Vectorization of Line Drawings via PolyVector Fields

MIKHAIL BESSMELTSEV, Université de Montréal & Massachusetts Institute of Technology
JUSTIN SOLOMON, Massachusetts Institute of Technology

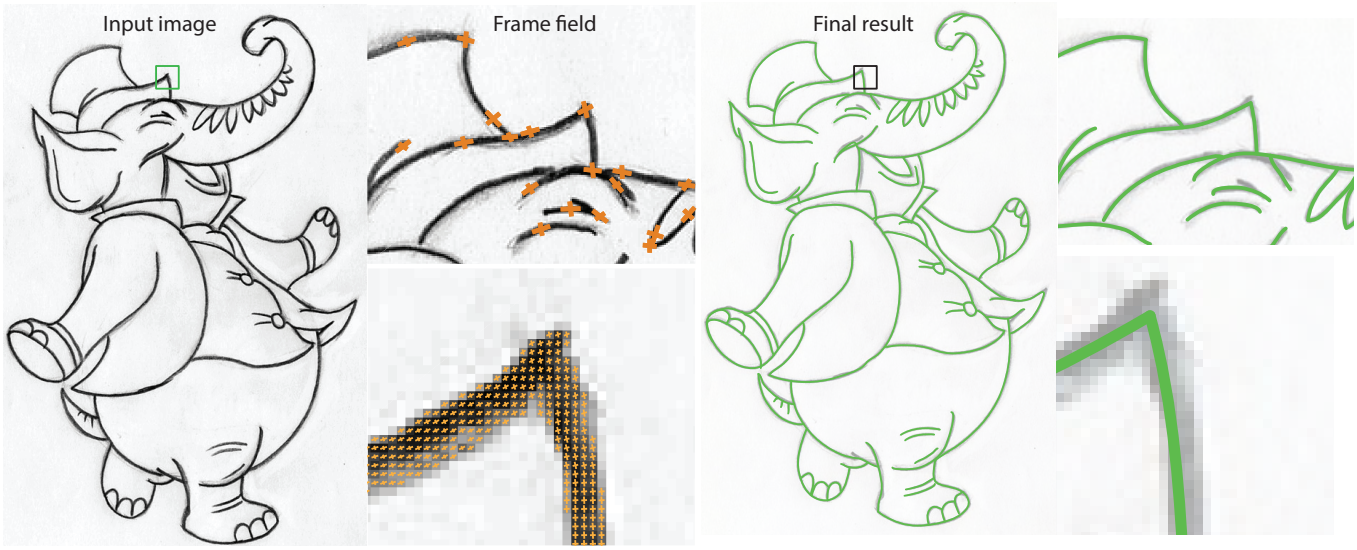


Fig. 1. Given a possibly noisy grayscale bitmap image, we compute a frame field aligned with the directions on the image, superimposing multiple directions around sharp corners as well as X- and T-junctions. We then use this frame field to extract the drawing topology and create the final vectorization with the computed topology. Frame field computation (shown for a subset of pixels in the upper zoom and the full field in the lower one) is the key component of the system. The frame field disambiguates X- and T-junctions even in the noisy areas, allowing tracing to be straightforward and robust. Input images are from www.easy-drawings-and-sketches.com, ©Ivan Huska.

Image tracing is a foundational component of the workflow in graphic design, engineering, and computer animation, linking hand-drawn concept images to collections of smooth curves needed for geometry processing and editing. Even for clean line drawings, modern algorithms often fail to faithfully vectorize junctions, or points at which curves meet; this produces vector drawings with incorrect connectivity. This subtle issue undermines the practical application of vectorization tools and accounts for hesitance among artists and engineers to use automatic vectorization software. To address this issue, we propose a novel image vectorization method based on state-of-the-art mathematical algorithms for frame field processing. Our algorithm is tailored specifically to disambiguate junctions without sacrificing quality.

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

Additional Key Words and Phrases: Vectorization, line drawing, PolyVector field, frame field

0730-0301/2017/0-ART0 $15.00
DOI: 0000001.0000001_2

ACM Reference format:
Mikhail Bessmeltsev and Justin Solomon. 2017. Vectorization of Line Drawings via PolyVector Fields. *ACM Trans. Graph.* 0, 0, Article 0 (2017), 12 pages.
DOI: 0000001.0000001_2

1 INTRODUCTION

Image vectorization algorithms date back to early 1990s and are among the core tools in vector processing software including Adobe Illustrator (Live Trace), CorelDRAW (PowerTRACE), and Inkscape. Despite their wide adoption in industry, algorithms for line drawing vectorization remain under active development and still admit major shortfalls [Favreau et al. 2016; Noris et al. 2013]. In several industries where vectorization is heavily needed, including traditional animation and engineering design, this task frequently is done manually, by painstakingly tracing a scanned image with drawing tools. This process is often considered to take less time than editing the automatic result from commercial vectorization tools.

A primary reason for frustration with line drawing vectorization algorithms is incorrect treatment of junctions, resulting in wrong topology, or connectivity (Fig.2(a,b)). Image understanding and perception rely on junctions and drawing topology to disambiguate depth and other cues [Xia et al. 2014]. In industries such as character animation, incorrect topology yields temporal incoherence and makes modern automatic coloring or in-betweening tools unusable [Orzan et al. 2013; Whited et al. 2010]. In engineering-oriented

industries, incorrect topology may be considered an incorrect result overall, because it may not correspond to a physically-realizable object.

The main challenge when disambiguating junctions in line drawings is noisy or insufficient local information, even for clean images [Noris et al. 2013]. The presence of noise, such as uneven curve edges, complicates matters even further, and the widely-used local approach to resolving junctions based on a one-pixel width image skeleton becomes unreliable [Favreau et al. 2016].

A recent method by Favreau et al. [2016] uses *global information* to resolve ambiguities at junctions. Their method successfully vectorizes sketches with numerous overdrawn strokes, where a heavy simplification of the result is needed. Unfortunately, for inputs requiring fidelity, their approach can lead to oversimplified results significantly deviating from the drawn contours (Fig. 2(b), 16).

While theoretically junctions may have various valences, as noted by previous work [Noris et al. 2013], the vast majority of junctions are X- and T-junctions (Fig. 1). Occlusion contours typically generate T-junctions, making them crucial for 3D shape perception [Bessmeltsev et al. 2015; Kanizsa 1979]. Hence, correct resolution of X- and T-junctions is a primary concern during image vectorization.

With these challenges in mind, in this paper we propose a robust image tracing method true to the image in unambiguous regions, with global treatment of T- and X-junctions even when local information is unclear (Fig. 2, right). Our technical innovation is to use frame fields to guide vectorization. Frame fields attach two pairs of vectors $\{\pm u, \pm v\}$ to each point on the plane. They have been recently used to generate anisotropic quadrilateral meshes and to estimate 3D normals from 2D sketches [Iarussi et al. 2015; Panozzo et al. 2014]. Although frame fields are natural for tracking the orientations of curves meeting at sharp junctions, to our knowledge they never have been applied to image vectorization.

Overview. As illustrated in Figure 1, the general idea of our method is to find a smooth frame field on the image plane, where at least one direction is aligned with nearby contours of the drawing. Around X- or T-shaped junctions, the two directions of the field will be aligned with the two intersecting contours. Then, we extract the topology of the drawing by tracing the frame field and grouping traced curves into strokes. Finally, we create a vectorization aligned with the frame field with the extracted topology.

2 RELATED WORK

Our work builds upon achievements in three areas: image vectorization, junction detection, and frame fields. A comprehensive review of these areas is outside the scope of this paper; here, we instead highlight work relevant to our proposed pipeline.

Frame fields. Our algorithm is built upon the construction of *frame fields* that assign two directions to every point in a planar region; these directions will guide our placement of strokes. Unlike *cross fields*, frame fields have no constraint on orthogonality or length of the direction vectors. We refer the reader to the recent survey by Vaxman et al. [2016] for broad discussion.

While cross fields have been extensively used in computer graphics [Hertzmann and Zorin 2000; Kass and Witkin 1987; Palacios and Zhang 2007], representations of frame fields and algorithms for their computation are relatively recent [Diamanti et al. 2015; Panozzo et al. 2014]. They serve as a natural representation of linear transformations on tangent spaces of a surface. Frame fields originally were proposed for guiding anisotropic quad meshing via inversion-free mesh parameterization [Panozzo et al. 2014]. Since then, frame fields have found additional applications, such as inferring 3D normals from a 2D sketch [Iarussi et al. 2015] and recovery of damaged historical documents [Pal et al. 2016].

