Keypoint-Driven Line Drawing Vectorization via PolyVector Flow

Keypoint-Driven Line Drawing Vectorization via PolyVector Flow

IVAN PUHACHOV, Université de Montréal, Canada
WILLIAM NEVEU, Université de Montréal, Canada
EDWARD CHIEN, Boston University, USA
MIKHAIL BESSMELTSEV, Université de Montréal, Canada

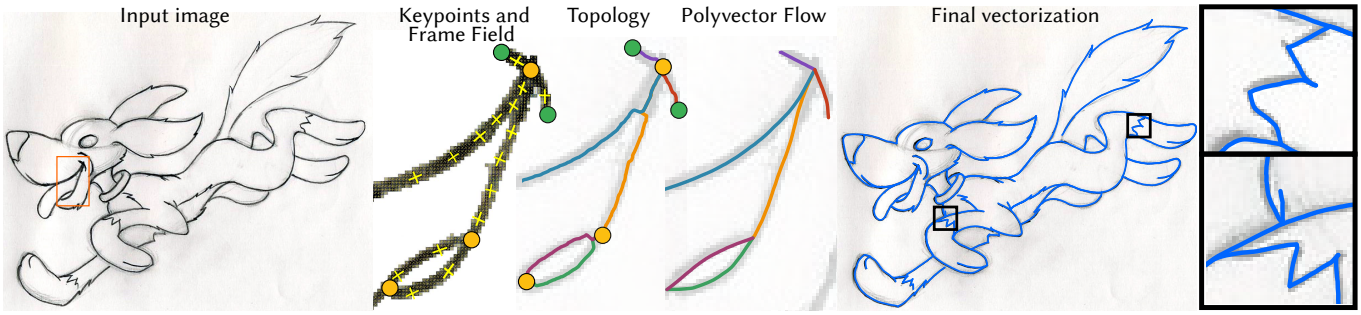


Fig. 1. Given a greyscale bitmap drawing, we use deep learning–based machinery to extract keypoints: junctions, curve endpoints, and sharp corners. We then compute a frame field aligned to the drawing and extract the drawing topology finding curves connecting the keypoints. Finally, we use our novel PolyVector flow that aligns those curves to the frame field, robustly disambiguating directions around keypoints. Input image is from www.easy-drawings-and-sketches.com (c) Ivan Huska.

Line drawing vectorization is a daily task in graphic design, computer animation, and engineering, necessary to convert raster images to a set of curves for editing and geometry processing. Despite recent progress in the area, automatic vectorization tools often produce spurious branches or incorrect connectivity around curve junctions; or smooth out sharp corners. These issues detract from the use of such vectorization tools, both from an aesthetic viewpoint and for feasibility of downstream applications (e.g., automatic coloring or inbetweening). We address these problems by introducing a novel line drawing vectorization algorithm that splits the task into three components: (1) finding keypoints, i.e., curve endpoints, junctions, and sharp corners; (2) extracting drawing topology, i.e., finding connections between keypoints; and (3) computing the geometry of those connections. We compute the optimal geometry of the connecting curves via a novel geometric flow — *PolyVector Flow* — that aligns the curves to the drawing, disambiguating directions around Y-, X-, and T-junctions. We show that our system robustly infers both the geometry and topology of detailed complex drawings. We validate our system both quantitatively and qualitatively, demonstrating that our method visually outperforms previous work.

CCS Concepts: • **Computing methodologies** → **Parametric curve and surface models**.

Authors' addresses: Ivan Puhachov, Université de Montréal, Canada, ivan.puhachov@umontreal.ca; William Neveu, Université de Montréal, Canada, william.neveu@umontreal.ca; Edward Chien, Boston University, USA, edchien@bu.edu; Mikhail Bessmeltsev, Université de Montréal, Canada, bmpix@iro.umontreal.ca.

0730-0301/2021/12-ART266 $15.00
https://doi.org/10.1145/3478513.3480529

Additional Key Words and Phrases: line drawing, vectorization, frame field, geometric flow

ACM Reference Format:
Ivan Puhachov, William Neveu, Edward Chien, and Mikhail Bessmeltsev. 2021. Keypoint-Driven Line Drawing Vectorization via PolyVector Flow. *ACM Trans. Graph.* 40, 6, Article 266 (December 2021), 17 pages. https://doi.org/10.1145/3478513.3480529

1 INTRODUCTION

Line drawings are commonplace in numerous industries, including cartoon animation and graphic design. These drawings are often created with traditional tools, such as pen and paper, or in raster software. Before the image can be printed in high resolution or used in downstream applications, such as coloring or animation, the artist often has to recreate their drawings in vector format. Despite recent progress, many artists, frustrated by the modern vectorization tools, prefer manual vectorization, although it is tedious and error-prone.

Typical results of automatic vectorization tools include spurious branches, smoothed out sharp corners, and incorrect geometry or connectivity around junctions (Fig. 3). The significance of these topological and geometrical artifacts goes well beyond aesthetics, as they impede the use of automatic vectorizations in downstream applications: For instance, sketch-based modeling applications heavily rely on junction positions and connectivity [Gryaditskaya et al. 2020; Lipson and Shpitalni 1996].

One of the core causes for these artifacts is incorrect or imprecise treatment of positions and directions around *keypoints* of three main types, i.e. curve endpoints, junctions, and sharp corners. The main challenge in robustly finding, disambiguating, and connecting keypoints is insufficient or noisy local information. For clean and noisy drawings alike, color intensity and its gradient around each

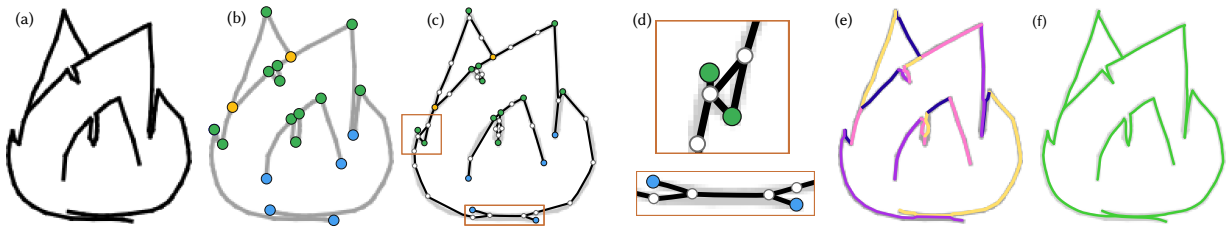


Fig. 2. Overview of our method: Starting with a raster image (a), our keypoint extraction network outputs the key topological features and their type: junctions (yellow), sharp corners (green), and endpoints (blue) (b). We then compute the drawing-aligned frame field, disambiguating directions around junctions, which we use to generate the rough topological graph of the drawing (c). We then extract the paths over this graph between the computed keypoints, resolving ambiguities (d) using keypoints type. We therefore compute the topology and the initial geometry of the vectorization (e). Finally, we use our novel PolyVector Flow to align the initial paths to the frame field, resulting in the final vectorization (f).

keypoint are unreliable. This issue complicates inferring geometry, such as curve positions and directions around keypoints, as well as topology, such as deciding whether two curves are connected or not (Fig. 3).

Traditional vectorization approaches relied on 1-skeleton [Favreau et al. 2016; Noris et al. 2013] or gradient [Noris et al. 2013]. A recent line of work robustly disambiguates directions around junctions via a *frame field* [Bessmeltsev and Solomon 2019; Stanko et al. 2020]. Their frame field attaches two directions to each pixel, so that at least one of the directions is aligned to the curve tangent. As a result, this frame field aligns naturally with the two intersecting curve directions around X- and T-junctions, typical for line drawings, disambiguating those directions. Our work is inspired by those; however, their line of work addressed only *directions* and did not robustly infer *positions* of the keypoints.

Our observation is that away from the keypoints — the main source of topological and geometrical ambiguity — topology is trivial to infer, and curve geometry has well-defined, purely local criteria: curves should be aligned with the drawing tangent and be centered with respect to the drawn stroke. If the locations and connections between all keypoints are known, the principal task left is to align centered curves to the image tangents. To this end, we introduce a novel geometric flow, *PolyVector Flow*, that aligns a given curve to the frame field capturing drawing directions, robustly disambiguating directions around keypoints.

Based on this flow, we propose a novel line drawing vectorization method that offers robust junctions, sharp corners, and endpoints resolution in terms of both positions and directions. Our system contains three stages: (1) determine positions and type of all keypoints via a deep learning system; (2) extract drawing topology, i.e. find connections between keypoints via optimization and (3) find geometry of each connection curve via PolyVector Flow.

Our technical innovation is two-fold:

- we introduce the novel PolyVector flow, derived from an anisotropic curve shortening flow, that aligns the given curve to the drawing tangents, captured by a frame field, and
- we leverage that flow along with a deep learning–based machinery and a novel topology extraction algorithm to introduce a line drawing vectorization system, focused on the robust treatment of endpoints, sharp corners, and junctions.

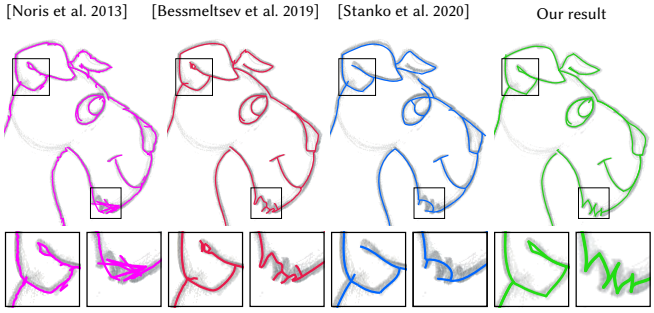


Fig. 3. Traditional approaches [Noris et al. 2013] suffer from geometrical and topological artifacts around keypoints: junctions, sharp corners, and endpoints. Frame field–based approaches [Bessmeltsev and Solomon 2019; Stanko et al. 2020] resolve directional ambiguities around keypoints, but not their positions, leading to incorrect topology. Our approach addresses all of these challenges (right).

Overview. Our system takes as an input a grayscale bitmap line drawing and produces a set of curves aligned to the drawing (Fig. 2). We first design and train a deep neural network outputting locations and type of all keypoints in the image (Sec. 3). We then compute the frame field, aligned to the drawing, and use it to compute the initial drawing graph, using previous work [Bessmeltsev and Solomon 2019] (Sec. 4.1). This initial graph captures all the connectivity of the drawing, but contains additional connections. We use this graph to find a set of paths between the computed keypoints that cover the drawing and respect the inferred keypoint types (Sec. 4). Finally, we create the vectorization aligned with the drawing by deforming those initial paths via the novel *PolyVector Flow* (Sec. 5).

2 RELATED WORK

Our work is inspired by the progress in two areas: line drawing vectorization and geometric flows. We focus only on the most relevant works. Vectorization methods for shaded images, which typically decompose the input bitmap into closed primitives [Jun et al. 2017; Lecot and Lévy 2006; Orzan et al. 2013; Zhang et al. 2009], as well as the methods for clip-art image vectorization, which typically assume images are composed solely of closed flat-colored regions [Dominici et al. 2020; Hoshyari et al. 2018], are outside our scope.

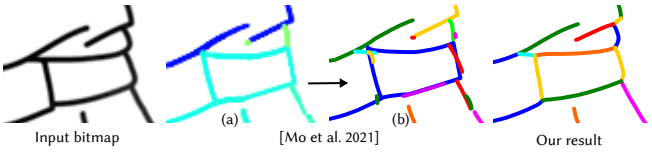


Fig. 4. Typical deep learning–based architectures focus mainly on minimizing visual differences between their result and the input image, and do not target reconstructing curve connectivity, often forming vectorizations that are problematic for editing or downstream applications [Mo et al. 2021] (their rasterized result with their stroke widths (a), the same result with fixed stroke widths (b)). In contrast, we aim to reconstruct correct topology (right).

We target precise vectorization of freeform line drawings with no or little shading and support both closed and open curves.

Line Drawing Vectorization. Vectorization of bitmap images containing curves is an extensively researched topic. The existing solutions in medical imaging or road map reconstruction use domain-specific knowledge which cannot be directly transferred onto line drawings [Chai et al. 2013; Türetken et al. 2013]. Unlike technical drawings, where often straight lines or a small number of simple primitives are assumed [Egiazarian et al. 2020; Hilaire and Tombre 2006], freeform line drawings are often composed of complex piecewise smooth curves, rendering those specialized methods largely inapplicable.

Vectorization algorithms for *clean* line drawings typically rely on 1-skeleton or raw image gradient to extract curve directions, endpoints, junctions, and connectivity [Bo et al. 2016; Donati et al. 2017, 2019; Najgebauer and Scherer 2019; Noris et al. 2013]. Noris et al. [2013] vectorize clean digital line drawings via a global approach to topology. Donati et al. [2017; 2019] use Pearson's Correlation coefficient to extract curvilinear pixel structures, skeletonize, and fit Bezier curves. Both 1-skeleton and raw image gradient, however, are unreliable and lead to geometric and topological artifacts in the presence of noise. Those artifacts in 1-skeletons can be only partially addressed via heavy processing or expensive optimization [Favreau et al. 2016].

Methods for simultaneous vectorization and simplification of *rough* overdrawn sketches often extend the traditional pipeline by first merging the adjacent strokes in the bitmap image, and only then computing 1-skeleton [Bartolo et al. 2007; Parakkat et al. 2018]. Alternatively, Goes et al. [2011] use an optimal transport framework, and Chen et al. [2018] take a region-based approach. Favreau et al. [2016] address the problem of noisy topology of 1-skeleton by introducing a global topology optimization.

