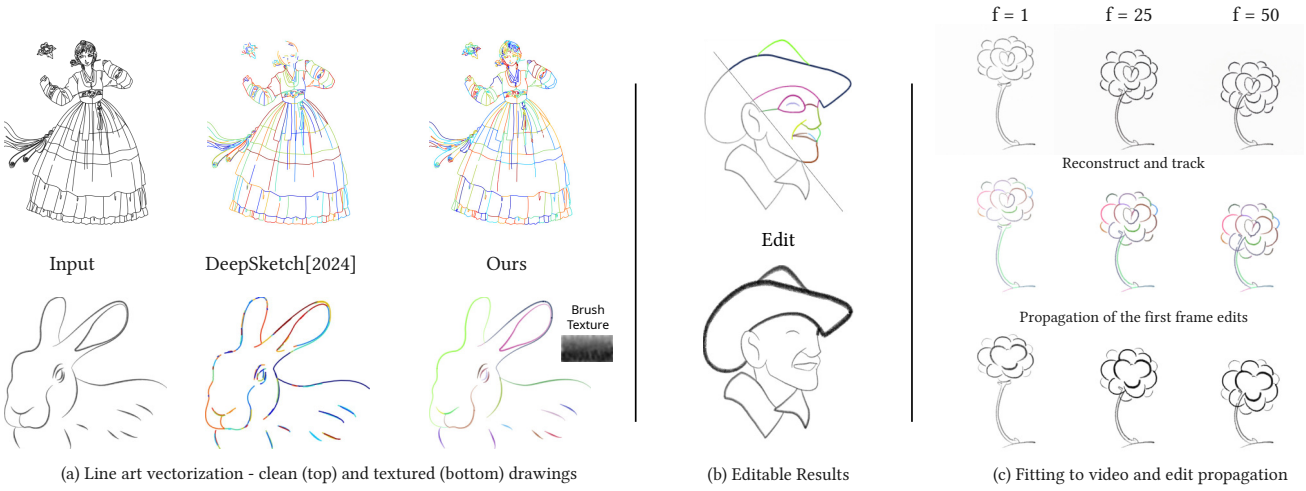
2D Gaussian Splatting for Bézier Spline Line Art Vectorization

TIANHAO CHEN, DisneyResearch|Studios, Switzerland and ETH Zürich, Switzerland
CLARA FERNANDEZ, DisneyResearch|Studios, Switzerland
MARTEINN OSKARSSON, Walt Disney Animation Studios, Canada
CHUCK TAPPAN, Walt Disney Animation Studios, United States of America
OLGA SORKINE-HORNUNG, ETH Zürich, Switzerland
MARKUS GROSS, DisneyResearch|Studios, Switzerland and ETH Zürich, Switzerland
CHRISTOPHER SCHROERS, DisneyResearch|Studios, Switzerland
ABDELAZIZ DJELOUAH, DisneyResearch|Studios, Switzerland



(a) Line art vectorization - clean (top) and textured (bottom) drawings (b) Editable Results (c) Fitting to video and edit propagation

Fig. 1. We propose a novel differentiable framework for line art vectorization. (a) Our parameterization models both stroke geometry and brush texture, outperforming state-of-the-art methods on both clean (top) and textured (bottom) drawings. (b) The resulting Bézier splines are compatible with vector graphic software, allowing users to easily modify both the geometry and appearance of the strokes. (c) Finally, thanks to our differentiable 2D Gaussian renderer, we can jointly optimize multiple frames, showing promising results for temporal tracking and editing of strokes.

Vectorization of line art and sketches is an important problem in computer graphics and remains an active area of research. In this paper, we leverage several recent advances in the field to introduce a new state-of-the-art method for line art vectorization. First, we show that it is possible to go beyond heuristics to extract a meaningful set of strokes from a raster image. We leverage both depth prediction and semantic feature extraction models to propose a more principled approach to split the skeleton graph of the sketch into subgraphs to initialize a set of strokes, producing results that better align with artistic intent. Second, we model strokes as Bézier curves with both geometric and appearance components, and employ 2D Gaussian splatting for fast, differentiable rendering. This formulation enables efficient fitting of strokes to the input while jointly optimizing control points and brush texture. We further demonstrate promising results on video by incorporating temporal tracking and adaptive keyframe introduction. Our evaluation shows that we achieve state-of-the-art results for image reconstruction, while being fast and producing strokes of high quality. More importantly, the fitting is highly compatible with user input, through the correction of the splines and the selection of optimization parameters.

CCS Concepts: • **Computing methodologies** → **Shape analysis**.

ACM Reference Format:
Tianhao Chen, Clara Fernandez, Marteinn Oskarsson, Chuck Tappan, Olga Sorkine-Hornung, Markus Gross, Christopher Schroers, and Abdelaziz Djelouah. 2026. 2D Gaussian Splatting for Bézier Spline Line Art Vectorization. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers (SIGGRAPH Conference Papers '26), July 19–23, 2026, Los Angeles, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3799902.3811211

Authors' Contact Information: Tianhao Chen, DisneyResearch|Studios, Zürich, Switzerland and ETH Zürich, Zürich, Switzerland, authors@disneyresearch.com; Clara Fernandez, DisneyResearch|Studios, Zürich, Switzerland, clara.fernandez@disneyresearch.com; Marteinn Oskarsson, Walt Disney Animation Studios, Vancouver, Canada, marteinn.oskarsson@disneyanimation.com; Chuck Tappan, Walt Disney Animation Studios, Burbank, United States of America, chuck.tappan@disneyanimation.com; Olga Sorkine-Hornung, ETH Zürich, Zürich, Switzerland, sorkine@inf.ethz.ch; Markus Gross, DisneyResearch|Studios, Zürich, Switzerland and ETH Zürich, Zürich, Switzerland, grossm@inf.ethz.ch; Christopher Schroers, DisneyResearch|Studios, Zürich, Switzerland, christopher.schroers@disneyresearch.com; Abdelaziz Djelouah, DisneyResearch|Studios, Zürich, Switzerland, aziz.djelouah@disneyresearch.com.

This work is licensed under a Creative Commons Attribution 4.0 International License.
SIGGRAPH Conference Papers '26, Los Angeles, CA, USA
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2554-8/2026/07
https://doi.org/10.1145/3799902.3811211

1 Introduction

Line art represents an important style component in drawing, painting and many other forms of visual rendering such as animation or games. Artists are very intentional in the way they draw the lines and want to have significant control over the rendering and editing process. This is visible in the variety of works that aim at introducing artist driven line art into the animation pipeline [Whited et al. 2012; Willett et al. 2023]. As a result, vector graphics is the most commonly adopted approach for line manipulation due to its precision and scalability. The motivation of this work stems from the need to vectorize raster images created using a variety of existing techniques: direct painting in raster software; procedural approaches based on filtering techniques applied to rendering buffers or recent machine learning based stylization. The need for line art vectorization techniques is clear, ideally with support for video.

Line drawing vectorization is not a novel problem, and a wide range of methods have been proposed over the years. Early approaches addressed this problem through various optimization techniques, often decomposing the problem into a sequence of independent subtasks, such as topology extraction, curve fitting, or stroke grouping [Bessmeltsev and Solomon 2019; Favreau et al. 2016; Guţan et al. 2023; Noris et al. 2013; Puhachov et al. 2021]. With the rise of deep learning, several works explored end-to-end frameworks that directly predict vector primitives or stroke representations from raster inputs [Carlier et al. 2020; Das et al. 2021; Egiazarian et al. 2020; Ha and Eck 2018; Liu et al. 2018; Lopes et al. 2019; Mo et al. 2021]. More recently, the state-of-the-art method *DeepSketch* [Yan et al. 2024] adopts a multi-stage pipeline, predicting a distance field to the center line followed by isocurve extraction and post-processing. Beyond line drawings, the more general problem of image vectorization has also received significant attention [Dominici et al. 2020; Zhu et al. 2022], including differentiable rendering approaches such as DiffVG [Li et al. 2020] and the recent Bézier Splatting Vector Graphics Renderer [Liu et al. 2025], which offers an appealing alternative for gradient-based optimization.

Despite these advances, relatively few methods explicitly target semantically meaningful, user-editable curve representations that preserve the expressive texture of hand-drawn line art. Moreover, most existing approaches focus on static images and do not naturally extend to animated content, where temporal coherence and consistent stroke tracking across time are essential.