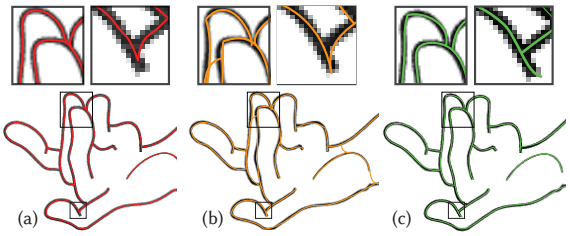


Fig. 2. (a) Local approaches to junction resolution, such as the one proposed by Noris et al. [2013], may result in incorrect or imprecise junctions. (b) Favreau et al.'s method [2016] may significantly deviate from the drawing. (c) Our result.

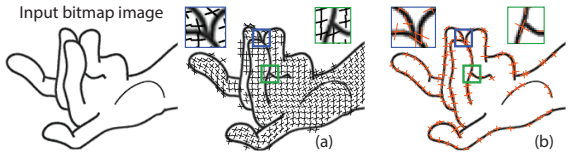


Fig. 3. The target application for Bendfields [Iarussi et al. 2015] leads to different frame field design assumptions (a) that are unsuitable for our type of vectorization. Compare with our result (b). For clarity, we only show a subset of frame field on (b); we compute frame field on every dark pixel.

Our work is driven by the frame field synthesis and interpolation tool set developed in [Diamanti et al. 2015; Panozzo et al. 2014]. Namely, we use their definition and representation of a *PolyVector field*, as described in Section 3.1.

The BendFields algorithm proposed by Iarussi et al. [2015] inspired some aspects of our approach. While their algorithm is targeted to 3D surface reconstruction from curvature lines, they initially generate a frame field aligned to directions in a bitmap image. Their goal, however, is to compute a frame field in the space *between* the input curves, while we solve for a frame field defined exclusively on dark pixels. This difference gives our method a significant performance boost by reducing the number of degrees of freedom, and it qualitatively affects the results near junctions with sharp angles. Due to differences in application, the formulation and weighting of their alignment term differs from ours (Fig. 3), and our use of PolyVectors has only real-valued variables per pixel rather than requiring a mixed-integer solver.

Image vectorization. Vectorization of bitmapped images has been studied extensively in graphics, vision, and other disciplines. Various input- and application-specific priors guide many vectorization methods, conforming to requirements of end users in medical imaging, road map reconstruction from GPS traces, processing of

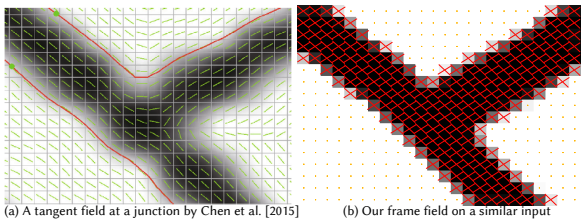


Fig. 4. (a) A tangent field [Chen et al. 2015] cannot capture a collection of directions present at a junction point. (b) On a similar input, the frame field is a natural representation of the directions at a junction.

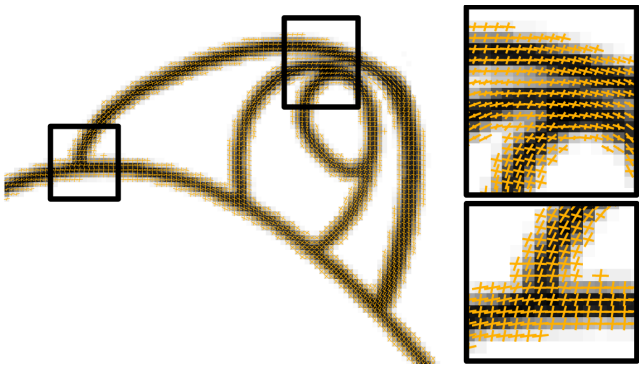


Fig. 5. The frame field we design has at least one field direction aligned with a nearby curve tangent. Near T- and X-junctions, our field is aligned with both tangent directions.

astronomical imagery, and other tasks [Bo et al. 2016; Chai et al. 2013; Türetken et al. 2013]. These methods are application-specific and cannot be applied directly to vectorization of hand-drawn line drawings. Other vectorization methods deal with shaded images, like photographs or cartoon images [Lecot and Lévy 2006; Orzan et al. 2013; Zhang et al. 2009]; their focus is to capture an image using simple colored primitives, which typically are assumed to be closed.

We focus on reconstruction of line drawings without shading, where lines may or may not be closed. In this area, strong priors about line shape, e.g. that lines only form circles or straight lines, might bring simplicity to vectorization of technical drawings [Hilaire and Tombre 2006], but do not apply to free-form line drawings.

For vectorization of curvy line drawings, existing methods vary by the amount of noise allowed in the input. Noisy line drawings with multiple overlapping strokes or hatching patterns require deviation from the drawn image in favor of simplicity [Bartolo et al. 2007; Favreau et al. 2016]. Guided by a similar motivation, De Goes et al. [2011] propose a method to extract a simplified curve network for noisy drawings. While their approach is natural for drawings with very fuzzy lines and significant noise, such behavior may not be desired for higher-quality drawings that do not contain overlapping strokes, which require more precise vectorization (Fig. 2(a)).

On the other side of the spectrum is an image vectorization method tailored for clean cartoon drawings by Noris et al. [2013]. Their global approach to topology allows them to, for instance, correctly disambiguate nearby parallel strokes. Their treatment of junctions, however, is still local and may result in incorrect or imprecise treatment (Fig. 2(b)). Furthermore, the discrete nature of the algorithm renders it unstable in presence of noise (Fig. 15).

A recent work by Donati et al. [2017] explores accurate vectorization of noisy sketches using Pearson's correlation coefficient with Gaussian kernels. While their method achieves impressive performance and is able to process sketches with multiple overlapping strokes, it makes no effort to correctly disambiguate junctions, parallel lines, or overall extract drawing topology. Instead they rely on the topology of a 1-pixel width skeleton, which is known to be prone to local artifacts [Favreau et al. 2016]. In contrast, we resolve junctions and parallel lines by generating a frame field, and use it to explicitly extract drawing topology.

A line of work close to our method is using tangent fields for image processing and vectorization [Chen et al. 2013, 2015; Kang et al. 2007]. For instance, Chen et al. [2015] propose an image vectorization method with a global variational approach to disambiguation of junctions. The primary issue with these approaches is the use of tangent fields, which cannot capture a *collection* of directions present at a junction point (Fig 4). As a result, the method by Chen et al. [2015] relies on user interaction and arbitrary thresholds to resolve junctions. Their method also does not consider the topology of the drawing, potentially yielding disconnected lines and/or spurious connections.

Building on this work, our method uses a more natural representation to track junctions in the drawings: a frame field defined at each stroke pixel. The two directions of the frame field efficiently disambiguate directions around T- or X-junctions, and the variational nature of our approach makes it resistant to noise.

Corner and junction detection in images. Corner detection is a basic step in classical computer vision pipelines [Szeliski 2010]; for example, the well-known Harris corner detector [Harris and Stephens 1988] is implemented in countless industry-standard vision libraries. The goal of these methods typically is to detect and characterize salient features, while for vectorization it is more important to calculate the exact center of the junction and to estimate the directions of the joining lines robustly. Furthermore, even if it is possible to identify junction points, image gradient directions near corners and junctions often are noisy, making it difficult to estimate the individual directions meeting at a junction point using purely local information.

3 ALGORITHM

Our system takes as input a grayscale bitmap line drawing and produces a set of strokes aligned to the drawing. We first solve an optimization problem to compute a frame field at each pixel in a narrow band around the set of stroke pixels, designed to capture directionality of the input and to superpose multiple directions near junctures. We then extract topology of the drawing by tracing the frame field and grouping curves into strokes. We then compute the final vectorization (Fig. 1).

3.1 Designing Frame Fields

Our vectorization algorithm begins by computing a smooth frame field, such that at every point near a stroke at least one field direction is aligned with a nearby curve tangent. Near T- and X-junctions, our field will align to *both* tangent directions present nearby in the image; this provides the flexibility needed to resolve image behavior near junctions (Fig. 5). We formulate computation of the field as a variational problem that, after discretization, can be approached using standard algorithms for nonlinear unconstrained optimization.

Initial steps. We start by thresholding the image to isolate those pixels involved in the line drawing. In particular, our algorithm will operate on a subset of pixels I, corresponding to dark pixels with intensity less than a fixed threshold, $\theta_{\text{noise}} \cdot I_{max}$, where I_{max} is the maximum image intensity. We additionally estimate a noisy tangent field τ from the drawing by taking its Sobel gradient g with kernel of size 3 and rotating by $\pi/2$. Note that, similarly to the discussion in [Zhang et al. 2007], the direction of this rotation (clockwise vs. counterclockwise) will not affect the computation of the frame field, which always couples forward and backward directions.