In contrast to these approaches, we target precise vectorization without simplification. Simplification can be addressed either via vector-based methods [Liu et al. 2018] or bitmap simplification methods. The latter filter rough bitmap sketches, producing a simplified bitmap image visually merging parallel strokes via blurring kernels [Chen et al. 2013; Kang et al. 2007] or deep learning [Simo-Serra et al. 2018, 2016; Xu et al. 2019]. We view these approaches as complementary to ours in case simplification is desired; we target precise vectorization of sketches with no intent to simplify them (Fig. 16).

Recent works have explored using deep learning for vectorization [Carlier et al. 2020; Das et al. 2021; Egiazarian et al. 2020; Gao et al. 2019; Guo et al. 2019; Kim et al. 2018; Li et al. 2020; Liu et al. 2017; Lopes et al. 2019; Reddy et al. 2021; Zhou et al. 2019a]. These approaches either cast vectorization as a segmentation task [Kim et al. 2018], or predict a fixed number of curves either via direct vector supervision [Carlier et al. 2020; Lopes et al. 2019] or via a differentiable rasterizer [Li et al. 2020]. Other approaches improve on that by designing RNNs to predict splines from images [Gao et al. 2019; Reddy et al. 2021]. These approaches typically target vectorization of simple doodles due to the limited complexity of the available datasets. Mo et al. [2021] use an RNN with a differentiable rendering to train on raster data only. All these approaches make no effort to reconstruct correct drawing topology, which is essential for downstream applications. In contrast, our system vectorizes complex line drawings with correct topology (Fig. 4 and Supplementary).

We are inspired by the approaches that separate junction detection and topology reconstruction [Guo et al. 2019; Liu et al. 2017; Zhou et al. 2019a]. Guo et al. [2019] use a fully convolutional architecture to infer junctions and centerlines, and disambiguate connectivity around junctions. Zhou et al. [2019a] detect straight-line wireframes in photographs. Overall, end-to-end approaches [Das et al. 2021; Guo et al. 2019], being trained on limited or fully synthetic datasets, struggle to generalize on noisy real pen-and-paper sketches or even on clean digital drawings with variable stroke width. In contrast, we only use learning to detect and classify keypoints, a task that, as we demonstrate, generalizes well (Sec. 3).

Addressing the issue of noisy image gradient, some works generate a smooth tangent direction field and trace its integral curves [Bao and Fu 2012; Chen et al. 2018, 2015]. We build upon a recent line of work that extended those via *frame fields*, assignment of two directions per pixel, one aligned to the curve tangent [Bessmeltsev and Solomon 2019; Stanko et al. 2020]. This disambiguates the directions of X- and T-junctions, since frame field directions capture the directions of the two intersecting curves. Starting with a frame field, Bessmeltsev and Solomon [2019] trace multiple integral curves, group them, form a graph than they then clean up and convert into the vectorization. Stanko et al. [2020] improves the tracing mechanism by first computing the uv-parameterization of the narrow band, and then extracting the parameterization isolines. These methods, however, only focus on the problem of ambiguous directions and not positions of the keypoints. Bessmeltsev and Solomon [2019] extract keypoints using a noisy 1-skeleton, and Stanko et al. [2020] make topological decisions based on a global parameterization that depends on a user-defined local scaling mask. As a result, both methods often produce incorrect connectivity, spurious branches, or smoothed out sharp corners (Fig. 3). In contrast, we extract keypoint positions and type directly from the drawing using a deep neural network, and treat the problem of vectorization as finding connectivity and geometry of curves connecting those points, aligned to the frame field. Our method shares the frame field design as well as graph computation with Bessmeltsev and Solomon [2019], but uses those in a fundamentally different framework. We demonstrate an in-depth comparison of the results to that work in Section 6.

Discretized Geometric Flows. There are several geometric surface or curve flows that have been utilized in geometry processing. The most relevant to our setting is that of mean curvature flow, corresponding to minimization of surface area or curve length. The applications have consisted mostly of methods for surface smoothing and fairing [Desbrun et al. 1999; Leng et al. 2013; Taubin 1995] and/or multi-resolution processing [Guskov et al. 1999; Kobbelt 2000; Kobbelt et al. 1998]. It has also found use in fluid dynamics [Ishida et al. 2017; Misztal et al. 2012; Thürey et al. 2010; Zhang et al. 2012].

None of the above consider an anisotropic mean curvature flow that varies throughout the manifold or domain. One formulation of such an anisotropic mean curvature flow considers gradient flow of an area/length functional where the area/length element varies with direction of the surface/curve normal. Several theoretical works analyze numerical schemes and convergence for such flows where the area/length element is translation-invariant and constant over the domain [Clarenz et al. 2004; Deckelnick et al. 2005; Dziuk 1999; Dziuk and Elliott 2013]. We follow their general approach in derivation of the PolyVector Flow, but modify it to allow for a length element that is not translation-invariant.

The use of an area/length element that varies over the domain is also considered in Clarenz et al. [2003]. There they detail two different algorithms that generalize mean curvature flow for feature-preserving surface fairing. One of their models is of a similar mathematical formulation, but our work presents a novel functional tailored specifically to the task at hand. Furthermore, the application of these ideas to frame field alignment is novel.

In computer vision, a modification of curve shortening flow inspired the rise of a successful line of methods for edge detection and image segmentation, called Active Contours [Caselles et al. 1995; Kichenassamy et al. 1995]. As shown in Kichenassamy et al. [1995], active contours can be thought of as a curve shortening flow with an edge-aware, spatially-varying Riemannian metric. This metric only scales the standard Riemannian metric, and the length element does not vary with direction. The barrier term in our model, which promotes centering of the curves, is an example of this sort of model.

Lastly, let us also mention a few other related flows which have been used for smoothing and fairing of surfaces: Willmore flow [Bobenko and Schröder 2005; Clarenz et al. 2004; Crane et al. 2013; Gruber and Aulisa 2020] and a diffusion flow, related to the bi-Laplacian [Schneider and Kobbelt 2001].

3 KEYPOINT EXTRACTION

The first stage of our algorithm takes a greyscale bitmap image as an input and produces a set of real-valued 2D positions of all the keypoints and their type, i.e., whether they are junctions, curve endpoints, or sharp corners. In the next stages of the algorithm, the final vectorization is created by connecting these keypoints using their inferred type. For junctions in particular, we do not specifically aim to extract Y-junctions due to their somewhat ambiguous location in the drawing, only focusing on X-, T-, and high-valence junctions (Fig. 5). The positions of Y-junctions are resolved later due to the PolyVector Flow (Sec. 5).

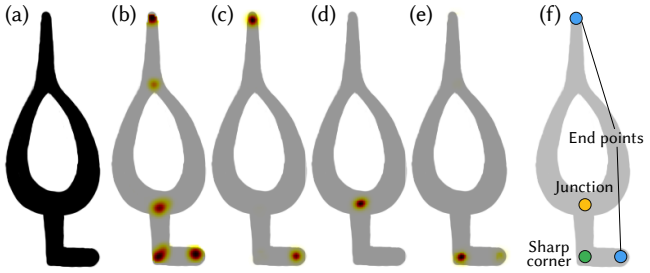


Fig. 5. Given an input image (a), in the first stage of our pipeline we train our network to predict four heatmaps: likelihood of any keypoint H_1 (b), likelihood of an endpoint H_2 (c), likelihood of a junction H_3 (d), and likelihood of a sharp corner H_4 (e). We finally use non-maximum suppression (f) to extract the locations of junctions (orange), curve endpoints (blue), and sharp corners (green). We do not specifically target Y-junctions.

We train an anchor-free object detection system, which produces four heatmaps of the same resolution as the input image (Fig. 5b). The first heatmap H_1 shows likelihood of *any* keypoint at a location, while the other three, H_2, H_3, H_4, show likelihood of keypoints of each of the three types respectively. We then extract positions of all the keypoints using H_1 via non-maximum suppression with the minimal distance between peaks of one pixel (Fig. 5c). For each keypoint, we then extract its type by finding the maximum of the three type heatmaps at that location. Note that while predicting the first heatmap may not seem necessary and can be theoretically replaced by a sum of the other three, we found this setup to be more robust and to produce fewer redundant predictions.

We base our architecture on a combination of state-of-the-art systems for object detection: CenterNet [Zhou et al. 2019b], Hourglass [Newell et al. 2016], and Feature Pyramid Network [Lin et al. 2017]. We additionally experimented with other keypoint detection or object detection architectures [Redmon et al. 2016; Xie and Tu 2017], but found this combination to perform better. We use a similar architecture that downsamples and upsamples the image at multiple levels, at each level extracting features and propagating them using skip connections. In addition to the final four heatmaps H_1, H_2, H_3, H_4, the network predicts two intermediate downscaled versions (0.25× and 0.5×) of H_1 that we use to facilitate training. The exact architecture is detailed in Appendix B.

3.1 Training details

Loss Function. Our loss is composed of six terms, each comparing the four final $H_1, \ldots, H_4$ and two intermediate heatmaps H_5, H_6 to the ground truth heatmaps $\hat{H}_i$, computed via placing Gaussians ($\sigma = 2$) on each annotated ground truth keypoint. Each loss is an L_1-like asymmetric norm that penalizes false negatives more than false positives:

$$\mathcal{L} = \sum_{i=1}^{6} w_i \phi(H_i - \hat{H}_i), \tag{1}$$

$$\phi(x) = \begin{cases} 0.6x, & \text{if } x \geq 0 \\ |x|, & \text{otherwise} \end{cases}$$

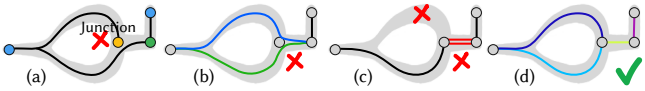


Fig. 6. We extract the topology, represented as a set of paths between keypoints, subject to three requirements. The paths should respect keypoint valence (a), correctly disambiguate junctions (b), and cover the whole drawing (c). Correct vectorization (d).

We use the following weights in our experiments: $w_1 = 1, w_2 = w_3 = w_4 = 0.2, w_5 = 0.4, w_6 = 0.6$.

Training. We train our network in a fully supervised manner on our dataset that consists of rasterized vector sketches. We use the Adam optimizer with learning rate $5 \cdot 10^{-4}$ and cosine annealing. We set the rest of parameters to their default *PyTorch* values. The model is trained for 18 epochs. We use random rotations, Gaussian noise, and other standard augmentations to prevent overfitting. We then fine-tune the model on a smaller dataset for 20 epochs. Details on both datasets and augmentation are presented in Appendix B. The images presented in our results and figures, except for the dataset sample in Fig. 22, were not used in training.

Detection robustness. We evaluate the detector on a separate hand labeled dataset of 40 *real* line drawings. The trained model reaches detection F_1 score of 0.87 and classification accuracy of 0.86 on the correctly detected keypoints. See Appendix B for more details. We design the rest of our pipeline to accommodate errors in the prediction or classification (Sec. 6.4).

4 KEYPOINT-DRIVEN DRAWING TOPOLOGY

In the next step of our algorithm, we infer the topology of the drawing, finding connections between the extracted keypoints. The geometry of all those connections will be refined in the final stage of the algorithm (Sec. 5). We outline the following requirements on such a set of paths (Fig. 6):

i *Keypoint-Driven.* Each path connects two keypoints; keypoint type should be respected.
ii *Junction resolution.* The paths should correctly traverse junctions.
iii *Efficient coverage.* Finally, subject to the other requirements, the set of paths should cover the drawing with minimal redundancy.

We address this problem in several steps. We first compute a frame field aligned to the drawing, which allows us to disambiguate directions around junctions (ii). Using this frame field, we form the initial drawing graph. This graph captures all the connectivity and details of the drawing, and covers the drawing completely (iii). We find a subgraph that connects all the keypoints and has the same coverage as the initial graph. Finally, we use this subgraph to extract the minimal set of paths between keypoints while keeping the coverage, thus satisfying all the above requirements.

4.1 Frame Field and Graph Construction

Frame Field Computation. The underlying structure necessary for both topology extraction (Sec. 4.2, 4.3) and subsequent PolyVector Flow (Sec. 5) is a frame field representing two dominant directions at each dark pixel of the image. At every point near a stroke the frame field contains at least one direction that is aligned with a nearby curve tangent. Next to T- and X-junctions, the frame field captures both intersecting curve directions (Fig. 7a). This structure allows us to resolve direction disambiguities around most junctions and sharp corners.

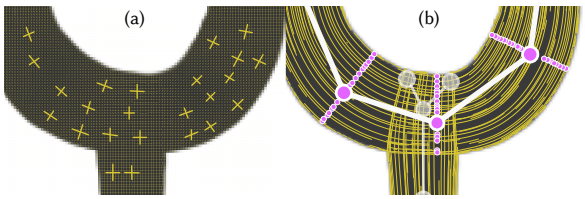


Fig. 7. Initial graph construction, using [Bessmeltsev and Solomon 2019]. For each dark pixel of the image ('narrow band'), we find a frame field (a, in yellow) aligned with the input drawing. We then trace it (b, in yellow) and form the initial graph by computing cross-sections of the traced curves and grouping the points within a cross-section (small pink circles) into a single graph vertex (larger circles). Graph vertices containing at least one shared traced curve in their cross-section get connected by an edge. Note that at T-junctions this forms two disconnected yet overlapping branches (b).