In this paper, we propose a differentiable Bézier spline brush stroke renderer based on 2D Gaussian splatting. Building on this formulation, we introduce an optimization framework for line art vectorization that jointly optimizes stroke geometry, line width, and brush texture. While related work has explored differentiable stroke appearance modeling, our method directly fits artistic line drawings within a unified Bézier spline representation. To achieve robust results, careful initialization is essential: we leverage recent advances in scene understanding to embed semantic cues into an initial skeleton graph extracted from the input image, and formulate stroke initialization as a graph partitioning problem [Blondel et al. 2008]. We model both the stroke geometry and appearance. In particular, we model width using a polynomial in arc length and appearance using a texture map. From a user perspective, both intermediate and final results are standard Bézier splines fully compatible with existing vector graphics and image editing software, allowing users to edit, constrain, or refine individual strokes at any stage of the process. It is also possible to freeze certain parameters (control points, width or brush texture) and optimize other ones.

To summarize, our contributions are as follows:

- A differentiable Bézier spline rendering method using 2D Gaussian splatting for textured line work;
- A line art vectorization method that leverages semantics for better stroke initialization;
- State-of-the-art performance for line art vectorization.

2 Related Work

Differentiable Vector Graphics and Image Decomposition. Vector graphics representations can be effectively optimized through differentiable rendering. DiffVG [Li et al. 2020] introduced general-purpose differentiable vector graphics and it is used for high-fidelity image vectorization [Hirschorn et al. 2024; Li et al. 2020; Ma et al. 2022; Reddy et al. 2021; Wang et al. 2025; Zhao et al. 2025] or stylization [Berio et al. 2025; Liu et al. 2021], but they suffer from substantial computational overhead. The recent Bézier Splatting [Liu et al. 2025] offers a compelling alternative by treating vector primitives as Gaussian-like splats. Combined with Gaussian splatting [Kerbl et al. 2023], it achieves significantly faster rendering. However, when applied to line art, this approach often produces redundant or degenerate strokes, making it ill-suited for topology-sensitive vectorization. Beyond rendering backbones, several methods investigate image decomposition. PolyFit [Dominici et al. 2020] converts raster clip-art into clean, structured vector shapes. Adopting TCB-splines, Zhu et al. [2022] achieve better control over continuity and tension in image vectorization. A series of subsequent works [Chen et al. 2025; Hirschorn et al. 2024; Ma et al. 2022; Wang et al. 2025; Wu et al. 2025; Zhao et al. 2025] explore layer-wise vectorization strategies. By incorporating priors from semantic segmentation [Chen et al. 2025; Wang et al. 2025; Zhao et al. 2025] or vision-language models [Wu et al. 2025], these methods achieve more semantics-aware image decomposition and vector reconstruction.

Optimization-based Methods. Earlier methods extract topology from the raster and fit vector curves to a graph by explicit optimization. Noris et al. [2013] preserve topology for clean line drawings but degrade on rough sketches or highly complex inputs. Favreau et al. [2016] balance fidelity and curve-network simplicity, yet depend on skeleton quality and user interaction. Tracing-based methods fit directional fields to the image [Bao and Fu 2023; Bessmeltsev and Solomon 2019; Guţan et al. 2023; Puhachov et al. 2021; Stanko et al. 2020] but can yield overly complex topology and remain sensitive to noise. Orthogonal to reconstruction, stroke consolidation [Liu et al. 2023, 2018; Van Mossel et al. 2021] cleans overdrawn clusters into simpler vectors, while semantic- and diffusion-guided sketch-vector methods [Arar et al. 2025; Vinker et al. 2022] structure strokes for synthesis rather than faithful vectorization.

Deep Learning-based Methods. More recent line art vectorization methods leverage deep learning. Some approaches [Egiazarian et al. 2020; Ha and Eck 2018; Liu et al. 2022; Lopes et al. 2019; Mo et al.

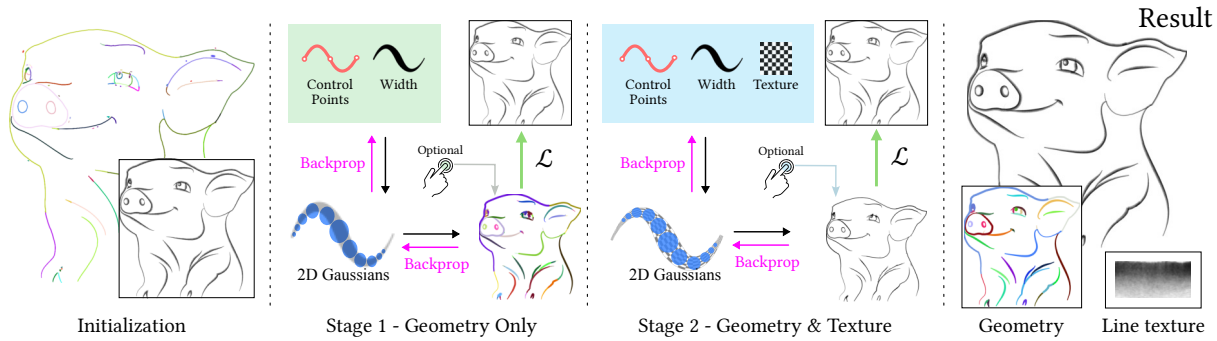


Fig. 2. Method overview. From an input raster, the *Initialization* extracts sub-graphs using depth and semantic features to initialize Bézier splines. The strokes are then iteratively refined via differentiable rendering. In *Stage 1*, we first optimize geometry via the control points and the width. This fitting includes stroke pruning and densification at regular intervals. Once done, *Stage 2* additionally fits a texture map to more precisely represent the brush appearance. The *Result* is a parametrized vectorization of the drawing that includes the lines (with varying thickness) and the brush texture.

2021] use encoder-decoder architectures to predict a latent representation of strokes. Mo et al. [2021] and Liu et al. [2022] use differentiable rendering [Li et al. 2020] to explicit supervise the vectorization process with raster reconstruction loss, but they often produce incomplete results that miss fine details. Other approaches [Magne and Sorkine-Hornung 2025; Puhachov et al. 2021; Yan et al. 2024] use neural networks to predict auxiliary cues, such as keypoints, distance fields or intersection classifications, to guide downstream vectorization. While effective in specific settings, these methods often exhibit poor generalization to other datasets.

Stroke Textures. Rendering of textured strokes is fundamental to digital drawing. Kilgard [2020] proposes a stroking theory that robustly defines and tessellates stroke regions. Ciallo [Ciao et al. 2024] employs a GPU-accelerated rasterizer to render repeated brush patterns along vector paths. NeuBE [Shugrina et al. 2022] leverages a GAN model [Goodfellow et al. 2020] to embed brush styles into a latent space, enabling interactive brush editing and design form existing artworks. More recently, Belhe et al. [2025] introduce a differentiable shader which can be used for inverse brush design; however, their approach requires a predefined vector path as input, while our method jointly learns both stroke geometry and texture.

Drawing Inbetweening. Inbetweening is an important feature in digital content creation software for animation. A variety of methods can be used such as generic point tracking based frame-interpolation [Briedis et al. 2025] or more specialized methods [Brodt and Bessmeltsev 2024; Siyao et al. 2023]. Mo et al. [2024] propose an approach for establishing correspondences between vector strokes across keyframes, but their method relies on a given vectorized input and is tested only on simple sequences with moderate motion and no occlusions. Our application is different as we start from a different set of assumptions (available raster frames) and our objective is to extract tracked editable lines.

3 Line Art Vectorization

An overview of the proposed line art vectorization pipeline is shown in Figure 2. Starting from the input image, we initialize strokes by fitting Bézier splines to a graph derived from the line art skeleton, where depth and semantic cues guide the decomposition into perceptually and artistically meaningful strokes. Initial stroke widths are estimated using a distance transform (Section 3.1).

These initial splines (Section 3.2) are subsequently refined using a differentiable renderer based on 2D Gaussian splatting (Section 3.3). Gaussian primitives are sampled along each spline, and their geometry is optimized by updating the control point positions and stroke width to match the target image. During optimization, we periodically prune low-opacity strokes, remove redundant control points, and introduce new strokes in under sampled regions.

In a second stage, we optimize the brush appearance via a learned texture map. This two-stage strategy improves robustness, as appearance optimization cannot compensate for geometric errors and vice versa. The framework also supports user interaction at any stage, allowing manual edits and selective parameter locking, and remains compatible with existing vector-editing tools.

3.1 Stroke Initialization

Greedy stroke initialization strategies that densely populate the image space [Li et al. 2020; Liu et al. 2025] help avoid vanishing gradients but typically produce an excessive number of poorly structured strokes, leading to inefficient and suboptimal optimization. We instead propose a topology- and semantics-aware initialization that aligns stroke topology with the structure of the input (Figure 3).

