Integer-Grid Sketch Simplification and Vectorization

Tibor Stanko, Mikhail Bessmeltsev, David Bommes, Adrien Bousseau

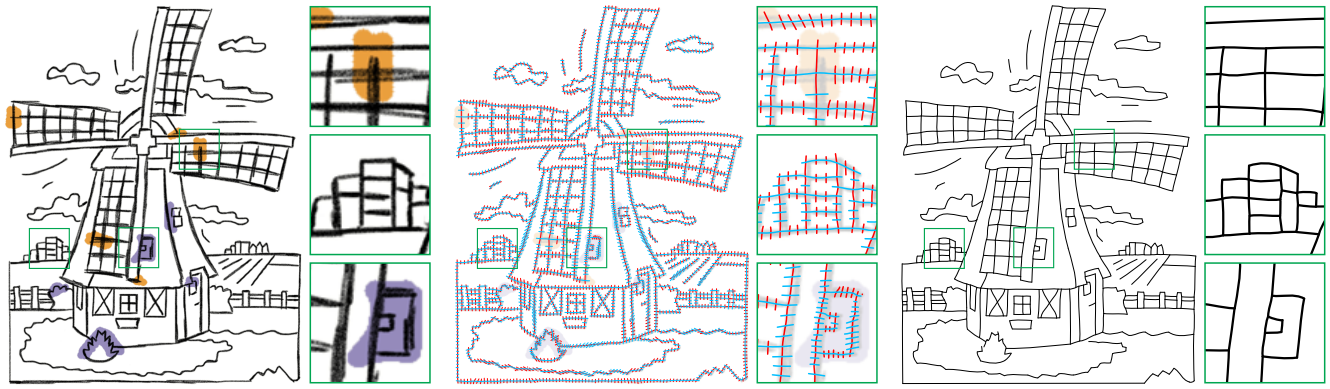
Eurographics Symposium on Geometry Processing 2020
Q. Huang and A. Jacobson
(Guest Editors)

Volume 39 (2020), Number 5

Integer-Grid Sketch Simplification and Vectorization

Tibor Stanko[1], Mikhail Bessmeltsev[2], David Bommes[3] and Adrien Bousseau[1]

[1] Université Côte d'Azur, Inria
[2] Université de Montréal
[3] University of Bern



a) input line drawing and mask *b) stroke-aligned parametrization* *c) output curve network*

Figure 1: *Starting from an input line drawing (left), we locally parametrize the sketch as a grid aligned with the strokes (middle). Neighboring parallel strokes are automatically snapped to the same isoline of the parametrization, while junctions are snapped to grid nodes. This parametrization facilitates the extraction of a clean network of Bézier curves (right). Using a simple mask, the user can locally specify the desired amount of simplification in the output (purple scribbles: less simplification, orange scribbles: more simplification). See supplemental materials for a result without the mask.*

Abstract

A major challenge in line drawing vectorization is segmenting the input bitmap into separate curves. This segmentation is especially problematic for rough sketches, where curves are depicted using multiple overdrawn strokes. Inspired by feature-aligned mesh quadrangulation methods in geometry processing, we propose to extract vector curve networks by parametrizing the image with local drawing-aligned integer grids. The regular structure of the grid facilitates the extraction of clean line junctions; due to the grid's discrete nature, nearby strokes are implicitly grouped together. We demonstrate that our method successfully vectorizes both clean and rough line drawings, whereas previous methods focused on only one of those drawing types.

CCS Concepts

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

1. Introduction

The goal of sketch vectorization is to convert a bitmap line drawing into a compact set of parametric curves. Existing methods tackle this problem in two steps: they first segment the drawing into individual curves, and then fit a parametric spline on each curve. However, segmenting a line drawing is a difficult task, especially in the presence of thick, over-sketched strokes. In such cases, existing methods tend to either produce too many curves and misinterpret junctions, or tend to miss details in an attempt to simplify the drawing.

Our algorithm addresses these challenges by reasoning at the scale of the entire sketch rather than at the scale of individual curves. In particular, while prior methods solve multiple, separate curve parametrization problems, we express sketch vectorization as a single *curve network* parametrization problem. This global approach

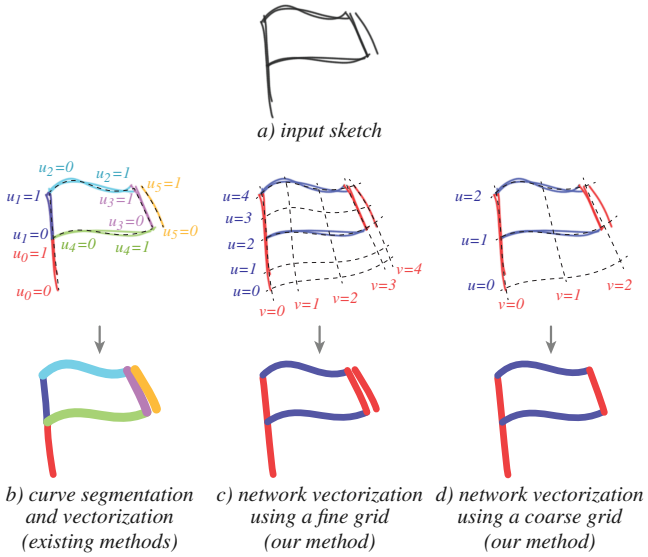


Figure 2: *Intuition behind our approach. Given a bitmap sketch (a), existing methods segment the drawing into individual curves, each being parametrized separately (b). We instead align the drawing with a grid, which parametrizes the entire curve network at once (c). The scale of the grid gives a direct control over how neighboring curves get merged together (d, top right of the sketch).*

provides increased robustness to over-sketching, and helps locate junctions.

Fig. 2 illustrates the intuition behind our approach on a simple sketch composed of several intersecting curves. Existing methods would segment the drawing and parametrize each curve with index i with its own parameter u_i (Fig. 2b). Our key idea is to instead segment the drawing into two families of curves roughly aligned with the two locally-dominant directions of the sketch (red and blue curves in Fig. 2c). We then assign a single parameter to each of these directions to obtain a 2D grid-like structure (Fig. 2c, u and v parameters).

The above construction allows us to parametrize all curves jointly, without requiring an explicit segmentation of the drawing into individual curves. To reach this goal, we first need to label pixels as being part of the first or second family of curves. We perform this labeling by analyzing local orientations of the drawing, as given by a smooth *frame field* aligned with the drawing. We then solve for a parametrization whose isolines follow the assigned frame field directions, and extract our curve network from the isolines tangent to the sketch. We introduce a novel regularization strategy to compute a frame field that captures well acute junctions while being as orthogonal as possible away from them.

We next constrain this representation further by imposing that the parameter running transversally to a curve in the drawing takes on an integer value along that curve. For example, in Fig. 2c, the u parameter takes on the integer value 2 along the bottom blue curve, and the value 4 along the top one. This constraint encourages neighboring parallel strokes to snap to the same integer isoline of the parametrization, effectively clustering them to form a single curve in the output. Similarly, this formulation automatically snaps intersecting curves to a single node of the integer grid, forming clean T- and X- junctions. The scale of the integer grid provides an intuitive control over the minimum spacing that we allow between nearby strokes before considering that they represent the same curve. A coarser scale yields a more aggressive simplification of the sketch, as shown in Fig. 2d where two red curves get snapped to the same isoline $v = 2$. This scale can even be adjusted locally to capture varying levels of details (Fig. 11).

While the simple example in Fig. 2 is well captured by a single integer grid, we cope with more complex configurations – including non-quad shapes – by restricting the parametrization to a narrow band of pixels around the strokes. As shown in Fig. 1c, this representation forms local grids at junctions, yet adapts to the intricate topology of entire curve networks.

In summary, we propose to cast line drawing vectorization as a grid parametrization problem. Our formulation brings several solutions to common vectorization challenges, including the automatic discovery of the line drawing topology, the precise reconstruction of junctions, and a direct control over the feature size under which neighboring strokes get merged together. In contrast to existing methods tailored to either clean or rough drawings, we show that our method produces high quality curve networks from both types of input. The source code of our implementation and the supplemental materials are available at `https://repo-sam.inria.fr/d3/grid-vectorization/`.

2. Related work

Here we discuss prior work related to the application we target – line drawing vectorization. We introduce technical background on quad-meshing algorithms that inspired our approach in Section 4.