We use the frame field design of Bessmeltsev and Solomon [2019], which uses a PolyVector field representation [Diamanti et al. 2015]. Briefly, their variational approach combines three terms: alignment to the image tangent, smoothness, and a regularizer weakly biasing the field towards orthogonal frames. This amounts to solving a small number of linear systems, which we solve via the Conjugate Gradient method. For performance reasons, we compute the frame field only within a narrow band – all dark pixels of the image, determined via an intensity threshold on the input greyscale image ($0.35 \cdot \max(intensity)$). Please refer to the original paper for details [Bessmeltsev and Solomon 2019].

Graph Construction. Following the construction of Bessmeltsev and Solomon [2019], we trace the frame field at every dark pixel and form the graph vertices by locally grouping cross-sections of traced curves corresponding to the same direction of the frame field (Fig. 7). We assign the position of each vertex to the centroid of the cross-section, and add edges between vertices sharing at least one traced curve. This produces a graph where each vertex is centered with respect to the drawn stroke. Each vertex is also associated with a stroke width, computed as half of the diameter of the cross-section.

This graph captures all the connectivity and details of the original image, but typically contains numerous additional branches (Fig. 7b, vertical transparent branch) and is suboptimally positioned, e.g. around Y-junctions (Fig. 9b). We leverage graph's properties to extract the correct topology utilizing our computed keypoints.

4.2 Extracting Topology

We look for a set of paths over the graph connecting the predicted keypoints. To this end, we first identify *key vertices*, i.e. mapping the detected keypoints to the graph, then find a subgraph connecting those vertices and covering the original drawing, and finally convert

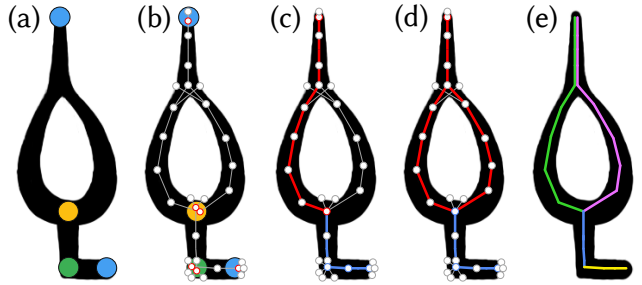
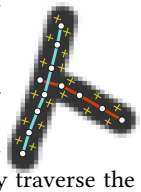


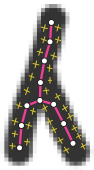
Fig. 8. Having computed the keypoints (a), we extract the topology in a few steps. First, we map each keypoint to one or two *key vertices* (red circles) (b) and move them to the keypoint locations. We then find Steiner trees, one for each connected component of the graph, connecting those vertices (c). Note that in this example the graph has two connected components, disconnected at the T-junction ((c), red and blue). We then further extend the Steiner trees forming subgraphs covering the whole drawing (d). Finally, we convert these subgraphs into a set of paths between the key vertices, while keeping the coverage (e).

this subgraph into a set of paths, taking into account the type of each keypoint.

Finding key vertices. To find paths, we need to associate each keypoint, i.e. a point in image space, with one or two key vertices on the graph (Fig. 8a). For each keypoint, we find the closest graph vertex containing the keypoint's pixel in its cross-section, as defined in Sec. 4.1.



While normally it is a single vertex, at T- and X-junctions, where two strokes intersect, the graph stucture contains two disconnected yet overlapping branches. In those cases, we then find two key vertices for a keypoint, i.e. one per branch (Fig. 8b, inset). This construction allows us to vectorize T- and X-junctions into two overlapping paths that correctly traverse the junctions while passing through the keypoint. Finally, having found the key vertices, we move each one to the associated keypoint, thus trusting its predicted location.



Finding Connecting Subgraphs. Our next step extracts subgraphs connecting key vertices and covering the drawing, thus leveraging the predicted keypoints to gracefully discard the topological artifacts of the original graph such as auxiliary branches, connections, and holes (Fig. 8c). We observe that our graph consists of multiple connected components, each corresponding to a single stroke in the drawing that can possibly split at Y-junctions (Fig. 8d, inset).

We thus find a subgraph connecting key vertices for each such component. We start by finding a tree of minimum length connecting all the key vertices within a component, also known as *Steiner Tree*. While in general the problem of computing Steiner trees is NP-hard, we use an efficient polynomial-time approximation via Minimum Spanning Trees [Kou et al. 1981].

Covering the Drawing. For each component, the computed Steiner tree connects the key vertices, but it may leave parts of the drawing uncovered since it breaks cycles by definition. We thus need to track

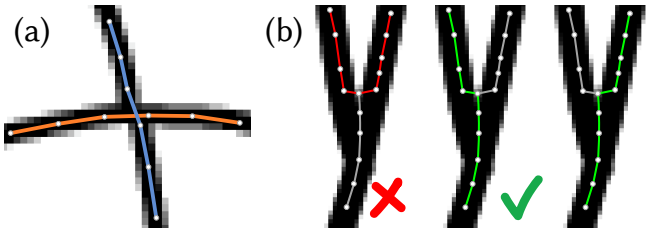


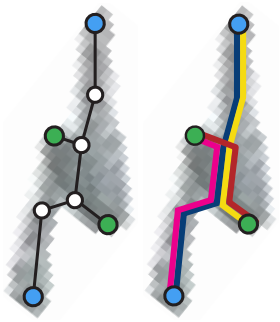
Fig. 9. Junction resolution: (a) X- and T-junctions, captured by the frame field, form disconnected components of the graph. Hence, all paths over this graph resolve those junctions correctly. To disambiguate Y-junctions (b), which appear as valence-3 vertices, we explicitly prohibit incorrect turns along the path.

the *coverage* of the subgraphs and extend it to cover the whole drawing. We build up the subgraph iteratively by successively including shortest paths between key vertices if they cover at least four new previously uncovered vertices.

A vertex is covered by the current subgraph if it is inside of a thickened version of the existing subgraph. For this, we first estimate for each graph vertex the corresponding stroke width in pixels, defined as a diameter of the vertex cross-section. To avoid local noise in the width estimation, we perform a single iteration of Laplacian smoothing of the widths on the graph via a single linear solve with $\lambda = 1$ [Desbrun et al. 1999]. Now, the thickened subgraph is constructed by linearly interpolating width between vertices, and considering the resulting region.

The shortest path selection is an instance of a *constrained shortest path problem*. In our implementation, we use a simple algorithm by AliAbdi et al. [2019]. We stop when no such path exists. For robustness, since keypoint detection (Sec. 3) may occasionally miss a keypoint, we additionally iteratively find and add longest paths between a key vertex and any other vertex if the path covers at least four new pixels.

4.3 Converting Subgraphs to Paths



In the last step of the topology extraction, we convert the subgraphs into a set of paths satisfying our requirements (Sec. 4). For each subgraph, we start by finding all *allowed* paths between the key vertices, i.e. correctly traversing the junctions. We then use mixed-integer optimization to select a minimal subset of those allowed paths having the same coverage as the subgraphs while respecting keypoint types.

Enumerating Allowed Paths. For each pair of key vertices within a connected component of our subgraph, we find all paths not containing any other key vertex, which correctly traverse the junctions (see inset). The graph construction (Sec. 4.1) facilitates 'automatic' correct resolution of X- and T-junctions (Fig. 9a). X- and T-junctions form non-connected, yet overlapping branches in the graph, hence

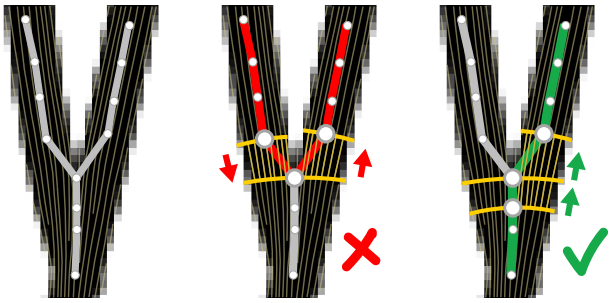


Fig. 10. To test if a path over the graph correctly traverses a Y-junction, we track the motion over the associated traced curves (background, brown). Precisely, if the average orientations of the shared traced curve motions (orange) are pointing in the opposite directions, we reject the path (center); otherwise, we accept it as allowed.

guaranteeing that all paths over the graph resolve those junctions correctly (Fig. 7b, 9a). Y-junctions, however, are not disambiguated by the frame field, and hence form valence-3 vertices in the graph. For each valence-3 vertex we therefore constrain the paths to only take the correct turns: i.e. from the 'stem' of Y to one of the branches, but not from one branch to another (Fig. 9b). This disambiguation allows PolyVector flow (Sec. 5) to correct the suboptimal initial locations of valence-3 vertices (see Fig. 11, where the flow moves valence-3 vertex to its correct location in the right corner).

Remembering that each graph vertex was computed via grouping samples on traced curves with the same root of the frame field, a path over the graph may be associated with simultaneous 'sliding' over the traced curves (Fig. 10). Thus, to robustly test if a path passing through a valence-3 vertex makes an allowed turn, we compare the average orientations of the motion over the traced curves: if they have a positive dot product, it is a motion between a stem of Y-junction and a branch, and therefore allowed; otherwise, we discard this path.

Extracting Final Paths. Finally, we select a minimal subset with the *same pixel coverage* as the subgraphs that respects the keypoint types. We start by associating with an allowed path p a binary variable $e_p \in \{0, 1\}$, signifying whether the path is selected (1) or not (0). Each path starts and ends with a key vertex, denoted as $\partial p = \{i, j\}$.

Keypoint types control key vertex valence in the final path set: an endpoint should have valence-1, a corner should have valence-2, and a junction should have valence-3 or higher. Using our binary variables, we can express the valence of a key vertex i in the final drawing as $v = \sum_{p|i\in\partial p} e_p$. This translates to a set of linear equality or inequality constraints $C_{valence}$. In practice, however, we replace all the equalities by inequalities, i.e. $v \geq 1$, $v \geq 2$, or $v \geq 3$ for endpoints, sharp corners, and junctions correspondingly to account for possible errors in the prediction of the classifier.

In order to select the subset of paths keeping the same coverage as the subgraphs, we first compute coverage of each allowed path. Then for each dark pixel (x, y), we enumerate all the paths covering it, forming a set $S(x, y) = \{p_1, \ldots, p_n\}$. We then add a linear inequality constraint that the pixel must be covered by at least one path: $\sum_{p\in S(x,y)} e_p \geq 1$. We denote the set of those constraints for all pixels as $C_{coverage}$.

Finally, we would like to select the paths most aligned with the drawing. To compute that, before performing this optimization, we align all allowed paths to frame field via PolyVector Flow (Sec. 5), and measure the energy of the final curve via Eq. 6a. We denote this value as c_p.

Bringing this all together, we solve the following mixed-integer programming problem:

$$\min_{e_p\in\{0,1\}} \sum_p c_p e_p$$
$$\text{s.t. } C_{valence}, C_{coverage}$$

We explicitly avoid infeasible optimization by adjusting the necessary valences. Precisely, if graph vertex valence is less than the valence predicted by the classifier, we adjust the prediction to its valence in the graph.

We solve this optimization problem using Gurobi solver [Gurobi Optimization 2021]. Note that while it is an instance of a NP-hard problem, this mixed integer problem is relatively small. We typically have 100 variables or fewer corresponding to all allowed paths. In our experiments it is always solved to optimality.

5 POLYVECTOR FLOW

We use the extracted set of paths over the graph as initial curves, centered by construction, to create the final vectorization. The key idea is to align the initial curves to the frame field while keeping them within the bounds of the drawn strokes. To this end, we formulate a novel geometric flow aligning the curve tangent at each point with one of the directions of the frame field (Fig. 11). The alignment is modeled as an *anisotropic* curvature flow, and an additional *barrier* term is used to keep the curves within the drawn strokes. This is the first use of such a flow for alignment to a frame field.

In the sections below, we detail our tailored functional and its L_2 gradient flow. A variational derivative is taken to arrive at the weak form, and this form is discretized in both time and space with the Finite Element Method (FEM).

5.1 Anisotropic Curvature Flows

The standard mean curvature flow is given by the parametric evolution equation $\dot{x} = -Hv$ where $\dot{x}$ is shorthand for $\frac{\partial x}{\partial t}$, H denotes mean curvature, and v is the unit normal to the hypersurface. In our setting of 1-dimensional curves in $\mathbb{R}^2$, this standard flow becomes $\dot{x} = -\kappa v$ where κ is just the usual 1D curvature, so we will drop the 'mean' and refer to just 'curvature flows'.

The standard curvature flow may be viewed as a curve-shortening flow, because it is also the L_2 gradient flow of the standard length functional. If we allow this length functional to vary depending on the direction of the curve tangent, we arrive at an *anisotropic* curvature flow. These have been previously studied in the literature (e.g., [Deckelnick et al. 2005; Dziuk 1999]) and we briefly define these models below, stating them in the setting of open curves with fixed endpoints.

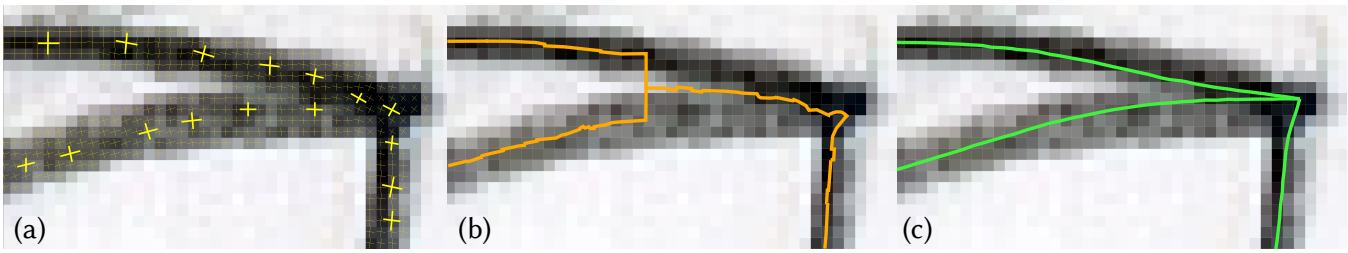


Fig. 11. The geometric flow we design aligns the initial curves (b) to the drawing tangents while keeping the curves centered (c). By using the frame field aligned with local tangents (a), our flow robustly refines curves around T-, X-, and Y-junctions.