Representation and variational problem. Following Diamanti et al. [2015], we represent the unknown frame field as a *PolyVector field*. Suppose we are given two directions u, v representing curve tangents of the drawing near a given pixel; we can identify the image plane with the complex numbers $\mathbb{C}$ and take $u, v \in \mathbb{C}$ as complex vectors. Consider the following complex polynomial $f(z)$:

$$f(z) := (z^2 - u^2)(z^2 - v^2) = z^4 + c_2 z^2 + c_0. \tag{1}$$

Here, the constants c_0 and c_2 determine u and v up to relabeling and sign. That is, every pair $(c_0, c_2) \in \mathbb{C}^2$ uniquely determines a *frame* $\{\pm u, \pm v\}$, agnostic to the labeling of u vs. v as well as their sign. We use $f(z; c_0, c_2)$ to denote the function above parameterized by the two coefficients c_0 and c_2.

Recovering the frame directions from (c_0, c_2) is equally straightforward:

$$\left\{ \begin{array}{l} c_0 = u^2 v^2 \\ c_2 = -(u^2 + v^2) \end{array} \right\} \longleftrightarrow \left\{ \begin{array}{l} u^2 = -\frac{1}{2}\left(c_2 + \sqrt{c_2^2 - 4c_0}\right) \\ v^2 = -\frac{1}{2}\left(c_2 - \sqrt{c_2^2 - 4c_0}\right) \end{array} \right\} \tag{2}$$

Of course, the relationship on the right is non-unique.

Optimizing for a (u, v) pair per pixel induces challenging issues involving labeling and sign; for example, this representation in the BendFields algorithm requires the use of a mixed-integer solver [Iarussi et al. 2015]. To avoid this complexity, we instead optimize for a (c_0, c_2) pair per pixel, which has no sign or ordering ambiguity. That is, the unknown in our optimization technique is a pair of complex-valued functions $c_0, c_2 : I \to \mathbb{C}$.

We propose the following variational problem:

$$\min_{c_0, c_2 : I \to \mathbb{C}} E_{\text{alignment}} + \lambda E_{\text{smoothness}} + \mu E_{\text{regularization}} =$$

$$\min_{c_0, c_2 : I \to \mathbb{C}} \int_I |f(e^{i\theta_\tau}; c_0(\vec{x}), c_2(\vec{x}))|^2 \, d\vec{x} + \lambda \sum_{i=0,2} \int_I \|\nabla c_i(\vec{x})\|^2 \, d\vec{x}$$

$$+ \mu \int_I |f(e^{i\theta_g}; c_0(\vec{x}), c_2(\vec{x}))|^2 \, d\vec{x} \tag{3}$$

Here, θ_w encodes the direction of a vector w, i.e. in complex language we can write $w = \|w\|_2 e^{i\theta_w}$.

Details about the individual terms in (3) are below, in the order they appear:

- Alignment: The first optimization term enforces alignment of the frame field with the tangent directions. This term is small when the polynomial $f(\cdot; c_0, c_2)$ has a root near $e^{i\theta_\tau}$, implicitly implying that one of the field directions $\{\pm u, \pm v\}$ is aligned with the tangent direction τ. Since (1) has no odd-degree terms, this term has no dependence on the sign of τ, as desired.
- Smoothness: The second optimization term is a Dirichlet energy measuring the smoothness of the functions $c_0(\vec{x})$ and $c_2(\vec{x})$ as a function of the location $\vec{x}$ in the image. Smoothly-varying (c_0, c_2) pairs imply a smooth set of frame directions. We use $\lambda = 50$ in all our experiments; while the method is fairly stable to the choice of λ, larger values may be desirable for particularly noisy inputs.
- Regularization: Away from junctions, there is only one prominent direction in the image. To prevent the frame field from collapsing into a line field, the regularization term expresses a slight preference for the field to be aligned with $\tau^\perp = g$ in the absence of other information.

To improve results by attenuating the influence of noisy directions near junctions, we weigh the smoothness term by $\frac{w(\vec{x})}{\max_{\vec{x}}(w(\vec{x}))}$ and alignment term by $1 - \frac{w(\vec{x})}{\max_{\vec{x}}(w(\vec{x}))}$, where

$$w(\vec{x}) := \left| \frac{\langle \tau^2(\vec{x}) \rangle}{\|\langle \tau^2(\vec{x}) \rangle\|} - \tau^2(\vec{x}) \right| \tag{4}$$

$$\langle \tau^2(z) \rangle := \int_{B(\vec{x})} \tau^2(\vec{x}') d\vec{x}'. \tag{5}$$

Here, $B(\vec{x})$ denotes a small (one-pixel) neighborhood of $\vec{x}$; in practice, we approximate this integral by averaging the neighboring values of τ^2 adjacent to the pixel centered at $\vec{x}$.

Optimization. We apply standard finite-difference discretization to evaluate the objective function (3) on a pixel grid. The end result is a quadratic, unconstrained optimization problem for a (c_0, c_2) pair per pixel. We use "natural" (Neumann) boundary conditions, essentially evaluating the gradient-based smoothness term only on pairs of adjacent pixels that are both included as degrees of freedom in the numerical problem.

We use the L-BFGS algorithm for optimization [Nocedal and Wright 2006], with a history of 6 iterates for the quasi-Newton Hessian approximation. Our code uses the LBFGS++ implementation described by Qiu et al. [2016]. We start from an initial guess of an axis-aligned cross field. The quadratic nature of our optimization problem would allow for more specialized techniques, e.g. preconditioned conjugate gradients, but as frame field computation is not

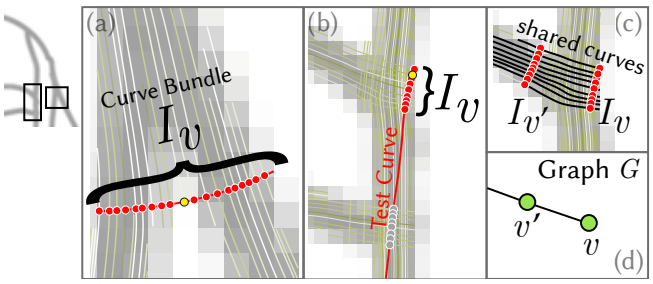


Fig. 6. Grouping curves into bundles and constructing graph G. (a,b) We trace a test curve (red) from each seed point (yellow), recording its intersection points with the curves (red and gray). Starting with a seed point, we then group the points that are adjacent along the test curve and closer than a pixel distance apart, forming *a curve bundle* I_v (red). (c,d) We then associate each curve bundle with a vertex in the graph G, connecting vertices if they have at least one pair of intersection points adjacent along their shared curve.

currently the efficiency bottleneck of our algorithm, we leave tuning of this step to future work.

In the end, we only require frame directions on the image pixels corresponding to strokes that we will trace. Hence, to improve optimization efficiency and to improve junction resolution even with acute angles, we take inspiration from *narrow band* level set methods [Adalsteinsson and Sethian 1995] and only include variables corresponding to pixels in I; that is, pixels corresponding to white areas in the input image are ignored. This greatly reduces the number of variables, yielding a significant boost in performance.

The optimization yields two scalar fields, c_0, c_2. At every dark pixel $i \in I$, we then use (2) to recover the frame field directions $\{\pm u, \pm v\}$.

4 EXTRACTING DRAWING TOPOLOGY

The next step of our algorithm extracts the topology of the drawing from the computed frame field. The key requirement is not only to extract the correct topology, but also to create a structure that allows for subsequent vectorization aligned with the frame field.

Starting from each dark pixel, we trace the frame field (Fig. 7(a)); these curves are grouped locally into *curve bundles*. Each curve bundle is associated to a vertex in a topological graph $G = (V, E)$, whose adjacency is determined by shared curve segments between different bundles (Fig. 7(b)). After topological simplification (Fig. 7(c)) and disambiguating parallel strokes, this graph has the topology of the line drawing (Fig. 7(d)) and allows for vectorization by following shared curves connecting each pair of curve bundles.

4.1 Initial Graph Construction

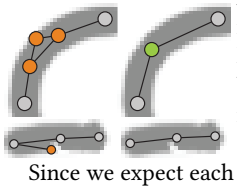
Away from junctions, the largest root of the frame field reliably is aligned with the curve tangent. Therefore, at every dark pixel $\vec{x}$, we choose the frame field root with the maximum magnitude; without loss of generality, we will label it $u(\vec{x})$. We then trace the frame field starting from $\vec{x}$ in both directions $\pm u$, using simple Euler's integration method with a step size $h = 0.1$, where each pixel is considered to have width 1 (Fig. 7(a)). We stop tracing as soon as a curve leaves the narrow band or comes indistinguishably close (within 0.01 distance in our implementation) to a curve with the same tangent. The latter condition is designed partly to prevent closed curves in the drawing from being traced over multiple times, and partly for efficiency reasons to avoid overtracing.