Sketch vectorization. Many line drawing vectorization algorithms follow a two-step process, where skeletonization or tracing is first applied to locate the curves of the drawing, which are subsequently approximated with parametric splines [HT06; NHS*13; DCP17; PPM18; BF12; NS19]. Nevertheless, skeletonization and tracing methods often produce spurious branches at junctions or in the presence of multiple overlapping strokes. Favreau et al. [FLB16] address this challenge with an optimization procedure that simplifies the skeleton topology as long as it remains faithful to the drawing. However, this optimization can only merge consecutive curves, not parallel ones, which motivated them to use a region-based skeleton algorithm to locate curves composed of overlapping strokes. While region-based skeletonization is very effective on rough drawings, it often removes fine details in clean drawings composed of little regions, and does not extract open curves. A similar region-based approach is taken by Chen et al. [CDQM18], where they simplify the drawing topology by iteratively discarding small narrow regions formed by adjacent parallel strokes. They remove the remaining open curves using a heuristic. Unfortunately, that method uses absolute distance thresholds carefully chosen on a per-input basis.

Kim et al. [KWÖG18] propose a deep-learning algorithm to segment a line drawing into individual curves, effectively bypassing

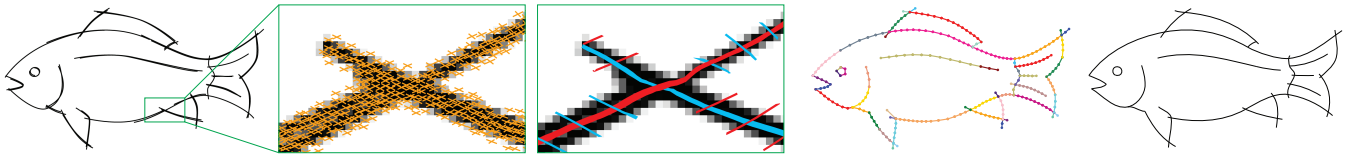


Figure 3: *Overview of our method. Starting with a bitmap line drawing, we first compute a frame field to recover the two local dominant directions at each pixel (a). We use this field to guide a grid-based parametrization whose isolines snap to strokes, and nodes snap to junctions (b). Tracing the isolines tangential to the strokes gives a parametrized topological graph of the drawing (c), to which we fit a network of vector curves (d).*

the challenges raised by skeletonization. Following a similar deep-learning methodology, Guo et al. [GZH*19] propose several deep networks to perform skeletonization, junction detection, and prediction of the local topology of each junction. These predictions are then used to guide curve tracing. However, both methods are designed and trained to process clean drawings and do not directly generalize to rough sketches (see [GZH*19, Figs. 14, 16]).

Closer to our work is the method by Bessmeltsev and Solomon [BS19], which vectorizes line drawings by tracing streamlines along a frame field aligned with the input drawing. Their algorithm first generates many streamlines of the frame field, and then groups them together to form the final curve network. However, this last grouping step does not simplify the drawing topology, and thus tends to produce many parallel curves in the presence of over-sketched strokes (Fig. 12). We draw inspiration from their use of a frame field to guide the vectorization. However, we propose to rely on a global parametrization rather than on local curve tracing, which brings robustness to over-sketched strokes.

Our approach also relates to the image vectorization algorithm of Wei et al. [WZG*19], which relies on a feature-aligned quadrangulation to convert photographs into colored meshes. However, vectorizing line drawings raises different requirements than vectorizing photographs. In particular, the sketches we target contain overdrawn strokes, which we simplify by snapping them to the same integer isolines of a global parametrization. In contrast, Wei et al. [WZG*19] only seek to align the mesh to image edges, which they achieve by local projection. Other aspects of our approach differ to better account for specificities of line drawings, such as the use of a frame field to capture non-orthogonal junctions, or the use of a narrow band surrounding pen strokes to adapt to intricate details.

Bitmap sketch filtering. Rough sketches are often composed of multiple, nearly parallel strokes. This observation motivated researchers to filter such sketches using anisotropic blurring kernels [KLC07; CGBG13; BCF*07]. A common idea of these methods is to orient the blurring kernels according to a vector field tangential to the strokes. However, tangent vector fields become ill-defined at junctions, where multiple dominant directions exist. Frame fields [VCD*16] offer a solution to this challenge by representing two dominant directions at each point of the drawing, allowing the accurate capture of T- and X-junctions. Singularities of the field can also capture Y- or high-valence junctions. Several recent methods take advantage of the frame field representation for sketch processing, including the aforementioned work of Bessmeltsev and Solomon [BS19] on line drawing vectorization, and the methods by Iarussi et al. [IBB15] and Li et al. [LPL*17] for sketch-based modeling. We take inspiration from this family of work and rely on a frame field to guide our grid-based parametrization.

While the above sketch filtering methods employ hand-crafted blurring kernels, Simo-Serra et al. [SISI16; SII18a] showed how very effective filters can be learned from examples using convolutional neural networks. Xu et al. [XXM*19] improves on that work by using a multi-layer discriminator trained with a perceptual loss. Their network better preserves global structures and fine details. Nevertheless, these local filters tend to either remove intended lines or retain spurious parallel strokes, as shown in Fig. 10. Since sketch filtering is typically a pre-process for sketch vectorization, we see bitmap filtering methods as largely complementary to ours.

Vector stroke aggregation. While we target the vectorization of *bitmap* drawings, our approach relates to the methods that process *vector* drawings created with digital devices. In particular, several algorithms have been proposed to aggregate pen strokes to form clean vector curves [BTS05; LWH15; LRS18; OK11; LLBG19]. A common, basic step in those algorithms is to cluster strokes based on proximity, continuity, or parallelism. While our method does not have access to individual pen strokes, we express our parametrization as an integer grid to implicitly cluster nearby parallel strokes together. As shown in our experiments (Section 7), our method produces vectorizations of comparable quality to stroke aggregation algorithms, even though we take as input raster images rather than more informative vector drawings.

Interactive filtering and vectorization. Distinguishing between intended details and spurious strokes is an ambiguous task, which motivates the development of interactive methods where users can iteratively correct the output of an automatic algorithm. The system proposed by Jun et al. [JHWS17] supports several brushes to merge, split or smooth vectorized regions, but as such is limited to closed contours. Alternatively, Simo-Serra et al. [SII18b] trained a deep convolutional neural network to filter sketches conditioned on user-provided scribbles. Their system supports several brushes to indicate strokes that should be added to or removed from the filtered output, but it does not support *merging* neighboring strokes. Our vectorization algorithm is the first to offer local control over the scale at which neighboring strokes should be merged.

3. Overview

Fig. 3 provides a visual overview of our grid-based algorithm for line drawing vectorization.

The first step of our approach is to propagate the local stroke tangents to obtain the two dominant directions at every point in the vicinity of the drawing (Fig. 3a). We represent this information with two directions, forming a *frame field* [VCD*16] over the drawing. We optimize this frame field to best align with the drawing while being as smooth as possible. We then assign each direction one of two labels to locally decompose the drawing into two families of roughly-parallel curves.

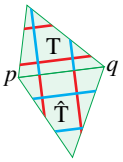
The second step of our approach is to parametrize the two families of curves globally (Fig. 3b). Our target application, sketch vectorization, imposes several desiderata on this grid-like parametrization. First, we would like the isolines of the two parameters, u and v, to align with the two local dominant directions of the sketch encoded by the frame field. Second, we would like the curves of the drawing to snap to integer isolines of the parametrization, which implicitly groups nearby strokes together and facilitates subsequent curve extraction. Finally, we would like to offer user control over the scale of the grid, which defines the minimum spacing under which nearby strokes get grouped. We express these desiderata as energy terms in an optimization, which combines discrete and continuous unknowns.

Our parametrization step assigns each pixel of the drawing to an integer isoline of a local parametric grid, and assigns each junction to an integer node. The last step of our approach is to extract the corresponding topological graph (Fig. 3c), and convert the associated pixels into a parametric curve network (Fig. 3d).

In what follows, we call *stroke pixels* the dark pixels of the input drawing. Our goal is to convert the input into a set of *vector curves*.

4. Background on Integer-Grid Parametrization

Our approach bridges two fields of research by showing how the challenging problem of sketch vectorization can benefit from grid-based parametrization algorithms developed by the quad-meshing community [BLP*13]. Many of these algorithms follow a common three-step procedure, where the dominant directions of the surface are first estimated by computing a feature-aligned direction field, this field is then used as a guidance to compute a seamless parametrization of the surface, from which integer isolines are extracted to form the quad mesh. Similarly to our approach, constraining the parametrization to take integer values along salient feature lines of the surface is a convenient way to snap the edges of the quad mesh to these lines [BZK09].



