2, p_3]$ to d . After all segments are processed, a $[\text{Z}]$ is appended to close the path.

Each path d is then associated with a fill color (r, g, b) , computed via sigmoid activation on the optimized feature vector \mathcal{F}_i , and opacity 1.0. The result is stored in a global path set \mathcal{S} , and all paths are assembled into the final SVG file S . This output is directly compatible with standard vector graphics tools such as Adobe Illustrator.

Algorithm 1 Convert Bézier Splatting to Standard SVG.

Input: Optimized control points $\mathcal{C} \in \mathbb{R}^{N \times (3k+1) \times 2}$, feature colors $\mathcal{F} \in \mathbb{R}^{N \times 3}$, canvas size (W, H)
Output: SVG file S containing cubic Bézier paths
for each curve $i = 1$ to N **do**
 $P_i \leftarrow \frac{\mathcal{C}_{i+1}}{2} \cdot (W, H)$
 $(r, g, b) \leftarrow \text{sigmoid}(\mathcal{F}_i) \times 255$
 Initialize path $d \leftarrow [\text{M } P_i^0]$
 for each segment $j = 1$ to k **do**
 $(p_1, p_2, p_3) \leftarrow (P_i^{3j+1}, P_i^{3j+2}, P_i^{3j+3})$
 $d \oplus [\text{C } p_1, p_2, p_3]$
 end for
 $d \oplus \text{Z}$
 $\mathcal{S} \leftarrow \mathcal{S} \cup \{\text{path}(d, \text{fill} = (r, g, b), \text{opacity} = 1.0)\}$
end for
Return: SVG file assembled from all paths

C Bézier Splatting Supports Flexible Attributes of Curves

All 2D Gaussians in our method are generated via a differentiable sampling algorithm, which makes it straightforward to incorporate user-defined shape attributes, such as linear-gradient fills in color or opacity. To demonstrate the extensibility of our model, we evaluate the multi-opacity scheme for open curves: each Bézier segment within an open curve is assigned an independent opacity value. This design enables users to either maintain consistent sampling point appearance across segments or apply customized interpolation strategies. This strategy significantly improves vectorization quality for open curves, as shown in Table 5. Since closed curves are the primary representation format in SVG, we adopt a uniform setting for opacity and color in those regions to ensure a full compatibility with existing vector graphic tools. Nonetheless, users can still follow the above multi-opacity scheme to implement linear-gradient fills for both opacity and color, enhancing the flexibility and expressiveness of vector graphics.

Table 5: An ablation study on the number of opacity parameters per curve, evaluated on 512 open curves with DIV2K dataset [23].

Method	$SSIM^\uparrow$	$PSNR^\uparrow$	$LPIPS^\downarrow$
3-opacity	0.65	23.79	0.50
1-opacity	0.63	22.97	0.51

D More Comparisons

We present more qualitative comparisons on the DIV2K [23] dataset. As shown in Fig. 9, Fig. 10, Fig. 11, Fig. 12, and Fig. 13, our method shows consistent improvements over the baselines, including better preservation of fine details, higher rendering fidelity, fewer visual artifacts, and more accurate geometric structures. Furthermore, as the number of curves increases, our method demonstrates a significantly improved representational ability, enabling more precise reconstruction of complex shapes, and fine-grained structures.

We further evaluate our method on another natural image dataset, Kodak [13]. Due to the slow vectorization speed of LIVE [17], we compare with DiffVG on open curves only. As shown in Table 6, our method consistently outperforms DiffVG across quantitative metrics including PSNR, SSIM, and LPIPS.

Table 6: A quantitative evaluation on the Kodak dataset [13] with 256 to 1024 curves. We report results on open curves.

