Fidelity vs. Simplicity: a Global Approach to Line Drawing Vectorization

Jean-Dominique Favreau Florent Lafarge Adrien Bousseau

Inria

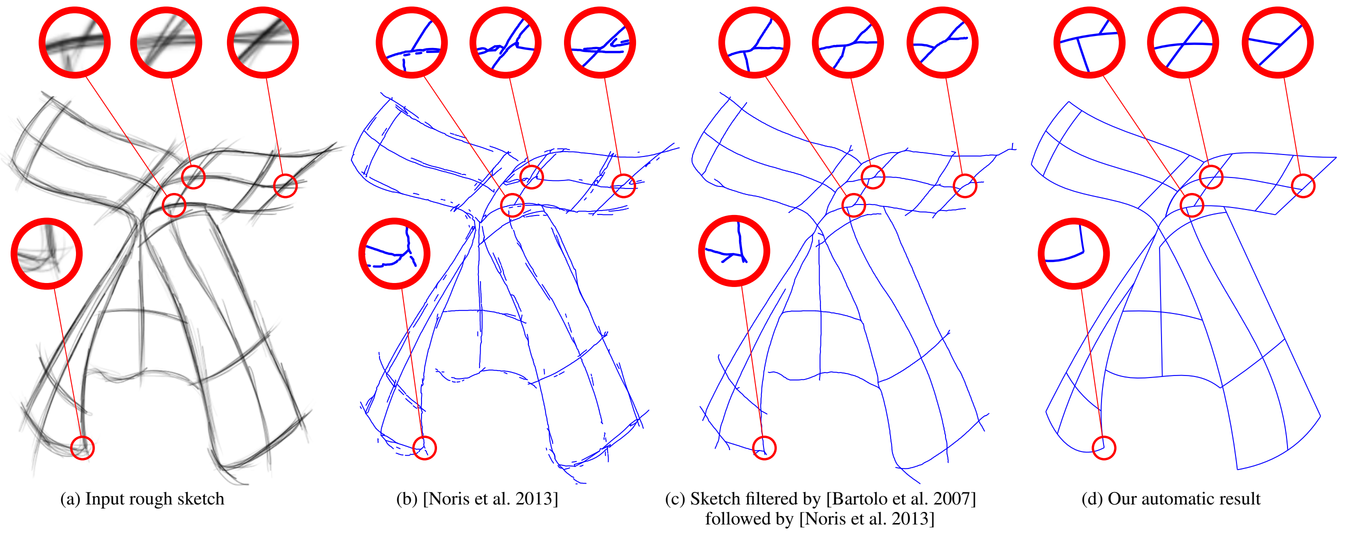

(a) Input rough sketch
(b) [Noris et al. 2013]
(c) Sketch filtered by [Bartolo et al. 2007] followed by [Noris et al. 2013]
(d) Our automatic result

Figure 1: *Rough sketches often contain overlapping strokes (a), which existing vectorization algorithms [Noris et al. 2013] represent as multiple curves (b). Pre-filtering the drawing with the method of Bartolo et al. [2007] improves the vectorization, but produces spurious curve segments at junctions (c). Since existing algorithms analyze junctions locally, they cannot recover the proper topology of these seemingly similar line configurations. By adopting a global formulation that optimizes for both fidelity to the input sketch and simplicity of the output curve network, our algorithm recovers proper topology while significantly reducing the overall number of curves and control points. Design sketch after Sori Yanagi's "Butterfly" stool.*

Abstract

Vector drawing is a popular representation in graphic design because of the precision, compactness and editability offered by parametric curves. However, prior work on line drawing vectorization focused solely on faithfully capturing input bitmaps, and largely overlooked the problem of producing a compact and editable curve network. As a result, existing algorithms tend to produce overly-complex drawings composed of many short curves and control points, especially in the presence of thick or sketchy lines that yield spurious curves at junctions. We propose the first vectorization algorithm that explicitly balances fidelity to the input bitmap with simplicity of the output, as measured by the number of curves and their degree. By casting this trade-off as a global optimization, our algorithm generates few yet accurate curves, and also disambiguates curve topology at junctions by favoring the simplest interpretations overall. We demonstrate the robustness of our algorithm on a variety of drawings, sketchy cartoons and rough design sketches.

Keywords: Line drawing, sketch, vectorization, hypergraph, Bézier curves

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

1 Introduction

Vector drawings offer many advantages over bitmaps by representing graphical elements with parametric curves, such as Bézier splines. First, parametric curves are compact and resolution independent, while bitmap drawings require high storage to avoid aliasing. Second, parametric curves are defined by a small number of control points, which makes them easy to edit. The low dimensionality of the curves also contributes to the distinctive clean and sharp look of vector drawings, which contrasts with the noisy, rough look of bitmap sketches. Finally, parametric curves form the input of many advanced applications such as sketch-based modeling [Xu et al. 2014] and cartoon animation [Dalstein et al. 2015].

However, drawing parametric curves and adjusting their control points requires more precision and user interaction than freehand sketching, which is why many artists still prefer to draw in a bitmap form, either with pen and paper or with digital painting tools like Adobe Photoshop and Autodesk SketchBook. Our goal is to convert such rough, freehand bitmap sketches to clean vector drawings, keeping three main objectives in mind:

- **Fidelity**. The parametric curves should approximate well the input drawing.
- **Simplicity**. The result should be composed of a small number of curves with few control points to preserve the compactness and editability of vector graphics.

SIGGRAPH '16 Technical Paper, **This is the author's version of the work**
ISBN: 978-1-4503-4279-7/16/07
DOI: http://dx.doi.org/10.1145/2897824.2925946

- **Interactivity**. The algorithm should support user guidance to disambiguate the multiple interpretations inherent to artistic inputs.

Unfortunately, existing vectorization algorithms [Hilaire and Tombre 2006; Noris et al. 2013; Bo et al. 2015] only partly satisfy these requirements. In particular, while most methods employ curve fitting to satisfy the first objective of *data fidelity*, this fitting is performed locally and is often sub-optimal with respect to our second objective of *simplicity*. This limitation of local approaches is especially noticeable at line junctions, where ambiguous configurations yield many short curve segments instead of long smooth curves intersecting at a single point, as illustrated in Figure 1b, c.

To achieve our objectives, we propose to cast line drawing vectorization as a global optimization that balances data fidelity with output simplicity. We express data fidelity as the goodness of fit of Bézier curve segments, and we express output simplicity as the number and degree of curve segments that compose the drawing. We initialize our algorithm with an over-segmentation of the drawing, where we fit one Bézier curve on each line segment that connects two junctions, sharp turns or endpoints. Our optimization then reduces the complexity of this initial solution by merging successive curves if they can be expressed as a single curve without loss of accuracy. In addition, we occasionally allow two curves to share the same line segment of the input. This mechanism is equivalent to collapsing the line segment, which is particularly effective to simplify the topology of the drawing around ambiguous junctions, as shown in Figure 1d. We describe a stochastic optimization to efficiently explore the many configurations of merged curves and evaluate their quality in terms of fidelity and simplicity.

While our algorithm produces high-quality vectorizations automatically, it achieves its full potential when guided by the user. In our interactive implementation, users can disambiguate junctions by imposing that two successive curves form a single curve, or that they form two separate curves. These local annotations are then propagated to the entire solution thanks to our global formulation. Users can also prevent local edits from having a global impact by fixing the parts of the solution that they want to preserve.