The integration step yields more curves than will be present in the final traced image. Hence, once all curves are traced, we split up curves into groups corresponding to strokes in the drawing. Since curves may naturally continue past acute junctions or *Y-junctions*—valence-3 junctions with three distinct directions—we create a topological graph by grouping curves locally (Fig. 6). Our goal is to group corresponding curves along the width of the stroke, perpendicular to the centerline; each group forms a vertex of the graph. We only group curves corresponding to the same direction of the frame field, thus separating intersecting strokes.

Starting from each curve endpoint (*seed*), at each of the 8 neighboring pixels, we select a matching direction of the frame field using the standard least-angle matching criterion [Diamanti et al. 2015]. We then trace a curve perpendicular to this local direction field, extending the field each time we move to a neighboring pixel by the same procedure: We look at the new 8-pixel neighborhood and match the frame field directions. Once the orthogonal test curve is traced, we find its intersection points with the curves with the matching direction. We then form the group of intersection points I_v, the curve bundle associated with a graph vertex $v \in V$, in the following way (Fig. 6): Starting with the seed point, we group adjacent intersection points if they are less than one pixel apart. This strategy effectively groups curves along the width of the stroke without relying on an estimate of the stroke width, based only on the simpler assumption that different parallel strokes are separated by at least one pixel (Fig. 6(a,b)). We then add edges between vertices that have at least one pair of intersection points adjacent along their shared curve (Fig. 6(c,d)). To efficiently test for intersections, in our implementation we cache segments overlapping with each pixel; we then test for intersections only with the segments in the pixels overlapping with the test curve.

Using this graph, one can define an "*induced*" vectorization of the drawing with the same topology as the graph. Namely, each edge of the graph defines a set of curve segments connecting adjacent curve bundles. By choosing one of those segments per edge and connecting, if necessary, the ends of those segments with straight lines, we obtain a vectorization with the topology of the graph (Fig. 11). This vectorization is not unique, since edges are typically associated to more than one shared curve (Fig. 11(b)); we discuss the vectorization process further in Section 5.

4.2 Topology Simplification



An uneven narrow band boundary can affect the topology of the graph. In particular, it might introduce extraneous loops and branches (inset, Fig. 7(b), red). To account for this, we perform the topology simplification procedure below (inset, Fig. 7(b)-(c)).

Since we expect each loop to correspond to a hole in the narrow band, we contract each loop if its induced vectorization contains fewer than n_{hole} white pixels (inset, top). To distinguish true topological *branches*—valence-two paths ending with a leaf node—from

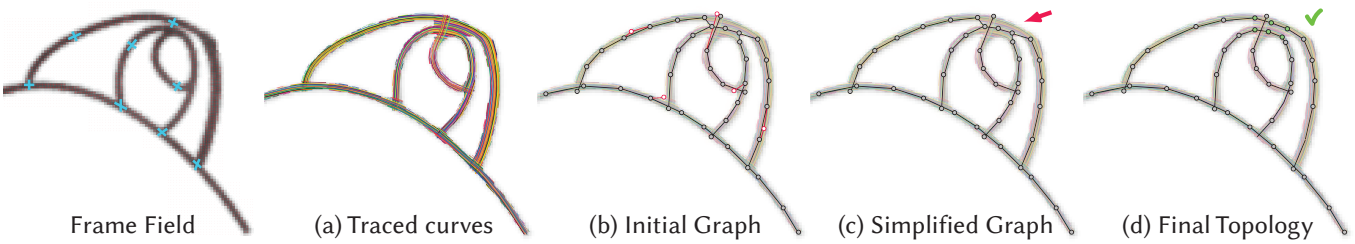
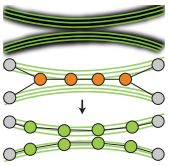


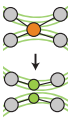
Fig. 7. Stages of extracting drawing topology. Starting with the frame field (left, zoom in for complete frame field), we first trace curves passing through each dark pixel (a). We then locally group curves into curve bundles, and use adjacency along curves to form the initial graph (b). We then perform topological simplification (c), and finally, disambiguate parallel strokes, resulting in the final graph (d).

extraneous ones, we use the following heuristic. For each vertex with valence greater than two, we repeatedly choose the shortest branch and prune it if its length outside the strokes formed by the other branches is too short (inset, bottom). In our implementation, for a given branch, we use a quarter of its full length, or a pixel, whichever is greater, as such threshold. To perform this test, we define stroke width at each vertex as the maximal distance between the intersection points of its curve bundle. Then, for each vertex of the branch we find the closest vertex not belonging to the branch and test if Euclidean distance between those is within sum of their stroke radii.

4.3 Disambiguating parallel strokes



In our final stage of topology extraction, we separate parallel strokes that are merged due to a connected narrow band segment (inset and Fig. 7(c)-(d)). This happens when two different but nearly parallel strokes touch or overlap (inset, top): In the overlap, the traced curves of the upper stroke will be naturally grouped with the traced curves of the lower stroke, forming the orange vertices. To resolve this, we first find paths of valence-2 vertices connecting pairs of vertices with valence 3 (inset, orange). Edges along these paths are split into two by duplicating their vertices; this procedure "unzips" the path connecting the degree-3 vertices (inset and Fig. 7(d), green vertices). The remaining edges at the degree-3 vertices that were not unzipped are assigned to new neighbors based on the connectivity of the underlying curve bundle.



At X-junctions, two strokes intersect but they do not share a vertex in our graph construction. Hence, vertices with valence 4 or higher are extremely rare. We split these vertices using the same unzipping technique, effectively treating them as a pair of degree-3 vertices connected by an edge of length zero.

4.4 Treating Frame Field Singularities

In contrast to frame fields applied to quad meshing [Diamanti et al. 2015], singularities in our frame field do not have meaningful interpretation; they usually are artifacts due to noise. Our insight is since singularities happen in the areas with significant noise, we can eliminate most of the singular points by relaxing the alignment term in those areas. Therefore, after the frame field optimization, we find singular pixels, set their alignment weight to zero, and re-run the optimization (3). We repeat this process, each time updating the alignment weights, until no more singularities can be resolved this way. Since each time we either reduce the number of non-zero alignment weights or stop, and since a frame field with no alignment term has no singularities, the process necessarily terminates. In practice, however, the alignment only needs to be relaxed for a small number of pixels, typically less than 1% of dark pixels.

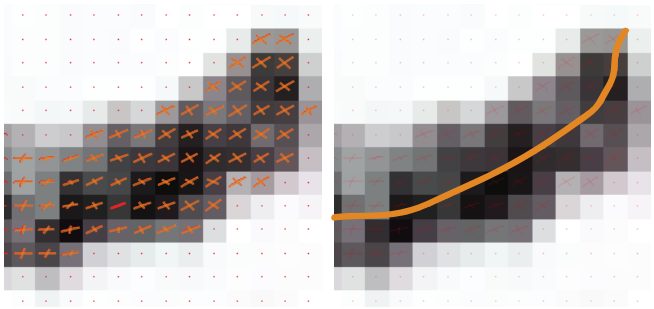


Fig. 8. (left) A typical singularity (highlighted in red) occurs at a point where the two frame directions coincide. (right) Final vectorization.

We address the remaining singularities (typically fewer than five singular pixels per image, Fig. 8) using a simple heuristic. First, we stop tracing curves at singular pixels. This ensures that no vertices in the topological graph with inconsistently matched frame field directions are connected. However, this may introduce a gap in a stroke in the final vectorization. To address this, we mark leaf vertices next to a singularity in the topological graph, and greedily connect each to the closest non-adjacent vertex.

5 VECTORIZATION

We use the extracted topological graph G to create the final vectorization. The key idea is to extract a vectorization that follows the traced curves as much as possible while having the topology of the graph. Additionally, we would like the vectorized curves close to the centerline of the stroke.

We initialize our procedure for embedding the topological graph by embedding vertices v whose degree does not equal two; these vertices correspond to isolated stroke endpoints as well as points where curve segments join together tangentially. Recall that v in the topological graph represents a *bundle* I_v of intersection points

between the traced curves and an orthogonal test curve (red points in Figure 6). Thus, a natural choice of embedding for each vertex v is one of the points in I_v. With this construction in mind, we approximate the drawing centerline at v as the barycenter b_v of I_v (Fig. 9, green points); b_v becomes the assigned position for v in our embedding (Fig. 11(b)).