Given a raster input image, we extract its line art skeleton [Zhang and Suen 1984] and construct an 8-connected graph G over the skeleton pixels. Stroke initialization is then formulated as a graph partitioning problem, where each partition corresponds to a single stroke. Partitioning the skeleton graph based solely on connectivity is ambiguous, particularly at junctions, and often leads to decompositions that are inconsistent with how an artist would draw the strokes. To obtain perceptually and artistically meaningful partitions, we augment the skeleton graph with additional cues that provide context beyond local geometry. Specifically, we integrate depth [Yang et al. 2024a,b] and semantic features [Darcet et al. 2024; Jose et al. 2025; Oquab et al. 2024] by defining an affinity score $A(i, j) \in [0, 1]$ for each edge $(i, j) \in G$, measuring how likely two connected nodes belong to the same stroke.

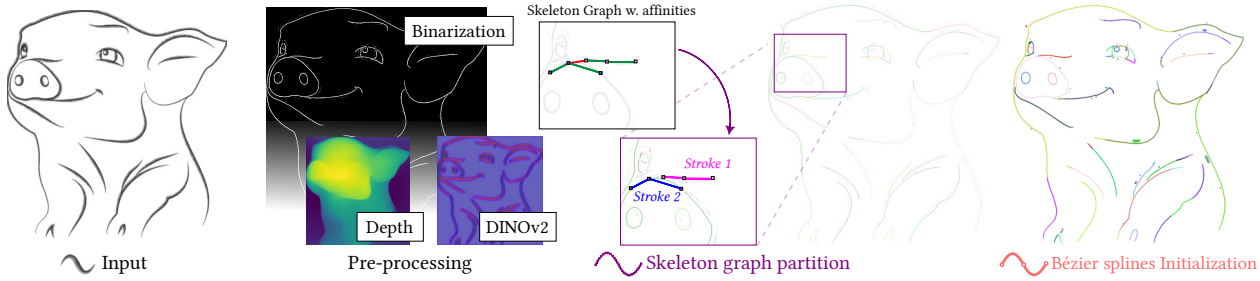


Fig. 3. Bézier Spline Initialization. Starting from the input image, the pre-processing stage consists of image binarization and depth and semantic feature map extraction. The skeleton graph is constructed with affinities computed from these maps, and it is partitioned into sub graphs that become the distinct strokes. This initialization of the strokes more likely corresponds to the way an artist would draw the lines. Finally Bézier splines are fitted to the different strokes.

To improve robustness, affinities are computed using local neighborhoods, where $\mathcal{N}^k_{G\setminus(i,j)}(i)$ denotes the exclusive k-hop neighbors of node i after removing edge (i,j). We compute silhouette scores [Rousseeuw 1987] on depth and semantic features between the neighborhoods of i and j

$$S_z(i,j) = S(\{z_p : p \in \mathcal{N}^k_{G\setminus(i,j)}(i)\}, \{z_q : q \in \mathcal{N}^k_{G\setminus(i,j)}(j)\}), \quad (1)$$

$$S_f(i,j) = S(\{\mathbf{f}_p : p \in \mathcal{N}^k_{G\setminus(i,j)}(i)\}, \{\mathbf{f}_q : q \in \mathcal{N}^k_{G\setminus(i,j)}(j)\}). \quad (2)$$

The silhouette score measures intra-set similarity relative to inter-set dissimilarity, with values close to 1 indicating well-separated sets and values close to -1 indicating highly mixed sets. The final affinity score is defined as

$$A(i,j) = \frac{1 - \lambda_z S_z(i,j) - \lambda_f S_f(i,j)}{2}, \quad (3)$$

where λ_z and λ_f balance depth and semantic contributions.

Before partitioning, we resolve ambiguous junctions. For nodes with degree ≥ 3, edges with affinity below a threshold τ are removed. For degree-3 junctions, we further enforce geometric consistency by estimating local stroke directions from nearby skeleton pixels and removing the edge whose opposite angle is closest to π, favoring straight-through continuations.

The cleaned graph is then partitioned into connected subgraphs [Blondel et al. 2008]. Each subgraph is converted into an Eulerian graph [Edmonds and Johnson 1973] and traversed to extract a minimal set of paths covering all edges. Short paths are discarded, and cubic B-splines [Dierckx 1982, 1995] are fitted to the remaining paths and converted to Bézier splines. Finally, stroke widths are initialized from the skeleton distance transform and modeled by a polynomial function along each stroke.

3.2 Stroke Rendering via Gaussian Splatting

Figure 4 presents a simple example of a two-segment Bézier stroke, illustrating its parametrization, texture-map stamping along the curve, and the corresponding 2D Gaussian splatting rendering used to model the stroke appearance.

Stroke Geometry Modeling. We characterize each stroke by a center line defining the stroke topology, and a width function controlling thickness variation. The center line is modeled as an n-segment cubic Bézier spline $\mathbf{B}(T;\mathcal{P})$, where $\mathcal{P} = \{\mathbf{P}_i\}_{i=0}^{3n}$ are the control points, and $T \in [0,1]$ is a normalized arc-length parameter. Thickness is modeled by a polynomial function of degree m:

$$w(T;\mathbf{a}) = \sum_{k=0}^{m} a_k T^k, \quad (4)$$

where $m = 0$ corresponds to constant-width strokes. The stroke region, defined as the set of points within a distance of $w(T)$ from the centerline $\mathbf{B}(T)$, can be parameterized using local stroke coordinates (T,s), where $s \in [-1,1]$ denotes the normalized offset along the normal direction. Points within the stroke region can be mapped to image space as

$$\mathbf{x}(T,s) = \mathbf{B}(T;\mathcal{P}) + s \cdot w(T;\mathbf{a}) \cdot \mathbf{n}(T;\mathcal{P}), \quad (5)$$

with $\mathbf{n}(T)$ the unit normal to the center line. In practice, since the arc-length parametrization of Bézier splines does not admit a closed-form expression, we recompute a dense lookup table that maps per-segment Bézier parameters to arc-length parameters. We then evaluate $\mathbf{B}(T)$ and $\mathbf{n}(T)$ by converting T back to the corresponding Bézier parameter via efficient search.

Stroke Appearance Modeling. We fix the stroke color to black and model appearance via an opacity texture map defined in local stroke coordinates. Texture repetition and alignment along the stroke are controlled by a scale k and offset d. Given (T,s), texture coordinates $(u,v) \in [-1,1]^2$ are computed as

$$(u,v) = (2 \cdot \mathrm{frac}(kT + d) - 1, s), \quad (6)$$

where $\mathrm{frac}(x) = x - \lfloor x \rfloor$ denotes the fractional part of x.

Gaussian-based Rendering. When rendering the stroke, isotropic 2D Gaussian primitives are sampled within the stroke region and composited via Gaussian splatting [Kerbl et al. 2023; Zhang et al. 2024]. Along the tangent direction, Gaussians are sampled equidistantly in the normalized arc-length parameter T, with a fixed spacing ΔT; along the normal direction, a number of Gaussians equal to the height of the texture map are uniformly sampled over $s \in [-1,1]$.

Compared to Bézier Splatting [Liu et al. 2025], our formulation explicitly accounts for stroke width and textured appearance. Each Gaussian is positioned at $\mathbf{x}(T,s)$, assigned a fixed black color, an opacity sampled from the stroke's texture map, and a scale. The scale is determined by the local stroke width and curvature at the

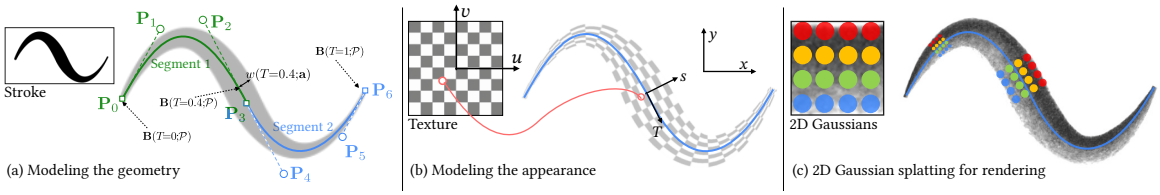


Fig. 4. Line parametrization and 2D Gaussian splatting for rendering. (a) We use Bézier splines to model the center line of the stroke. Given the normalized arc-length parameter $T \in [0, 1]$, the width of the line at $\mathbf{B}(T; \mathcal{P})$ is defined by the width function $w(T; \mathbf{a})$. (b) Stroke appearance is modeled using a texture map that is repeated and translated along the stroke. (c) For rendering, 2D Gaussians are sampled on the texture map, and splatted on the image according to the local stroke width and curvature. As all the components are differentiable, we can optimize stroke parameters end-to-end via gradient descent.

sampling location

$$\sigma(T, s) = \frac{w(T)}{\rho} \left(1 + s w(T) \kappa(T)\right), \tag{7}$$

where $\kappa(T)$ is the local curvature and ρ is an optional trainable density parameter controlling the base scale of the Gaussians. This curvature-aware scaling improves coverage on curved strokes by enlarging Gaussians on the outer side and shrinking them on the inner side. The opacity of each Gaussian is evaluated by bilinearly sampling the stroke's texture map at the corresponding texture coordinates. Finally, the Gaussians are composited using standard α-blending to produce the final rendered image.