In summary, we make the following contributions

- The first formulation of line drawing vectorization that balances fidelity to the input with simplicity of the output.
- A global optimization that simulates topological changes of the drawing to find the simplest interpretation of ambiguous junctions.
- An interactive interface to let users collaborate with the optimization to quickly achieve a desired result.

2 Related work

Line drawing vectorization. A number of methods have been proposed to vectorize various types of drawings. Many algorithms target technical diagrams composed of straight lines and circular arcs [Hilaire and Tombre 2006], while freeform splines are more common in cartoon images [Bao and Fu 2012; Noris et al. 2013; Bo et al. 2015]. All these approaches follow a similar three-step procedure. First, a 1-pixel width *skeleton* of the drawing is extracted and junction points between multiple lines are identified. Second, vectorial primitives (lines, arcs, curves) are fitted on each line segment bounded by two junctions. Finally, primitives that meet at a junction are merged based on heuristics on tangent alignment or curvature agreement.

However, because these three steps are applied in sequence, errors in one step propagate to the subsequent steps. In particular, the topology of the skeleton extracted in the first step remains fixed, despite the fact that it is often erroneous at junctions, as illustrated in Figure 1b. The originality of our approach is to allow topological changes at junctions during curve fitting and merging. In addition, while existing methods refine each junction independently based on local information, we propose a global optimization to favor the junction configuration that yields the simplest interpretation overall.

By focusing on line drawings, our goal differs from the problem of vectorizing color images like photographs. While lines are best captured by parametric curves, color regions can be represented with various primitives, including linear color gradients [Lecot and Lévy 2006], parametric patches [Sun et al. 2007] or PDEs [Orzan et al. 2008]. We refer the interested reader to [Liao et al. 2012] for a detailed discussion of color image vectorization. It is worth noting that none of the existing methods explicitly minimizes the number of vectorial primitives in the output.

Line drawing simplification. With the advent of digital drawing tools, several methods have been proposed to simplify drawings composed of vectorial pen strokes [Barla et al. 2005; Orbay and Kara 2011; Liu et al. 2015]. While such methods also face the challenge of merging strokes to form long curves, the additional knowledge provided by the shape and orientation of the input strokes greatly facilitates proper handling of junctions. Nevertheless, our algorithm produces results of comparable quality when applied on rasterized drawings, despite the fact that our input bitmaps offer less information than digital strokes.

Line-network extraction. Our problem is also related to line-network extraction which has received significant attention in computer vision to identify roads in aerial images, blood vessels or neurons in medical images, or galaxy filament in astronomic images [Peng et al. 2010; Turetken et al. 2013; Chai et al. 2013]. These methods build on strong shape priors to favor particular forms of line-networks, but these priors are often too specific for freehand drawings. In addition, these methods focus on localizing the lines and modeling the network topology rather than converting the lines to parametric curves and minimizing their complexity.

Global optimization with complexity term. While penalizing complexity is novel in the context of image vectorization, it has proven beneficial in other applications such as image segmentation [Delong et al. 2012], mesh decomposition [Zhou et al. 2015] and reflection separation [Levin et al. 2004] among others. The rational behind these methods is that, when faced with an ill-posed inverse problem, humans often favor the *simplest* interpretation. In our context, low complexity translates in compactness and editability, which are critical features of vector graphics. Closer to our goal is the work of Iarussi et al. [2015], who decomposes a line drawing of a jewelry piece into a small number of paths suitable for fabrication with metal wires. However, their algorithm takes clean line drawings as input and does not improve curve fitting and junction configurations during optimization.

3 Overview

Figure 2 illustrates the main steps of our method. Our algorithm takes as input bitmap line drawings, either scanned from pen-on-paper drawings or created with digital drawing tools like Adobe Photoshop or Autodesk SketchBook (Figure 2a).

The main challenge of automatic vectorization is to extract the *topology* of the curve network, i.e. identify how the black pixels

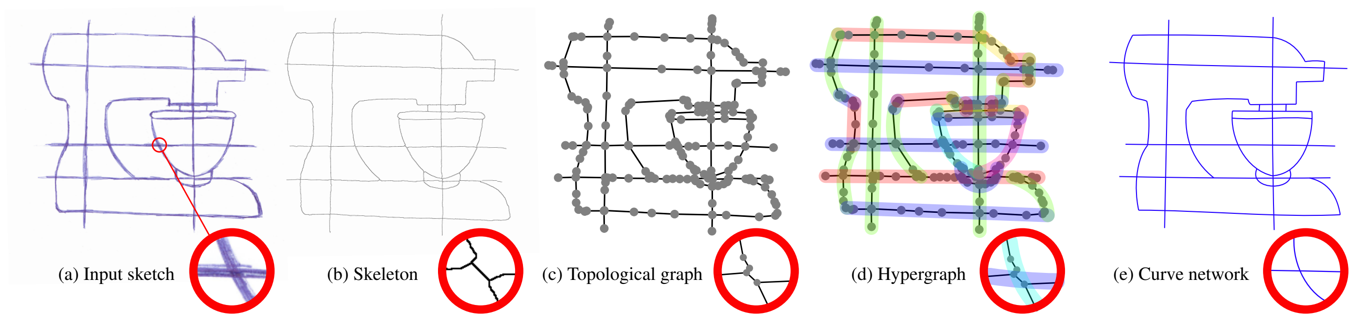


Figure 2: *Overview of our method. Our algorithm takes as input bitmap drawings (a). We first extract the 1-pixel width skeleton of the drawing to locate the curves and their junctions (b). We encode this information as a graph where edges correspond to curve segments and nodes to junctions, endpoints and sharp turns (c). The core of our algorithm consists in merging groups of successive edges to form* hyperedges *of a* hypergraph *(d). Note that several hyperedges can share the same edge of the original graph. Each hyperedge corresponds to a Bézier curve in the output (e). Edges that are shared by several hyperedges are implicitly collapsed by curve fitting, resulting in precise junctions despite extraneous branching of the skeleton.*

of the drawing should be grouped together to form different curves. Once this topology is extracted, the *geometry* of each curve is obtained by least-squares fitting, such that each curve best captures the black pixels it represents. Existing methods typically perform topology extraction and curve fitting as two sequential steps. Our key novelty is to perform these steps *jointly* to balance the compactness of the topology with the accuracy of the fitting.

Similarly to prior work [Hilaire and Tombre 2006; Noris et al. 2013; Bo et al. 2015], the first step of our algorithm consists in filtering the bitmap drawing to extract its 1-pixel width skeleton (Figure 2b). This skeleton locates the center of the lines and their junctions. We encode this topological information as a graph where each edge represents a line segment and vertices represent line junctions, endpoints and sharp turns (Figure 2c). However, fitting a curve on each edge of this graph often provides a poor solution to our objective of low complexity. First, junctions often break long curves into smaller ones, which results in an over-segmentation of the drawing. Second, thick lines produce extraneous branching of the skeleton at junctions, which translates to spurious edges in the topological graph. Prior work relies on local analysis of junctions to remove short edges and to join continuous curve segments [Noris et al. 2013]. Instead, we adopt a global optimization approach to jointly minimize the number of curves and the fitting error.