What remains is to embed the degree-2 vertices and the edges of the graph. Our objective in this step is to embed edges as curves that approximately follow the traced curves without diverging too far from the stroke centerline. Each individual traced curve, even if it started at the stroke center, might drift away from the centerline. To account for that, we select a curve on a per-edge basis; our vectorization can "hop" from one traced curve to another at the vertices of the topological graph, in which case the two curve segments are connected using a straight line segment. The end result is a tracing that is composed piecewise of traced curves connected with short ligaments that subsequently can be smoothed. The details of this procedure are outlined below.

5.1 Auxiliary Graph

We cast the remaining embedding computation as a shortest-path problem over an *auxiliary graph* $G^{Aux} = (V_G^{Aux}, E_G^{Aux})$ constructed as follows:

- Vertices (V_G^{Aux}): The vertices of the auxiliary graph are defined as the union $\cup_{v \in V} I_v$, corresponding to the set of *all* intersection points between the traced curves and the test curves (yellow points in Fig. 9).
- Edges (E_G^{Aux}): Recall the vertices of G^{Aux} are clustered into sets I_v. For two vertices v_1, v_2 connected by an edge $(v_1, v_2) \in E$ in the topological graph (rather than G^{Aux}), we insert a bipartite graph of edges connecting all vertices in I_{v_1} to all vertices in I_{v_2}. Symbolically, we can write:

$$E_G^{Aux} = \{(p, q) : p \in I_{v_1}, q \in I_{v_2}, (v_1, v_2) \in E\}.$$

  Intuitively, by following an edge in the auxiliary graph, we connect the two intersection points with a segment of some traced curve shared by their curve bundles and, possibly, a straight line segment.
- Edge weights (w_e): The edge weight is designed so that the shortest path on the graph produces a curve that is smooth and centered. Thus, the weight of an edge e is a weighted sum of two terms:

$$w_e = E_{\text{connections}} + \eta E_{\text{centering}}.$$

  Roughly, the first term penalizes hopping from one curve to another when they are far away, in some sense favoring smoother connections; and the second term is designed to penalize connecting pairs of vertices that are far from the centerline (Figure 9).

  More concretely, the first term, $E_{\text{connections}}$, is computed as a sum of distances to the closest shared curve (for an orange edge on Fig. 9 it is the magenta curve):

$$E_{\text{connections}} = \min_{r_{1,2} \in I_{v_{1,2}}, c(r_1)=c(r_2)} [\|r_1 - p\| + \|r_2 - q\|],$$

  where $c(r)$ is the traced curve containing the intersection point r of some curve bundle. The centering term penalizes the distance from a vertex to the corresponding barycenter, i.e. for an edge $e = (p, q) \in E_G^{Aux}$ from bundles I_{v_1} and I_{v_2} with stroke widths w_{v_1}, w_{v_2} respectively,

$$E_{\text{centering}} = \frac{\|p - b_{v_1}\| + \|q - b_{v_2}\|}{w_{v_1} + w_{v_2}}.$$

  The centering weight η affects the exact locations of Y-junctions in ambiguous areas (Fig. 10); we use $\eta = 0.07$ in all our experiments.

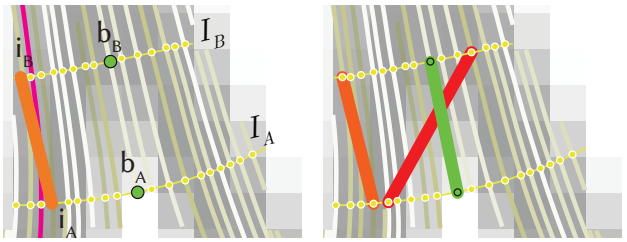


Fig. 9. A few vertices and edges of the auxiliary graph. Left: a cost of an edge (orange) between two vertices in the auxiliary graph is computed as a weighted sum of distances between $i_{A,B}$ and the barycenters $b_{A,B}$ ($E_{\text{centering}}$) and a sum of distances to the closest shared curve (magenta, $E_{\text{connections}}$). This penalizes edges corresponding to vectorizations far from center (orange) or vectorizations deviating from the traced curves (red). Using stroke width, computed in §4.2, in the centering term relaxes centering around Y-junctions, where the stroke width is locally high.

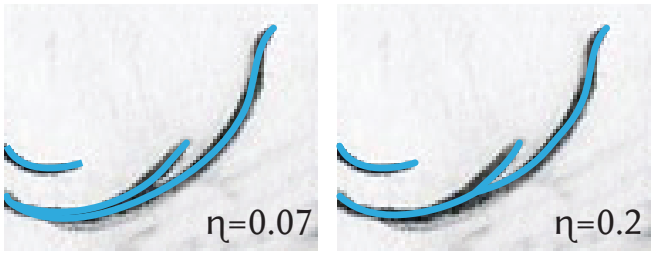


Fig. 10. The centering weight η controls the locations of Y-junctions.

5.2 Extracting the Vectorization

As an initial embedding of the full graph G, we simply connect the previously-embedded vertices (with degree not equal two) using shortest paths in this weighted auxiliary graph. This produces a vectorization with the correct topology, which is centered and follows the traced curves (Fig. 11(a,b)).

After this initial pass computes an embedding, we make a second pass to refine the result. Principally, we improve the locations of the valence-3 vertices, which can be suboptimal since they were chosen independently (Fig. 11(b)). Our procedure for moving the valence-3 vertices to improved locations is illustrated in Figure 11 and described below.

Valence-3 vertices typically correspond to acute junctions or Y-junctions. In this stage, we find optimal locations that provide a smooth transition between the joining curves. Thus, we attempt to further improve the total shortest-path cost over the graph while preserving topology. To do so, we allow each degree-3 vertex (inset, yellow) to snap to any barycenter along its outgoing degree-2 chains

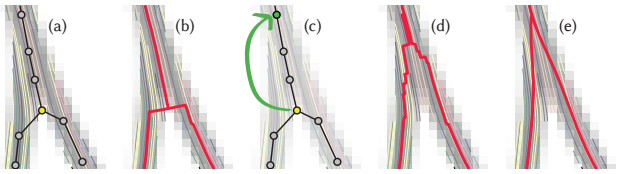


Fig. 11. Generating final vectorization from the topological graph. (a) Topological graph, (b) A vectorization, (c) Optimization result, (d) Optimized vectorization, (e) Final smooth result

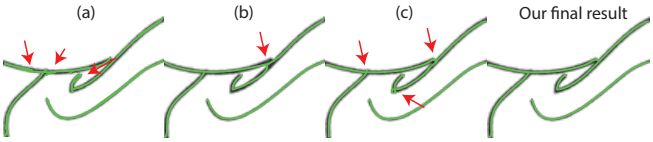
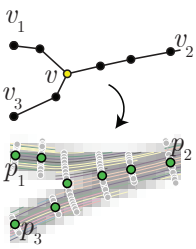


Fig. 12. (a) A result of our algorithm without topology simplification (Sec. 4.2). (b) A result without optimizing the locations of the degree-2 vertices (Sec. 5.2). (c) A result without post-processing stage (end of Sec. 5.2). (right) Our final result with all stages enabled.

in G (inset, green) and find the optimal locations for valence-3 vertices minimizing the total shortest-path cost on the auxiliary graph (G^{Aux}, gray vertices in the inset below).



If two degree-3 vertices are connected by a chain of vertices of valence two, we fix the location of the vertex closest to the middle of the chain that was not split during the procedure described in §4.2; this avoids having to solve a global problem to place all the degree-3 vertices simultaneously and is well-justified since traditional 1-skeleton-based image vectorization methods [Noris et al. 2013] perform well away from junctions. After this step, every valence-3 vertex $v \in V$ is connected, via chains of degree-2 vertices, to vertices $v_1, v_2, v_3 \in V$ with fixed locations $p_1, p_2, p_3 \in V_G^{Aux}$ (inset). Denoting the set of all the vertices in the degree-2 chains connecting v to v_i as $V_c \subseteq V$, we restrict the set of possible embedding locations for the valence-3 vertex v to the set of all curve bundle barycenters $B = \{b_{v'} | v' \in V_c\} \subseteq V_{G^{Aux}}$ (inset, green points). In particular, $p_i \in B$. We iterate over B to solve a discrete problem for the embedding of v:

$$\min_{p \in B} \sum_{i=1,2,3} d_{G^{Aux}}(p, p_i), \tag{6}$$

where $d_{G^{Aux}}(p, p_i)$ is the shortest-path distance on G_{Aux}.