Method	256			512			1024			
	$SSIM^\uparrow$	$PSNR^\uparrow$	$LPIPS^\downarrow$	$SSIM^\uparrow$	$PSNR^\uparrow$	$LPIPS^\downarrow$	$SSIM^\uparrow$	$PSNR^\uparrow$	$LPIPS^\downarrow$	
Open	DiffVG	0.601	23.43	0.535	0.645	24.70	0.495	0.699	26.04	0.439
	Ours	0.679	26.18	0.457	0.743	27.90	0.383	0.797	29.24	0.310
Closed	DiffVG	0.622	24.11	0.513	0.666	25.34	0.475	0.719	26.66	0.420
	Ours	0.621	24.19	0.519	0.664	25.61	0.485	0.708	26.91	0.448

E More Image Vectorization Results

Fig. 14, Fig. 15, and Fig. 16 show more results on natural images from DIV2K [23] and Kodak [13], as well as animation images from DanbooruRegion [31], respectively. The results demonstrate that both the global structure and local texture of the images are well reconstructed, highlighting the effectiveness of our approach in capturing fine details and complex shapes.

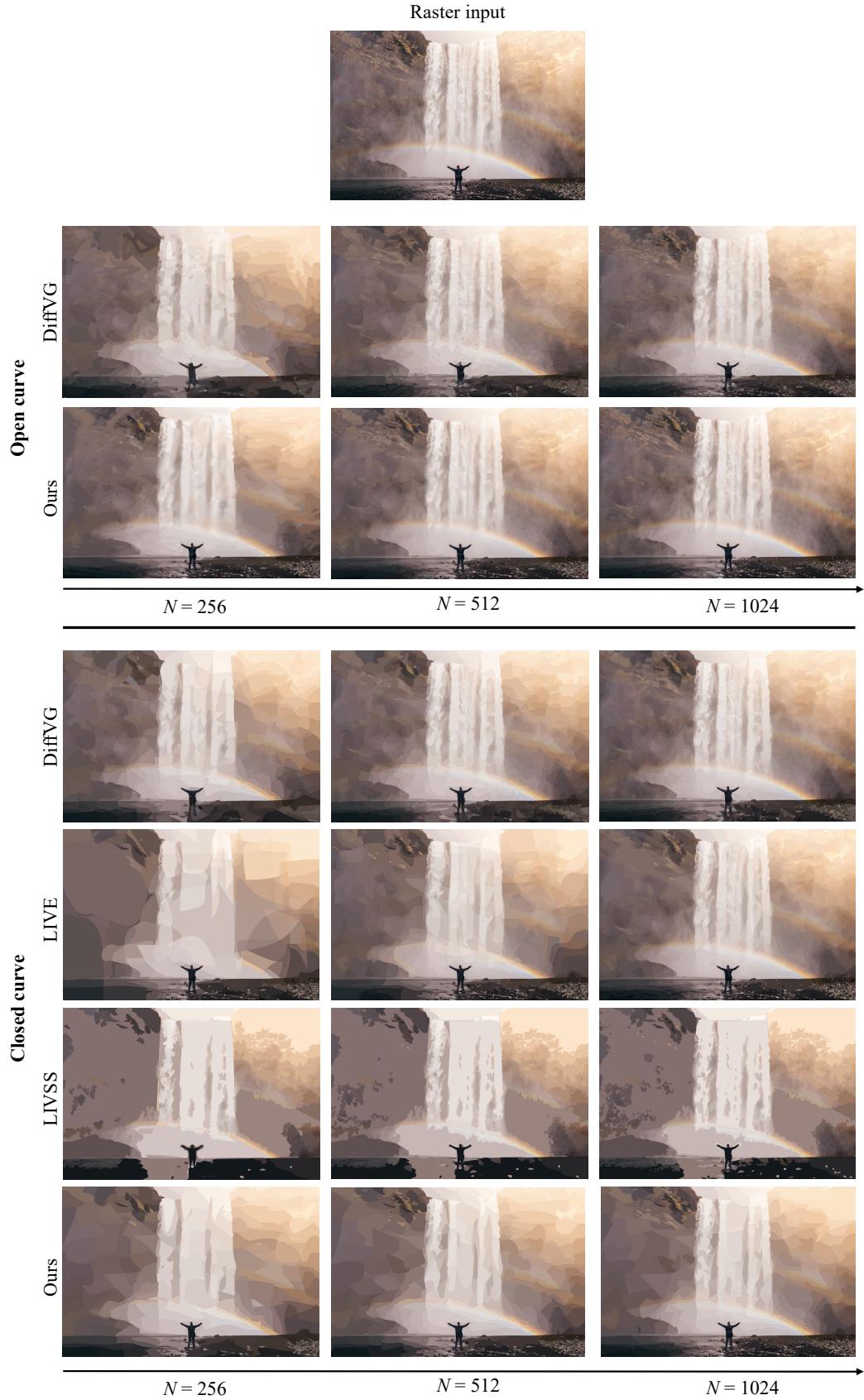


Figure 9: A qualitative comparison of our method and the existing differentiable VG rasterization method on DIV2K dataset [23].