Since complex surfaces often cannot be parametrized by a single continuous chart, the above algorithms start by assigning one chart per triangle of the input mesh, and ensure that they form a globally seamless parametrization by enforcing compatibility between their respective integer grids. Given two adjacent triangles, their integer grids connect seamlessly if they can be related by a rotation of an integer multiple of $\frac{\pi}{2}$ and an integer translation. The composition of the integer rotation and translation is called a *grid-preserving transition function*:

$$\begin{aligned}(\hat{u}_p, \hat{v}_p) &= \mathrm{R}^k(u_p, v_p) + (i, j),\\ (\hat{u}_q, \hat{v}_q) &= \mathrm{R}^k(u_q, v_q) + (i, j),\end{aligned} \tag{1}$$

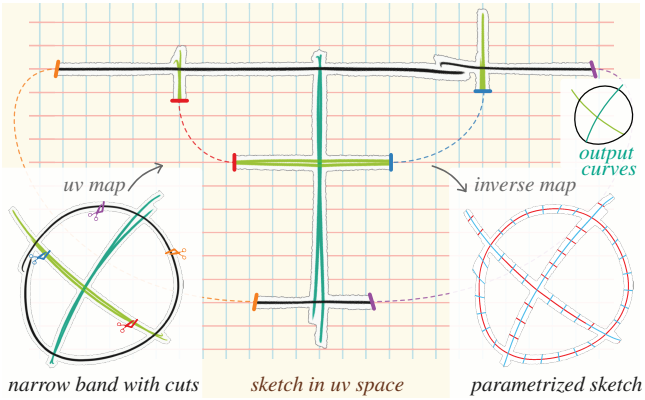


Figure 4: *In order to globally parametrize the input line drawing, each connected component of the narrow band needs to be cut open into a topological disk. A seamless parametrization is obtained by constraining the two sides of a cut to be related by a grid-preserving transition function, see Eq. (1).*

where p and q are the vertices shared by the two triangles (see inset), and R^k denotes rotation by $k\frac{\pi}{2}$. The integer k is often called the *period jump* [LVRL06].

The optimal rotation between two adjacent triangles can be deduced from the optimized frame field by comparing their respective frames. The integer translation (i, j) is typically optimized using mixed-integer solvers. However, enforcing the constraints (1) naively yields two integer variables per edge, making the optimization very slow. Fortunately, many of these variables can be eliminated by forming a single chart per connected component of the mesh, such that interior edges have fixed transition functions. The only remaining integer variables are then located along boundaries of this chart – the so-called *cut edges* – which correspond to discontinuities of the parametrization (Fig. 4). One way to build this chart is to start from a random triangle and connect it to all other triangles by tracing a dual spanning tree in a breadth-first manner [BZK09]. Every time the tree crosses an edge, the frame of the triangle it enters is *combed*, i.e. rotated by a multiple of $\frac{\pi}{2}$ such that the corresponding period jump vanishes, while the integer translation of the edge is fixed to 0. The number of integer variables can be further reduced by also eliminating the integer translation along cut edges that form open paths that do not end at a frame field singularity. We follow this standard procedure to drastically decrease the number of integer variables of our problem. For example, it removed more than 99% integer variables for the windmill sketch shown in Fig. 1.

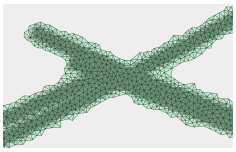
The last step of grid-based meshing algorithms is to extract a quad mesh by mapping an integer grid to the input surface using the inverse of the parametrization. The extraction has to overcome several issues, notably inaccuracies due to floating-point arithmetic and presence of inverted or degenerate parametric triangles. To address these issues, Ebke et al. [EBCK13] proposed a robust algorithm using exact predicates. The algorithm starts by mapping grid nodes (i.e. points with integer coordinates) to the surface, then connects nearby grid nodes with edges by locally tracing integer isolines, and finally connects edges into quads. We adopt this approach, even though only the nodes and edges are relevant in our context.

5. Pre-Processing

Our method starts with a few pre-processing steps to locate the strokes, estimate their width, and generate a triangle mesh around them that will serve as support for the parametrization.

We first identify stroke pixels by thresholding the input image, using a manual threshold for simplicity. We then compute the L1 distance of each stroke pixel to its closest background pixel, which provides an estimate of the local stroke width along the stroke centerlines. We turn these local estimates into a global one by averaging the centerline values over the entire drawing. We refer to this quantity as the average stroke width $\bar{\omega}$, which will serve to adjust the scale of the parametrization.

Next, we discretize the image space via a triangulation that we compute using TRIANGLE [She96]. Using a custom triangulation (rather than pixels) for discretization allows us to decouple the complexity of the optimization from the resolution of the input drawing. We set a bound on the size of triangles by requiring the circumdiameter to be smaller than $\bar{\omega}/4$. We provide results for different mesh resolutions in the supplemental materials, showing that the quality of the parametrization decreases with decreasing mesh resolution.



Finally, we define a narrow band by selecting triangles with a barycenter at a distance less than $\beta\bar{\omega}$ from a stroke pixel (see inset). We use the average stroke width $\bar{\omega}$ to make the user-specified parameter β resolution-independent. For clean inputs, we typically use β between 0.0 and 0.3. Over-sketched inputs require a wider band, which we obtain using β between 0.5 and 1.2. This restricted triangulation is the domain on which we compute our grid parametrization.

6. Grid-Based Vectorization

We now detail the main components of our method – drawing-aligned frame field, grid parametrization, and isoline extraction.

6.1. Frame Field

Computation. The first step of our method is to compute a frame field representing the two dominant directions at each point in the vicinity of the drawing. We achieve this goal using the energy formulation proposed by Bessmeltsev and Solomon [BS19], which uses the *PolyVector field* representation of Diamanti et al. [DVPS14]. In a nutshell, the energy combines three terms responsible for aligning the frame field with the strokes of the drawing, smoothing the frame field away from the strokes, and a regularizer encouraging the frame field to form orthogonal crosses rather than to collapse to a line field.

However, we have observed that encouraging frame orthogonality uniformly can degrade results in the presence of sharp angles in the drawing, as shown in Fig. 5a. Instead, we encourage orthogonality only where needed using iterative re-weighting of the regularization term. We initialize the frame field by setting the regularization weight to 0 for all vertices. We then update the frame field iteratively, checking which frames are degenerate at each iteration and incrementing their weights by 1. We consider that a frame is degenerate

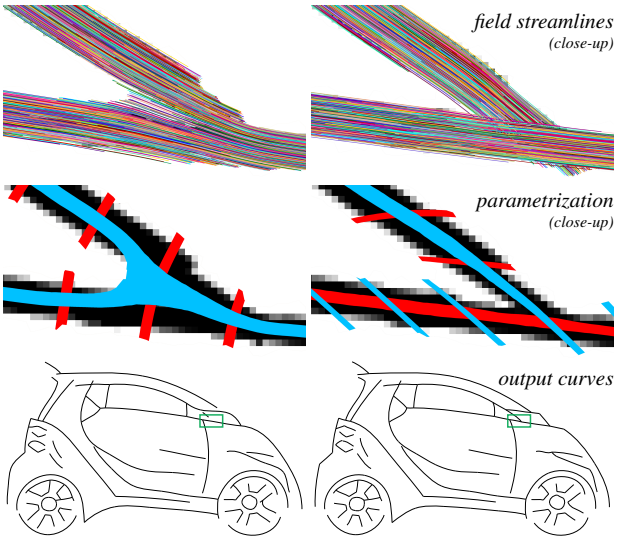
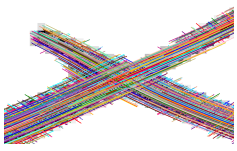


Figure 5: *Around junctions with sharp angles, the uniform regularizer of Bessmeltsev and Solomon [BS19] tends to favor orthogonality over alignment to strokes, yielding incorrect junctions in the parametrization (a). Our frame field formulation respects the alignment constraints near junctions while preventing the field from collapsing away from them (b). This significantly improves the overall quality of vectorizations containing sharp angles. Note that the frame field is visualized via streamlines in tangent direction only. See supplemental materials for the input bitmap.*

if the angle between the two vectors is less than 30 degrees, or if the magnitude of one of the vectors is less than 10% of the magnitude of the other vector. We stop iterating as soon as less than 1% of frames are degenerate, or when the maximum number of iterations is reached (set to 100 in our experiments). In practice, this procedure usually stops after 5 to 10 iterations (see Table 2 for statistics). Each iteration requires solving a sparse positive-definite least-squares problem, which we do using the Cholesky factorization. Fig. 5b shows how our novel adaptive regularization better aligns the frame field and subsequent parametrization to the drawing even at acute junctions.

We refer the interested reader to the paper by Bessmeltsev and Solomon [BS19] for other implementation details.