Given the initial topological graph, the core of our algorithm seeks to group successive edges together when their geometry can be represented by a single curve without loss of accuracy. We introduce a new representation based on the concept of *hypergraph* [Bretto 2013] to encode this grouping. In this representation, each group of edges forms a *hyperedge*, as illustrated in Figure 2d. A key advantage of this formulation is that two hyperedges can share one or more edges of the initial topological graph. This feature is critical to resolve extraneous branching at junctions, as it allows our optimization to simplify the overall curve network by assigning small spurious edges to multiple intersecting curves (see Figure 3 and close-ups in Figure 2).

4 Algorithm

The goal of our algorithm is to produce a curve network that is compact and accurate. To this end, we start with an over-segmentation of the drawing where each segment between two consecutive junctions, sharp turns or endpoints is a curve. This initialization satisfies well our objective of accuracy, but often contains more curves than needed. Our optimization then consists in merging these initial curves to reduce complexity without sacrificing accuracy.

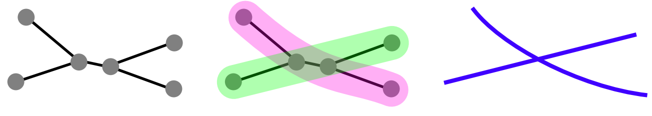
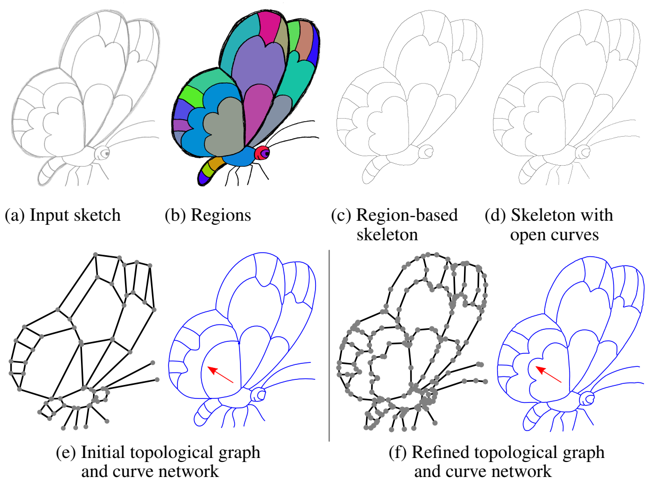


Figure 3: *Given the topological graph of the drawing (a), our algorithm groups successive edges to form* hyperedges. *We fit a Bézier curve on each hyperedge (c). Edges that are shared by several hyperedges, such as the central edge in this example collapse to a single point after fitting.*

4.1 Initialization by over-segmentation

Extracting the skeleton. Following standard practice, we initialize our curve network from the 1-pixel width skeleton of the drawing. Many solutions exist to compute such a skeleton. For clean line drawings, popular methods include morphological thinning [Hilaire and Tombre 2006] and iterative stroke pixel clustering [Noris et al. 2013]. However, these line-based methods tend to produce many extraneous branches on sketchy drawings. Inspired by [Liu et al. 2015], we adopt a more robust region-based approach where we define the skeleton as the frontiers between adjacent regions of the drawing, as illustrated in Figure 4. We first detect the regions of the drawing by running the trapped-ball segmentation algorithm [Zhang et al. 2009], which is robust to small leakage between regions. We then iteratively dilate the regions until they meet and assign the pixels adjacent to two or more regions to the skeleton. The number of dilation iterations gives us an estimate of the local thickness of the lines. However, this region-based algorithm does not capture open curves. As a second step, we identify pixels of open curves as the ones that are at a distance greater than the local thickness of the closest skeleton point. We then compute the skeleton of these additional pixels using morphological thinning. Figure 4d shows the skeleton we obtain for a typical drawing.

Initializing the curve network. The drawing skeleton forms a network of 1-pixel width lines. The next step towards a vectorial representation is to identify which pixels should be grouped together to form curves. Following the terminology of Noris et al. [2013], we call this grouping the *topology* of the drawing and we represent it as a graph $\mathbf{g} = (V, E)$ where nodes V correspond to the junctions and endpoints of the skeleton, and edges E correspond to the skeleton branches. Each edge $e \in E$ is associated with a single Bézier curve segment B^e. We compute the geometry of each curve


(a) Input sketch (b) Regions (c) Region-based skeleton (d) Skeleton with open curves

(e) Initial topological graph and curve network (f) Refined topological graph and curve network

Figure 4: *Extraction of the skeleton and topological graph. We adopt a region-based approach to be robust to sketchy lines (b,c), which we complement with a morphological approach for open curves (d). The initial graph only encodes junctions and endpoints (e). We refine it to include sharp turns (f).*

by chaining the corresponding pixels and minimizing the fitting error

$$\epsilon(e) = \sum_{p \in S^e} (1 - \frac{w_p}{2}) \| B^e(t_p) - p \|_2^2 \qquad (1)$$

where S^e is the chain of pixels associated with edge e, $t_p \in [0, 1]$ is the normalized position of pixel p along the pixel chain, and w_p is the thickness of the line at p normalized with respect to the maximal thickness over the entire drawing. We weight the fitting error by the line thickness to account for the fact that the skeleton is less precise along thick lines. We compute the initial curves by fitting Bézier curves of degree three, as illustrated in Figure 4e, although our optimization later considers curves of lower degree for higher compactness.

We further improve the accuracy of this initialization by recursively splitting the graph edges until the average fitting error of all Bézier curves is below 2 pixels. This operation ensures that we capture sharp turns along the skeleton branches, as shown in Figure 4f. We define the splitting point on an edge such that the fitting error of the two resulting curves is the lowest, as found by a binary search.

4.2 Simplification by hypergraph exploration

Given our initial, over-segmented vectorization, we now need to merge successive curve segments to reduce overall complexity and remove extraneous branching at junctions. Since each curve segment corresponds to an edge in the topological graph, merging multiple curve segments is equivalent to grouping edges of the topological graph. To model this operation, we rely on the concept of *hypergraph*, illustrated in Figure 3. In its most general definition, a hypergraph is a generalization of a graph in which an edge (also called a *hyperedge*) can connect any number of vertices. In this work, we adopt a more restrictive definition where each vertex is covered by at least one hyperedge, and each hyperedge connects at least two vertices. In addition, we impose that each hyperedge corresponds to a sequence of adjacent edges in the initial topological graph. These conditions are guaranteed by the perturbation operators of our stochastic optimization, described in Section 4.3.

Let $\mathbf{x} = (V, H_\mathbf{x})$ be a hypergraph of the topological graph $\mathbf{g}$, where V is the set of nodes and $H_\mathbf{x}$ the set of hyperedges. We associate each hyperedge $h \in H_\mathbf{x}$ with a chain of pixels $S_\mathbf{x}^h$ by concatenating the pixels of the skeleton associated with the edges grouped into h. Each pixel chain $S_\mathbf{x}^h$ yields a fitted Bézier curve $B_\mathbf{x}^h$ in the curve network. The degree of the Bézier curve is a free parameter that allows the optimization to consider straight, quadratic and cubic curves, all being supported by the SVG format.

Problem formulation. Our goal is to explore the space $\mathcal{H}$ of hypergraphs generated from the initial graph $\mathbf{g}$ to find $\mathbf{x} \in \mathcal{H}$ that offers the best trade-off between the simplicity of the curve network and its fidelity to the input drawing. We measure the quality of this trade-off with an energy U composed of two terms:

$$U(\mathbf{x}) = (1 - \lambda) U_{\text{fidelity}}(\mathbf{x}) + \lambda U_{\text{simplicity}}(\mathbf{x}) \qquad (2)$$

where λ is a model parameter that balances the two terms.