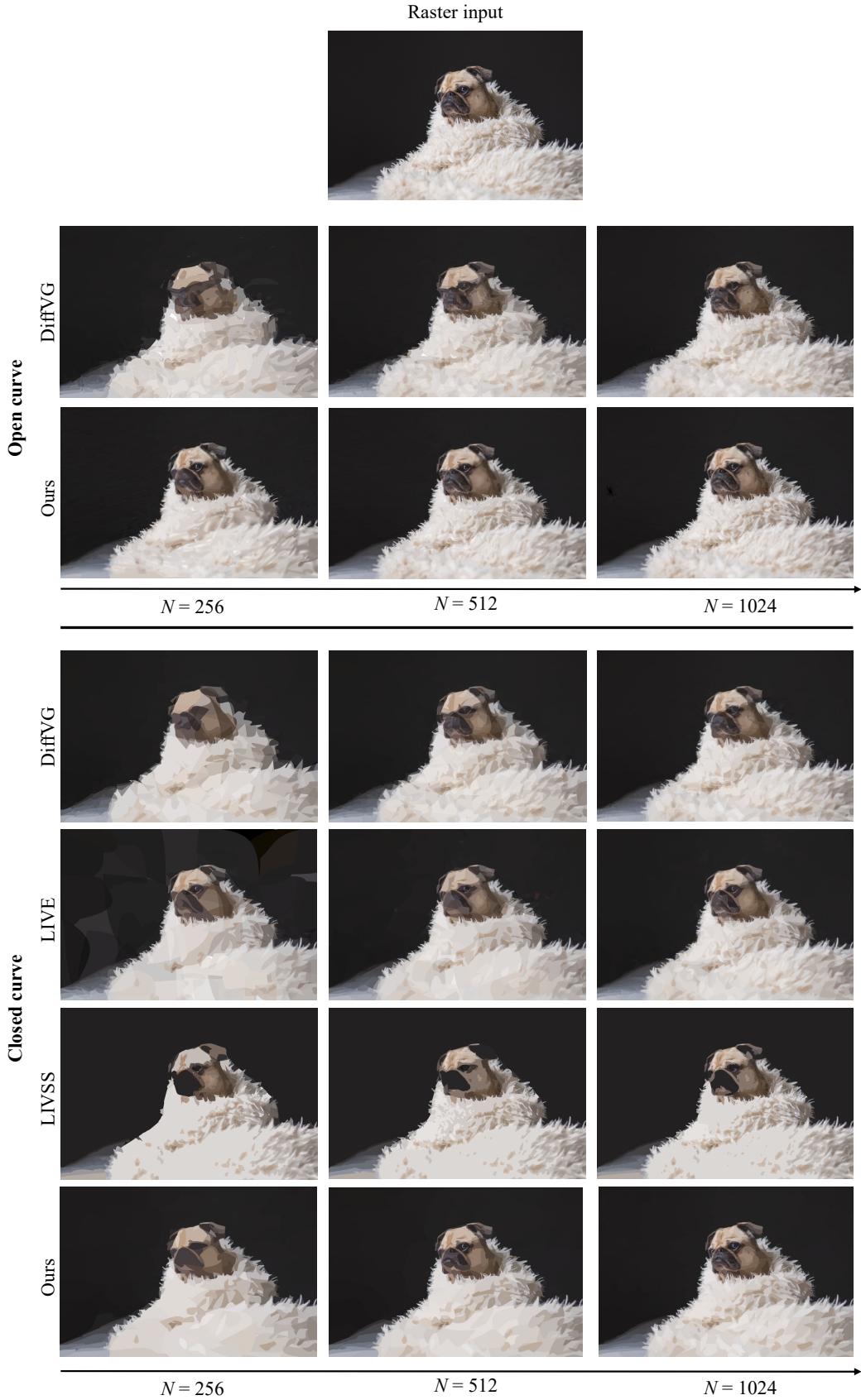


Figure 10: More qualitative comparisons of our method and the existing differentiable VG rasterization method on DIV2K dataset [23].