Since all operations are differentiable, stroke geometry and appearance can be optimized end-to-end via gradient descent.

3.3 Stroke Optimization

With the differentiable 2D Gaussian splatting renderer and stroke initialization in place, line fitting is performed by backpropagating gradients to the Bézier control points and width parameters. For simple line art, a reconstruction loss is sufficient, while optimizing textured strokes additionally requires a geometric regularization term to prevent appearance and geometry from compensating for each other. We first describe the loss functions and then outline the overall fitting procedure, including stroke densification and simplification.

Reconstruction Loss. This loss term penalizes the discrepancy between the rendered image $\hat{I}$ and the input image I:

$$\mathcal{L}_{\text{recon}}(I, \hat{I}) = \mathcal{L}_{\text{MSE}}(I, \hat{I}) + \lambda_{\text{LPIPS}} \mathcal{L}_{\text{LPIPS}}(I, \hat{I}), \tag{8}$$

with $\mathcal{L}_{\text{MSE}}$ the mean squared error and $\mathcal{L}_{\text{LPIPS}}$ a perceptual loss [2018].

Geometric Regularization Loss. The goal of this loss is to reward desired geometric properties of the estimated curves where λ_{cont}, λ_{int}, and λ_{curv} balance the contributions of each term.

$$\mathcal{L}_{\text{geo}}(\mathcal{P}) = \lambda_{\text{cont}} \mathcal{L}_{\text{cont}}(\mathcal{P}) + \lambda_{\text{int}} \mathcal{L}_{\text{int}}(\mathcal{P}) + \lambda_{\text{curv}} \mathcal{L}_{\text{curv}}(\mathcal{P}). \tag{9}$$

Given a stroke $\mathbf{B}(T)$ parametrized by control points $\mathcal{P} = \{\mathbf{P}_i\}_{i=0}^{3n}$, the continuity loss aims to improve the G^1 continuity at segment junctions, encouraging control points next to segment junctions to be collinear:

$$\mathcal{L}_{\text{cont}}(\mathcal{P}) = 1 - \frac{1}{n-1} \sum_{k=1}^{n-1} \frac{(\mathbf{P}_{3k+1} - \mathbf{P}_{3k}) \cdot (\mathbf{P}_{3k} - \mathbf{P}_{3k-1})}{\|\mathbf{P}_{3k+1} - \mathbf{P}_{3k}\| \|\mathbf{P}_{3k} - \mathbf{P}_{3k-1}\|}. \tag{10}$$

The self-intersection loss $\mathcal{L}_{\text{int}}(\mathcal{P})$ penalizes the presence of self-intersections for each Bézier segment [Hirschorn et al. 2024]. Details about the loss term are presented in supplementary material.

The curvature loss penalizes high curvatures

$$\mathcal{L}_{\text{curv}}(\mathcal{P}) = \frac{1}{s} \sum_{i=1}^{s} \text{ReLU}(\kappa(T_i; \mathcal{P}) - \kappa_{\max}), \tag{11}$$

where $\kappa(T_i; \mathcal{P})$ is the curvature at point $\mathbf{B}(T_i; \mathcal{P})$, $\kappa_{\max}$ is a user-defined maximum curvature threshold, and $\{T_i\}_{i=1}^{s}$ are s uniformly sampled points along the stroke.

Stroke Simplification and Densification. To reduce redundancy and improve efficiency, we periodically prune strokes with low mean opacity and simplify stroke geometry by removing redundant control points based on a Chamfer-distance criterion. To better cover undersampled regions, we detect foreground pixels missing in the rendering using a distance transform and introduce new Bézier strokes in those areas. Implementation details and hyper-parameters are provided in Section 5.1.

4 Extension to Line Art Animation

Recent advances in generative models allow easy creation of line art-style videos from RGB clips or text prompts, but the results are often difficult to edit due to the black-box nature of these methods. To address this, we extend our vectorization approach to videos, enabling vectorized frames with temporal stroke tracking for flexible editing and manipulation. To model the temporal dynamics of strokes, we additionally predict an array of displacement vectors $\{\Delta\mathbf{P}_f\}_{f=1}^{F}$ for each control point and an array of visibility factors $\{\Delta o_f\}_{f=1}^{F}$ for each Bézier segment, where F is the total number of frames. Namely, for any control point $\mathbf{P}$ and texture function $o(T, s)$ at the reference frame, their values at frame f are given by:

$$\mathbf{P}_f = \mathbf{P} + \Delta\mathbf{P}_f, \tag{12}$$

$$o_f(T, s) = o(T, s) \cdot \text{sigmoid}(\Delta o_f). \tag{13}$$

To optimize the dynamic stroke parameters, we first vectorize the reference frame. We then leverage the off-the-shelf point tracking method of DOT [Le Moing et al. 2024]: we sample a set of points along the strokes in the reference frame and track their positions across all frames; we then fit Bézier splines to these tracked points, which provides an initial estimate of the displacement vectors. During optimization, we minimize the expected loss function over all

Table 1. Quantitative comparison of different methods on the clean benchmark dataset [Yan et al. 2024]. The best, second best, and third best results are highlighted in yellow, light gray, and brown, respectively.

Method	Chamfer Dist. ↓	Str. Len. Diff. ↓	Str. Count	LPIPS $\times 10^{-3}$ ↓	PSNR ↑	SSIM $\times 10^{-2}$ ↑
VirtualSketch[2021]	149.29	34.06%	84.86	124.33	25.87	89.96
PolyFlow[2021]	1.75	10.59%	141.37	21.93	31.28	97.90
PolyFields[2019]	1.55	4.58%	143.84	20.58	30.92	97.28
DeepSketch[2024]	1.45	3.14%	94.58	10.07	32.44	98.37
Ours	0.94	1.40%	113.65	5.15	39.70	99.62

Table 2. Quantitative comparison of different methods on the textured benchmark dataset [2024]. The best, second best, and third best results are highlighted in yellow, light gray, and brown, respectively.

Method	Chamfer Dist. ↓	Str. Len. Diff. ↓	Str. Count	LPIPS $\times 10^{-3}$ ↓	PSNR ↑	SSIM $\times 10^{-2}$ ↑
VirtualSketch[2021]	622.12	14.51%	103.32	159.04	18.40	86.23
PolyFlow[2021]	4.52	11.60%	201.29	117.06	18.99	88.94
PolyFields[2019]	4.47	11.06%	156.92	119.12	19.08	88.73
DeepSketch[2024]	2.05	4.68%	108.02	109.36	19.44	89.96
Ours	1.13	5.08%	121.85	18.43	29.02	98.77

frames:

$$\mathcal{L} = \mathbb{E}_f \left[\mathcal{L}_f \right], \tag{14}$$

where $\mathcal{L}_f$ is the loss function formulated in Section 3.3 for frame f.

When fitting videos with significant motion or occlusions, the first frame may not provide sufficient stroke coverage for the entire video, and the point tracking may also be inaccurate. To address these challenges, we iteratively add strokes from other reference frames. Specifically, we monitor the reconstruction error of each frame during optimization. After a fixed number of iterations, we select the frame with the highest error as the new reference frame t_{new}. We then vectorize this frame and track the resulting strokes to all other frames. To avoid duplicating existing strokes, similar to the densification process in Section 3.3, we compute the distance transform of the rendered image from current strokes at frame t_{new}, and only add new strokes whose average distance to existing strokes exceeds the threshold $d_{\min}$.