Fidelity term. We measure the accuracy of a curve network as the sum of the fitting error of all its hyperedges

$$U_{\text{fidelity}}(\mathbf{x}) = \sum_{h \in H_\mathbf{x}} \epsilon(h) \qquad (3)$$

where $\epsilon(h)$ is given by Equation 1.

Simplicity term. The main novelty of our approach resides in explicitly optimizing for simple curve networks. We measure the simplicity of a network by the number of hyperedges, where lower is simpler. We also favor curve networks whose Bezier curves have low degrees since they are more compact and can be edited with fewer control points. The complexity term is defined as a sum of these two types of information weighted by the model parameter μ

$$U_{\text{simplicity}}(\mathbf{x}) = \sum_{h \in H_\mathbf{x}} 1 + \mu \text{Deg}(B_\mathbf{x}^h) \qquad (4)$$

where $\text{Deg}(B_\mathbf{x}^h)$ is the degree of the Bézier curve $B_\mathbf{x}^h$. As illustrated in Figure 5, a high μ value increases the presence of straight lines.

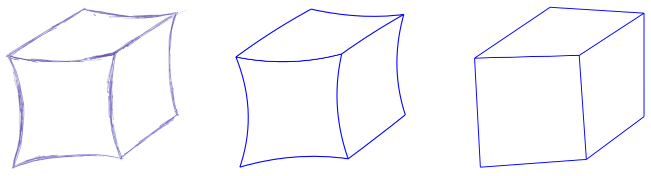

(a) Input sketch (b) Low value of μ (c) High value of μ

Figure 5: *The parameter μ controls the penalization of high degree curves. A high μ favors straight lines.*

4.3 Exploration mechanism

Searching for the hypergraph that minimizes energy U is a non trivial optimization problem as U is non convex and contains global terms. Exhaustive exploration of the hypergraph space is only tractable for very simplistic input drawings as evaluating each configuration requires solving a least squares fitting problem (Equation 3). We adopt a more scalable strategy based on the Metropolis-Hastings algorithm [Hastings 1970]. In a nutshell, this algorithm makes a random exploration of the solution space by iteratively perturbing the current configuration $\mathbf{x} \in \mathcal{H}$ into $\mathbf{x}' \in \mathcal{H}$. The perturbed hypergraph $\mathbf{x}'$ becomes the current configuration with a certain probability depending on the energy variation between the two configurations, and a relaxation parameter T. In addition to scalability, such a Monte Carlo sampler easily supports user-provided constraints, as explained further in Section 5. We now detail perturbation operators and a relaxation schedule adapted to our problem. Algorithm 1 details the main steps of our optimization.

Algorithm 1 Exploration mechanism

```
Compute initial topological graph $\mathbf{g}$ (Sec. 4.1)
Initialize relaxation parameter $T = T_{\text{init}}$
Initialize $\mathbf{x} = \mathbf{g}$
repeat
    Generate $\mathbf{x}'$ from $\mathbf{x}$ with a random perturbation operator
    Fit Bézier curves $B^h_{\mathbf{x}'}$ on the perturbed hyperedges
    Draw a random value $p \in [0, 1]$
    if $p < \exp\left(\frac{U(\mathbf{x})-U(\mathbf{x}')}{T}\right)$ then update $\mathbf{x} \leftarrow \mathbf{x}'$
    else update $\mathbf{x} \leftarrow \mathbf{x}$
    Update $T \leftarrow C \times T$
until $T < T_{end}$
Finalize output representation (Sec. 4.4)
```

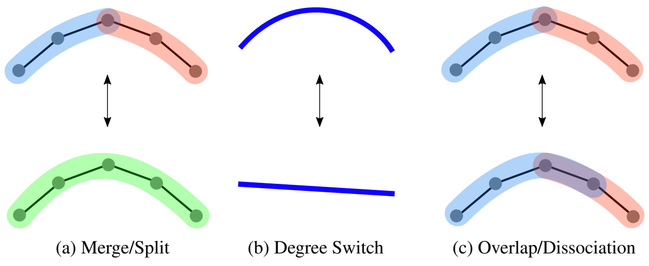


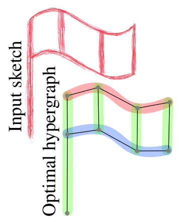
Figure 6: *Our optimization explores the solution space with three reversible perturbation operators: merging or splitting hyperedges (a), changing the degree of the curve associated with a hyperedge (b) and creating or removing overlap between hyperedges (c).*

Perturbation operators. Our optimization seeks to simplify the curve network by merging Bézier curve segments and reducing their degree. We explore these objectives with three types of operators (Figure 6):

- Hyperedge merge and split. This operator splits a hyperedge into two adjacent hyperedges, and reversibly merges two adjacent hyperedges into one. These two operations are implemented by splitting or merging the sets of edges included in each hyperedge. As a result, a hyperedge containing only one edge of the initial graph cannot be split.
- Bézier degree switch. This operator modifies the Bézier degree of a hyperedge to take any value from degree one (straight line) to degree three (cubic Bézier).
- Hyperedge overlap and dissociation. This operator integrates an edge of a hyperedge into a second hyperedge, and reversibly dissociates an edge associated to two hyperedges from one of them. This operator is particularly effective at simplifying topology at junctions.

Starting from the initial hypergraph $\mathbf{x}_0 = \mathbf{g}$, these three operators are sufficient to guarantee that (i) any hypergraph in $\mathcal{H}$ is reachable with a finite number of perturbations from any hypergraph of $\mathcal{H}$, (ii) the reverse pertubations exist, and (iii) perturbations only affect a hypergraph locally.

Relaxation schedule. The relaxation parameter T controls both the speed and the quality of the exploration. Although the Metropolis-Hastings algorithm is guaranteed to converge to the global minimum of our energy when using a logarithmic decrease [Salamon et al. 2002], we prefer to use a geometric decrease of rate C to achieve reasonable running times. While this approximation removes the guarantee of reaching the global minimum, it finds solutions close to this optimum in practice. To quantify this approximation, we performed 1000 runs of our algorithm on the simple sketch shown as inset, for which we computed the global minimum. The correct solution was found in 78% of the cases. The remaining 22% corresponded to local minima close to the global solution, with small visual differences on the resulting curves. In our experiments, we fix the initial temperature $T_{\text{init}} = 1$ and the decrease rate $C = 0.999^{\frac{1}{\text{Card}(V)}}$. Figure 7 shows the evolution of the configurations during the optimization.



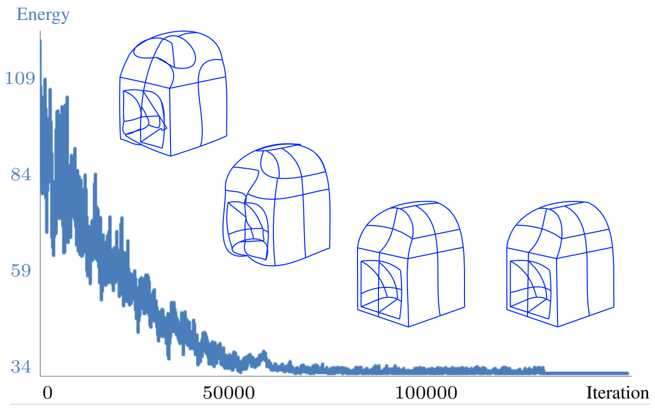


Figure 7: *Evolution of energy U (Equation 2) during the Metropolis-Hastings optimization on a typical sketch. At the beginning of the optimization, perturbations are easily accepted (high energy). The process then becomes progressively selective until converging towards a configuration of interest. Although the two right configurations have both a low energy and are visually identical, their Bezier curves do not have exactly similar degrees.*

4.4 Finalization

We now describe two additional features to refine the curve network by imposing curve connectivity and continuity.