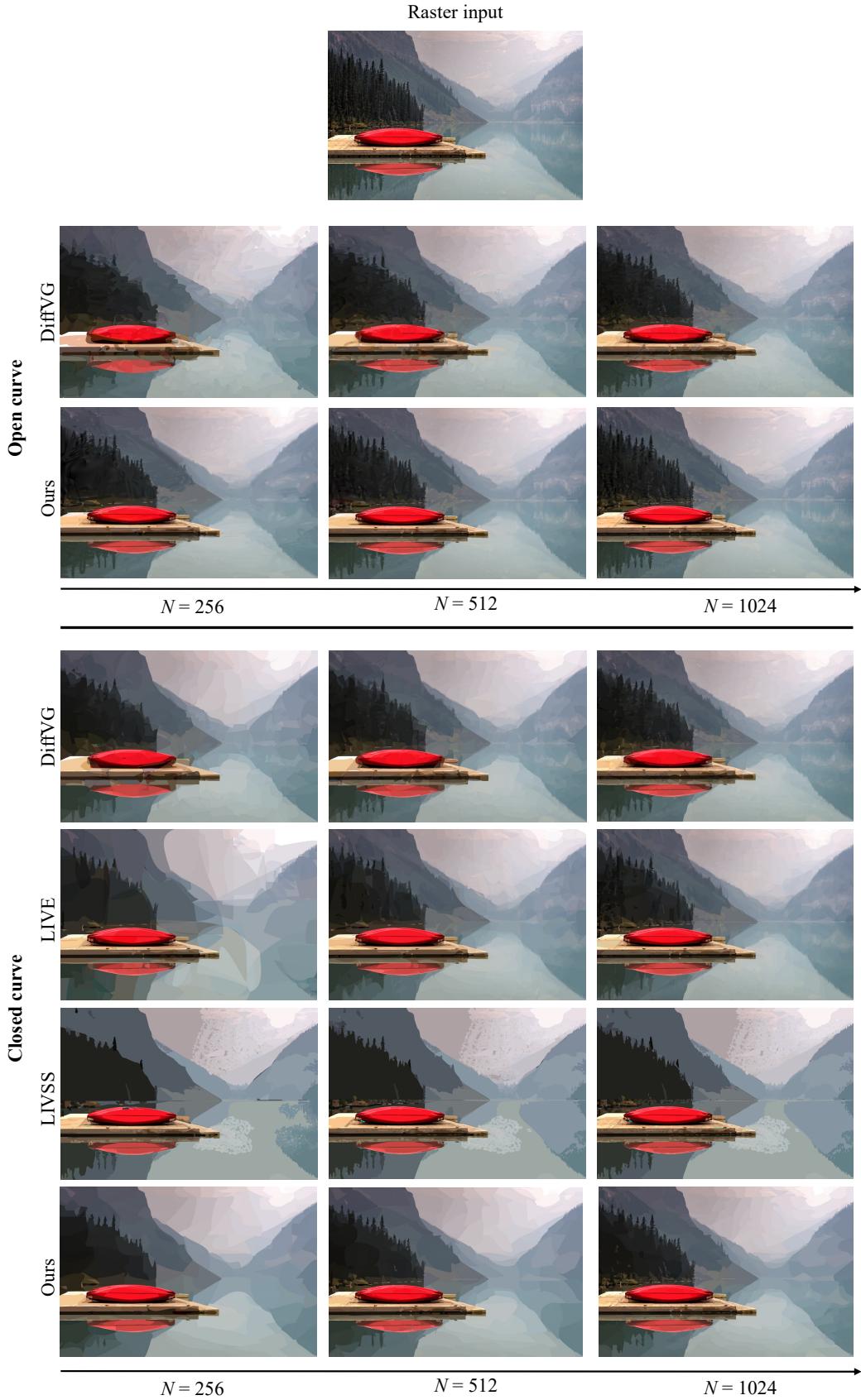


Figure 11: More qualitative comparisons of our method and the existing differentiable VG rasterization method on DIV2K dataset [23].