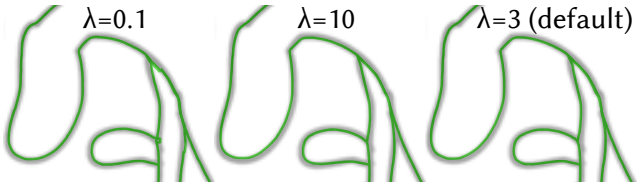


Fig. 12. Parameter λ controls smoothness of our result while still keeping the curves within the bounds of the narrow band.

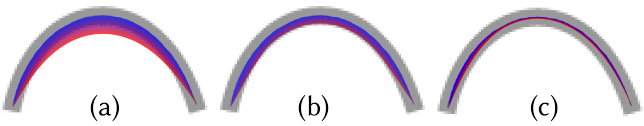


Fig. 13. Different flows visualized from blue (initial curve) to red (final), run for 50,000 iterations. Standard curve shortening flow (a) does not respect the boundary of the given stroke and gradually contracts any path into a straight line. With our barrier term added (b), curve shortening flow stays within the bounds of the drawn stroke but deviates from the centerline and drawn tangents. Our flow (c) aligns the curve to the local tangents while keeping the curve smooth and within the bounds of the narrow band ($\lambda = 0.2$).

Consider an L_2 gradient flow of an anisotropic length functional:

$$\begin{aligned} \min_x \quad & \int_\Gamma \gamma(v)\, dl, \\ \text{s.t.} \quad & x(0) = A, x(L) = B \end{aligned}$$

where $x(\theta), \theta \in [0, L]$ parameterizes the curve Γ, v is its pointwise unit normal, and $\gamma : \mathbb{R}^2 \to \mathbb{R}$ is an anisotropic length function that is positive and 1-homogeneous:

(a) *Positiveness*: $\gamma(v) > 0$, if $v \neq 0$
(b) *Homogeneity of degree 1*:

$$\gamma(\lambda v) = |\lambda|\gamma(v), \text{for all } \lambda \neq 0, v \neq 0 \tag{4}$$

Note that γ accomodates non-unit arguments so that it may be used with non-arclength parametrizations. The standard curvature flow results from the standard length functional $\gamma(v) = |v|$.

In the following sections, we generalize this model so that it may smooth our curves and promote alignment to the frame field. In particular, we consider an anisotropy function $\gamma(x, v) : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$ that also varies positionally (i.e., in x) to account for this changing frame field.

5.2 Our Anisotropy Function

To measure alignment with the frame field and construct an appropriate $\gamma(x, v)$, we use the PolyVector field representation [Diamanti et al. 2015]. Our frame field, described in Sec. 4.1, is represented as a pair of complex scalar fields $c_0(x), c_2(x) \in \mathbb{C}$ defined over the dark pixels of the image. At each pixel those scalars define a quartic polynomial, whose roots $\pm u, \pm v \in \mathbb{C}$, identifying image plane with a complex plane, are the two dominant directions in the drawing:

$$f(x, z) := z^4 + c_2(x)z^2 + c_0(x) = (z^2 - u(x)^2)(z^2 - v(x)^2)$$

In particular, consider a curve $x(\theta), \theta \in [0, L]$ and its normalized tangent $\tau = \frac{x'(\theta)}{\|x'(\theta)\|}$. The norm $|f(x, \tau)|$ is locally minimized when τ aligns with one of the frame field directions.

Based on this observation, we may construct the following positive and 1-homogeneous anisotropy function:

$$\gamma(x, v) := \|v\| \left(\left| f\left(x, i\frac{v}{\|v\|}\right)\right|^2 + b(x) \right),$$

Above, multiplication by i is a rotation of the normal to the tangent.

The *barrier* function $b(x)$ keeps the curve inside the bounds of the drawn strokes while also acting as a smoothness regularizer: it is nearly zero within the bounds of the drawn strokes, increasing outside the bounds. We represent it via a sum of Gaussians centered at the graph vertex locations $p(v) \in \mathbb{R}^2$ for all the vertices v in the path. We set the standard deviation $\sigma(v)$ using the per-vertex stroke width $w(v)$ (Sec. 4.2) using 3σ-rule: $\sigma(v) = w(v)/6$.

$$b(x) = \max\left(1 - \sum_v \exp\left(-\frac{\|x - p(v)\|^2}{2\sigma(v)^2}\right), 0\right) + \lambda, \tag{5}$$

where $\lambda > 0$ is a regularizing parameter that pushes the flow towards the standard curve-shortening flow (Fig. 12, Fig. 13).

5.3 Variational Derivative and Weak Form

Our generalized model considers the L_2 gradient flow of the following anisotropic position-varying length functional:

$$\min_x \quad \int_\Gamma \gamma(x, v) dl \tag{6a}$$

$$\text{s.t.} \quad x(0) = A, x(L) = B. \tag{6b}$$

Our arguments below closely follow those in Dziuk [1999], Deckelnick et al. [2005] so we adopt their notation, and summarize them briefly, making it specific to 1D curves embedded in $\mathbb{R}^2$. There are important differences arising due to our position-varying γ and the setting of open curves with fixed endpoints, so we are sure to highlight these as we proceed. We begin with calculation of the variational derivative of our functional, c.f. Lemma 8.2 in Deckelnick et al. [2005].

Proposition 5.1. *For a normal variation described by $\phi : \Gamma \to \mathbb{R}$ such that $x(\theta, t) = x(\theta, 0) + t\phi(x(\theta, 0))\nu(x(\theta, 0))$ and $\phi(A) = \phi(B) = 0$, we have:*

$$\frac{d}{dt}\left(\int_{\Gamma(t)} \gamma(x, \nu)\, dl\right)\Big|_{t=0} = \int_\Gamma \nabla_\Gamma \cdot D\gamma(\nu)\phi + \frac{\partial \gamma}{\partial \nu}\phi\, dl \qquad (7)$$

where ∇_Γ refers to the gradient projected onto Γ, $D\gamma$ refers to the gradient with respect to the normal argument ν, and $\frac{\partial \gamma}{\partial \nu}$ refers to the partial derivative in the normal direction ν.

The key difference in our model is the presence of the second term above, as in the usual setting of anisotropic curvature flows $\frac{\partial \gamma}{\partial \nu} = 0$. This proposition follows from an application of a transport theorem and integration-by-parts. A more detailed argument is given in Appendix A.1.

The above proposition shows that the gradient flow is of the form:

$$\dot{x} = -\left(\nabla_\Gamma \cdot D\gamma(\nu) + \frac{\partial \gamma}{\partial \nu}\right)\nu.$$

The weak form that we will discretize results from integrating the dot product against a vector variation $\psi : [0, L] \to \mathbb{R}^2$ for which $\psi(0) = \psi(1) = 0$. The existence of the second-order derivative in the first term creates issues for discretization, so another integration-by-parts results in the following equivalence.

Lemma 5.2. *Given a test function ψ as described above,*

$$\int_{\Gamma(t)} \nabla_\Gamma \cdot D\gamma(\nu)(\nu \cdot \psi)\, dl = \int_0^L D\gamma(x'(\theta)^\perp) \cdot (\psi'(\theta)^\perp)\, d\theta \qquad (8)$$

where $(a, b)^\perp = (-b, a)$, and $x'(\theta) = \frac{\partial x}{\partial \theta}$ and $\psi'(\theta) = \frac{\partial \psi}{\partial \theta}$ denote tangents to their respective curves.

A proof sketch for this lemma is given in Appendix A.1. The final weak form that we will discretize is the following:

$$\underbrace{\int_\Gamma \dot{x} \cdot \psi\, dl}_{\text{first term}} + \underbrace{\int_0^L D\gamma(x'(\theta)^\perp) \cdot (\psi'(\theta)^\perp)\, d\theta}_{\text{second term}} + \underbrace{\int_\Gamma \frac{\partial \gamma}{\partial \nu}(\nu \cdot \psi)\, dl}_{\text{third term}} = 0 \qquad (9)$$

5.4 Discretization

Using first-order Finite Element Method (FEM), we discretize our curve $x(\theta, t) = \sum_{i=0}^M x_i(t)\phi_i(\theta)$ with hat functions ϕ_i in the parameter domain θ. With $[0, L]$ divided into M subintervals of length L/M with endpoints $\theta_j = jL/M$, these hat functions are linear over the subintervals and satisfy $\phi_i(\theta_j) = \delta_{ij}$. The discretized curve forms a polyline interpolating the vertices $x_i(t) \in \mathbb{R}^2$ which vary in time.

In this setting, for test functions, we consider a basis of discrete variations $\psi_{i,k} = \phi_i(\theta)e^k$, where e^k is a basis vector. $\psi_{i,k}$ describes the discrete variation that shifts vertex i in the kth basis direction. We consider the weak form for each of these $2(M+1)$ basis variations.

Considering the first term in Eq. 9, we get:

$$\frac{1}{6}\dot{x}_{j-1}q_j + \frac{1}{3}\dot{x}_j(q_j + q_{j+1}) + \frac{1}{6}\dot{x}_{j+1}q_{j+1},$$

where $q_j = \|x_j - x_{j-1}\|$. The constants arise from standard integrals of 1D hat functions.

For the second term of Eq. 9 note that $(\psi'_{j,k})^\perp = \phi'_j(\theta)e_k^\perp \neq 0$ only on $[\theta_{j-1}, \theta_{j+1}]$:

$$e_k \cdot \left((D\gamma(x_{j+1}^\perp - x_j^\perp))^\perp - D\gamma(x_j^\perp - x_{j-1}^\perp))^\perp\right),$$

since $a \cdot b^\perp = -a^\perp \cdot b$.

Finally, denoting discretized tangents as $\tau_j = \frac{x_j - x_{j-1}}{q_j}$ and normals as $\nu_j = \tau_j^\perp$, the third term in Eq. 9 becomes

$$\frac{1}{2}e_k \cdot \left(\nu_j\left(\nabla_x \gamma_j \cdot x_j^\perp - x_{j-1}^\perp\right) + \nu_{j+1}\left(\nabla_x \gamma_{j+1} \cdot x_{j+1}^\perp - x_j^\perp\right)\right),$$

where we put $\nabla_x \gamma(x, \nu)|_{j-1}^j \approx \nabla_x \gamma(\frac{x_j + x_{j-1}}{2}, \frac{x_j^\perp - x_{j-1}^\perp}{|x_j^\perp - x_{j-1}^\perp|}) \stackrel{\text{def}}{=} \nabla_x \gamma_j$.

Now let us finalize our discretization. We denote $F_j = \langle \nabla_x \gamma_j, \nu_j \rangle$, and further $g_j = \gamma(\nu_j), g'_j = \langle \gamma'(\nu_j), \tau_j \rangle$, and, for convenience, $X = x_1 - ix_2, x^\perp = -iX, G = g + ig'$. Finally, we use forward Euler integration of the flow using δ as the time step. After straightforward calculations, we can write out the final tridiagonal complex linear system:

$$X_{j-1}^n\left(\frac{iF_j}{2} - \frac{G_j}{q_j}\right) + X_j^n\left(\frac{1}{2\delta}(q_j + q_{j+1}) + \frac{G_j}{q_j} + \frac{G_{j+1}}{q_{j+1}} - \frac{1}{2}iF_j + \frac{1}{2}iF_{j+1}\right) - X_{j+1}^n\left(\frac{G_{j+1}}{q_{j+1}} + \frac{iF_{j+1}}{2}\right) = \frac{1}{2\delta}X_j^{n-1}(q_{j+1} + q_j). \qquad (10)$$

Eq. 10 is the final discretized version of our PolyVector Flow. Performing an iteration of the flow amounts to solving this linear system, which we implement via an LU factorization.

Adaptive Resampling. We observe a known artifact of standard curve shortening flow: in some cases, our flow may develop zero-length edges next to long edges. To address that, we adapt the standard approach of de Goes et al. [2008]: we maintain edge length to be under one pixel, resampling the curve as needed. Additionally, following Deckelnick et al. [2005], we regularize the length element computation: $q_j = \sqrt{\|x_j - x_{j-1}\|^2 + \varepsilon}$, where $\varepsilon = 0.2$ in our experiments.

Time Step. We use base time step $\delta = 5 \cdot 10^{-3}$, which we scale by the inverse shortest-path distance to the high-valence (i.e. ≥ 3) vertices on our graph, since we have noted that the geometry of the initial graph is more reliable far from high-valence vertices. We run our flow for a fixed number of 1500 iterations.

6 VALIDATION AND RESULTS

6.1 Qualitative Evaluation

We have automatically generated a number of vectorizations for line drawings of different line styles and levels of noise (Fig. 1, 15,

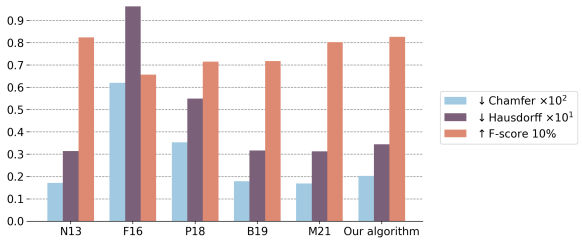


Fig. 14. We used benchmark [Yan et al. 2020] to measure mean Chamfer and Hausdorff distances (lower is better), as well as F_1 score 10% (higher is better). By these metrics, our results are comparable to previous work (columns: [Noris et al. 2013], [Favreau et al. 2016], [Parakkat et al. 2018], [Bessmeltsev and Solomon 2019], [Mo et al. 2021], ours). We note, however, as indicated by previous work [Mo et al. 2021], that these metrics do not measure how useful or correct the vectorizations are, instead measuring how well they cover dark pixels of the input bitmap.