5 Experiments

5.1 Implementation Details

We implement our method in PyTorch [Paszke et al. 2019]. The optimization is performed using the Adam optimizer [Kingma 2014]. The learning rates for stroke control points $\mathcal{P}$, width parameters $\mathbf{a}$, texture maps, texture scaling and offset factors (k, d), and the density factor ρ are set to 1×10^{-3}, 0.1, 0.1, 0.02, and 0.01, respectively. For solid line art, the texture map is fixed to a single pixel of shape 1×1, and the stroke width is parameterized by a constant value. The density factor ρ, which controls the scale of the Gaussians, is fixed to 2×10^{-3}. For textured line art, the texture map resolution is set to 16×32 pixels, the stroke width is parameterized using a 8-degree polynomial, and the density factor ρ is treated as a trainable parameter. During initialization, the weights of the depth and feature affinity terms are set to $\lambda_z = 0.7$ and $\lambda_f = 0.3$, respectively. The threshold used for determining low-affinity edges is set to $\tau = 0.2$.

For solid line art, we optimize for 200 iterations (Stage 1 only) using the reconstruction loss only, while for textured line art, we optimize for 1,000 iterations (300 for Stage 1 and 700 for Stage 2) using both the reconstruction loss and the geometric loss. For video optimization, we additionally optimize per-frame displacement vectors and visibility factors with learning rates of 0.01 and 1×10^{-3}, respectively. Optimization is done for 15k iterations, and an additional reference frame is introduced at iteration 5k.

Datasets and Metrics. To evaluate our method, we use the benchmark dataset proposed by Yan et al. [2024]. which contains 369 line drawings with vector ground truth varying in complexity and

Table 3. Ablation study on different components of our method: The optimization (O) using random initialization, the graph-based initialization (G), the edge affinity-based initialization (A), and the stroke pruning and densification (PD). We also compare the optimization using our differentiable renderer with DiffVG [Li et al. 2020]. The best results are in **bold**.

Method	Chamfer Dist. ↓	Str. Len. Diff. ↓	Str. Count ↓	LPIPS $\times 10^{-3}$ ↓	PSNR ↑	SSIM $\times 10^{-2}$ ↑
O	1139.74	12.90%	140.91	70.71	28.68	94.68
G	1.39	2.69%	120.67	20.67	29.57	96.99
G+A	1.34	9.10%	113.92	23.73	30.24	97.28
G+A+DiffVG	1.17	4.78%	113.92	11.25	35.01	98.94
G+O	1.28	3.85%	120.67	5.95	39.59	**99.64**
G+A+O	1.08	**1.32%**	113.92	5.59	39.48	99.58
G+A+O+PD	**0.94**	1.40%	**113.65**	**5.15**	**39.70**	99.62

Table 4. Average running time per image (in seconds) for different methods on different resolutions of the clean dataset [Yan et al. 2024]. For both our method and DiffVG we use a fixed total number of 200 steps which we find to be best based on visual quality.

Method	256px	384px	512px	768px	1024px
VirtualSketch[2021]	2.45	3.92	7.19	16.49	26.92
PolyFlow[2021]	10.88	22.03	34.20	46.97	68.07
PolyFields[2019]	1.09	2.01	3.73	6.95	8.56
DeepSketch[2024]	1.64	2.20	4.93	9.23	16.76
Ours (G+A)	1.68	1.94	2.74	3.59	4.69
Ours (G+A+DiffVG)	13.95	18.85	26.32	39.41	60.77
Ours (full)	4.11	5.11	6.37	8.45	10.73

Table 5. Runtime and number of strokes. For a fixed resolution of 512px, we use random initialization for a varied number of strokes. Fitting is done using DiffVG and our approach. We report and compare runtime and Chamfer distance after full convergence (500) steps.

	Chamfer Distance ↓			Runtime (s) ↓		
Number of strokes	512	768	1024	512	768	1024
DiffVG[Li et al. 2020]	168.69	102.97	70.11	37.48	44.16	50.25
Ours	92.3	63.2	51.17	13.5	15.43	17.72

subjects. We also adopt the same evaluation metrics using the chamfer distance and stroke length difference (see supplementary for details), in addition to visual similarity metrics. To evaluate runtime performance, we measure the average time required to vectorize a single image on an NVIDIA RTX 3090 GPU.

5.2 Evaluation and Comparisons

We compare our method against several state-of-the-art line art vectorization methods: VirtualSketch [Mo et al. 2021], PolyFlow [Puhachov et al. 2021], PolyFields [Bessmeltsev and Solomon 2019] and DeepSketch [Yan et al. 2024]. In a first step, we evaluate the methods on the clean lines set. Table 1 summarizes the quantitative results on the benchmark dataset, where our method outperforms all prior approaches across all metrics except the stroke count. Despite this, it maintains a reasonable number of strokes compared to PolyFlow [Puhachov et al. 2021] and PolyFields [Bessmeltsev and Solomon 2019], which generate a larger number of strokes without better reconstruction quality. We note that a lower stroke count may also result from missing strokes (see Figure 1), and therefore does not necessarily indicate improved compactness. Notably, our method decreases the chamfer distance by 35.17% and the stroke length difference by 55.41% compared to the second best method DeepSketch [Yan et al. 2024]. The random stroke initialization strategy of DiffVG leads to poor results (see supplemental material).

Figure 5 presents a representative selection of results from the clean line art set. Our method yields more accurate vectorization results by better capturing fine details and complex topologies. Additional visual results are presented in supplementary material.

In a second stage we compare our method on the textured line art set. These results are presented in Table 2. The difference here is the complexity of the lines, which have both varying width and more complex brush textures. As we are able to fit both the texture and the varying width, our method naturally achieves better reconstruction quality. The quality of the lines still remains high as demonstrated by the other metrics such as chamfer distance.

Figure 6 illustrates performance under textured scenarios. Our approach effectively recovers both the geometry and appearance of the brush strokes. In this figure we represent the different lines in different colors to better showcase the quality of the lines. For downstream tasks, lines should not be too short and ideally attached to a semantically meaningful part of the drawing. The resulting number of control points is also a key feature. Too many points render edits cumbersome. For reference we provide the control points from the ground truth. One can also note that a method like PolyFlow [Puhachov et al. 2021] generates an extremely large number of points. Figure 8 shows more examples of brush textures.

Runtime and Differentiable Rendering. Our method shows comparable or better runtime performance than existing line art vectorization methods (Table 4). It is possible to use DiffVG as the back end renderer within our framework, with our initialization and losses. We use the setting leading to best performance with a learning rate of 0.02 and run the method to full convergence (350 steps). Figure 12 shows that each optimization step is slower in DiffVG with a smaller improvements in quality. Furthermore, Table 5 evaluates the runtime based on the total number of strokes (fixed resolution of 512px). For a full control on the number of strokes, we use predefined stroke counts with a random initialization.

Ablation Study. We conduct an ablation study analyzing the contribution of individual components of our method. Quantitative evaluation on the benchmark dataset is reported in Table 3. The first row shows that, due to sparseness of the gradients, optimization from random initialization often converges to suboptimal solutions. In contrast, graph-based initialization significantly improves reconstruction quality. Incorporating edge affinity further enhances the results, also illustrated in Figure 9, where semantic based affinity helps solve intersection issues indicated by red circles, and the depth helps resolve the remaining issues indicated with blue circles. Stroke pruning and densification lead to additional improvements by adaptively removing redundant strokes and adding missing details. We also compare our optimization with DiffVG [Li et al. 2020], using its default MSE-based configuration and a learning rate of 0.01, which we chose so that the training loss decreases steadily.

Video Results. Figure 7 illustrates the advantage of the extension of vectorization to videos. Using a per image fitting, the reconstruction quality is high, however each frame contains different sets of lines, and editing cannot be propagated from frame to frame. Thanks to our Gaussian based rendering, it is possible to jointly optimize all frames. This leads to fewer lines, but the reconstruction quality is poor. First, by tracking points over time and then introducing keyframes, our method significantly reduces the number of strokes required to represent the entire video sequence.

Compatibility with User Editing. The fitting representation is highly compatible with user edits (Figure 10). The resulting Bézier curves are easy to edit both globally and locally. As we optimize both geometry and texture, it is possible to keep the original line width and only change the brush texture and vice versa. Users can also intervene during the optimization process or continue a few fitting steps after corrections or edits (see Figure 11). More importantly, edits can be propagated over time (Figure 1).

Discussion. Issues remain with rough sketches containing complex textures or noisy backgrounds. Although we remove spurious branches and small connected components during initialization, remaining artifacts may still lead to hallucinated strokes in the final results. For video, a major challenge lies in handling occlusions. Occlusions may occur at arbitrary locations along a stroke. The predicted displacements can partially compensate for this issue, but severe occlusions may still lead to flickering or visual artifacts.

6 Conclusion

We presented a differentiable framework for vectorizing textured line art based on Gaussian splatting, combining topology- and semantics-aware stroke initialization, Bézier spline modeling, and end-to-end optimization of geometry and appearance. Our method enables accurate reconstruction, intuitive editing, and efficient optimization. We further demonstrated how the framework naturally extends to video by integrating temporal tracking and automatically introducing keyframes for frames with high reconstruction error, producing promising results on video.