Orientation labeling. Once computed, we use the frame field to assign strokes to one of the two dominant local directions. This labeling will be later used by our parametrization energy to snap the corresponding coordinate to an integer value. Our goal is to estimate which direction of the frame field is most aligned with the stroke in a small neighborhood of each stroke triangle.



To do so, we start two streamlines from the barycenter of the triangle, and trace them along the two directions of the frame field, taking care of applying rotational part of the transition functions between triangles (see inset). We stop the tracing when a streamline

leaves the narrow band or when the distance from the barycenter is bigger than a threshold, fixed to $4\bar{\omega}$ in our implementation. We then count the number of stroke pixels k_0, k_1 encountered along each of the streamlines, and assign the stroke triangle to the direction that covers more stroke pixels. We keep the triangle unlabeled if the ratio $k_0/(k_0+k_1) \in [0, 0.5]$ of number of stroke pixels $k_0 \leq k_1$ covered by the two streamlines is above a threshold, fixed to 0.48 in our implementation. We provide results for different thresholds in supplemental materials, showing that our preset improves the snapping of the parametrization to the input curves. As a pre-process, we apply a morphological dilation of $\lfloor \bar{\omega}/4 \rfloor$ pixel(s) to the input drawing to thicken the strokes.

6.2. Parametrization

Equipped with a guidance frame field, the core of our method computes a parametrization of the drawing that maps the sketched lines to integer parametric isolines (Fig. 3c). We optimize this parametrization according to two energy terms, responsible for aligning the parametric isolines to the frame field and snapping stroke pixels to integer isolines. In addition, we constrain the isolines to be continuous over the triangulated narrow band. We now describe each of these components in detail.

Alignment term. Let $\phi = [\mathbf{s}, \mathbf{t}]$ be the two normalized directions of the (combed) frame field in a triangle $T \in \mathcal{T}$. We seek to find a map $f : c \mapsto (u, v)$ that assigns parametric coordinates to triangle corners $c \in \mathcal{C}$. In order to locally explain the input sketch by a grid, we would like the two coordinates u and v of the parametrization to vary most in the directions given by ϕ. This implies that the Jacobian $\mathcal{J}_f = \begin{pmatrix} \partial u/\partial x & \partial u/\partial y \\ \partial v/\partial x & \partial v/\partial y \end{pmatrix}$ has to map the sketch-aligned frame ϕ to the canonical frame $\mathrm{I} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$, yielding the energy term

$$E_{\text{align}} = \sum_{T \in \mathcal{T}} \|\mathcal{J}_f \tilde{\phi} - \mathrm{I}\|_{\mathrm{F}}^2. \tag{2}$$

Here, $\tilde{\phi} = s\phi$ denotes a scaled frame. The factor $s > 0$ controls the scale of the grid: small values of s result in a fine grid, while big values of s result in a coarse grid. In practice, we make s proportional to the average stroke width $\bar{\omega}$ such that a bigger scale is used over thick strokes. We additionally provide user control over the scale factor by means of a multiplicative factor λ, yielding the expression $s = \lambda\bar{\omega}$. We use $\lambda = 1$ for clean drawings and $\lambda > 1$ for inputs that require simplification (see Table 2).

Snapping term. An important intuition behind our method is that nearby strokes can be grouped together by assigning them to the same, quantized isoline of the parametrization. A similar goal exists in quad meshing algorithms that seek to snap integer isolines of the parametrization to feature lines of the 3D surface [BZK09]. A common strategy to achieve this goal is to constrain the parametrization to take on an integer value along those points. However, imposing this constraint strictly makes the triangles covering neighboring pixels collapse to a single line, which yields numerical challenges for the subsequent extraction of the curve network. We instead express this constraint as a soft energy term that attracts the constrained

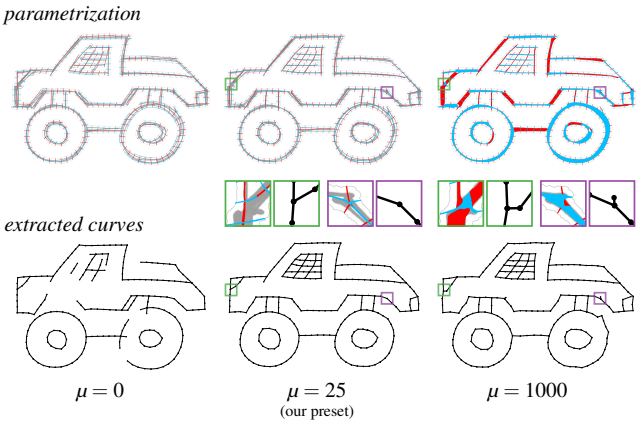


Figure 6: *If the snapping term is disabled ($\mu = 0$) the isolines fail to closely follow the input curves. Too much snapping ($\mu = 1000$) creates degeneracies in the parametrization. We use $\mu = 25$ to balance these two extremes.*

parameters of stroke triangles to the value of their nearest integer:

$$E_{\text{snap}} = \sum_{c \in \mathcal{C}_s} w_{\text{snap}} \|u - \bar{u}\|^2 + \sum_{c \in \mathcal{C}_t} w_{\text{snap}} \|v - \bar{v}\|^2, \tag{3}$$

where $\mathcal{C}_s$ and $\mathcal{C}_t$ denote the sets of stroke triangle corners assigned to the $\mathbf{s}$ and $\mathbf{t}$ directions of the frame field, respectively, and $\bar{u}$ and $\bar{v}$ are auxiliary integer variables associated with each stroke triangle corner. We reduce the influence of this term away from the strokes by setting $w_{\text{snap}} = \exp(-d^2/(2\sigma^2))$ where $\sigma = 0.1$ and $d \in [0, 1]$ denotes the distance to the center line of the stroke, computed as the normalized distance transform of the binary image.

Integer elimination strategy. While the snapping term introduces new integer variables, in practice, many are redundant. Consider a triangle where the u (resp. v) coordinates of two of the corners get snapped to the *same* integer value. We could eliminate one of these variables from the optimization by forcing these two corners to have equal coordinates, if only we could detect such configurations beforehand. We achieve this goal by solving a per-triangle parametrization problem, which gives us local estimates of the coordinates. To do so, we transform the triangle such that the two directions of the frame field map to the vectors $(1/s, 0)$ and $(0, 1/s)$. We then force equality between corner coordinates if the distance between their local estimates is less than 0.5.

While this heuristic might bias the solution towards local decisions, we found it to work well in practice while drastically reducing the complexity of the optimization. Combined with the elimination of integer translations described in Section 4, this strategy typically removes between 95% and 99% of all integer variables.

Final optimization. We compute the parametrization by minimizing an energy computed as a weighted combination of the alignment term and the snapping term, with a small L^2 regularization:

$$E_{\text{param}} = E_{\text{align}} + \mu E_{\text{snap}} + \varepsilon \sum_{c \in \mathcal{C}} \|(u, v)\|^2, \tag{4}$$

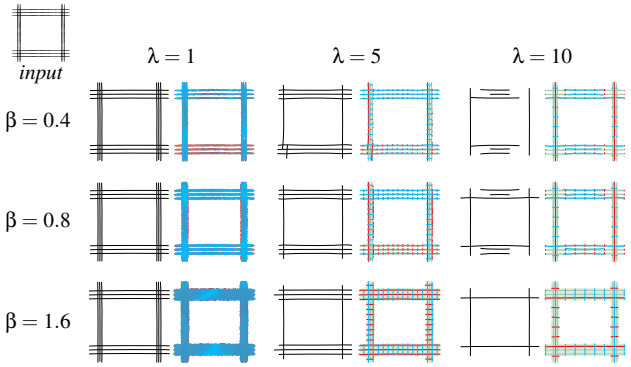
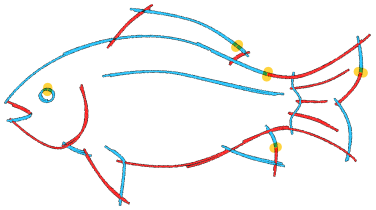


Figure 7: *By letting the user specify the narrow band size* β *and the scale multiplier* λ*, our method provides direct control over the level of simplification. We demonstrate this feature on a toy example. For each pair* (β,λ) *we show the output curves and the parametrization. The narrow band is shown in green. Please zoom in on the electronic version of this figure to see the parametrization in detail.*

subject to the constraint that the grid-preserving transition functions (1) are satisfied for the cut edges. In all of our examples, the regularization weight is fixed to $\varepsilon = 10^{-4}$ and the snapping weight is fixed to $\mu = 25$, see Fig. 6. To optimize (4), we use the greedy mixed-integer solver CoMISo [BZK12].