17, 18). We used automatic contrast adjustment in Adobe Photoshop as a preprocessing step. The drawings include digital clean drawings ('sheriff, 'muten', and 'dracolion' in Fig. 15), as well as complex noisy scanned drawings ('running dog' in Fig. 1, 'dog', 'rabbit', 'donkey', and 'leaf' in Fig. 17, 'elephant', 'turkey', 'banana tree', 'puppy', 'kitten', 'hippo' in Fig. 18).

6.2 Comparison to Prior Art

We compare our vectorization system with the recent methods [Bessmeltsev and Solomon 2019; Mo et al. 2021; Stanko et al. 2020], as well as older systems [Favreau et al. 2016; Noris et al. 2013] and Adobe Illustrator for completeness. We ran the methods by Noris et al. [2013] with the default parameters, and Favreau et al. [2016] over a set of parameter values [1]. For a fair comparison, even though Stanko et al. [2020] can potentially accept a user-defined scaling image, we run it in the automatic mode using a few different constant values for scale and narrow band size [2]. For all those methods we used the binaries or code provided by the authors. Finally, we directly compare with the results of Mo et al. [2021] as presented in their paper and on their website. We ran our method with the default parameters on all inputs. We present some of the comparison results in Fig. 15 and Fig. 17, the rest are presented in the supplementary materials.

On the clean digital drawings, our method produces results comparable to ones of Noris et al. [2013], Bessmeltsev and Solomon [2019], Stanko et al. [2020]. Even for those, our resolution of zigzags, in particular, is more robust (see e.g. 'fire' on Fig. 15). As noted by the authors, Noris et al. [2013] is unstable under any noise in the input image and thus suffers from numerous artifacts for scanned drawings (see Supplementary).

As compared to Bessmeltsev and Solomon [2019], our method has more robust junction and topology detection (see Fig. 3, 17, and Supplementary). Bessmeltsev and Solomon [2019] use the initial graph for junction and endpoint positions, leading to spurious branches and smoothed out corners. Our PolyVector flow formulation allows us to be more robust especially around corners and Y-junctions (see e.g. zigzags on the donkey or rabbit, Y-junction in the dog in Fig. 17).

The method of Stanko et al. [2020] is more targeted towards simultaneous simplification and vectorization, and hence tends to simplify significant features even for non-ambiguous regions (see zooms in Fig. 17). Our method contains fewer topological artifacts due to our keypoint-driven topology extraction instead of their parameterization, such as spurious or absent connections and erroneous branches.

We compared to the recent work by Mo et al. [2021] in the supplementary. As demonstrated by Fig. 4, their method does not target reconstructing correct topology, making it problematic to use for graphical design, animation, or most downstream applications.

Since our vectorization method targets precise vectorization as opposed to simplification, we see the graph optimization algorithm by Favreau et al. [2016] as complementary to ours. For precise vectorization, however, the method by [Favreau et al. 2016] tends to oversimplify the results and deviates from the drawing (see Supplementary).

If simplification is needed for vectorization of rough sketches, our method can be used in conjunction with raster-based simplification systems [Simo-Serra et al. 2018, 2016; Xu et al. 2019] as a preprocessing step (Fig. 16). We see this rough drawing simplification task as separate and not a focus of our current work. Alternatively, one may use vector-based simplification methods [Liu et al. 2018] on our results.

6.3 Quantitative Analysis

We used a recent benchmark to quantitatively validate our method [Yan et al. 2020], and, as suggested by the benchmark, measured the Chamfer and Hausdorff distances, as well as F_1-score (10%) between our algorithmic results and the dark pixels in the input bitmaps. As discussed by previous work [Mo et al. 2021], these distances do not correspond to how useful, editable, or correct the vectorization result is, and are more indicative of how well the vectorization covers the input bitmap. In terms of these metrics, however, our method is comparable with the other state-of-the-art methods (Fig. 14). Note that to compare to Mo et al. [2021], we use the Chamfer distance from their paper, and compute Hausdorff distance and F-score using the trained model provided on their website.

6.4 Robustness, Input and Parameter Sensitivity

Our method is robust to changes in the input bitmap, resolution (Fig. 19), and input stroke widths (for our inputs, widths range from 1px to 12px) as our keypoint detector is robust under those conditions, and the core of our method, i.e. PolyVector Flow, is variational in nature. Our method has very few tunable parameters. We demonstrate the effect of changing the parameter λ in our PolyVector Flow formulation in Fig. 12.

While accuracies or our keypoint detection and classification are high (see Sec. 3.1), both scores can be improved by collecting a larger dataset of real drawings. However, we designed our algorithm to gracefully alleviate prediction/classification errors.

[1] {maxNumOpenCurves = 0, minLengthOpenCurves=30, minRegionSize=7} and {maxNumOpenCurves = 30, minLengthOpenCurves=5, minRegionSize=3}. λ = 0.5

[2] -n 0.3, 0.5, 0.8 and -s 0.5, 1.0, 2.0

Fig. 15. On clean digital line drawings, our method produces comparable results to the previous work [Bessmeltsev and Solomon 2019; Noris et al. 2013].

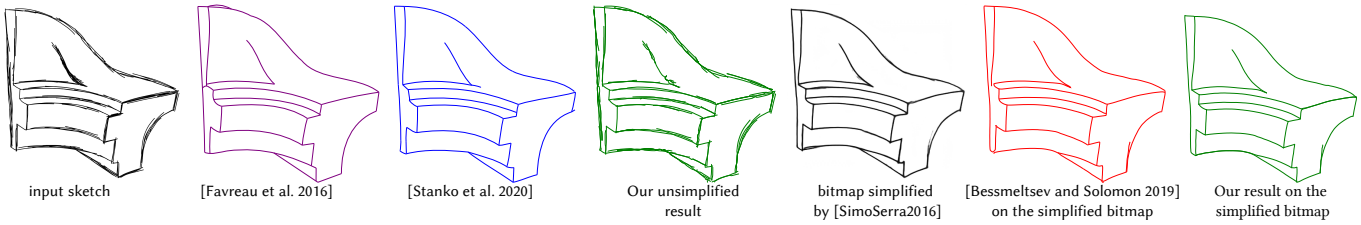


Fig. 16. Comparison of our method with alternative approaches on a rough bitmap sketch. Since simplification is not the focus of our method, we preprocess the input using a bitmap-to-bitmap simplification method [Simo-Serra et al. 2016]

The detector may miss keypoints (false negative), especially at stroke endings that smoothly fade out (hippo's neck, see Supplementary). We use pixel coverage (Sec. 4.2), fixing most of these errors. The detector may also produce additional points (false positive), often close to the centerline, resulting in a marginally increased curve complexity but not decreased quality. Rare incorrect off-center points might introduce local geometric or topological artifacts (e.g. dog tail, right, Fig. 17).

We accommodate *classification* errors by using inequalities in extracting final paths (Sec. 4.3), retaining pixel coverage, and avoiding infeasible optimization by adjusting the necessary valences (Sec. 4.2).

In rare cases, for areas already covered by other curves, classification errors might lead to short missing paths in the final vectorization (see Fig. 19, near elephant's eye). The detector may also produce additional points (false positive detection), often close to the centerline, resulting in a marginally increased curve complexity

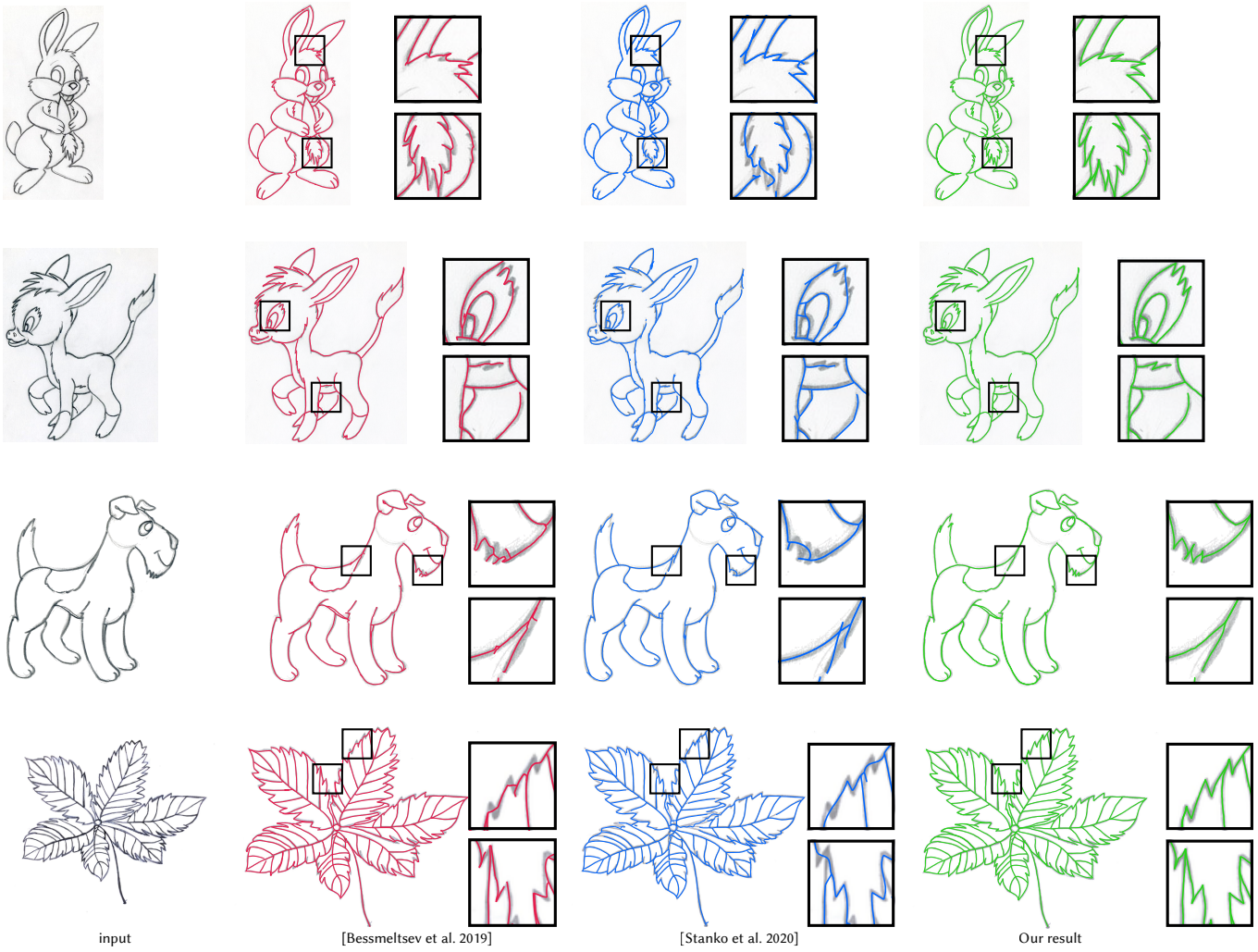


Fig. 17. Compared to the previous approaches based on frame fields [Bessmeltsev and Solomon 2019; Stanko et al. 2020], our method more robustly captures sharp corners and junctions. Input images 'rabbit', 'donkey', 'dog' are from www.easy-drawings-and-sketches.com ©Ivan Huska.

but not decreased quality. Rare incorrect off-center points might introduce local geometric or topological artifacts (e.g. dog tail, Fig. 17).

6.5 Runtime

On an Intel(R) Core(TM) i9-9900K CPU @3.60 GHz with 128GB RAM, our unoptimized implementation usually takes around a minute to vectorize medium-to-high resolution images, and is comparable to the other vectorization algorithms (Table 1). The bottleneck of our algorithm is currently solving the constrained shortest path problem (Sec. 4.2). We used the same parameters for all those images.

Ablation Study. To illustrate the contribution of each stage of our algorithm, we perform an ablation study (Fig. 20).

6.6 Limitations and Future Work

Similarly to most previous work, our method does not address the problem of shaded regions (see e.g. nose of the kitten in Fig. 18); we leave this to future work. We also share the dependency of many other methods (e.g. [Bessmeltsev and Solomon 2019; Stanko et al. 2020], Adobe Illustrator) on the background-foreground threshold, which might be problematic for drawings with a very noisy background.

In future work, it might be also interesting to consider semantic cues and prior knowledge to make better informed connectivity decisions (see e.g. blow-up of the eyebrow, Fig. 15).