References

Ellie Arar, Yarden Frenkel, Daniel Cohen-Or, Ariel Shamir, and Yael Vinker. 2025. SwiftSketch: A Diffusion Model for Image-to-Vector Sketch Generation. In *ACM SIGGRAPH 2025 Conference Papers*. 82:1–82:12.

Bin Bao and Hongbo Fu. 2023. Line Drawing Vectorization via Coarse-to-Fine Curve Network Optimization. In *Computer Graphics Forum*, Vol. 42. Wiley Online Library, e14787.

Yash Belhe, Ishit Mehta, Wesley Chang, Iliyan Georgiev, Michael Gharbi, Ravi Ramamoorthi, and Tzu-Mao Li. 2025. Automatic Sampling for Discontinuities in Differentiable Shaders. *ACM Transactions on Graphics (TOG)* 44, 6 (2025), 1–19.

Daniel Berio, Michael Stroh, Sylvain Calinon, Frederic Fol Leymarie, Oliver Deussen, and Ariel Shamir. 2025. Neural Image abstraction using long smoothing B-splines. *ACM Transactions on Graphics (TOG)* 44, 6 (2025), 1–11.

Mikhail Bessmeltsev and Justin Solomon. 2019. Vectorization of line drawings via polyvector fields. *ACM Transactions on Graphics (TOG)* 38, 1 (2019), 1–12.

Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment* 2008, 10 (2008), P10008.

Karlis Martins Briedis, Abdelaziz Djelouah, Raphaël Ortiz, Markus Gross, and Christopher Schroers. 2025. Controllable Tracking-Based Video Frame Interpolation. In *Proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers*. 1–11.

Kirill Brodt and Mikhail Bessmeltsev. 2024. Skeleton-driven inbetweening of bitmap character drawings. *ACM Transactions on Graphics (TOG)* 43, 6 (2024), 1–19.

Alexandre Carlier, Martin Danelljan, Alexandre Alahi, and Radu Timofte. 2020. Deepsvg: A hierarchical generative network for vector graphics animation. *Advances in Neural Information Processing Systems* 33 (2020), 16351–16361.

Ye Chen, Zhangli Hu, Zhongyin Zhao, Yupeng Zhu, Yue Shi, Yuxuan Xiong, and Bingbing Ni. 2025. Easy-editable Image Vectorization with Multi-layer Multi-scale Distributed Visual Feature Embedding. In *Proceedings of the Computer Vision and Pattern Recognition Conference*. 23345–23354.

Shen Ciao, Zhongyue Guan, Qianxi Liu, Li-Yi Wei, and Zeyu Wang. 2024. Ciallo: GPU-accelerated rendering of vector brush strokes. In *ACM SIGGRAPH 2024 Conference Papers*. 1–11.

Timothée Darcet, Maxime Oquab, Julien Mairal, and Piotr Bojanowski. 2024. Vision Transformers Need Registers. In *International Conference on Learning Representations (ICLR)*.

Ayan Das, Yongxin Yang, Timothy M Hospedales, Tao Xiang, and Yi-Zhe Song. 2021. Cloud2curve: Generation and vectorization of parametric sketches. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 7088–7097.

Paul Dierckx. 1982. Algorithms for smoothing data with periodic and parametric splines. *Computer Graphics and Image Processing* 20, 2 (1982), 171–184.

Paul Dierckx. 1995. *Curve and surface fitting with splines*. Oxford University Press.

Edoardo Alberto Dominici, Nico Schertler, Jonathan Griffin, Shayan Hoshyari, Leonid Sigal, and Alla Sheffer. 2020. Polyfit: Perception-aligned vectorization of raster clip-art via intermediate polygonal fitting. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 77–1.

Jack Edmonds and Ellis L Johnson. 1973. Matching, Euler tours and the Chinese postman. *Mathematical programming* 5, 1 (1973), 88–124.

Vage Egiazarian, Oleg Voynov, Alexey Artemov, Denis Volkhonskiy, Aleksandr Safin, Maria Taktasheva, Denis Zorin, and Evgeny Burnaev. 2020. Deep vectorization of technical drawings. In *European conference on computer vision*. Springer, 582–598.

Jean-Dominique Favreau, Florent Lafarge, and Adrien Bousseau. 2016. Fidelity vs. simplicity: a global approach to line drawing vectorization. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 1–10.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2020. Generative adversarial networks. *Commun. ACM* 63, 11 (2020), 139–144.

Olga Guțan, Shreya Hegde, Erick Jimenez Berumen, Mikhail Bessmeltsev, and Edward Chien. 2023. Singularity-Free Frame Fields for Line Drawing Vectorization. In *Computer Graphics Forum*, Vol. 42. Wiley Online Library, e14901.

David Ha and Douglas Eck. 2018. A Neural Representation of Sketch Drawings. In *International Conference on Learning Representations*.

Or Hirschorn, Amir Jevnisek, and Shai Avidan. 2024. Optimize & reduce: a top-down approach for image vectorization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 2148–2156.

Cijo Jose, Théo Moutakanni, Dahyun Kang, Federico Baldassarre, Timothée Darcet, Hu Xu, Daniel Li, Marc Szafraniec, Michaël Ramamonjisoa, Maxime Oquab, et al. 2025. Dinov2 meets text: A unified framework for image-and pixel-level vision-language alignment. In *Proceedings of the Computer Vision and Pattern Recognition Conference*. 24905–24916.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkuehler, and George Drettakis. 2023. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Transactions on Graphics (TOG)* 42, 4 (2023), 1–14.

Mark J Kilgard. 2020. Polar stroking: New theory and methods for stroking paths. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 145–1.

Diederik P Kingma. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

Guillaume Le Moing, Jean Ponce, and Cordelia Schmid. 2024. Dense optical tracking: Connecting the dots. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 19187–19197.

Tzu-Mao Li, Michal Lukáč, Michaël Gharbi, and Jonathan Ragan-Kelley. 2020. Differentiable vector graphics rasterization for editing and learning. *ACM Transactions on Graphics (TOG)* 39, 6 (2020), 1–15.

Chenxi Liu, Toshiki Aoki, Mikhail Bessmeltsev, and Alla Sheffer. 2023. Stripmaker: Perception-driven learned vector sketch consolidation. *ACM Transactions on Graphics (TOG)* 42, 4 (2023), 1–15.

Chenxi Liu, Enrique Rosales, and Alla Sheffer. 2018. Strokeaggregator: Consolidating raw sketches into artist-intended curve drawings. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–15.

Difan Liu, Matthew Fisher, Aaron Hertzmann, and Evangelos Kalogerakis. 2021. Neural strokes: Stylized line drawing of 3d shapes. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 14204–14213.

Hanyuan Liu, Chengze Li, Xueting Liu, and Tien-Tsin Wong. 2022. End-to-end line drawing vectorization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. 4559–4566.

Xi Liu, Chaoyi Zhou, Nanxuan Zhao, and Siyu Huang. 2025. Bézier Splatting for Fast and Differentiable Vector Graphics Rendering. *Advances in Neural Information Processing Systems* (2025).

Raphael Gontijo Lopes, David Ha, Douglas Eck, and Jonathon Shlens. 2019. A learned representation for scalable vector graphics. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 7930–7939.

Xiaohua Ma, Yu Zhou, Yuki Hashimoto, and Hongbo Fu. 2022. LIVE: Layer-wise Image Vectorization with Iterative Refinement. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2047–2056.

Tanguy Magne and Olga Sorkine-Hornung. 2025. Single-Line Drawing Vectorization. In *Computer Graphics Forum*, Vol. 44. Wiley Online Library, e70228.

Haoran Mo, Chengying Gao, and Ruomei Wang. 2024. Joint stroke tracing and correspondence for 2d animation. *ACM Transactions on Graphics* 43, 3 (2024), 1–17.

Haoran Mo, Edgar Simo-Serra, Chengying Gao, Changqing Zou, and Ruomei Wang. 2021. General virtual sketching framework for vector line art. *ACM Transactions on Graphics (TOG)* 40, 4 (2021), 1–14.

Gioacchino Noris, Alexander Hornung, Robert W Sumner, Maryann Simmons, and Markus Gross. 2013. Topology-driven vectorization of clean line drawings. *ACM Transactions on Graphics (TOG)* 32, 1 (2013), 1–11.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. 2024. DINOv2: Learning Robust Visual Features without Supervision. *Transactions on Machine Learning Research Journal* (2024).