A few post-processing steps conclude our second pass. Since intersecting strokes are separated in the graph construction and thus are traced independently, they may continue past the points where they should meet (see inset). To prune the resulting curve fragments, we add the intersection points into the graph and repeat the branch pruning procedure (§4.2). Finally, we optionally smooth the curves using Adobe Illustrator's 'Simplify Path' feature with the 95% curve precision and zero angle threshold (Fig. 11, (e) and Fig. 13). Alternatively, one may use the Douglas–Peucker algorithm, followed by Laplacian smoothing; this strategy produces similar results.

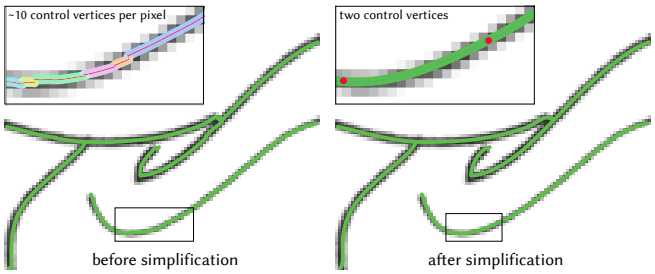


Fig. 13. In our implementation, we first output non-smooth curves (left) with control vertices sampled very densely (left, zoom, control vertices in red). We then use Adobe Illustrator's 'Simplify' feature that smooths and simplifies the curves (right). For example, after simplification the cut-out piece of curve (right, zoom) will only have two control vertices, making it easier to manipulate.

	input res.	n. of dark pixels	Noris et al. time	Favreau et al. time	our time
Muten	1024^2	35868	13s	375s	24s
Mouse	1024^2	50298	17s	341s	64s
Dracolion	1024^2	39402	15s	415s	25s
Sheriff	1024^2	50198	19s	437s	49s
Puppy	660x624	29908	26s	224s	41s
Hippo	700x535	25114	24s	120s	43s
Banana Tree	589x865	18619	15s	244s	23s
Penguin	500x714	24134	23s	181s	56s
Kitten	700x554	29023	38s	250s	81s
Elephant	500x753	34569	33s	270s	55s

Table 1. Algorithm statistics for different curve networks.

6 VALIDATION AND RESULTS

Qualitative Evaluation. We have automatically generated a number of vectorizations for line drawings of different style and level of noise (Figs. 14, 15, 16, 1, and 24 (green curves)). These included noisy, complex drawings ('Puppy,' 'Elephant,' 'Banana Tree'), some with varying stroke width ('Hippo,' 'Penguin'), www.easy-drawings-and-sketches.com, ©Ivan Huska, as well as high-resolution clean digital images ('Sheriff,' 'Dracolion,' 'Muten,' 'Mouse'). For all noisy images from the drawing tutorial website, to simplify line separation from the background, we automatically adjusted contrast in Adobe Photoshop. Alternatively, one may use an implementation of histogram equalization.

The puppy example (Fig. 15) has what Noris et al. [2013] call *spikes* on the sides of the face (Fig. 17). In the presence of noise, distinguishing those from true junctions is problematic, and the heuristic suggested by Noris et al. [2013] breaks down. Instead of relying on similar heuristics, we allow for a simple user interaction: the user is able to edit the narrow band as a bitmap image. For this example, using a couple of brush strokes within a few seconds, the

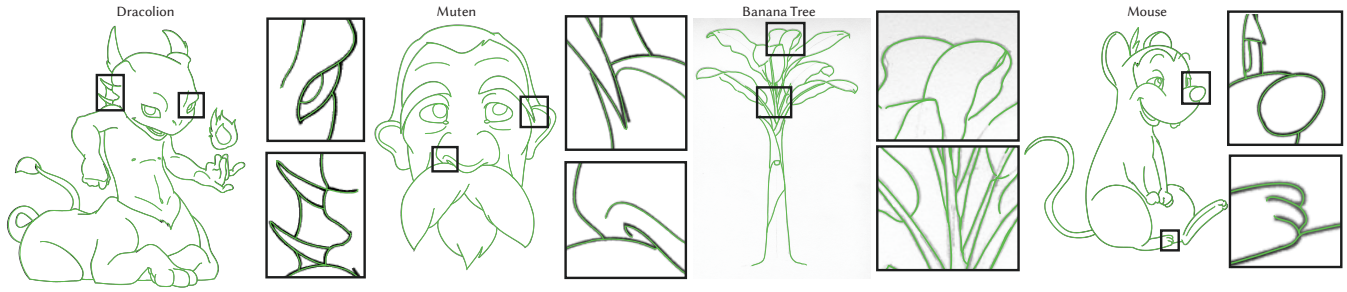


Fig. 14. A gallery of additional results.

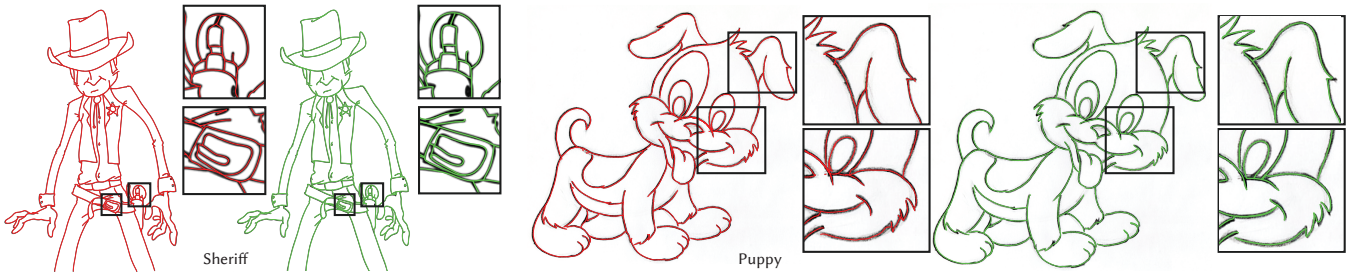


Fig. 15. The method by Noris et al. [2013] (red curves) is intended to work on clean drawings only. We (green curves) obtain results of similar quality to Noris et al. on clean inputs (left, sheriff). Our method is more robust to significant noise in the drawings (right, puppy).

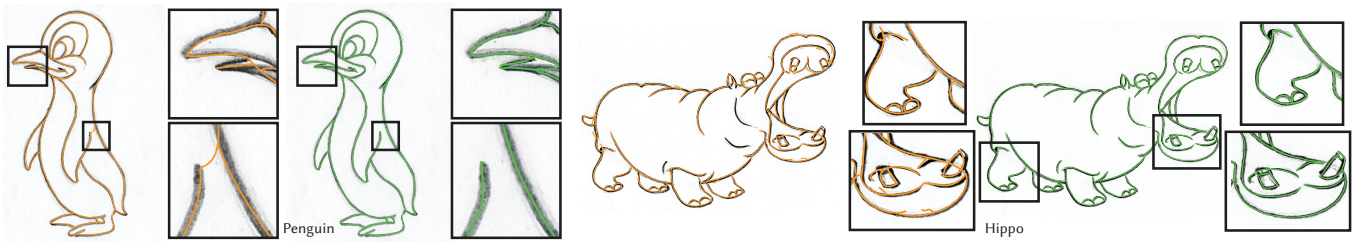


Fig. 16. Our method (green curves) is aimed at truthfully vectorizing images even in the presence of noise. The method by Favreau et al. [2016] (orange curves) is intended to work on drawings requiring significant topological simplification, and as such can be seen as complementary to our method.

Fig. 17. In the presence of noise, distinguishing junctions (a) from spikes (b) is problematic. Instead of relying on heuristics, we allow user to edit the narrow band, resulting in an arguably better interpretation (b).

user adjusted the narrow band to achieve the desired effect (Fig. 17, (b)). All other input images were processed in a fully automatic way.

Comparison to Prior Art. We compare our method to the most relevant recent work on vectorization, described in [Noris et al. 2013] and [Favreau et al. 2016] (Fig. 15,16).

To run the method by Favreau et al. [2016], we try two sets of input parameters: the default parameters in their implementation[1] and ones manually selected to improve results;[2] we keep the 'fidelity-simplicity' weight at the default value of 0.5 (λ in their formula (2)). To run the method by Noris et al. [2013], we try a set of parameter values, including the default parameters in their implementation,[3] and choose the best result. We tried both thresholding the initial images using our parameter value of θ_{noise}, as well as not thresholding. Optionally, we additionally run a post-processing step on the results by Noris et al. [2013] and Favreau et al. [2016]. For their methods, we chose the best results out of all those options on a per-input basis. We run our method with default parameters on all inputs.