Complexity analysis. The computational complexity of Eq. (4) depends on multiple factors, most importantly topology and geometry of the input drawing, and the setting of user parameters β and λ. We can approximately determine the complexity by estimating the number of integer variables in a typical input drawing. As an illustration, consider the FISH drawing (inset) whose narrow band mesh has 31389 vertices and 55107 triangles. The parametrization problem (4) has 62778 scalar variables and 64 integer variables, out of which 10 are cut integers (Eq. (1)), and 54 are snapping integers (Eq. (3)). In general, a cut needs to be added for each singularity, and for each hole in (a connected component of) the narrow band; each cut contributes two integer variables. The FISH has three connected components (main body, eye, back line) with a total of five holes (four for the main body, one for the eye), yielding the five cuts highlighted in orange in the inset. There are no singularities in its frame field, no extra cuts are therefore required. We also estimate the number of snapping integers. Thanks to our integer elimination strategy, we only need to add a single integer variable per connected region of triangles assigned to the same tangent direction and not separated by a cut. Tangent labels are represented by red/blue dots in the inset; black dots are unlabeled triangles. The FISH has 26 red regions and 28 blue regions, yielding a total of 54 snapping integers. Note that without our integer elimination strategy, we would need to introduce one snapping integer for each labeled triangle (i.e. each red/blue dot).



6.3. Curve Network Extraction

After computing the parametrization, we determine which integer isolines are covered by the sketch, extract the resulting topological graph, and convert it to parametric curves (Fig. 3c,d).

Topological graph. Because our parametrization is actually composed of multiple, per-triangle grids, we first need to connect the isoline segments of neighboring triangles to extract complete curves. We perform this extraction by inverse-mapping the grid to the sketch space with the algorithm of Ebke et al. [EBCK13]. Since our goal is to obtain a topological graph of the input sketch, we only trace edges that belong to isolines in the tangent direction, and discard all edges in the transversal direction.

Curve fitting. The last step of our method consists in fitting Bézier curves to the edges of the topological graph. A strength of our approach is that the parametrization tells us whether successive edges of the graph correspond to the same isoline. We exploit this information by chaining edges corresponding to the same isoline and by fitting a single spline curve to each chain. Denoting by $\{\mathbf{v}_i\}_{i=0}^{n}$ the vertices of a chain and by t_i their associated normalized arc-length parameters such that $t_0 = 0, t_n = 1, t_i < t_{i+1}$, we compute a piecewise-cubic spline $\gamma(t)$ that minimizes the fitting error and satisfies the *endpoint interpolation*:

$$\min_{\gamma} \sum_{i=0}^{n} \|\gamma(t_i) - \mathbf{v}_i\|^2 \quad \text{s.t.} \quad \gamma(0) = \mathbf{v}_0 \;\text{ and }\; \gamma(1) = \mathbf{v}_n.$$

Our goal is to obtain a compact result with a small number of spline segments without sacrificing precision. We start by fitting a single segment, and we check if the fitting error is smaller than a threshold (fixed to $\beta\bar{\omega}$ in our implementation), in which case we stop the fitting. Otherwise, we recompute the spline, increasing the number of segments by 1. We stop the iteration as soon as the fitting error falls below the threshold or the number of segments reaches the number of edges in the chain. Before the fitting, we split all chains at vertices of degree > 2, and we discard chains that weakly cover stroke pixels. To measure the coverage, we assign each stroke pixel to the nearest chain, and we make a 4-bin histogram of chain pixels using their associated isoline parameters normalized to $[0,1]$. The chain is discarded if any of the bins is empty.

7. Results and experiments

We implemented our method in C++ using EIGEN [GJ*10] for linear algebra, TRIANGLE [She96] for meshing, CGAL [CGA20] and LIBIGL [JP*20] for geometry processing, CoMISo [BZK12] for solving the mixed-integer optimization, and ALGLIB [Boc20] for spline fitting. The source code of our implementation and the supplemental materials are available at https://repo-sam.inria.fr/d3/grid-vectorization/.

Fig. 13 shows a variety of drawings vectorized with our method. While the CITYSCAPE includes grid-like structures particularly well suited to our approach, other drawings show that our method also succeeds on curved shapes, sharp angles and Y-junctions. Our current implementation takes less than a minute for simple inputs, up to several minutes for the most complex ones like the CITYSCAPE (see Table 2), which is on par with prior vectorization methods, cf. Table 1.

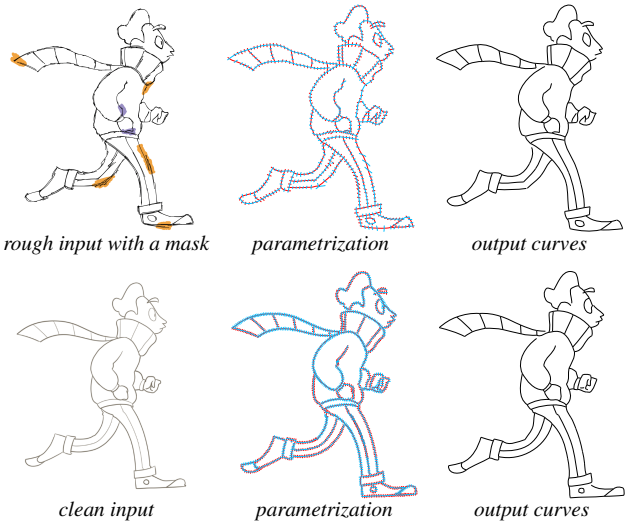


Figure 8: *Using a sparse mask on the heavily over-sketched* RUNNING KID *(top), our vectorization is close to the one obtained using a clean version of the same input (bottom). (Purple scribbles: less simplification, orange scribbles: more simplification.)*

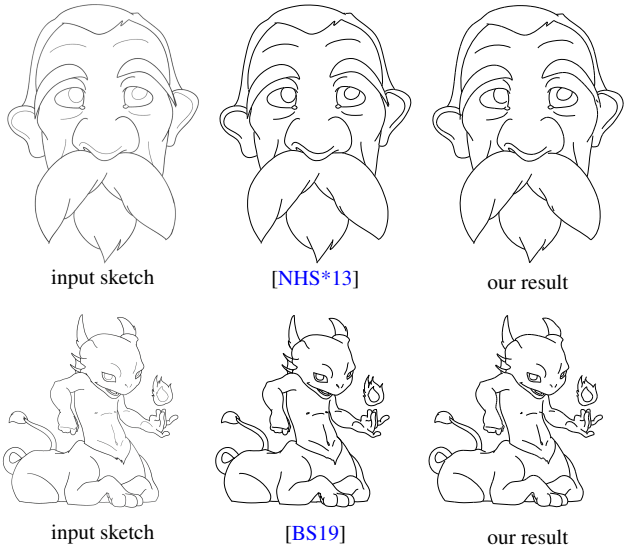


Figure 9: *Our method is competitive with prior work on clean drawings.*

Simplification control. Our method provides direct control over the level of detail that the user wishes to reconstruct. Fig. 7 shows the effect of varying the narrow band size β and the scale multiplier λ on a toy example. Intuitively, smaller values of β and λ provide weaker simplification, and are suitable when the user wants to reconstruct all of the strokes (for instance in clean drawings). Higher values of β and λ yield stronger simplification, and are suitable for merging over-sketched strokes. In our experiments, we typically first find a suitable value of the binarization threshold. By default, we use the threshold 165, which we increase (typically to 220) when processing drawings with weaker contrast. We then fix the size of the narrow band such that it covers neighboring regions where we want to merge strokes. Finally, we adjust the scale multiplier λ according to the desired level of detail.

Fig. 8 demonstrates the robustness of our method to the sketchiness in the input: we obtain similar results for both clean and rough version of the RUNNING KID.

Comparisons. Fig. 9 shows that our algorithm performs similarly to previous methods on clean drawings, while Fig. 12 shows that it outperforms prior work on over-sketched inputs. In the latter case, we compare to two recent bitmap vectorization algorithms [FLB16; BS19], as well as to a sketch simplification algorithm that takes vector strokes as input [LRS18], which we rasterized to be fed to the bitmap-based methods. We also show the results obtained with the method by Bessmeltsev and Solomon [BS19] on a bitmap filtered using Simo-Serra et al. [SII18b] and Xu et al. [XXM*19]. The results shown for Liu et al. [LRS18] come from their paper. We ran the method by Bessmeltsev and Solomon [BS19] with default parameters and the methods by Favreau et al. [FLB16], Simo-Serra et al. [SII18b], and Xu et al. [XXM*19] with a set of parameters that we adjusted to perform best overall.