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).

Ivan Puhachov, William Neveu, Edward Chien, and Mikhail Bessmeltsev. 2021. Keypoint-driven line drawing vectorization via PolyVector flow. *ACM Transactions on Graphics (TOG)* 40, 6 (2021), 1–17.

Pradyumna Reddy, Michael Gharbi, Michal Lukac, and Niloy J Mitra. 2021. Im2vec: Synthesizing vector graphics without vector supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 7342–7351.

Peter J Rousseeuw. 1987. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics* 20 (1987), 53–65.

Maria Shugrina, Chin-Ying Li, and Sanja Fidler. 2022. Neural Brushstroke Engine: Learning a Latent Style Space of Interactive Drawing Tools. *ACM Transactions on Graphics (TOG)* 41, 6 (2022).

Li Siyao, Tianpei Gu, Weiye Xiao, Henghui Ding, Ziwei Liu, and Chen Change Loy. 2023. Deep geometrized cartoon line inbetweening. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 7291–7300.

Tibor Stanko, Mikhail Bessmeltsev, David Bommes, and Adrien Bousseau. 2020. Integer-Grid Sketch Simplification and Vectorization. In *Computer graphics forum*, Vol. 39. Wiley Online Library, 149–161.

Dave Pagurek Van Mossel, Chenxi Liu, Nicholas Vining, Mikhail Bessmeltsev, and Alla Sheffer. 2021. Strokestrip: Joint parameterization and fitting of stroke clusters. *ACM Transactions on Graphics (TOG)* 40, 4 (2021), 1–18.

Yael Vinker, Ehsan Pajouheshgar, Jessica Y Bo, Roman Bachmann, Amit H Bermano, Daniel Cohen-Or, Amir Zamir, and Ariel Shamir. 2022. CLIPasso: Semantically-aware object sketching. *ACM Transactions on Graphics (TOG)* 41, 4 (2022), 1–11.

Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing* 13, 4 (2004), 600–612.

Zhenyu Wang, Jianxi Huang, Zhida Sun, Yuanhao Gong, Daniel Cohen-Or, and Min Lu. 2025. Layered image vectorization via semantic simplification. In *Proceedings of the*

Computer Vision and Pattern Recognition Conference. 7728–7738.

Brian Whited, Eric Daniels, Michael Kaschalk, Patrick Osborne, and Kyle Odermatt. 2012. Computer-assisted animation of line and paint in Disney's Paperman. In *ACM SIGGRAPH 2012 Talks*. 1–1.

Nora S Willett, Kurt Fleischer, Haldean Brown, Ilene L E, and Mark Meyer. 2023. Curvecrafter: A system for animated curve manipulation. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. 1–11.

Ronghuan Wu, Wanchao Su, and Jing Liao. 2025. LayerPeeler: Autoregressive Peeling for Layer-wise Image Vectorization. In *Proceedings of the SIGGRAPH Asia 2025 Conference Papers*. 1–20.

Chuan Yan, Yong Li, Deepali Aneja, Matthew Fisher, Edgar Simo-Serra, and Yotam Gingold. 2024. Deep sketch vectorization via implicit surface extraction. *ACM Transactions on Graphics (TOG)* 43, 4 (2024), 1–13.

Lihe Yang, Bingyi Kang, Zilong Huang, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. 2024a. Depth anything: Unleashing the power of large-scale unlabeled data. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 10371–10381.

Lihe Yang, Bingyi Kang, Zilong Huang, Zhen Zhao, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. 2024b. Depth anything v2. *Advances in Neural Information Processing Systems* 37 (2024), 21875–21911.

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. 2018. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 586–595.

Tongjie Y Zhang and Ching Y. Suen. 1984. A fast parallel algorithm for thinning digital patterns. *Commun. ACM* 27, 3 (1984), 236–239.

Xinjie Zhang, Xingtong Ge, Tongda Xu, Dailan He, Yan Wang, Hongwei Qin, Guo Lu, Jing Geng, and Jun Zhang. 2024. Gaussianimage: 1000 fps image representation and compression by 2d gaussian splatting. In *European Conference on Computer Vision*. Springer, 327–345.

Kaibo Zhao, Liang Bao, Yufei Li, Xu Su, Ke Zhang, and Xiaotian Qiao. 2025. Less is More: Efficient Image Vectorization with Adaptive Parameterization. In *Proceedings of the Computer Vision and Pattern Recognition Conference*. 18166–18175.

Haikuan Zhu, Juan Cao, Yanyang Xiao, Zhonggui Chen, Zichun Zhong, and Yongjie Jessica Zhang. 2022. Tcb-spline-based image vectorization. *ACM Transactions on Graphics (TOG)* 41, 3 (2022), 1–17.

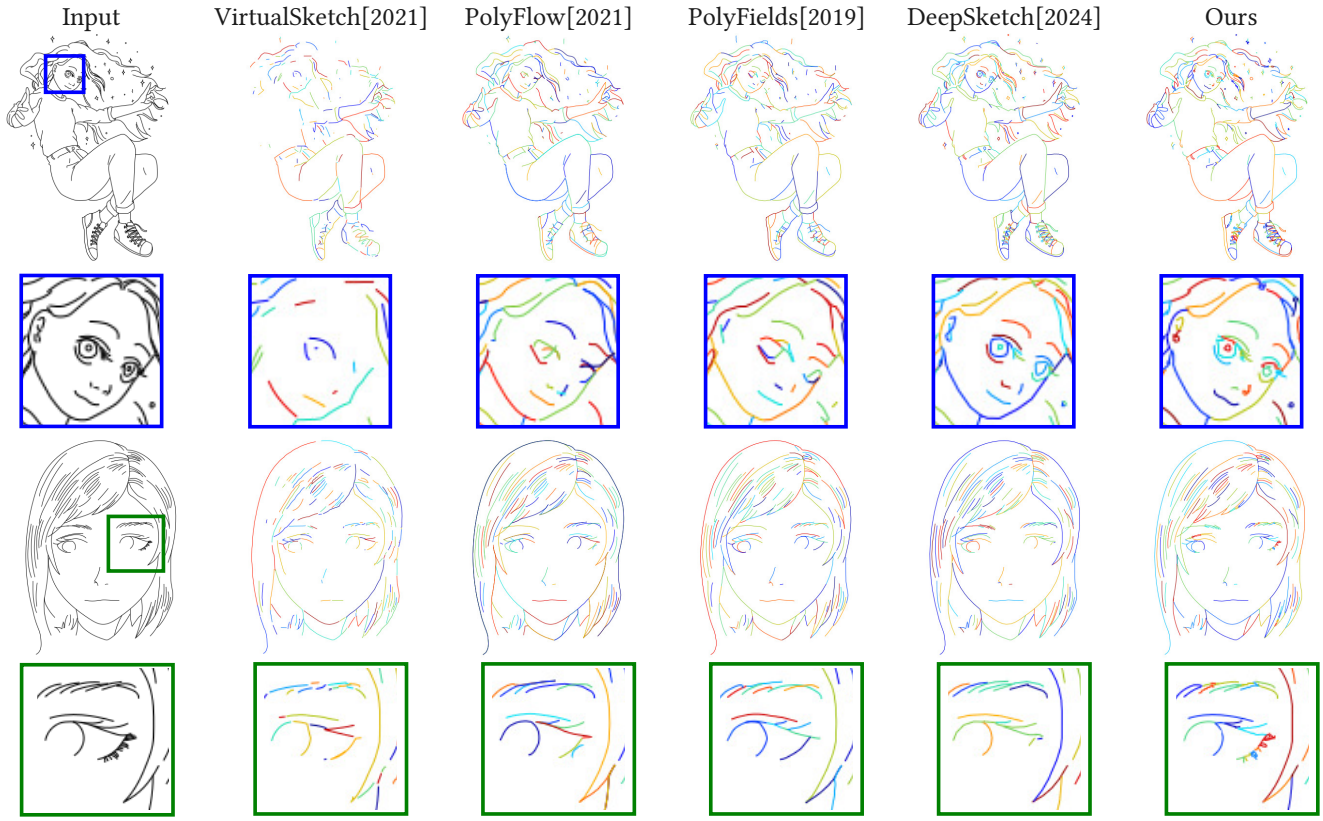


Fig. 5. Qualitative results on the clean line art set. Our method yields results more faithful to the input while maintaining high quality lines.