Curve connectivity. Our iterative optimization fits a Bézier curve on each hyperedge independently. While this computation is fast, it does not ensure that curves connect at junctions, as shown in Figure 8 (left). To address this problem, we include two connectivity constraints to the fitting:

- If two hyperedges are connected at their extremities, the corresponding control points of the two Bézier cuves must be the same.
- If the extremity of hyperedge h_1 connects with a non extremity of hyperedge h_2, the corresponding control point P of h_1 must be on the curve $B^{h_2}_{\mathbf{x}}$ (i.e. $\exists t \in [0, 1]$ s.t. $P = B^{h_2}_{\mathbf{x}}(t)$).

The first constraint potentially links all curves together. Satisfying it thus requires solving for the position of all control points at once, which is computationally much more expensive than independently fitting the curves impacted by a perturbation. In addition, the second term makes the optimization non-linear.

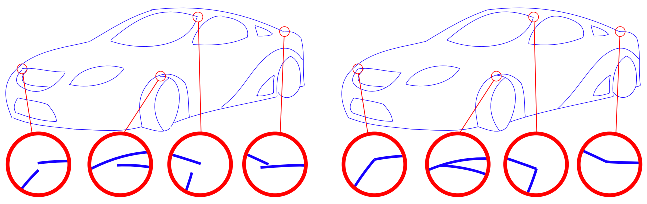
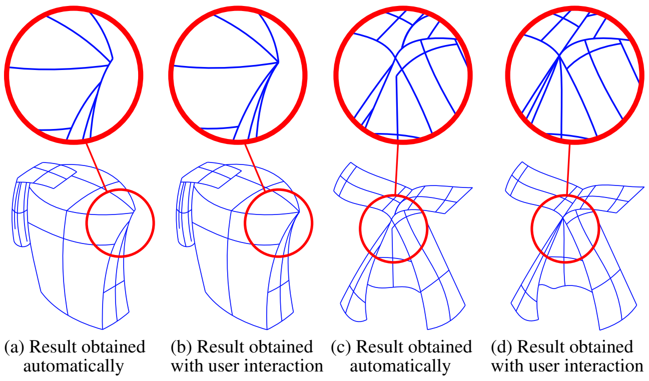
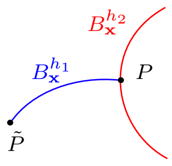


Figure 8: *Fitting each curve independently does not preserve the connectivity of the drawing (left). Adding connectivity constraints ensures that the final result has the same connectivity as the initial skeleton (right).*



(a) Result obtained automatically (b) Result obtained with user interaction (c) Result obtained automatically (d) Result obtained with user interaction

Figure 9: *User interaction. The user ensures that multiple curves intersect by imposing that they share the same segment of the initial vectorization.*



We linearize the problem by first minimizing the fitting error subject to the first constraint only. Then, for each hyperedge h_1 verifying the second constraint, we perform a binary search of $t \in [0, 1]$ such that the control point $P = B_{\mathbf{x}}^{h_2}(t)$ of h_1 minimizes the fitting error $\epsilon(h_1)$ (see inset for notations).

Since accounting for the connectivity constraints makes the fitting too costly to be performed at each iteration of the Metropolis-Hastings optimization, we only apply the constraints once the optimal hypergraph has been found. In practice, these constraints have a limited impact on the overall curve network and thus do not degrade accuracy significantly. Using similar conditions as the convergence stability experiment realized in Section 4.3, the global minimum was found in 80% of cases when applying these constraints at each iteration and in 78% of cases when applying them after the optimization, while the computation was 30 times slower in the former case.

Tangent continuity. Since our optimization considers Bézier curves of at most degree three, it decomposes curves with more than one inflection point into multiple segments. As an optional feature, we enforce tangent continuity of successive segments by aligning their tangents if they are almost co-linear. Similarly to the connectivity constraints, imposing curve continuity yields a non-linear optimization which we only perform once the optimal hypergraph has been found.

5 User interaction

One of the benefits of the Metropolis-Hastings algorithm is that it can easily incorporate user-provided constraints. Since the algorithm is iterative, users can stop the optimization process at any time to specify constraints, and let the optimization continue to see their effect. We support three types of constraints:

- **Merge.** The user can select two curves from the initialization and impose that they end up in the same curve after optimization. If the two curves are not consecutive, we select all other curves along the shortest path between the selected ones.
- **Split.** The user can select two curves from the initialization and impose that they end up in different curves after optimization.
- **Freeze.** When the user is satisfied about part of a solution, she can freeze it by selecting the curves that should no longer be perturbed.

The optimization then only considers the perturbations that do not violate the constraints. Figure 9 illustrates the effect of user-provided constraints.

We also found that exposing the relaxation parameter T offers users useful control on the explorative behavior of the optimization. When T is high, the algorithm accepts drastic perturbations to escape local minima, while when T is low, the algorithm only retains small perturbations that improve the solution locally. With this control, users can force the algorithm to consider other alternatives when they are not satisfied with a solution, or in contrast can accelerate convergence by reducing T when they feel that the solution is close to optimal. Please see the accompanying video for a demonstration of this control.

6 Results and experiments

All results shown in the paper were obtained with the automatic algorithm, except the two examples in Figure 9. We used a fixed $\lambda = 0.6$ for all results except the mechanical piece in Figure 10, where we used $\lambda = 0.3$ to capture the curve discontinuities on its side. We provide all our input bitmaps and output curves as supplemental material. We have applied our algorithm on a variety of drawings from different domains, as illustrated in Figure 10 with a selection of cartoon, engineering, architectural and fashion design sketches. Note that since the drawing of the stool in Figure 1 and the shoe in Figure 10 are dominated by closed region, we did not activate the detection of open curves, which is why all dangling segments have been removed. We now compare our method to prior work on line drawing simplification and vectorization and evaluate robustness, impact of parameters and performance.

Comparisons with existing work. Figure 14 provides a visual comparison with a state-of-the-art vectorization algorithm [Noris et al. 2013] and with the *Image Trace* feature of Adobe Illustrator CC. We first performed the comparison on a sketchy drawing, and then also evaluated the impact of pre-filtering the sketch with the method of Bartolo et al. [2007] to group the sketchy strokes into thick lines. Both algorithms produce multiple curves along sketchy lines and short spurious curves at junctions on the filtered sketches. In contrast, our method produces almost identical results on the two versions of each sketch, and recovers junctions with precision.

Figure 15 also compares our method with a recent line drawing simplification algorithm [Liu et al. 2015]. We insist on the fact that [Liu et al. 2015] takes as input digital drawings composed of vectorial strokes. Still, we obtain similar results even though we take bitmap drawings as input. Our results are even more accurate at junctions thanks to the connectivity constraints described in Section 4.4.