Given a raster drawing, our method produces vector drawings close to the results obtained by Liu et al. [LRS18] from a vector input. In contrast, the results by Favreau et al. [FLB16] contain residual open curves that their optimization does not attempt to merge with nearby parallel curves. The method by Bessmeltsev and Solomon [BS19] targets clean drawings, and as such keeps most of the original strokes. Please see the supplemental materials for additional comparisons.

While the bitmap filtering algorithms by Simo-Serra et al. [SII18b] and Xu et al. [XXM*19] succeed in merging most of the over-sketched strokes in Fig. 12, they might create topological artifacts that the method by Bessmeltsev and Solomon [BS19] cannot correct (see for instance the close-ups on the SMALL CAR in the third row). Moreover, since Simo-Serra et al. [SII18b] and Xu et al. [XXM*19] use a single global scale to specify the level of simplification, they might fail to properly reconstruct all of the curves in the presence of varying levels of sketchiness, even when the scale is adjusted to obtain the best possible result. Fig. 10 demonstrates three examples of such failure cases. Our method correctly merges the over-sketched strokes and reconstructs the clean strokes, without needing a mask on these three inputs.

User control. Over-sketched drawings often contain varying levels of details, for which a global scale multiplier λ might not be adequate. For such drawings, we allow the user to provide a mask indicating different scales for the grid parametrization. Fig. 11 shows examples generated with such masks; all other figures were generated without masks except Fig. 1 and Fig. 8 (top). Purple scribbles indicate small details, which we capture by reducing the grid scale and the narrow band size to one half of the global values. Orange scribbles indicate regions where stronger simplification is desired, which we achieve by doubling the scale of the grid and the size of the narrow band.

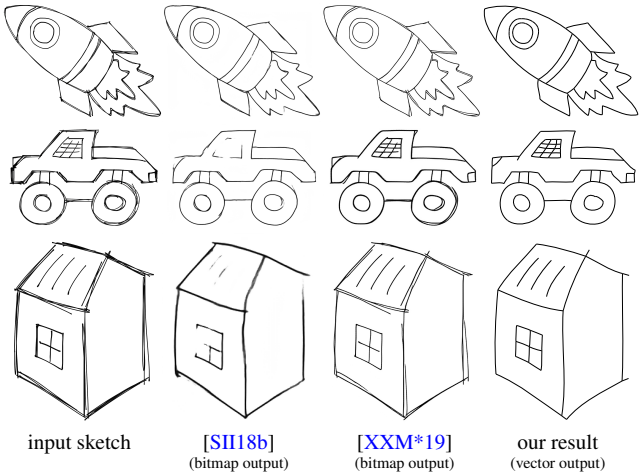


Figure 10: *Our method can handle sketches containing both clean and over-sketched strokes. In contrast, existing learning-based filtering methods often fail to recover the correct topology for such sketches, resulting in artifacts in the output bitmaps. Note that we have adjusted the parameters of both filtering methods [SII18b; XXM*19] to obtain the best results. Please see the supplemental materials for additional comparisons.*

Figure 11: *Our method supports local control over the scale at which strokes get merged. (Purple scribbles: less simplification, orange scribbles: more simplification.)*

Limitations and future work. A strength of our approach is its ability to merge parallel strokes by increasing the scale of the grid parametrization. However, small open curves are sometimes missed when their length falls bellow the distance between grid nodes. The grid might also struggle to explain sharp turns or fine details, especially in low-resolution drawings (Fig. 17).

Our method relies on a narrow band of triangles to localize the grid parametrization. Setting this band too large may make remote parts of the drawing interact together, preventing the parametrization to closely follow the lines, as illustrated in Fig. 15. Since the size of the narrow band is specified by the global parameter β, for certain inputs, it might be difficult to find a value of β that works well in all parts of the drawing (Fig. 14).

The integer elimination strategy can be seen as a way to narrow down the search space during mixed-integer optimization, allowing us to significantly speed up the computation. Although this heuristic might induce a loss of precision, we did not experience such problems in practice.

While our method is comparable to prior work in terms of running time (cf. Table 1), complex sketches can take several minutes to process (cf. Table 2). Some performance could be gained via code optimization – our implementation is a research prototype, and we believe there is room for improvement. Moreover, our experiments suggest that, at least for some inputs, results of comparable quality might be obtained at faster rates using a coarser triangulation; see the supplemental materials for an example. Nevertheless, more experiments are required to determine a strategy for optimizing mesh density to gain performance. This could even mean a different mesh generation algorithm, e.g. by meshing the narrow band represented as a signed distance function.

Our current global approach requires recomputing the parametrization for the entire mesh when the user updates the mask, even when the changes are localized to small regions in the drawing. One could instead *freeze* the already-computed parametric coordinates in triangles outside of regions affected by the mask. This procedure would remove most degrees of freedom in the mixed-integer problem, potentially enabling more interactive workflows. We leave such extensions for future work.

Even though our frame field regularization strategy improves the resolution of sharp Y-junctions compared to [BS19] (see Fig. 5), it can still fail around very sharp ones, leading to locally incorrect field topology. In such cases, the tangent direction is the same for all three branches of the junction, similarly to Fig. 5a. Since our parametrization cannot bifurcate an isoline, such field configurations lead to local parametric inversions; in practice, the resulting isolines "loop back" on themselves until they reach a mesh boundary (see the black close-ups in Fig. 14). In future, we plan to locally detect such configurations in order to dynamically modify the topology of the frame field.

Finally, high-valence junctions might be difficult to capture using our vertex-based frame field (Fig. 16). This issue could be addressed using a face-based discretization, or a directional field with more than four directions, as already suggested by Bessmeltsev and Solomon [BS19], and/or a different type of parametrization, e.g. with triangular or hexagonal cells [NPPZ12]. Another option might be to allow manual positioning of singularities on such junctions, or

even automatic detection of such configurations. Since high-valence junctions are much less frequent than junctions between 3 or 4 lines, we leave such experiments for future work.

8. Conclusion

Grid-based parametrization is a popular tool in geometry processing as it allows the conversion of triangle meshes into clean, feature-aligned quad meshes. In this paper, we drew a parallel between this task and line drawing vectorization, where a long standing challenge has been to locate curves and junctions in the presence of multiple overlapping strokes. Adapting the modern geometry-processing toolbox to this new domain yields a novel algorithm that succeeds on both clean and over-sketched drawings and offers an intuitive control over the scale under which strokes are merged. We hope that our work will inspire the adaptation of other mesh processing algorithms to the domain of line drawings.

Acknowledgments

We thank the anonymous reviewers for their invaluable feedback, Justin Solomon for discussions in early stages of the project, Xuemiao Xu for providing comparisons to [XXM*19], and Yulia Gryaditskaya for providing the BIRD and the FLOWER drawings in Fig. 13. We also thank George Drettakis, Yulia Gryaditskaya, Felix Hähnlein, David Jourdan, Baptiste Nicolet and Simon Rodriguez for their feedback on preliminary drafts of the paper. This work was supported by the European Research Council (ERC) starting grants D3 (ERC-2016-STG 714221) and AlgoHex (ERC-2019-STG 853343), the Natural Sciences and Engineering Research Council of Canada (NSERC) grant RGPIN-2019-05097 ("Creating Virtual Shapes via Intuitive Input"), the Fonds de recherche du Québec – Nature et technologies (FRQNT) grant 2020-NC-270087, and research and software donations from Adobe.

input	Fig.	total time (sec.) [NHS*13]	[FLB16]	[BS19]	ours
WINDMILL	1	47	72	104	228
SMALLCAR2	5	14	120	71	30
DRACOLION	9	11	80	25	108
ROCKET	10	54	787	192	53
MUSCLEMAN	12	23	119	188	26
CHURCH	13	11	70	59	523