[1] maxNumOpenCurves = 0, minLengthOpenCurves=30, minRegionSize=7
[2] maxNumOpenCurves = 30, minLengthOpenCurves=5, minRegionSize=3
[3] All combinations of: Maximal Interact Distance $\in \{0.5, 1.0, 1.1, 1.2\}$, Maximal Active Distance $\in \{0.4, 1.0, 1.1\}$, Direction Threshold $\in \{0.04, 0.05, 0.06\}$

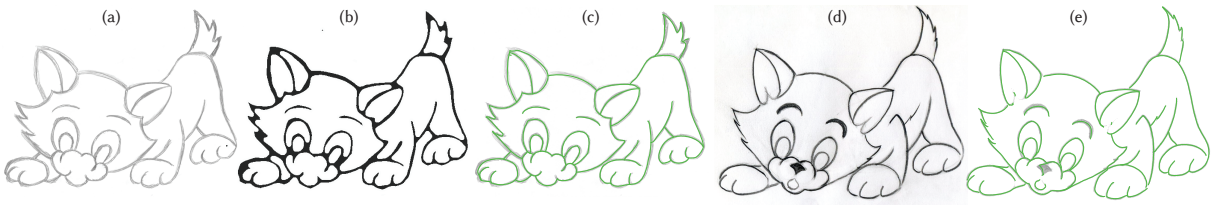


Fig. 18. Our method is robust to minor input changes: compare our result (c) for the input image (a), and our result (e) for a similar input image (d). Input image (d) is from www.easy-drawings-and-sketches.com, ©Ivan Huska; the other input image (a) is from [Favreau et al. 2016]. Since our method is not aimed at drawings with fuzzy lines, we have filtered the left sketch using [Bartolo et al. 2007] (b) but trace curves starting from dark pixels of the original sketch.

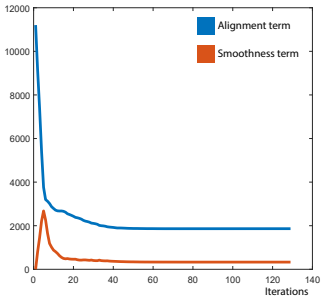


Fig. 19. A plot of energy terms in Equation 3 during iterations for a sample input (a cut of 'Penguin' image). The smoothness term is scaled by λ.

Fig. 20. Our method is robust to significant changes in the frame field smoothness weight λ. Increasing the weight makes junctions sharper at a cost of losing fine details in the drawing, so higher values of λ may be appropriate for very noisy drawings.

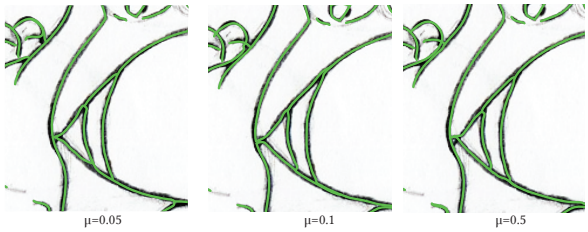


Fig. 21. Our method is robust to significant changes in the frame field regularizer weight μ.

On the clean digital inputs, our results are comparable to the ones by Noris et al. (Fig. 15, left). On the noisy inputs (Fig. 15), right, our variational method reliably disambiguates junctions, even with missing details and varying stroke width. The method by Noris et al. [2013] fails to disambiguate the complicated regions due to its discrete nature and heavy reliance on image gradients, e.g. puppy's eyes (red). Our method faithfully captures the principal directions

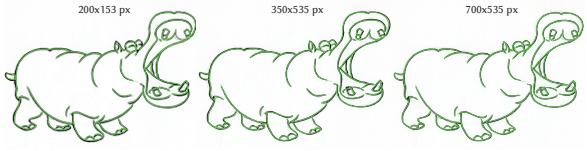


Fig. 22. In general, our method is robust to significant changes in resolution. However, at low resolutions small details might be indistinguishable from noise, and thus missing in the final result.

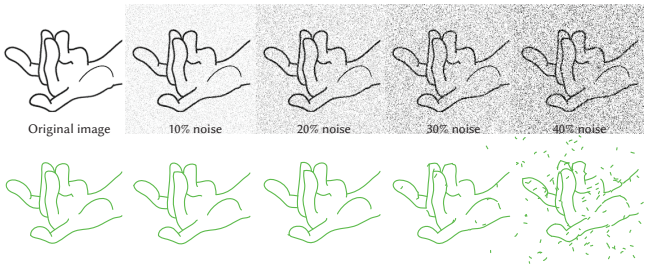


Fig. 23. Our method is robust under significant noise. Even with heavy noise, when discerning correct topology becomes problematic, junction accuracy remains stable.

and junctions even in those regions. We provide additional comparison results in the auxiliary materials.

We see the method by Favreau et al. [2016] as complementary to our method: their method works best when significant simplification of the curve network is needed, while our goal is to stay true to the drawing even in the presence of noise (Fig. 16). For sketches with multiple overlapping strokes, our method aims to reconstruct all the single pen strokes, and in some cases it may not be the desired behavior (Fig. 24 (c)). However, for those cases, our method may serve as a better input vectorization for further topological simplification [Favreau et al. 2016; Simo-Serra et al. 2016].

Our method is robust to changes in the input bitmap (Fig. 18), due in large part to its variational nature. Note that since drawings with multiple overlapping strokes, such as inputs in [Favreau et al. 2016], were not the focus of our work, we ran [Bartolo et al. 2007] on the left image, while tracing the curves from the dark pixels in the original sketch.

Noise robustness. We have evaluated noise robustness of our algorithm by testing on images polluted by various degrees of Gaussian

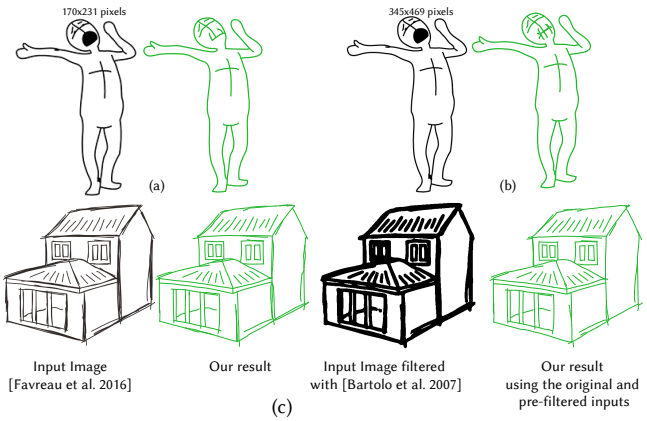


Fig. 24. While our method is robust even for low-resolution images, similarly to most methods in the category, we do not support shaded areas (a,b). Input image ©Ksenia Popova. Since our method is aimed at truthfully vectorizing images, rough sketches with multiple overlapping strokes may require additional simplification (c). To a certain degree, we can merge nearby parallel strokes by pre-filtering the input image with [Bartolo et al. 2007] ((c), right), but tracing from the original narrow band, similarly to Fig. 18.

noise (Fig. 23). In general, our method is robust to Gaussian noise; junction directions are particularly stable.

Benefits of individual steps. For completeness, we demonstrate effects of disabling significant steps of our algorithm in Figure 12.

Parameters and Processing Time. On a 4-core Intel i7-6700 @ 3.4Ghz with 32Gb RAM, our implementation usually takes from twenty seconds for lower-resolution images to a little over a minute on high resolution images. Due to the narrow-band optimization, our performance depends not on the image resolution, but rather on the number of dark pixels. Statistics for the images we tested are summarized in Table 1. We use the same parameters for all the images: $\theta_{\text{noise}} = 0.35, n_{\text{hole}} = 4$.

While most parameters in our method have a straightforward and intuitive effect on the result, we have two main nonlinear weights: the frame field smoothness weight λ, and the regularizer weight μ. In Figure 20, we demonstrate that our method produces reasonable output under significant variations of λ. Namely, increasing the weight sharpens the junctions, at a possible cost of losing some fine details in the drawings. Therefore, higher values of λ may be appropriate for noisier drawings. Our method also produces reasonable outputs for significantly different values of μ (Fig. 21).

While we kept parameters fixed, n_{hole} could be adjusted for drawings of different resolutions or noise structure. One could devise a heuristic to decrease it for lower resolutions, for instance, or use machine learning to calculate an optimal parameter for a class of drawings; we leave this for future work. We did not observe that keeping n_{hole} fixed caused any issues in our experiments.

Limitations. Similarly to most methods of this category [Bo et al. 2016; Favreau et al. 2016; Noris et al. 2013], our method works best on drawings without shading (Fig. 24(a,b), the nose of the cat in Fig. 18). On shaded images, user interaction might be necessary to achieve correct vectorization. Very low resolution images might be also challenging to vectorize (Fig. 22).