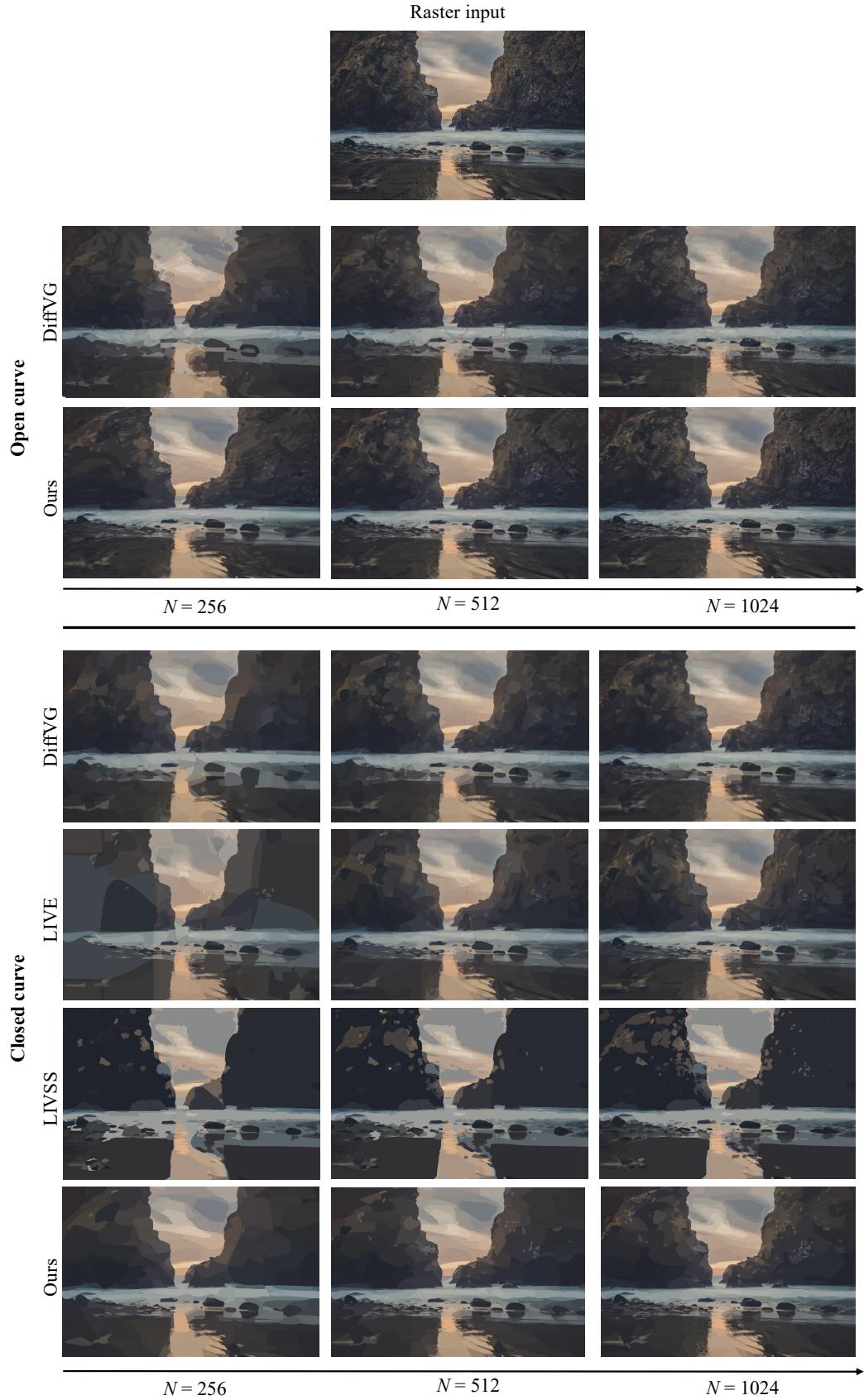


Figure 12: More qualitative comparisons of our method and the existing differentiable VG rasterization method on DIV2K dataset [23].