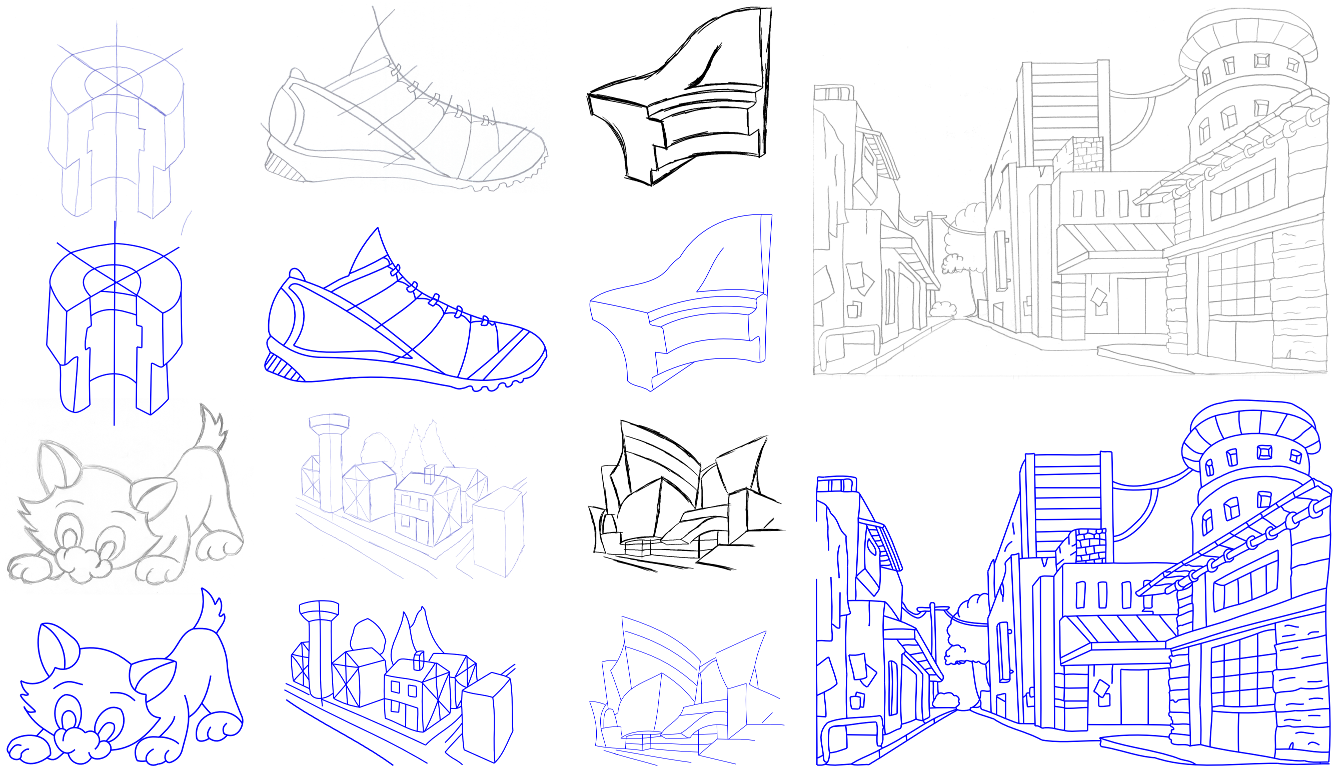


Figure 10: *A selection of line drawings from different domains and our vectorization. Input drawings in the third column courtesy of [Orbay and Kara 2011].*

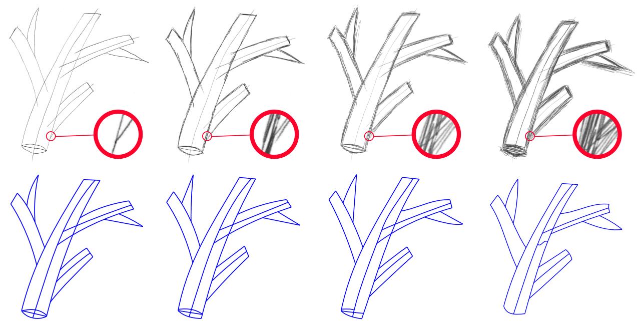


Figure 11: *Robustness to sketchiness. Our algorithm generates very similar curve networks for various levels of sketchiness, even though some details are lost in very sketchy drawings.*

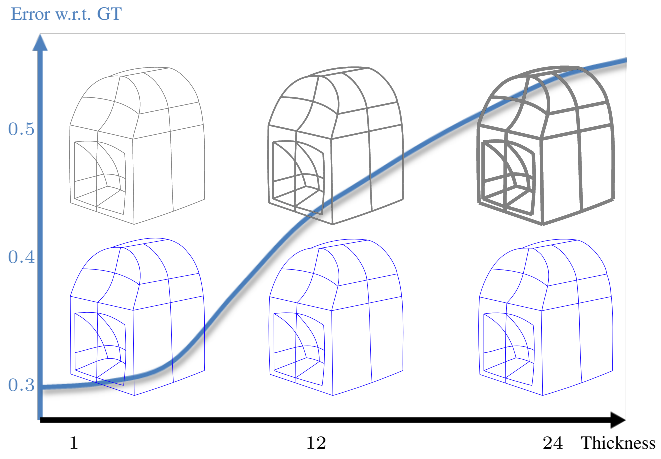


Figure 12: *Our algorithm produces almost identical output curves for increasingly thick lines. Even at thickness* 24*, the junctions remain accurate and the average error is bellow* 0.6 *pixels.*

Robustness. Sketchy and thick lines are very challenging for existing vectorization algorithms because they result in noisy skeletons with many extraneous branches, especially at junctions. Figure 11 shows that our algorithm produces consistent curve networks for increasing levels of sketchiness. Figure 12 provides a quantitative evaluation of the impact of line thickness. We performed this evaluation by rasterizing a vector drawing with increasing line thickness and measuring the distance between the recovered curve network and the ground truth, expressed in pixels. The average error remains below 0.6 pixels for a thickness of 24 pixels.

Impact of parameters. Our algorithm offers a trade-off between accuracy and simplicity, controlled by the parameter λ in Equation 2. Figure 13 illustrates the effect of this parameter. A low λ yields a very low fitting error but a high number of curves. In contrast, increasing λ greatly reduces the number of curves but the resulting network deviates more from the input. We again measured error with respect to a ground truth vector drawing that we rasterized to serve as input to our algorithm. Note that at low λ, the top-right part of the shape is best approximated by small linear segments, while a high λ produces a smoother, albeit less accurate output. We fixed the other parameters μ to 0.2 for all results in the paper except Figure 5, and the ball radius of trapped-ball segmentation algorithm to 3 pixels.

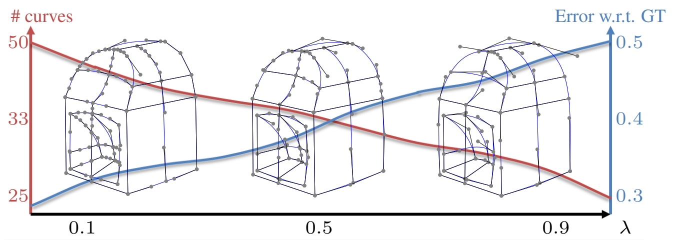


Figure 13: *Parameter λ controls the balance between fidelity and simplicity. Increasing λ augments error to Ground Truth while reducing the number of Bézier curves, and by extension, the number of control points (shown as grey dots).*

	# hyperedge	# interaction	time
Figure 9a (automatic)	25	–	$45s$
Figure 9b (interactive)	25	4	$34s$
Figure 9c (automatic)	27	–	$25s$
Figure 9d (interactive)	26	6	$32s$
Figure 10 right (automatic)	673	–	$95s$

Table 1: *Timing with and without user interactions.*

Performances. Depending on the complexity of the input bitmap, our algorithm takes a few seconds to a few minutes to produce output curves automatically. However, since our optimization is iterative, the user does not have to wait until completion to edit the result. Instead, she can stop the algorithm at any time to add constraints and appreciate their effect. The user can also speed-up the optimization by increasing the relaxation parameter T when the current configuration is satisfactory. As shown in Table 1, the results obtained with user interaction did not take more time than the ones obtained automatically.