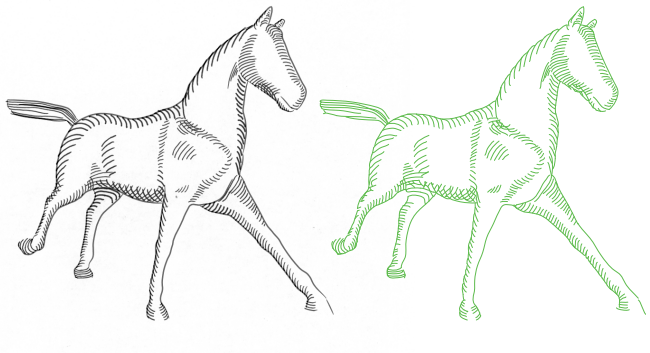


Fig. 25. Apart from line drawings, our method can be used to vectorize drawings in cross-hatching technique, when at most points there are only two curves crossing. Input drawing by Olga Vesselova [Kalogerakis et al. 2012].

A common alternative to shading is to convey information about shape and lighting via hatching [Kalogerakis et al. 2012]. Those drawings are quite different from typical line drawings: in a technique called cross-hatching, artists would often draw ink strokes of three or more distinct hatching directions in one region. Since our frame field captures only two directions at a point, our method is not designed to vectorize natural hatching images. Even so, our method is naturally suited for vectorizing hatching examples when mostly only two sets of directions are used in every region (Fig. 25).

7 CONCLUSION AND FUTURE WORK

We have presented a novel method for automatically vectorizing raster images, based on PolyVector field design. As we demonstrate on a gallery of examples, it reliably and efficiently disambiguates T- and X-junctions in both clean and noisy drawings, while staying true to curve shapes and connectivity. Our pipeline finds immediate application in artistic and engineering workflows, automatically providing a high-quality tracing without oversimplification or noise.

The presented method can be naturally extended to image domains where high-valence junctions are common, such as creating maps from GPS traces, by raising the degree of the PolyVector field polynomial (Eq. 1). Our preliminary experiments indicate it is indeed a promising direction, but special care must be taken to find consistent matchings of the frame field roots in the presence of noise.

Another interesting potential extension is to vectorization of animated sequences: while a temporal coherence term is trivial to add in our frame field design framework, tracing might need to be modified to avoid temporal artifacts.

8 ACKNOWLEDGMENTS

The authors acknowledge the generous support of Army Research Office grant W911NF-12-R-0011 (“Smooth Modeling of Flows on Graphs”), from the MIT Research Support Committee (“Structured Optimization for Geometric Problems”), and from the Skoltech–MIT

Next Generation Program ("Simulation and Transfer Learning for Deep 3D Geometric Data Analysis").

REFERENCES

David Adalsteinsson and James A. Sethian. 1995. A Fast Level Set Method for Propagating Interfaces. *J. Comput. Phys.* 118, 2 (1995), 269 – 277. https://doi.org/10.1006/jcph.1995.1098

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. (2007), 123–130. https://doi.org/10.1145/1384429.1384456

Mikhail Bessmeltsev, Will Chang, Nicholas Vining, Alla Sheffer, and Karan Singh. 2015. Modeling Character Canvases from Cartoon Drawings. *ACM Trans. Graph.* 34, 5, Article 162 (Nov. 2015), 16 pages. https://doi.org/10.1145/2801134

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

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. https://doi.org/10.1109/CVPR.2013.247

Jiazhou Chen, Gael Guennebaud, Pascal Barla, and Xavier Granier. 2013. Non-oriented MLS Gradient Fields. *Computer Graphics Forum* (Dec. 2013), p. https://hal.inria.fr/hal-00857265

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

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

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)*.

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)* (2016). http://www-sop.inria.fr/reves/Basilic/2016/FLB16

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* (2011). https://doi.org/10.1111/j.1467-8659.2011.02033.x

Chris Harris and Mike Stephens. 1988. A combined corner and edge detector. In *In Proc. of Fourth Alvey Vision Conference*. 147–151.

Aaron Hertzmann and Denis Zorin. 2000. Illustrating Smooth Surfaces. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '00)*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 517–526. https://doi.org/10.1145/344779.345074

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

Emmanuel Iarussi, David Bommes, and Adrien Bousseau. 2015. BendFields: Regularized Curvature Fields from Rough Concept Sketches. *ACM Trans. Graph.* 34, 3, Article 24 (May 2015), 16 pages. https://doi.org/10.1145/2710026

Evangelos Kalogerakis, Derek Nowrouzezahrai, Simon Breslav, and Aaron Hertzmann. 2012. Learning Hatching for Pen-and-Ink Illustration of Surfaces. *ACM Transactions on Graphics* 31, 1 (2012).

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 (NPAR '07)*. ACM, New York, NY, USA, 43–50. https://doi.org/10.1145/1274871.1274878

Gaetano Kanizsa. 1979. *Organization in vision : essays on gestalt perception*. Praeger, New York.

Michael Kass and Andrew Witkin. 1987. Analyzing Oriented Patterns. *Comput. Vision Graph. Image Process.* 37, 3 (March 1987), 362–385. https://doi.org/10.1016/0734-189X(87)90043-0

Gregory Lecot and Bruno Lévy. 2006. Ardeco: Automatic Region Detection and Conversion. In *Proceedings of the 17th Eurographics Conference on Rendering Techniques (EGSR '06)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 349–360. https://doi.org/10.2312/EGWR/EGSR06/349-360

Jorge Nocedal and Stephen J. Wright. 2006. *Numerical Optimization* (2nd ed.). Springer, New York.

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

Kazim Pal, Nicole Avery, Pete Boston, Alberto Campagnolo, Caroline De Stefani, Helen Matheson-Pollock, Daniele Panozzo, Matthew Payne, Christian Schüller, Chris Sanderson, Chris Scott, Philippa Smith, Rachael Smither, Olga Sorkine-Hornung, Ann Stewart, Emma Stewart, Patricia Stewart, Melissa Terras, Bernadette Walsh, Laurence Ward, Liz Yamada, and Tim Weyrich. 2016. Digitally Reconstructing The Great Parchment Book: 3D recovery of fire-damaged historical documents. *Literary and Linguistic Computing: the journal of digital scholarship in the humanities*, Oxford University Press (13 Dec. 2016), 1–31.

Jonathan Palacios and Eugene Zhang. 2007. Rotational Symmetry Field Design on Surfaces. *ACM Trans. Graph.* 26, 3, Article 55 (July 2007). https://doi.org/10.1145/1276377.1276446

Daniele Panozzo, Enrico Puppo, Marco Tarini, and Olga Sorkine-Hornung. 2014. Frame Fields: Anisotropic and Non-orthogonal Cross Fields. *ACM Trans. Graph.* 33, 4, Article 134 (July 2014), 11 pages. https://doi.org/10.1145/2601097.2601179

Yixuan Qiu, Naoaki Okazaki, and Jorge Nocedal. 2016. LBFGS++, A Header-only C++ Library for L-BFGS Algorithm. (2016). https://yixuan.cos.name/LBFGSpp/

Edgar Simo-Serra, Satoshi Iizuka, Kazuma Sasaki, and Hiroshi Ishikawa. 2016. Learning to Simplify: Fully Convolutional Networks for Rough Sketch Cleanup. *ACM Trans. Graph.* 35, 4, Article 121 (July 2016), 11 pages. https://doi.org/10.1145/2897824.2925972

Richard Szeliski. 2010. *Computer Vision: Algorithms and Applications* (1st ed.). Springer-Verlag New York, Inc., New York, NY, USA.

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

Amir Vaxman, Marcel Campen, Olga Diamanti, Daniele Panozzo, David Bommes, Klaus Hildebrandt, and Mirela Ben-Chen. 2016. Directional Field Synthesis, Design, and Processing. *Computer Graphics Forum* (2016). http://graphics.tudelft.nl/Publications-new/2016/VCDPBHB16

Brian Whited, Gioacchino Noris, Maryann Simmons, Robert W. Sumner, Markus Gross, and Jarek Rossignac. 2010. BetweenIT: An Interactive Tool for Tight Inbetweening. *Computer Graphics Forum* 29, 2 (2010), 605–614. https://doi.org/10.1111/j.1467-8659.2009.01630.x

Gui-Song Xia, Julie Delon, and Yann Gousseau. 2014. *Accurate Junction Detection and Characterization in Natural Images*. Vol. 106. Kluwer Academic Publishers, Hingham, MA, USA. 31–56 pages. https://doi.org/10.1007/s11263-013-0640-1

Eugene Zhang, James Hays, and Greg Turk. 2007. Interactive Tensor Field Design and Visualization on Surfaces. *IEEE Transactions on Visualization and Computer Graphics* 13, 1 (Jan. 2007), 94–107. https://doi.org/10.1109/TVCG.2007.16

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