7 CONCLUSION

We have presented a novel method for automatically vectorizing bitmap images, based on a novel PolyVector Flow and a deep learning–based keypoint detection network. As we demonstrate, it reliably

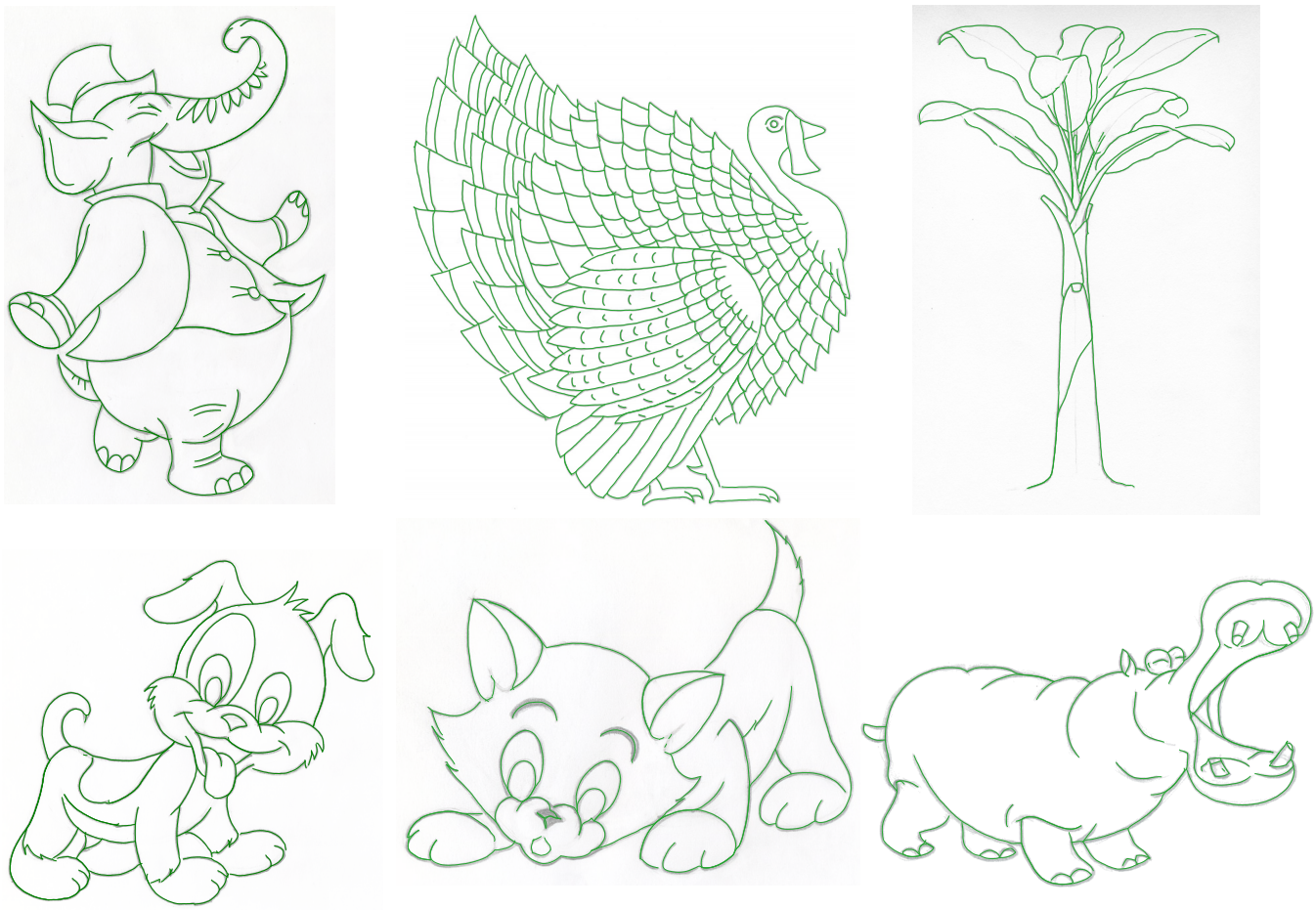


Fig. 18. A gallery of additional results. Input images from www.easy-drawings-and-sketches.com ©Ivan Huska.

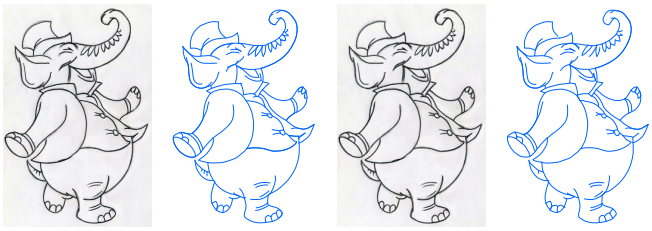


Fig. 19. Our system is robust to resolution changes. Left image (original) is 500×753 pixels, the right one is 250×377 pixels. While the quality of the result degrades with decreasing resolution, the directions and positions of most keypoints are stable.

and efficiently resolves topological and geometric ambiguities around sharp corners, junctions, and stroke endpoints.

Our key contribution, PolyVector Flow, can be used as a drop-in geometry refinement method for other vectorization algorithms, be adapted to quad-meshing algorithms, or used in medical/satellite imaging applications, e.g. road network reconstruction.

Table 1. Time performance in seconds, left to right: [Noris et al. 2013], [Favreau et al. 2016], [Bessmeltsev and Solomon 2019], [Stanko et al. 2020].

Input	N13	F16	B19	S20	Ours
Running dog	15	108	18	70	52
Sheriff	14	74	50	120	55
Muten	10	105	28	62	63
Dracolion	12	160	26	110	45
Dog	25	106	83	50	64
Rabbit	17	94	33	60	43
Donkey	24	100	89	33	69
Leaf	10	55	20	96	73
Elephant	23	130	50	52	70
Turkey	25	86	58	504	150
Banana tree	12	108	22	84	160
Puppy	18	110	35	55	52
Kitten	25	100	75	37	68
Hippo	17	75	44	22	44

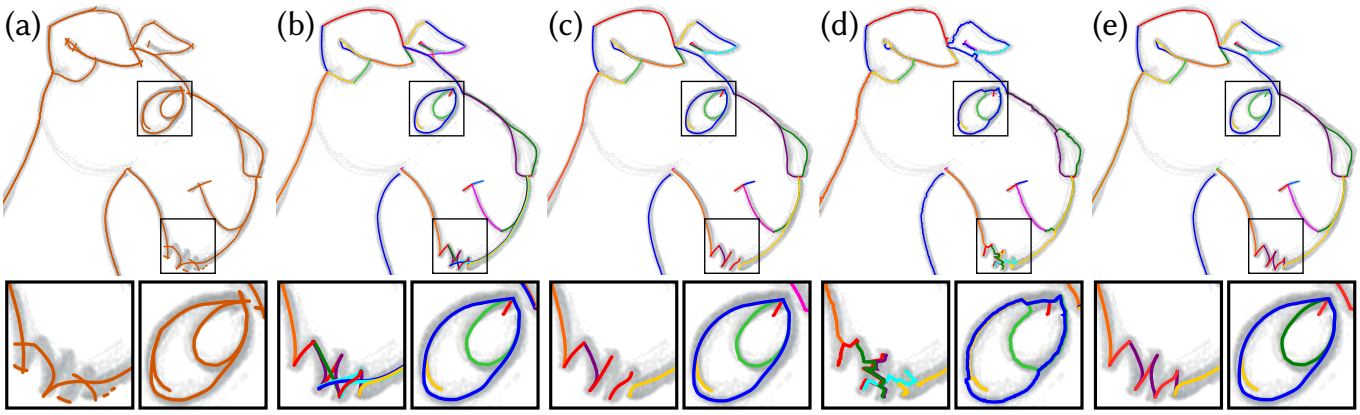


Fig. 20. Ablation study: (a) disabling keypoint extraction (Sec. 3), using only the graph and coverage to infer endpoints and junctions; (b) disabling 'extracting final paths' optimization (Sec. 4.2); (c) disabling valence constraints (Sec. 4.2); (d) disabling Polyvector Flow (Sec. 5) leads to both incorrect topology and geometry; (e) Our result.

Our system can be immediately useful for artists and engineers alike, robustly creating high-quality tracings even in the presence of noise.

ACKNOWLEDGMENTS

We would like to thank Dennis Gaitsgory and Gerhard Dziuk for insightful early discussions. We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC) under Grant No.: RGPIN-2019-05097 ("Creating Virtual Shapes via Intuitive Input") and the Fonds de recherche du Québec - Nature et technologies (FRQNT) under Grant No.: 2020-NC-270087.

REFERENCES

Amin AliAbdi, Ali Mohades, and Mansoor Davoodi Monfared. 2019. Constrained shortest path problems in bi-colored graphs: a label-setting approach. *GeoInformatica* (12 2019), 1–19. https://doi.org/10.1007/s10707-019-00385-8

Bin Bao and Hongbo Fu. 2012. Vectorizing line drawings with near-constant line width. In *IEEE Int. Conf. on Image Processing*. 805–808.

Alexandra Bartolo, Kenneth P. Camilleri, Simon G. Fabri, Jonathan C. Borg, and Philip J. Farrugia. 2007. Scribbles to Vectors: Preparation of Scribble Drawings for CAD Interpretation. *Proceedings of the 4th Eurographics Workshop on Sketch-based Interfaces and Modeling (SBIM '07)* (2007), 123–130. https://doi.org/10.1145/1384429.1384456

Mikhail Bessmeltsev and Justin Solomon. 2019. Vectorization of line drawings via polyvector fields. *ACM Transactions on Graphics* 38, 1 (2019). https://doi.org/10.1145/3202661 arXiv:1801.01922

Pengbo Bo, Gongning Luo, and Kuanquan Wang. 2016. A graph-based method for fitting planar B-spline curves with intersections. *Journal of Computational Design and Engineering* 3, 1 (2016), 14 – 23. https://doi.org/10.1016/j.jcde.2015.05.001

Alexander I. Bobenko and Peter Schröder. 2005. Discrete Willmore Flow. In *Proceedings of the Third Eurographics Symposium on Geometry Processing* (Vienna, Austria) *(SGP '05)*. Eurographics Association, Goslar, DEU, 101–es.

Alexandre Carlier, Martin Danelljan, Alexandre Alahi, and Radu Timofte. 2020. DeepSVG: A Hierarchical Generative Network for Vector Graphics Animation. arXiv:2007.11301 [cs.CV]

V. Caselles, R. Kimmel, and G. Sapiro. 1995. Geodesic active contours. In *Proceedings of IEEE International Conference on Computer Vision*. 694–699.

Dengfeng Chai, Wolfgang Förstner, and Florent Lafarge. 2013. Recovering Line-Networks in Images by Junction-Point Processes. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*. 1894–1901.

Jiazhou Chen, Mengqi Du, Xujia Qin, and Yongwei Miao. 2018. An improved topology extraction approach for vectorization of sketchy line drawings. *Visual Computer* 34, 12 (2018), 1633–1644. https://doi.org/10.1007/s00371-018-1549-z

Jiazhou Chen, Gaël Guennebaud, Pascal Barla, and Xavier Granier. 2013. Non-Oriented MLS Gradient Fields. *Computer Graphics Forum* 32, 8 (2013), 98–109.

JiaZhou Chen, Qi Lei, YongWei Miao, and QunSheng Peng. 2015. Vectorization of line drawing image based on junction analysis. *Science China Information Sciences* 58, 7 (2015), 1–14. https://doi.org/10.1007/s11432-014-5246-x

U. Clarenz, U. Diewald, G. Dziuk, M. Rumpf, and R. Rusu. 2004. A Finite Element Method for Surface Restoration with Smooth Boundary Conditions. *Comput. Aided Geom. Des.* 21, 5 (May 2004), 427–445. https://doi.org/10.1016/j.cagd.2004.02.004

Ulrich Clarenz, Gerhard Dziuk, and Martin Rumpf. 2003. On generalized mean curvature flow in surface processing. In *Geometric Analysis and Nonlinear Partial Differential Equations*. Springer.

Keenan Crane, Ulrich Pinkall, and Peter Schröder. 2013. Robust Fairing via Conformal Curvature Flow. 32, 4, Article 61 (July 2013), 10 pages. https://doi.org/10.1145/2461912.2461986

Ayan Das, Yongxin Yang, Timothy Hospedales, Tao Xiang, and Yi-Zhe Song. 2021. Cloud2Curve: Generation and Vectorization of Parametric Sketches. i (2021). arXiv:2103.15536

Fernando de Goes, Siome Goldenstein, and Luiz Velho. 2008. A simple and flexible framework to adapt dynamic meshes. *Computers & Graphics* 32, 2 (2008), 141 – 148. http://www.sciencedirect.com/science/article/pii/S0097849308000174

Klaus Deckelnick, Gerhard Dziuk, and Charles M. Elliott. 2005. Computation of geometric partial differential equations and mean curvature flow. *Acta Numerica* 14 (2005), 139–232. https://doi.org/10.1017/S0962492904000224

Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H. Barr. 1999. Implicit Fairing of Irregular Meshes Using Diffusion and Curvature Flow. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '99)*. ACM Press/Addison-Wesley Publishing Co., USA, 317–324. https://doi.org/10.1145/311535.311576

Olga Diamanti, Amir Vaxman, Daniele Panozzo, and Olga Sorkine-Hornung. 2015. Integrable PolyVector Fields. *ACM Trans. Graph.* 34, 4, Article 38 (July 2015), 12 pages. https://doi.org/10.1145/2766906

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 Transaction on Graphics* 39, 4 (2020). https://doi.org/10.1145/3386569.3392401

Luca Donati, Simone Cesano, and Andrea Prati. 2017. An Accurate System for Fashion Hand-Drawn Sketches Vectorization. In *The IEEE International Conference on Computer Vision (ICCV)*. 2280–2286. https://doi.org/10.1109/ICCVW.2017.268

Luca Donati, Simone Cesano, and Andrea Prati. 2019. A complete hand-drawn sketch vectorization framework. *Multimedia Tools and Applications* 78, 14 (2019), 19083–19113. https://doi.org/10.1007/s11042-019-7311-3

Gerhard Dziuk. 1999. Discrete Anisotropic Curve Shortening Flow. *SIAM J. Numer. Anal.* 36, 6 (1999), 1808–1830.