Limitations. Our algorithm is not designed to deal with missing data, such as broken strokes. Filling such holes would require extrapolating the curves, which adds significant complexity to the optimization. Note however that the trapped-ball segmentation algorithm [Zhang et al. 2009] for skeleton extraction is robust to small holes. Another limitation of our current optimization is that we only consider Bézier curves, while other primitives such as circular arcs would be better adapted to regular structures in technical drawings. Our algorithm also does not consider high-order geometric regularities such as parallelism, orthogonality or symmetry. Detecting and enforcing such regularities at each iteration of the optimization would be costly if implemented naively.

Our current implementation seeks a uniform trade-off between fidelity and simplicity over the entire drawing. Nevertheless, our Metropolis-Hastings optimization could easily adapt this trade-off locally by taking as additional input a spatially-varying λ parameter, which could be painted by the user or estimated from local image statistics. Finally, while our region-based skeleton extraction effectively merges overlapping strokes in sketchy drawings, it can also remove intended lines since there is an inherent ambiguity between noisy strokes and fine details. When dealing with clean drawings, a smaller trapped-ball should be used.

7 Conclusion and future work

Skillful vector artists create drawings composed of few curves because they result in clean, compact and easily editable artworks. This observation motivated us to propose the first vectorization algorithm that explicitly attempts to minimize the number of curves and their degree. This new, global objective is also extremely effective in disambiguating line junctions, where prior methods tend to produce spurious short curves. While the resulting optimization involves non-convex and non-local terms, we describe an efficient exploration algorithm to support interactive user control.

Our algorithm takes as input bitmap drawings, which allows it to deal with both scanned drawings as well as rasterized digital drawings. Nevertheless, we hope that our energy formulation will inspire novel algorithms dedicated to the simplification of digital drawings composed of vector strokes.

Our idea of minimizing the complexity of the output representation also has great potential for the vectorization of color images, in particular to extract layers that compactly represent transparency and occlusion effects [Richardt et al. 2014]. However, this new domain raises specific challenges, since the optimization should evaluate many interpretations of the shape and color of image regions.

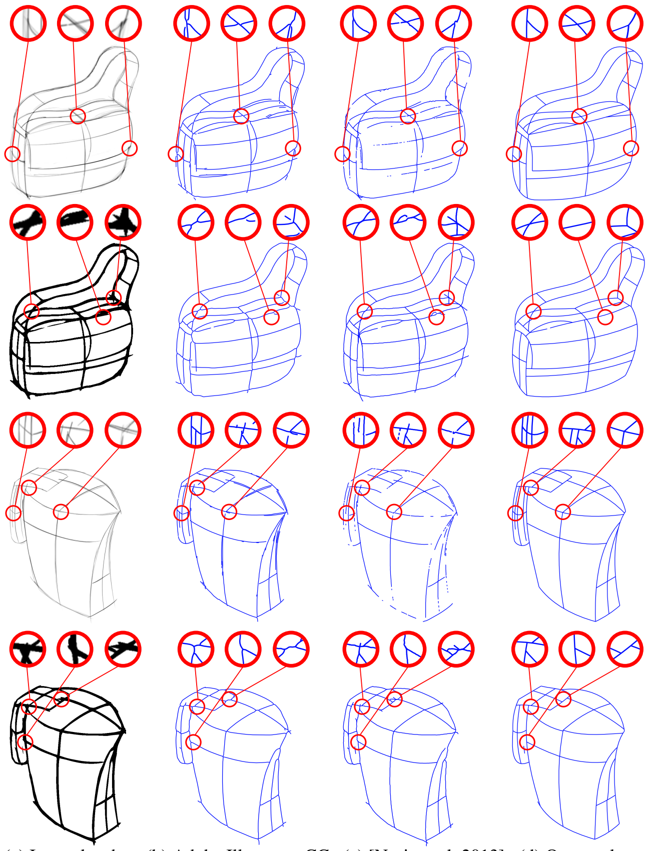


Figure 14: *Comparison to existing vectorization algorithms. In the 2nd and 4th row, the sketches were pre-processed with the Gabor filter bank from [Bartolo et al. 2007] to group neighboring strokes into thick lines. Existing methods produce multiple curves on sketchy lines and extraneous curves at junctions of thick lines. In contrast, our method recovers precise junctions by favoring the simplest interpretation.*

Acknowledgments

Many thanks to the authors of [Bartolo et al. 2007] and [Noris et al. 2013] for running their code on our inputs, and to [Orbay and Kara 2011] for providing their results in vectorial form. This