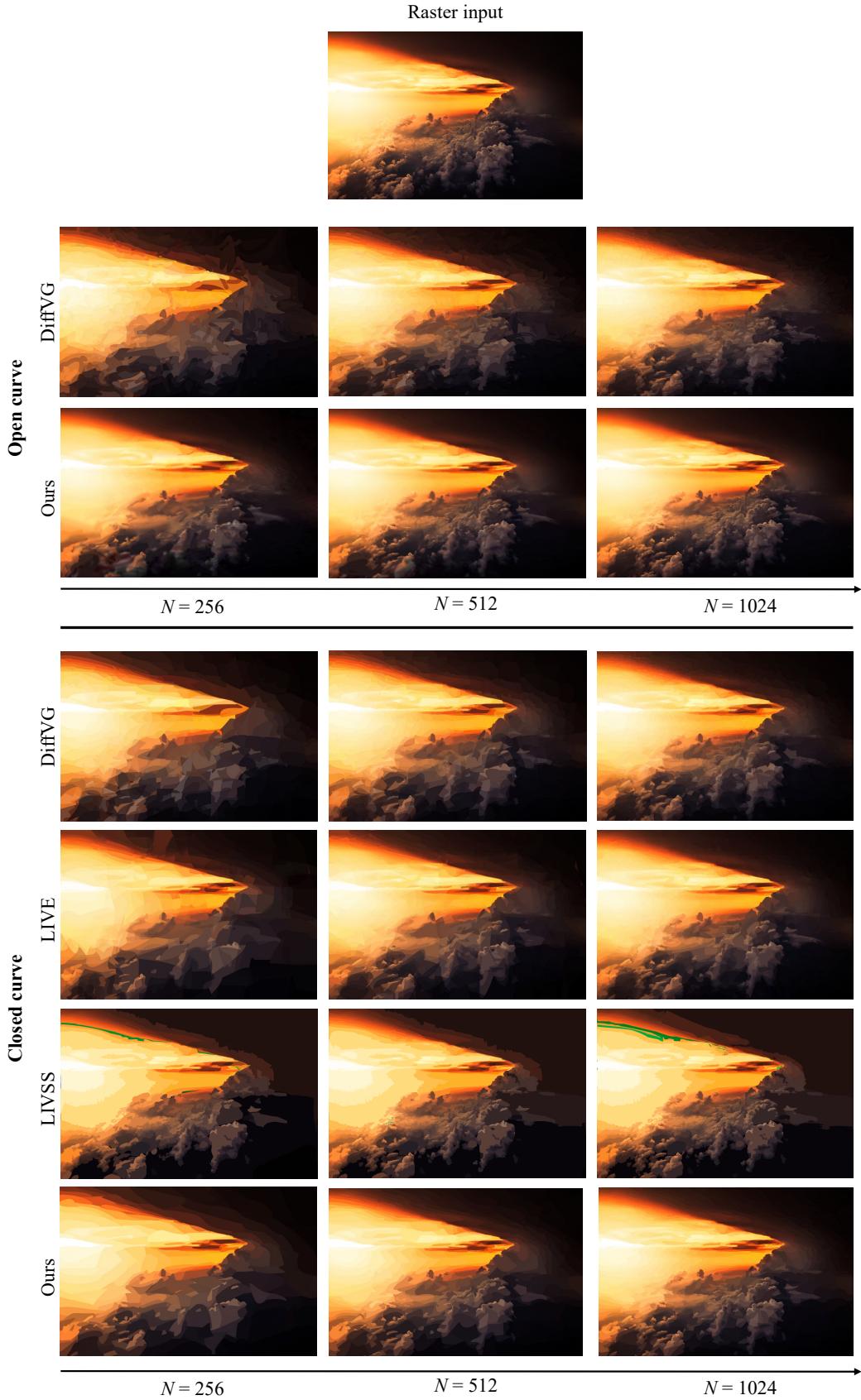


Figure 13: More qualitative comparisons of our method and the existing differentiable VG rasterization method on DIV2K dataset [23].