Table 1: *In terms of total running time per input, we are competitive with prior vectorization methods. While our algorithm takes longer on complex drawings, it often outperforms existing methods on simpler sketches. See Table 2 for additional statistics for each input. (Note that the timings differ slightly from the ones in Table 2 since they were measured on different machines.)*

input	Fig.	resolution w	h	settings β	λ	timings (sec.) field	uv	**all**
WINDMILL	1	700	850	0.3	1	23	208	**241**
FISH	3	1000	559	0.3	2	7	8	**18**
CROSS	4	500	500	1.0	5	4	3	**9**
SMALLCAR2	5	1000	600	0.3	2	14	19	**38**
KID (ROUGH)	8	1000	1140	0.3	2.5	32	54	**93**
KID (CLEAN)	8	584	640	0.3	1	11	25	**41**
MUTEN	9	1024	1024	0.3	1	22	33	**66**
DRACOLION	9	1024	1024	0.3	1	34	113	**165**
ROCKET	10	1800	1300	1.0	4	32	35	**75**
TRUCK	10	1000	686	0.8	3	63	38	**108**
HOUSE	10	1300	1578	1.2	5	45	53	**105**
CAR	11	1000	431	0.5	2	19	24	**46**
EAGLE1	11	1000	514	1.0	4	53	51	**108**
DUCK	11	1000	1092	0.3	1.5	15	16	**36**
TOOL	11	1000	1204	1.2	12	14	17	**35**
MECHA	12	1020	1000	0.8	3	44	52	**103**
MUSCLEMAN	12	1000	1127	0.3	2	12	21	**37**
SMALLCAR1	12	1000	560	0.8	3	23	22	**48**
BUILDING	12	1000	780	0.5	2	36	90	**134**
BIRD	13	1000	1250	0.3	2	19	85	**113**
BOWTIE	13	1000	567	1.2	3	85	24	**113**
CHURCH	13	1000	1000	0.5	2	45	470	**533**
CITYSCAPE	13	2000	1200	0	1	49	680	**760**
FLOWER	13	1800	1600	0.5	3	95	322	**436**
MOUSE	13	1024	1024	0.3	1	14	28	**49**
WOMAN	13	1000	1373	0.8	3	45	282	**343**
WALL	13	1300	1000	0.5	2	28	118	**157**
EAGLE2	15	500	315	0.3	2	5	7	**14**
JUNCTIONS (ALL)	16	250	250	0.5	2	1	1	**3**
CHARACTER	17	336	855	0.5	1	73	415	**507**
FAIRY	17	1000	955	0.3	1	37	31	**73**

Table 2: *Quantitative statistics for sketches shown in the paper. The last column shows the time it took to run the entire method, including pre-processing and extraction.*

References

[BCF*07] BARTOLO, A., CAMILLERI, K., FABRI, S., BORG, J., and FARRUGIA, P. "Scribbles to Vectors: Preparation of Scribble Drawings for CAD Interpretation". *Proc. Sketch-based Interfaces and Modeling (SBIM)*. 2007, 123–130. ISBN: 978-1-59593-915-9 3.

[BF12] BAO, B. and FU, H. "Vectorizing line drawings with near-constant line width". *IEEE International Conference on Image Processing*. 2012, 805–808 2.

[BLP*13] BOMMES, D., LÉVY, B., PIETRONI, N., PUPPO, E., SILVA, C., TARINI, M., and ZORIN, D. "Quad-Mesh Generation and Processing: A Survey". *Computer Graphics Forum* 32.6 (2013), 51–76 4.

[Boc20] BOCHKANOV, S. *ALGLIB*. 2020. URL: https://www.alglib.net/ 7.

[BS19] BESSMELTSEV, M. and SOLOMON, J. "Vectorization of Line Drawings via Polyvector Fields". *ACM Transactions on Graphics* 38.1 (Jan. 2019), 9:1–9:12 3, 5, 8–11.

[BTS05] BARLA, P., THOLLOT, J., and SILLION, F. "Geometric Clustering for Line Drawing Simplification". *Proc. Eurographics Symposium on Rendering (EGSR)*. 2005, 183–192 3.

[BZK09] BOMMES, D., ZIMMER, H., and KOBBELT, L. "Mixed-integer Quadrangulation". *ACM Transactions on Graphics (Proc. SIGGRAPH)* 28.3 (July 2009), 77:1–77:10 4, 6.

[BZK12] BOMMES, D., ZIMMER, H., and KOBBELT, L. "Practical Mixed-Integer Optimization for Geometry Processing". *Curves and Surfaces*. 2012, 193–206. ISBN: 978-3-642-27413-8 7.

[CDQM18] CHEN, J., DU, M., QIN, X., and MIAO, Y. "An improved topology extraction approach for vectorization of sketchy line drawings". *The Visual Computer* 34.12 (Dec. 2018), 1633–1644 2.

[CGA20] CGAL PROJECT. *CGAL User and Reference Manual*. 2020. URL: https://doc.cgal.org/ 7.

Figure 12: *Results on over-sketched drawings. Our method is on par with the one by Liu et al. [LRS18], which takes vector strokes as input. Other vectorization methods [FLB16; BS19] fail to merge neighboring parallel strokes. Although bitmap filtering methods [SII18b; XXM*19] can be used to pre-process a rough input before feeding it to a vectorization algorithm, they often create artifacts. For the methods by Favreau et al. [FLB16], Simo-Serra et al. [SII18b], and Xu et al. [XXM*19] we have adjusted the parameters to obtain the best results, and we ran the method by Bessmeltsev and Solomon [BS19] with the default settings. Please see Fig. 9 for additional comparisons to vectorization methods, Fig. 10 for additional comparisons to bitmap filtering methods, and the supplemental materials for complete results.*

[CGBG13] CHEN, J., GUENNEBAUD, G., BARLA, P., and GRANIER, X. "Non-Oriented MLS Gradient Fields". *Computer Graphics Forum* 32.8 (2013), 98–109 3.

[DCP17] DONATI, L., CESANO, S., and PRATI, A. "An Accurate System for Fashion Hand-Drawn Sketches Vectorization". *IEEE International Conference on Computer Vision (ICCV)*. Oct. 2017, 2280–2286 2.

[DVPS14] DIAMANTI, O., VAXMAN, A., PANOZZO, D., and SORKINE-HORNUNG, O. "Designing N-PolyVector Fields with Complex Polynomials". *Computer Graphics Forum (Proc. SGP)* 33.5 (Aug. 2014), 1–11 5.

[EBCK13] EBKE, H.-C., BOMMES, D., CAMPEN, M., and KOBBELT, L. "QEx: Robust Quad Mesh Extraction". *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)* 32.6 (Nov. 2013), 168:1–168:10 4, 7.

[FLB16] FAVREAU, J.-D., LAFARGE, F., and BOUSSEAU, A. "Fidelity vs. Simplicity: A Global Approach to Line Drawing Vectorization". *ACM Transactions on Graphics (Proc. SIGGRAPH)* 35.4 (July 2016), 120:1–120:10 2, 8, 10, 11.

[GJ*10] GUENNEBAUD, G., JACOB, B., et al. *Eigen v3*. 2010. URL: http://eigen.tuxfamily.org 7.

[GZH*19] GUO, Y., ZHANG, Z., HAN, C., HU, W., LI, C., and WONG, T.-T. "Deep Line Drawing Vectorization via Line Subdivision and Topology Reconstruction". *Computer Graphics Forum (Proc. Pacific Graphics)* 38.7 (2019), 81–90 3.

[HT06] HILAIRE, X. and TOMBRE, K. "Robust and accurate vectorization of line drawings". *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28.6 (2006), 890–904 2.

Figure 13: *Gallery of additional results, showing that our method can handle a variety of configurations including over-sketching and varying stroke width.*

[IBB15] IARUSSI, E., BOMMES, D., and BOUSSEAU, A. "BendFields: Regularized Curvature Fields from Rough Concept Sketches". *ACM Transactions on Graphics* 34.3 (May 2015), 24:1–24:16 3.

[JHWS17] JUN, X., HOLGER, W., WILMOT, L., and STEPHEN, S. "Interactive Vectorization". *ACM SIGCHI*. 2017, 6695–6705 3.

[JP*20] JACOBSON, A., PANOZZO, D., et al. *libigl: A simple C++ geometry processing library*. 2020. URL: https://libigl.github.io 7.

[KLC07] KANG, H., LEE, S., and CHUI, C. K. "Coherent line drawing". *ACM Symp. Non-Photorealistic Animation and Rendering (NPAR)*. 2007, 43–50 3.

[KWÖG18] KIM, B., WANG, O., ÖZTIRELI, A. C., and GROSS, M. "Semantic Segmentation for Line Drawing Vectorization Using Neural Networks". *Computer Graphics Forum (Proc. Eurographics)* 37.2 (2018), 329–338 2.

[LLBG19] LIU, Y., LI, X., BO, P., and GAO, X. "Sketch simplification guided by complex agglomeration". *Science China Information Sciences* 62.5 (Apr. 2019) 3.

[LPL*17] LI, C., PAN, H., LIU, Y., SHEFFER, A., and WANG, W. "BendSketch: Modeling Freeform Surfaces Through 2D Sketching". *ACM Transactions on Graphics (Proc. SIGGRAPH)* 36.4 (July 2017), 125:1–125:14 3.

[LRS18] LIU, C., ROSALES, E., and SHEFFER, A. "StrokeAggregator: Consolidating Raw Sketches into Artist-intended Curve Drawings". *ACM Transactions on Graphics (Proc. SIGGRAPH)* 37.4 (July 2018), 97:1–97:15 3, 8, 11.