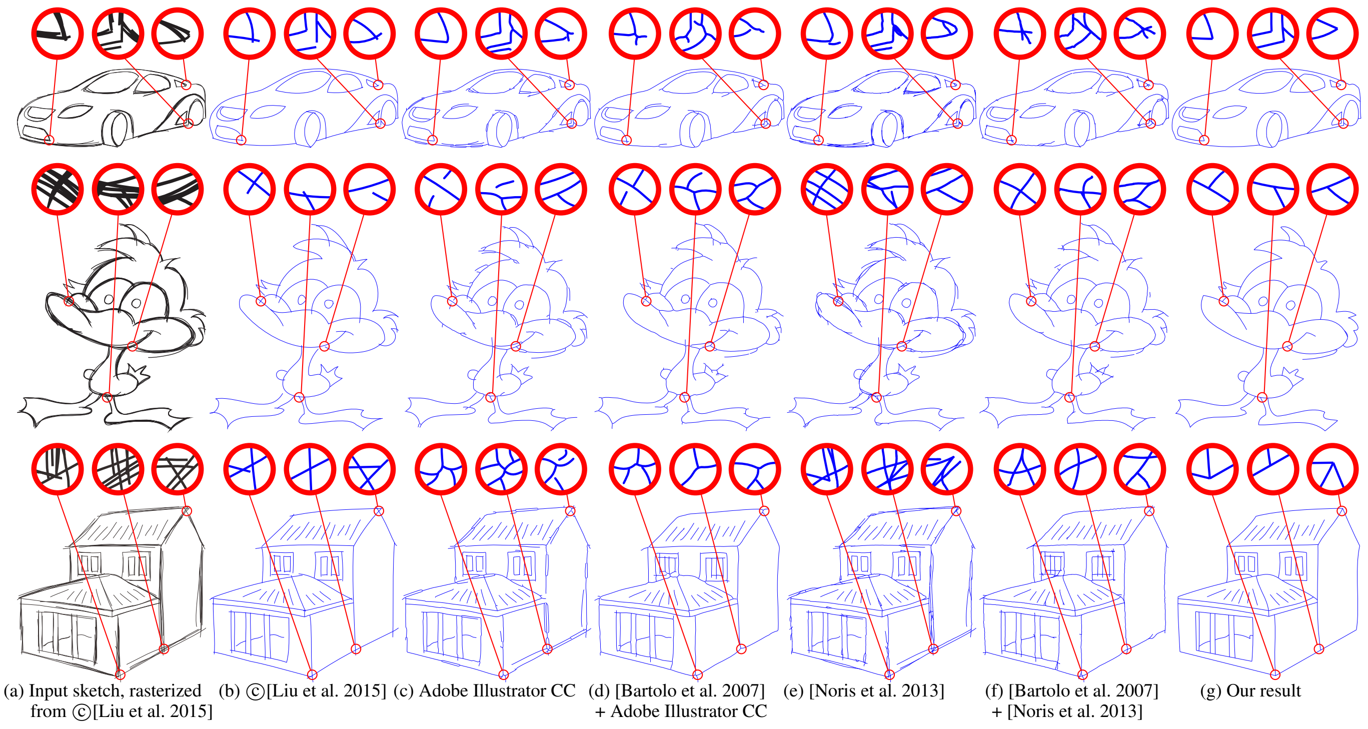

(a) Input sketch, rasterized from ©[Liu et al. 2015]
(b) ©[Liu et al. 2015]
(c) Adobe Illustrator CC
(d) [Bartolo et al. 2007] + Adobe Illustrator CC
(e) [Noris et al. 2013]
(f) [Bartolo et al. 2007] + [Noris et al. 2013]
(g) Our result

Figure 15: *Comparison to existing line drawing simplification and line drawing vectorization algorithms. In column (d) and (f), the sketches were pre-processed with the Gabor filter bank from [Bartolo et al. 2007] to group neighboring strokes into thick lines.* ***Note that [Liu et al. 2015] takes vector strokes as input****, while all other methods deal with bitmaps. Our approach produces results on par with [Liu et al. 2015] even though we process bitmaps rather than vector strokes. Compared to existing vectorization algorithms, our method produces more accurate curves and junctions despite the high ambiguity of the sketchy input.*

work was partially supported by research and software donations from Adobe.

References

BAO, B., AND FU, H. 2012. Vectorizing line drawings with near-constant line width. In *Image Processing (ICIP), 2012 19th IEEE International Conference on*, IEEE, 805–808.

BARLA, P., THOLLOT, J., AND SILLION, F. 2005. Geometric clustering for line drawing simplification. In *Proceedings of the Eurographics Symposium on Rendering*.

BARTOLO, A., CAMILLERI, K. P., FABRI, S. G., BORG, J. C., AND FARRUGIA, P. J. 2007. Scribbles to vectors: preparation of scribble drawings for cad interpretation. In *Proceedings of the 4th Eurographics workshop on Sketch-based interfaces and modeling*.

BO, P., LUO, G., AND WANG, K. 2015. A graph-based method for fitting planar b-spline curves with intersections. *Journal of Computational Design and Engineering*.

BRETTO, A. 2013. *Hypergraph Theory: an introduction*. Springer.

CHAI, D., FORSTNER, W., AND LAFARGE, F. 2013. Recovering line-networks in images by junction-point processes. In *Proc. of the IEEE conference on Computer Vision and Pattern Recognition (CVPR)*.

DALSTEIN, B., RONFARD, R., AND VAN DE PANNE, M. 2015. Vector graphics animation with time-varying topology. *ACM Transactions on Graphics (Proc. SIGGRAPH) 34*, 4.

DELONG, A., OSOKIN, A., ISACK, H. N., AND BOYKOV, Y. 2012. Fast approximate energy minimization with label costs. *International Journal Computer Vision (IJCV) 96*, 1.

HASTINGS, W. 1970. Monte Carlo sampling using Markov chains and their applications. *Biometrika 57*, 1.

HILAIRE, X., AND TOMBRE, K. 2006. Robust and accurate vectorization of line drawings. *IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI) 28*, 6.

IARUSSI, E., LI, W., AND BOUSSEAU, A. 2015. Wrapit: Computer-assisted crafting of wire wrapped jewelry. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia) 34*, 6.

LECOT, G., AND LÉVY, B. 2006. Ardeco: automatic region detection and conversion. In *Proceedings of the Eurographics Symposium on Rendering Techniques*.

LEVIN, A., ZOMET, A., AND WEISS, Y. 2004. Separating reflections from a single image using local features. In *Proc. of the IEEE conference on Computer Vision and Pattern Recognition (CVPR)*.

LIAO, Z., HOPPE, H., FORSYTH, D., AND YU, Y. 2012. A subdivision-based representation for vector image editing. *IEEE Transactions on Visualization and Computer Graphics (TVCG) 18*, 11.

LIU, X., WONG, T.-T., AND HENG, P.-A. 2015. Closure-aware sketch simplification. *ACM Transactions on Graphics (TOG) 34*, 6.

NORIS, G., HORNUNG, A., SUMNER, R. W., SIMMONS, M., AND GROSS, M. 2013. Topology-driven vectorization of clean line drawings. *ACM Transactions on Graphics (TOG) 32*, 1, 4.

ORBAY, G., AND KARA, L. B. 2011. Beautification of design sketches using trainable stroke clustering and curve fitting. *IEEE Transactions on Visualization and Computer Graphics (TVCG) 17*, 5.

ORZAN, A., BOUSSEAU, A., WINNEMÖLLER, H., BARLA, P., THOLLOT, J., AND SALESIN, D. 2008. Diffusion curves: a vector representation for smooth-shaded images. *ACM Transactions on Graphics (Proc. SIGGRAPH) 27*, 3.

PENG, T., JERMYN, I., PRINET, V., AND ZERUBIA, J. 2010. Extended phase field higher-order active contour models for networks - its application to road network extraction from VHR satellite images. *International Journal Computer Vision (IJCV) 88*, 1.

RICHARDT, C., LOPEZ-MORENO, J., BOUSSEAU, A., AGRAWALA, M., AND DRETTAKIS, G. 2014. Vectorising bitmaps into semi-transparent gradient layers. *Computer Graphics Forum (Proc. Eurographics Symposium on Rendering) 33*, 4.

SALAMON, P., SIBANI, P., AND FROST, R. 2002. *Facts, Conjectures, and Improvements for Simulated Annealing*. SIAM Monographs on Mathematical Modeling and Computation, Philadelphia, United States.

SUN, J., LIANG, L., WEN, F., AND SHUM, H.-Y. 2007. Image vectorization using optimized gradient meshes. *ACM Transactions on Graphics (Proc. SIGGRAPH) 26*, 3.

TURETKEN, E., BENMANSOUR, F., ANDRES, B., PFISTER, H., AND FUA, P. 2013. Reconstructing Loopy Curvilinear Structures Using Integer Programming. In *Proc. of the IEEE conference on Computer Vision and Pattern Recognition (CVPR)*.

XU, B., CHANG, W., SHEFFER, A., BOUSSEAU, A., MCCRAE, J., AND SINGH, K. 2014. True2form: 3d curve networks from 2d sketches via selective regularization. *ACM Transactions on Graphics (Proc. SIGGRAPH) 33*, 4.

ZHANG, S.-H., CHEN, T., ZHANG, Y.-F., HU, S.-M., AND MARTIN, R. R. 2009. Vectorizing cartoon animations. *IEEE Transactions on Visualization and Computer Graphics (TVCG) 15*, 4.

ZHOU, Y., YIN, K., HUANG, H., ZHANG, H., GONG, M., AND COHEN-OR, D. 2015. Generalized cylinder decomposition. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia) 34*, 6.