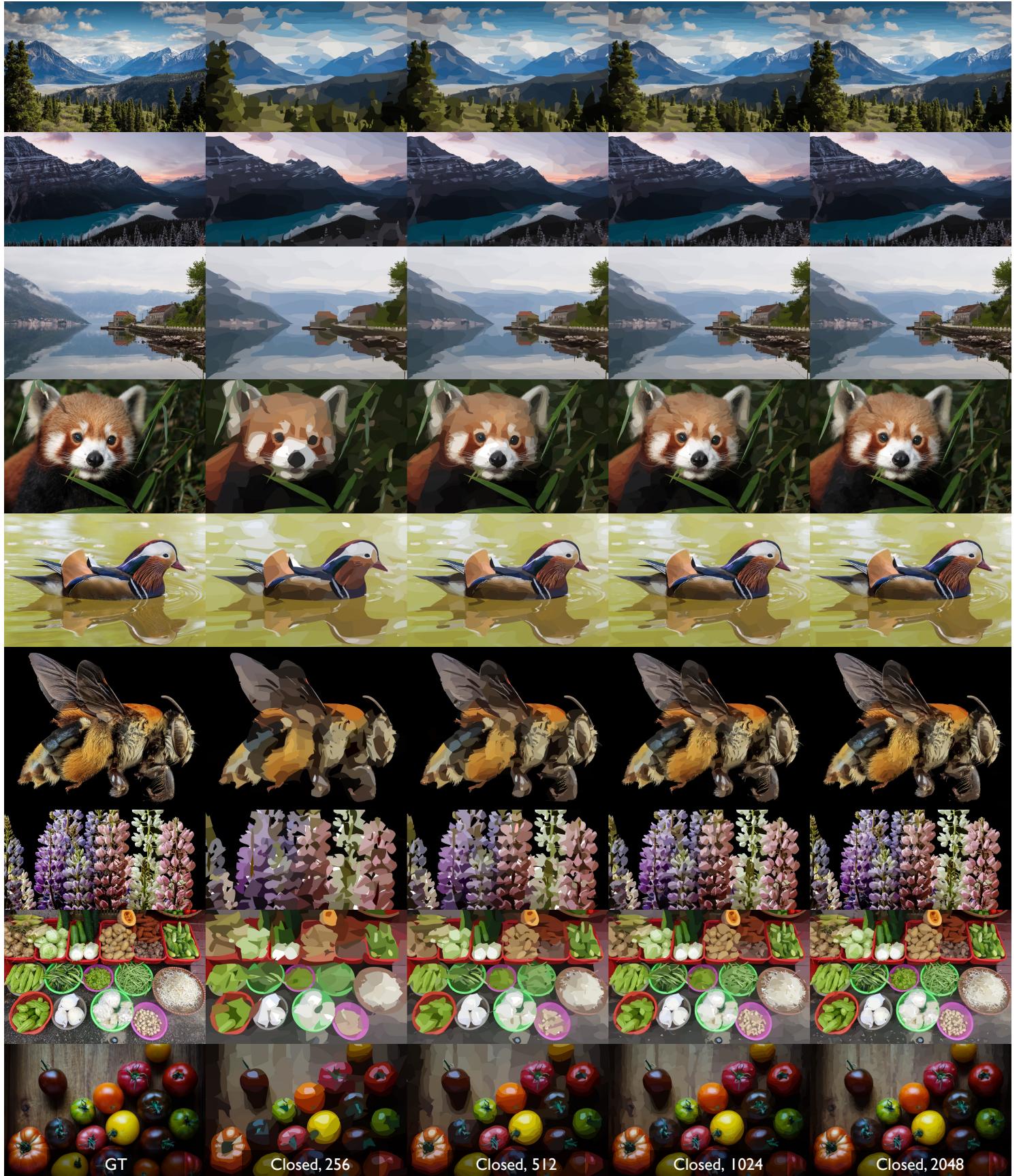


Figure 14: More image vectorization results by Bézier Splatting on DIV2K dataset [23].



Figure 15: More image vectorization results by Bézier Splatting on Kodak [13] dataset.



Figure 16: More image vectorization results by Bézier Splatting on DanboоРegion dataset [31].