Gerhard Dziuk and Charles M. Elliott. 2013. Finite element methods for surface PDEs. *Acta Numerica* 22 (2013), 289–396. https://doi.org/10.1017/S0962492913000056

Vage Egiazarian, Oleg Voynov, Alexey Artemov, Denis Volkhonskiy, Aleksandr Safin, Maria Taktasheva, Denis Zorin, and Evgeny Burnaev. 2020. Deep Vectorization of Technical Drawings. *Lecture Notes in Computer Science* (2020), 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 (SIGGRAPH Conference Proceedings)* 35, 4, Article 120 (July 2016), 10 pages.

https://doi.org/10.1145/2897824.2925946
Jun Gao, Chengcheng Tang, Vignesh Ganapathi-Subramanian, Jiahui Huang, Hao Su, and Leonidas J. Guibas. 2019. DeepSpline: Data-Driven Reconstruction of Parametric Curves and Surfaces. *CoRR* abs/1901.03781 (2019). arXiv:1901.03781
Songwei Ge, Vedanuj Goswami, C. Lawrence Zitnick, and Devi Parikh. 2021. Creative Sketch Generation. arXiv:2011.10039 [cs.CV]
Fernando de Goes, David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. 2011. An Optimal Transport Approach to Robust Reconstruction and Simplification of 2D Shapes. *Computer Graphics Forum* 30, 5 (July 2011), 1593–1602. https://doi.org/10.1111/j.1467-8659.2011.02033.x
Anthony Gruber and Eugenio Aulisa. 2020. Computational P-Willmore Flow with Conformal Penalty. *ACM Trans. Graph.* 39, 5, Article 161 (Aug. 2020), 16 pages. https://doi.org/10.1145/3369387
Yulia Gryaditskaya, Felix Hahnlein, Chenxi Liu, Alla Sheffer, and Adrien Bousseau. 2020. Lifting Freehand Concept Sketches into 3D. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)* 39 (12 2020).
Yi Guo, Zhuming Zhang, Chu Han, Wenbo Hu, Chengze Li, and Tien-Tsin Wong. 2019. Deep Line Drawing Vectorization via Line Subdivision and Topology Reconstruction. In *Computer Graphics Forum*, Vol. 38. Wiley Online Library, 81–90.
LLC Gurobi Optimization. 2021. Gurobi Optimizer Reference Manual. http://www.gurobi.com
Igor Guskov, Wim Sweldens, and Peter Schröder. 1999. Multiresolution Signal Processing for Meshes. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '99)*. ACM Press/Addison-Wesley Publishing Co., USA, 325–334. https://doi.org/10.1145/311535.311577
David Ha and Douglas Eck. 2017. A Neural Representation of Sketch Drawings. *ArXiv e-prints* (April 2017). arXiv:1704.03477 [cs.NE]
Xavier Hilaire and Karl Tombre. 2006. Robust and accurate vectorization of line drawings. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28, 6 (June 2006), 890–904. https://doi.org/10.1109/TPAMI.2006.127
Shayan Hoshyari, Edoardo Alberto Dominici, Alla Sheffer, Nathan Carr, Duygu Ceylan, Zhaowen Wang, and I-Chao Shen. 2018. Perception-Driven Semi-Structured Boundary Vectorization. *ACM Transaction on Graphics* 37, 4 (2018). https://doi.org/10.1145/3197517.3201312
Sadashige Ishida, Masafumi Yamamoto, Ryoichi Ando, and Toshiya Hachisuka. 2017. A Hyperbolic Geometric Flow for Evolving Films and Foams. *ACM Trans. Graph.* 36, 6, Article 199 (Nov. 2017), 11 pages. https://doi.org/10.1145/3130800.3130835
Xie Jun, Winnemöller Holger, Li Wilmot, and Schiller Stephen. 2017. Interactive Vectorization. In *ACM SIGCHI*. 6695–6705.
Henry Kang, Seungyong Lee, and Charles K. Chui. 2007. Coherent Line Drawing. In *Proceedings of the 5th International Symposium on Non-photorealistic Animation and Rendering* (San Diego, California) *(NPAR '07)*. ACM, New York, NY, USA, 43–50. https://doi.org/10.1145/1274871.1274878
S. Kichenassamy, A. Kumar, P. Olver, A. Tannenbaum, and A. Yezzi. 1995. Gradient flows and geometric active contour models. In *Proceedings of IEEE International Conference on Computer Vision*. 810–815. https://doi.org/10.1109/ICCV.1995.466855
Byungsoo Kim, Oliver Wang, A. Cengiz Öztireli, and Markus Gross. 2018. Semantic Segmentation for Line Drawing Vectorization Using Neural Networks. *Computer Graphics Forum (Proc. Eurographics)* 37, 2 (2018), 329–338.
Leif Kobbelt. 2000. Discrete fairing and variational subdivision for freeform surface design. *The Visual Computer* 16 (2000).
Leif Kobbelt, Swen Campagna, Jens Vorsatz, and Hans-Peter Seidel. 1998. Interactive Multi-Resolution Modeling on Arbitrary Meshes. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '98)*. Association for Computing Machinery, New York, NY, USA, 105–114. https://doi.org/10.1145/280814.280831
L. Kou, G. Markowsky, and L. Berman. 1981. A Fast Algorithm for Steiner Trees. *Acta Inf.* 15, 2 (June 1981), 141–145. https://doi.org/10.1007/BF00288961
Gregory Lecot and Bruno Lévy. 2006. Ardeco: Automatic Region Detection and Conversion. In *Proceedings of the 17th Eurographics Conference on Rendering Techniques* (Nicosia, Cyprus) *(EGSR '06)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 349–360. https://doi.org/10.2312/EGWR/EGSR06/349-360
Juelin Leng, Yongjie Zhang, and Guoliang Xu. 2013. A Novel Geometric Flow Approach for Quality Improvement of Multi-Component Tetrahedral Meshes. *Comput. Aided Des.* 45, 10 (Oct. 2013), 1182–1197. https://doi.org/10.1016/j.cad.2013.05.004
Tzu-Mao Li, Michal Lukáč, Gharbi Michaël, and Jonathan Ragan-Kelley. 2020. Differentiable Vector Graphics Rasterization for Editing and Learning. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 39, 6 (2020), 193:1–193:15.
Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. 2017. Feature Pyramid Networks for Object Detection. arXiv:1612.03144 [cs.CV]
H Lipson and M Shpitalni. 1996. Optimization-based reconstruction of a 3D object from a single freehand line drawing. *Computer-Aided Design* 28, 8 (1996), 651–663. https://doi.org/10.1016/0010-4485(95)00081-X
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), 97. https://doi.org/10.1145/3197517.3201314
Chen Liu, Jiajun Wu, Pushmeet Kohli, and Yasutaka Furukawa. 2017. Raster-to-Vector: Revisiting Floorplan Transformation. In *2017 IEEE International Conference on Computer Vision (ICCV)*. 2214–2222. https://doi.org/10.1109/ICCV.2017.241
Raphael Gontijo Lopes, David Ha, Douglas Eck, and Jonathon Shlens. 2019. A Learned Representation for Scalable Vector Graphics. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.
M. K. Misztal, K. Erleben, A. Bargteil, J. Fursund, B. Bunch Christensen, J. A. Bærentzen, and R. Bridson. 2012. Multiphase Flow of Immiscible Fluids on Unstructured Moving Meshes. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Lausanne, Switzerland) *(SCA '12)*. Eurographics Association, Goslar, DEU, 97–106.
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 (Proceedings of ACM SIGGRAPH 2021)* 40, 4 (2021), 51:1–51:14.
Patryk Najgebauer and Rafał Scherer. 2019. Inertia-based Fast Vectorization of Line Drawings. *Computer Graphics Forum (Proc. Pacific Graphics)* 38, 7 (2019), 203–213.
Alejandro Newell, Kaiyu Yang, and Jia Deng. 2016. Stacked Hourglass Networks for Human Pose Estimation, Vol. 9912. 483–499. https://doi.org/10.1007/978-3-319-46484-8_29
Gioacchino Noris, Alexander Hornung, Robert W. Sumner, Maryann Simmons, and Markus Gross. 2013. Topology-driven Vectorization of Clean Line Drawings. *ACM Trans. Graph.* 32, 1, Article 4 (Feb. 2013), 11 pages. https://doi.org/10.1145/2421636.2421640
Alexandrina Orzan, Adrien Bousseau, Pascal Barla, Holger Winnemöller, Joëlle Thollot, and David Salesin. 2013. Diffusion Curves: A Vector Representation for Smooth-shaded Images. *Commun. ACM* 56, 7 (July 2013), 101–108. https://doi.org/10.1145/2483852.2483873
Amal Dev Parakkat, Uday Bondi Pundarikaksha, and Ramanathan Muthuganapathy. 2018. A Delaunay triangulation based approach for cleaning rough sketches. *Computers & Graphics* 74 (2018), 171 – 181.
Pradyumna Reddy, Michael Gharbi, Michal Lukac, and Niloy J. Mitra. 2021. Im2Vec: Synthesizing Vector Graphics without Vector Supervision. arXiv:2102.02798 [cs.CV]
Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You Only Look Once: Unified, Real-Time Object Detection. arXiv:1506.02640 [cs.CV]
Robert Schneider and Leif Kobbelt. 2001. Geometric fairing of irregular meshes for free-form surface design. *Computer Aided Geometric Design* 18, 4 (2001), 359 – 379. https://doi.org/10.1016/S0167-8396(01)00036-X
Edgar Simo-Serra, Satoshi Iizuka, and Hiroshi Ishikawa. 2018. Mastering Sketching: Adversarial Augmentation for Structured Prediction. *ACM Transactions on Graphics (TOG)* 37, 1 (2018).
Edgar Simo-Serra, Satoshi Iizuka, Kazuma Sasaki, and Hiroshi Ishikawa. 2016. Learning to Simplify: Fully Convolutional Networks for Rough Sketch Cleanup. *ACM Transactions on Graphics (SIGGRAPH)* 35, 4 (2016).
Tibor Stanko, Mikhail Bessmeltsev, David Bommes, and Adrien Bousseau. 2020. Integer-Grid Sketch Simplification and Vectorization. *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Geometry Processing)* 39, 5 (jul 2020), 149–161. http://www-sop.inria.fr/reves/Basilic/2020/SBBB20
Gabriel Taubin. 1995. A Signal Processing Approach to Fair Surface Design. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '95)*. Association for Computing Machinery, New York, NY, USA, 351–358. https://doi.org/10.1145/218380.218473
Nils Thürey, Chris Wojtan, Markus Gross, and Greg Turk. 2010. A Multiscale Approach to Mesh-Based Surface Tension Flows. In *ACM SIGGRAPH 2010 Papers* (Los Angeles, California) *(SIGGRAPH '10)*. Association for Computing Machinery, New York, NY, USA, Article 48, 10 pages. https://doi.org/10.1145/1833349.1778785
Engin Türetken, Fethallah Benmansour, Bjoern Andres, Hanspeter Pfister, and Pascal Fua. 2013. Reconstructing Loopy Curvilinear Structures Using Integer Programming. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*. 1822–1829. https://doi.org/10.1109/CVPR.2013.238
Saining Xie and Zhuowen Tu. 2017. Holistically-Nested Edge Detection. *Int. J. Comput. Vision* 125, 1–3 (Dec. 2017), 3–18.
Xuemiao Xu, Minshan Xie, Peiqi Miao, Wei Qu, Wenpeng Xiao, Huaidong Zhang, Xueting Liu, and Tien-Tsin Wong. 2019. Perceptual-aware Sketch Simplification Based on Integrated VGG Layers. *IEEE Transactions on Visualization and Computer Graphics* (2019).
Chuan Yan, David Vanderhaeghe, and Yotam Gingold. 2020. A Benchmark for Rough Sketch Cleanup. *ACM Transactions on Graphics* 39, 6 (Nov. 2020). https://doi.org/10.1145/3414685.3417784 To be presented at SIGGRAPH Asia 2020.
Song-Hai Zhang, Tao Chen, Yi-Fei Zhang, Shi-Min Hu, and Ralph R. Martin. 2009. Vectorizing Cartoon Animations. *IEEE Transactions on Visualization and Computer Graphics* 15, 4 (July 2009), 618–629. https://doi.org/10.1109/TVCG.2009.9
Yizhong Zhang, Huamin Wang, Shuai Wang, Yiying Tong, and Kun Zhou. 2012. A Deformable Surface Model for Real-Time Water Drop Animation. *IEEE Transactions on Visualization and Computer Graphics* 18, 8 (Aug. 2012), 1281–1289. https://doi.org/10.1109/TVCG.2011.141

Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. 2019b. Objects as Points. In *arXiv preprint arXiv:1904.07850.*

Yichao Zhou, Haozhi Qi, and Yi Ma. 2019a. End-to-end wireframe parsing. *Proceedings of the IEEE International Conference on Computer Vision* 2019-October (2019), 962–971. https://doi.org/10.1109/ICCV.2019.00105 arXiv:1905.03246

A APPENDIX

A.1 Derivation of the PolyVector Flow

To be self-contained, we summarize the arguments for Proposition 5.1 and Lemma 5.2 in this section. There is a similar (but not identical) line of reasoning in [Deckelnick et al. 2005], so we refer the reader to this work for further details, if desired. We will highlight the differences that arise with our setting of a position-varying γ and open curves with fixed endpoints, and also provide further commentary on the arguments as a service to the community.