[LVRL06] LI, W.-C., VALLET, B., RAY, N., and LÉVY, B. "Representing Higher-Order Singularities in Vector Fields on Piecewise Linear Surfaces". *IEEE Transactions on Visualization and Computer Graphics* 12.5 (Sept. 2006), 1315–1322 4.

[LWH15] LIU, X., WONG, T.-T., and HENG, P.-A. "Closure-aware Sketch Simplification". *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)* 34.6 (Nov. 2015), 168:1–168:10 3.

[NHS*13] NORIS, G., HORNUNG, A., SUMNER, R., SIMMONS, M., and GROSS, M. "Topology-driven Vectorization of Clean Line Drawings". *ACM Transactions on Graphics* 32.1 (Feb. 2013), 4:1–4:11 2, 8, 10.

[NPPZ12] NIESER, M., PALACIOS, J., POLTHIER, K., and ZHANG, E. "Hexagonal Global Parameterization of Arbitrary Surfaces". *IEEE Transactions on Visualization and Computer Graphics* 18.6 (2012), 865–878 9.

[NS19] NAJGEBAUER, P. and SCHERER, R. "Inertia-based Fast Vectorization of Line Drawings". *Computer Graphics Forum (Proc. Pacific Graphics)* 38.7 (2019), 203–213 2.

[OK11] ORBAY, G. and KARA, L. B. "Beautification of Design Sketches Using Trainable Stroke Clustering and Curve Fitting". *IEEE Transactions on Visualization and Computer Graphics* 17.5 (May 2011), 694–708 3.

[PPM18] PARAKKAT, A. D., PUNDARIKAKSHA, U. B., and MUTHUGANAPATHY, R. "A Delaunay triangulation based approach for cleaning rough sketches". *Computers & Graphics (Proc. SMI)* 74 (2018), 171–181 2.

[She96] SHEWCHUK, J. R. "Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator". *Applied Computational Geometry:*

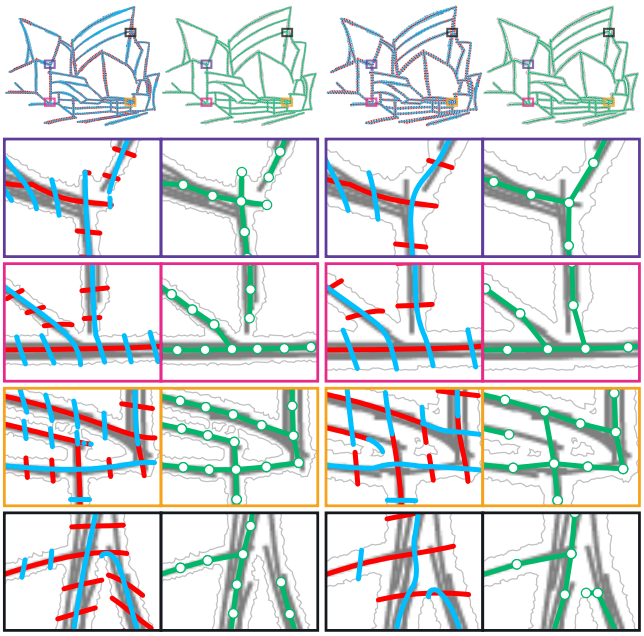


Figure 14: *Our algorithm can introduce artifacts if the narrow band is too small (purple and pink close-ups, smaller band) or too large (orange close-ups, wider band). Moreover, sharp Y junctions are often misrepresented by our frame field, in which case the parametrization cannot capture the correct topology (black close-ups).*

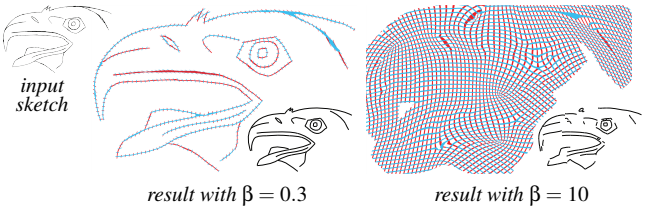


Figure 15: *The narrow band is crucial to restrict the grid parametrization to a local neighborhood. When the band is too large, the grid struggles to explain remote parts of the drawing, resulting in distortions.*

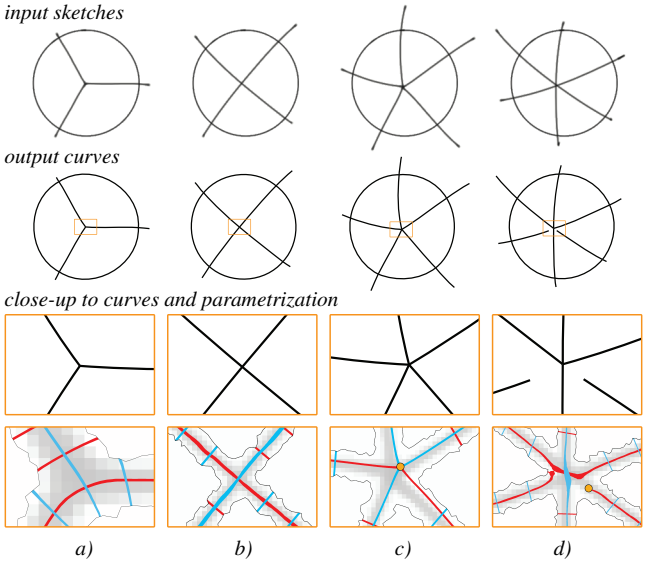


Figure 16: *While our method works best with junctions of degree 3 and 4 (a, b), other types of junctions (c) can be captured via frame field singularities, shown as orange dots. Since our vertex-based frame field only supports singularities of index $\pm\frac{1}{4}$, higher-order junctions might be difficult to capture (d).*

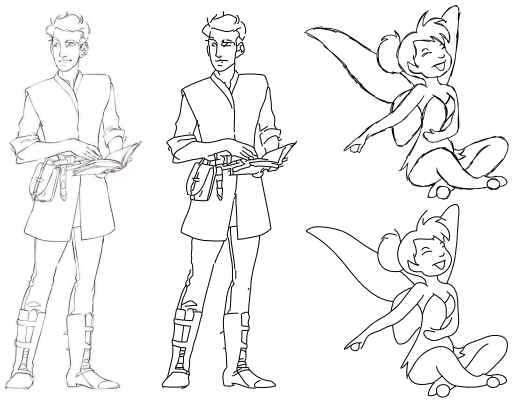


Figure 17: *Limitations of our method. For the low-resolution* CHARACTER *on the left, our method struggles to vectorize fine details. The* FAIRY *on the right contains junctions with sharp turns that are not captured in the parametrization, resulting in gaps in the vectorization.*

Towards Geometric Engineering. 1996, 203–222. ISBN: 978-3-540-70680-9 5, 7.

[SII18a] SIMO-SERRA, E., IIZUKA, S., and ISHIKAWA, H. "Mastering Sketching: Adversarial Augmentation for Structured Prediction". *ACM Transactions on Graphics* 37.1 (Jan. 2018), 11:1–11:13 3.

[SII18b] SIMO-SERRA, E., IIZUKA, S., and ISHIKAWA, H. "Real-Time Data-Driven Interactive Rough Sketch Inking". *ACM Transactions on Graphics (Proc. SIGGRAPH)* 37.4 (July 2018), 98:1–98:14 3, 8, 9, 11.

[SISI16] SIMO-SERRA, E., IIZUKA, S., SASAKI, K., and ISHIKAWA, H. "Learning to Simplify: Fully Convolutional Networks for Rough Sketch Cleanup". *ACM Transactions on Graphics (Proc. SIGGRAPH)* 35.4 (July 2016), 121:1–121:11 3.

[VCD*16] VAXMAN, A., CAMPEN, M., DIAMANTI, O., PANOZZO, D., BOMMES, D., HILDEBRANDT, K., and BEN-CHEN, M. "Directional Field Synthesis, Design, and Processing". *Computer Graphics Forum (Proc. Eurographics)* 35.2 (2016), 545–572 3, 4.

[WZG*19] WEI, G., ZHOU, Y., GAO, X., MA, Q., XIN, S., and HE, Y. "Field-aligned Quadrangulation for Image Vectorization". *Computer Graphics Forum (Proc. Pacific Graphics)* 38.7 (2019), 171–180 3.

[XXM*19] XU, X., XIE, M., MIAO, P., QU, W., XIAO, W., ZHANG, H., LIU, X., and WONG, T.-T. "Perceptual-aware Sketch Simplification Based on Integrated VGG Layers". *IEEE Transactions on Visualization and Computer Graphics* (2019) 3, 8–11.