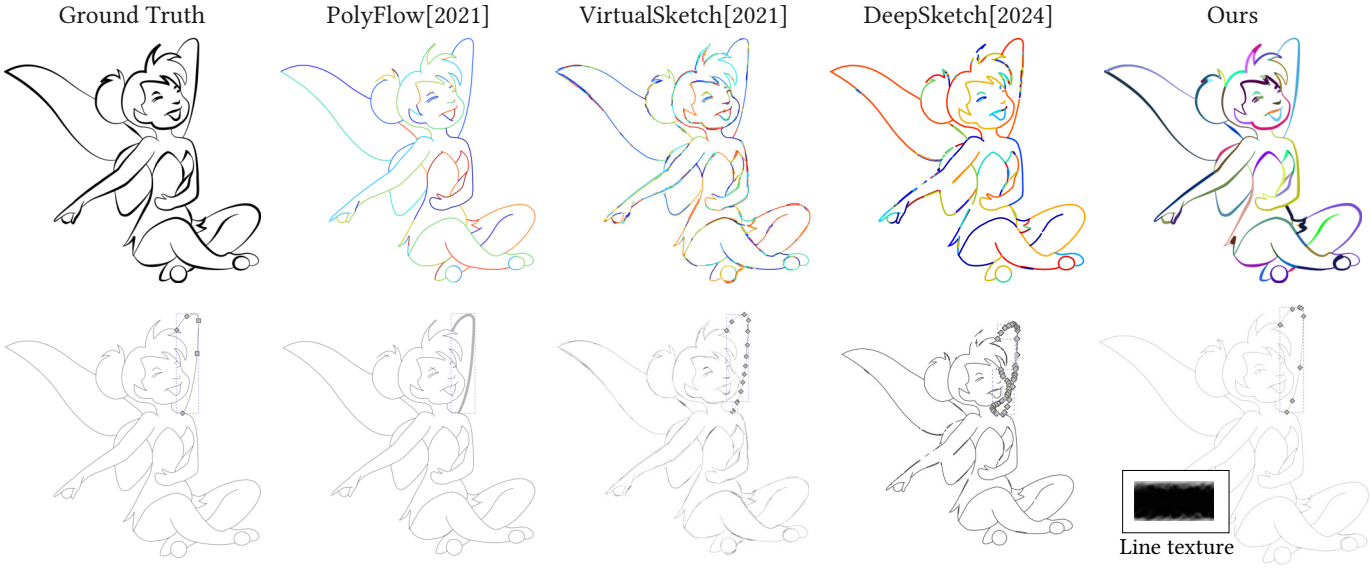


Fig. 6. Qualitative results on the textured line art set. Our approach effectively recovers both the geometry and the brush stroke texture. (*First row*) Different lines are represented in distinct colors. (*Second row*) The resulting number of control points is an important aspect. For reference we provide the control points from the ground truth. PolyFlow [Puhachov et al. 2021] generates an extremely dense set of control points.

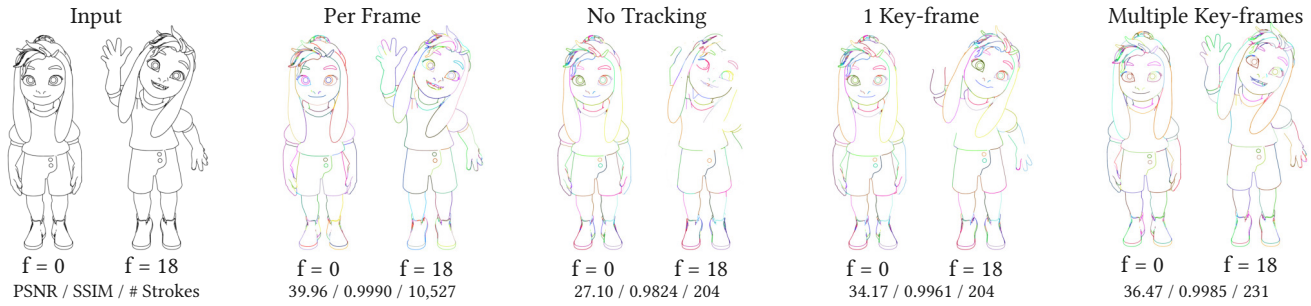


Fig. 7. Video results. Per-frame fitting produces high reconstruction quality but inconsistent stroke sets. Joint optimization across all frames using Gaussian-based rendering enforces temporal consistency, but results in missing strokes and degraded reconstruction quality. By integrating temporal tracking and adaptive keyframe insertion, our method maintains high reconstruction fidelity while producing a compact and temporally consistent stroke representation.

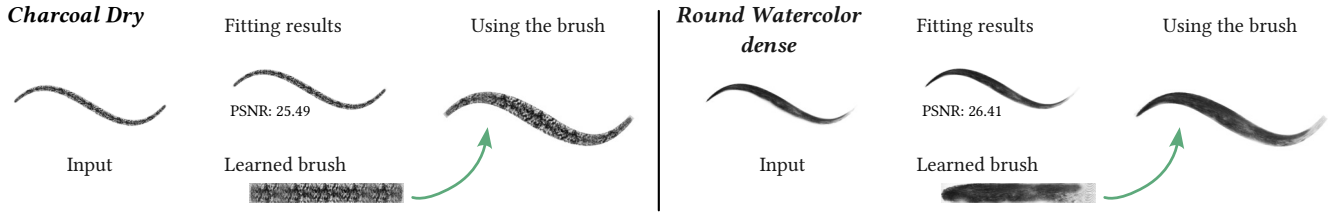


Fig. 8. Brush texture fitting. Given the original stroke on the left side, our optimization process is fitting the original drawing and estimates the brush texture. This brush can be exported as an image and used in DCC software.

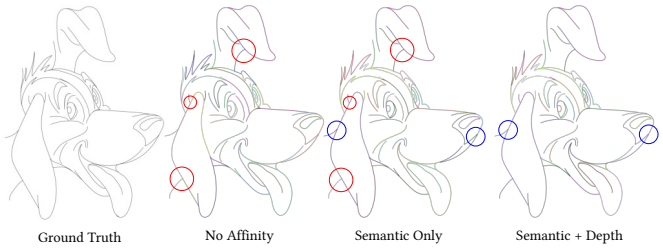


Fig. 9. Edge affinity contribution to the initialization. Semantic based affinities help solve intersection issues indicated by the red circles. Additionally using depth helps resolve the remaining issues indicated with blue circles.

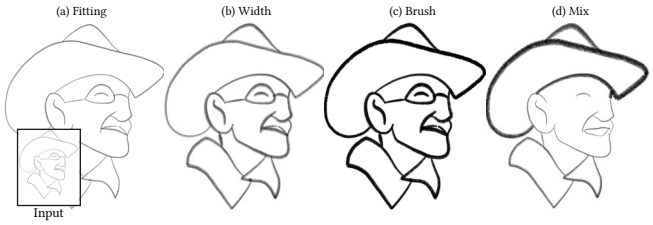


Fig. 10. Starting from the input image, (a) the fitting process converts the line art into Bézier splines which can be edited to change (b) the width (c) the brush type or (d) a combinations of operations.

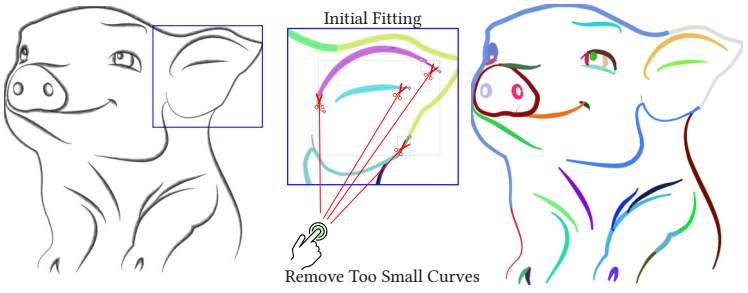


Fig. 11. Editing during optimization. Since our method directly optimizes the Bézier curve parameters, users can intervene at any stage of the process and seamlessly resume optimization. Individual curves or specific control points can be frozen to preserve desired structures. In this example, the user removes short curves before continuing the optimization.

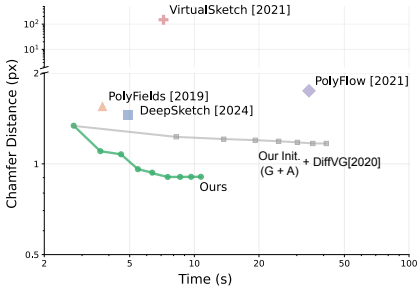


Fig. 12. Runtime comparison and impact on chamfer distance. Both our method and DiffVG are evaluated at regular 50-iteration intervals, from the initialization (leftmost point) to the final result (rightmost point). This shows the advantage of our differentiable rendering approach compared to DiffVG in the case of line art.