As done there, we use an evolving level set description of a closed curve where a time-varying signed distance function $d(x,t) : \mathbb{R}^2 \times (-\epsilon, \epsilon)$ is used. This tracks an evolving curve with the curve image described by the zero level set: $x([0,L],t) = \{x \in \mathbb{R}^2 | d(x,t) = 0\}$. For our case of open curves, we use these constructions in a tubular neighborhood of the initial curve $x(\theta, 0)$.

We will make use of integration-by-parts, which must be modified to include boundary terms in our open curve case. We use the following notation for the gradient operator and partial derivative operators restricted to Γ: $\nabla_\Gamma f = (\underline{\mathrm{D}}_1 f, \underline{\mathrm{D}}_2 f)$.

Theorem A.1 (Integration-by-parts). *Given a curve Γ as above, and a function f that is continuously differentiable in a neighborhood of Γ:*

$$\int_\Gamma \underline{\mathrm{D}}_i f \, dl = \int_\Gamma f \kappa \nu_i \, dl + \int_{\partial\Gamma} f x_i' \qquad i = 1, 2 \tag{11}$$

Note that the integral over $\partial\Gamma$ is just a signed sum: $f(x(L))x_i'(L) - f(x(0))x_i'(0)$. This may be proven easily with an arclength parametrization assumption (so that $x''(\theta) = -\kappa(\theta)\nu(\theta)$) and an application of standard integration-by-parts.

We will also use a transport theorem:

Theorem A.2 (Transport Theorem). *Given a family of evolving curves $x(\theta,t), \theta \in [0,L], t \in [-\epsilon, \epsilon]$ such that $x(0,t) = A, x(L,t) = B$ for all t, and a differentiable function g:*

$$\frac{d}{dt}\int_{\Gamma(t)} g \, dl = \int_{\Gamma(t)} \frac{\partial g}{\partial t} + gV\kappa + \frac{\partial g}{\partial \nu} V \, dl, \tag{12}$$

where V is the normal velocity and κ is the curvature.

The form of this theorem does not differ from that used in [Deckelnick et al. 2005], because of the fixed endpoints.

Lastly, we note two properties that are easy to verify and used repeatedly:

$$\nabla_\Gamma f \cdot \nu = 0, \quad \text{for any function } f \tag{13}$$

$$D\gamma(\nu) \cdot \nu = \gamma(\nu), \quad \text{for } \gamma \text{ 1-homogeneous and positive} \tag{14}$$

A.1.1 Proof of Prop. 5.1. Recall that we consider a normal variation of the curve where a function $\phi : \Gamma \to \mathbb{R}$ is used to describe the variation of the curve in the normal direction: $x(\theta,t) = x(\theta,0) + t\phi(x(\theta,0))\nu(x(\theta,0))$. For fixed endpoints, we further assume that $\phi = 0$ at the curve endpoints.

We consider our integrand as a composition with the gradient of the signed distance function: $\gamma(x,\nu) = \gamma(x, \nabla d(x,t))$. An application of the Transport Theorem gives us three terms to consider:

$$\frac{d}{dt}\left(\int_{\Gamma(t)} \gamma(x,\nu)\, dl\right)\Big|_{t=0} = \int_\Gamma \frac{\partial \gamma}{\partial t} + \gamma\phi\kappa + \frac{\partial \gamma}{\partial \nu}\phi \, dl.$$

For the first term in the integrand:

$$\frac{\partial \gamma}{\partial t}(\cdot, 0) = D\gamma(\nu) \cdot \nabla \frac{\partial d}{\partial t}(\cdot, 0) = -D\gamma(\nu) \cdot \nabla_\Gamma \phi.$$

The first equality follows from the chain rule and commuting derivatives. The second follows from noting that $\frac{\partial d}{\partial t}(\cdot,0) = -\phi$ and $\nabla \frac{\partial d}{\partial t}(\cdot,0) \cdot \nu = 0$ on Γ.

If we use integration-by-parts we may combine and simplify the integral of the first two terms.

$$\begin{aligned}
&\int_\Gamma \gamma\phi\kappa - (D\gamma(\nu) \cdot \nabla_\Gamma \phi)\ dl \\
&\qquad = \int_\Gamma \phi\kappa \sum_i (D\gamma(\nu)_i \nu_i) - \sum_i D\gamma(\nu)_i \underline{\mathrm{D}}_i \phi \, dl \\
&\qquad = \int_\Gamma \sum_i \underline{\mathrm{D}}_i \left(D\gamma(\nu)_i \phi\right) - \sum_i D\gamma(\nu)_i \underline{\mathrm{D}}_i \phi \, dl \\
&\qquad = \int_\Gamma \sum_i \underline{\mathrm{D}}_i \left(D\gamma(\nu)_i\right) \phi = \int_\Gamma \nabla_\Gamma \cdot D\gamma(\nu) \phi \, dl
\end{aligned}$$

The first equality follows from Equation (13). The second follows from integration-by-parts, noting that boundary terms disappear as ϕ vanishes at endpoints. The third follows from the product rule for $\underline{\mathrm{D}}_i$.

A.1.2 Proof of Lemma 5.2. First, we make an argument for Eq. 8.18 of [Deckelnick et al. 2005]. Noting the symmetry condition $\underline{\mathrm{D}}_k \nu_l = \underline{\mathrm{D}}_l \nu_k$ and the projected derivative equality $\underline{\mathrm{D}}_k x_l = \delta_{kl} - \nu_k \nu_l$, we get the following chain of equalities:

$$\begin{aligned}
\nabla_\Gamma \cdot D\gamma(\nu)\nu_l &= \nu_l \sum_k \underline{\mathrm{D}}_k D\gamma(\nu)_k \\
&= \sum_k \underline{\mathrm{D}}_k \left(D\gamma(\nu)_k \nu_l\right) - D\gamma(\nu)_k \underline{\mathrm{D}}_l \nu_k \\
&= \sum_k \underline{\mathrm{D}}_k \left(D\gamma(\nu)_k \nu_l\right) - \underline{\mathrm{D}}_l(\gamma(\nu)) \\
&= \sum_k \underline{\mathrm{D}}_k \left(D\gamma(\nu)_k \nu_l\right) - \underline{\mathrm{D}}_k(\gamma(\nu)(\delta_{kl} - \nu_k\nu_l)) - \gamma(\nu)\nu_l \underline{\mathrm{D}}_k \nu_k \\
&= \sum_k \underline{\mathrm{D}}_k \left(D\gamma(\nu)_k \nu_l\right) - \underline{\mathrm{D}}_k(\gamma(\nu)\underline{\mathrm{D}}_k x_l) - \gamma(\nu)\kappa\nu_l
\end{aligned}$$

The first equality follows from the product rule for $\underline{\mathrm{D}}_k$ and the symmetry condition. The second follows from the chain rule for $\underline{\mathrm{D}}_l$. For the third equality, one expands the second term arising from the projected derivative equality, using the product rule for $\underline{\mathrm{D}}_k$, and find that it cancels with the third term $\gamma(\nu)\nu_l \underline{\mathrm{D}}_k \nu_k$. Equation (14) is used throughout.

When we take the dot product of the above against a test function $\phi = (\phi_1, \phi_2)$ which vanishes at the curve endpoints, we get the

following equality:

$$\begin{aligned}
&\int_{\Gamma(t)} \nabla_\Gamma \cdot D\gamma(v)(v\cdot\psi)\,dl \\
&\quad= \sum_{k,l} \int_{\Gamma(t)} -D\gamma(v)_k v_l \underline{\mathrm{D}}_k\phi_l + D\gamma(v)_k v_l \kappa v_k \phi_l \\
&\qquad\qquad + \gamma(v)\underline{\mathrm{D}}_k x_l \underline{\mathrm{D}}_k \phi_l - \gamma(v)\kappa v_l \phi_l\,dl \\
&\quad= \sum_{k,l} \int_{\Gamma(t)} -D\gamma(v)_k v_l \underline{\mathrm{D}}_k\phi_l + \gamma(v)\underline{\mathrm{D}}_k x_l \underline{\mathrm{D}}_k \phi_l\,dl
\end{aligned}$$

The first equality follows by the product rule for $\underline{\mathrm{D}}_k$ and integration-by-parts. Again, vanishing of the test function at endpoints, ensures that boundary terms disappear. The second equality comes about as the second and third terms cancel by Equation (14).

Finally, let us express these quantities in terms of a parametrized curves $x(\theta,t)$ and variations $\phi(\theta)$, as is done in the derivation of Eq. 8.23 in [Deckelnick et al. 2005].

$$\begin{aligned}
&\int_{\Gamma(t)} \nabla_\Gamma \cdot D\gamma(v)(v\cdot\psi)\,dl \\
&\quad= \sum_{k,l} \int_{\Gamma(t)} -D\gamma(v)_k v_l \underline{\mathrm{D}}_k\phi_l + \gamma(v)\underline{\mathrm{D}}_k x_l \underline{\mathrm{D}}_k \phi_l\,dl \\
&\quad= -\sum_{k,l} \int_{\Gamma(t)} (D\gamma(v)_k v_l - \gamma(v)\delta_{kl})\underline{\mathrm{D}}_k\phi_l\,dl \\
&\quad= \int_0^{2\pi} (D\gamma(x'(\theta,t)^\perp)\cdot\phi'(\theta)^\perp\,d\theta
\end{aligned}$$

The second equality follows by the projected derivative equality and Equation (13). The third follows from noting that $\nabla_\Gamma\phi_l = \frac{\phi_l'(\theta)}{|x'(\theta,t)|}\tau$ where τ is the unit normal and $\tau = (\tau_1,\tau_2) = (-v_2, v_1)$, and expanding the summands in the integrand with Equation (14).

B DETAILS ON KEYPOINT EXTRACTION

Architecture. We use the architecture in Fig. 21. Input image at training is $1\times288\times288$. As described in Sec. 3, the network produces 6 heatmaps: 4 final and 2 intermediate. We use LeakyReLU as the activation function.

Validation. We use F_1 score to evaluate the detector performance, and compute classification accuracy on correctly detected keypoints. We consider a ground truth point correctly predicted (true positive), if there is predicted point within 5 pixel distance. Number of false positives is the difference between all predicted points (of the same class, if classification) and the number of true positives. Validation set consists of 40 real line drawings labeled manually.

Dataset. We use 75,000 vector format doodles from [Ha and Eck 2017] and 17,700 from [Ge et al. 2021]. We compute curve intersection points (junctions), endpoints, and sharp corners (measured by the angle of 135°). We rasterize each sketch in Adobe Illustrator using 10 different artistic brushes of different widths, randomly assigned on a per-stroke basis, at 288x288 resolution. We extend this dataset by adding manually labeled 250 pencil line drawings, which helps generalization. Resulting dataset of 927,000 png images with labels form Dataset-A. We form another dataset Dataset-B for

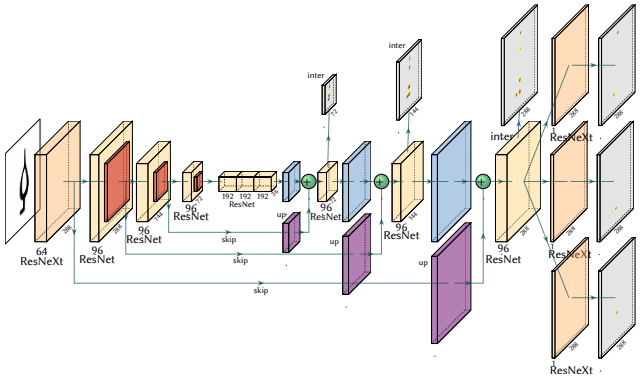


Fig. 21. Model architecture. ResNext block has 64 internal channels (1x1 conv, 3x3 padding 1 conv, 1x1 conv) + skip connection + batch normalization. Downsampling is performed via ResNet followed by 2x2 conv with stride 2. Upsampling is performed via Pixel Shuffle transforming (C,H,W) tensor into $(C/4, 2\times H, 2\times W)$, which is followed by conv layer with 3×3 kernel and padding 1, outputting a tensor $(C, 2\times H, 2\times W)$. At the lowest resolution (1/8x), we use 3 ResNet blocks with fixed spatial resolution, but twice as many channels. After upsampling we first add skip connection tensor of the same size, and then pass it to the ResNet block obtain a feature tensor with 96 channels. This procedure is repeated 3 times to obtain a feature tensor of the original spatial size and 96 channels 96x288x288.

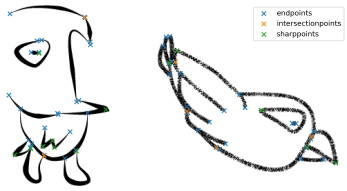


Fig. 22. A sample from the training dataset for the keypoint detector.

fine-tuning using 1000 vector drawings from [Ge et al. 2021], which we manually preprocess to remove shading or fills.

Training procedure and Inference. We start from training on Dataset-A with 80x80 crops from images at different resolutions (96x96, then 128x128 and then 288x288). Each image is normalized to $[0,1]$, series of standard image augmentations are applied at random (gaussian blurring, contrast and brightness, gaussian noise, salt&pepper noise, random rotation, random crop to size 80x80). Point positions are transformed to heatmaps and adjusted according to augmentations performed on corresponding image. We use loss in Eq. 1, only within the narrow band. We train for 18 epochs, with batch size 60.

Fine-tuning is done on Dataset-B on 80x80 crops from 288x288 images. We use the same data augmentation but different weights for classification heatmaps: 0.6, 0.6, 1.2 for H_2, H_3, H_4 respectively. We fine-tune the model for 20 epochs, and batch size 40. For both stages, we use learning rate $5\cdot10^{-4}$. At inference we perform contrast normalization on the original image and pad it to enable 8x downsampling.