DOI: 10.1111/cgf.70228
Pacific Graphics 2025
M. Christie, N. Pietroni, and Y.-S. Wang
(Guest Editors)

COMPUTER GRAPHICS *forum*

# Single-Line Drawing Vectorization

Tanguy Magne and Olga Sorkine-Hornung

ETH Zurich, Switzerland

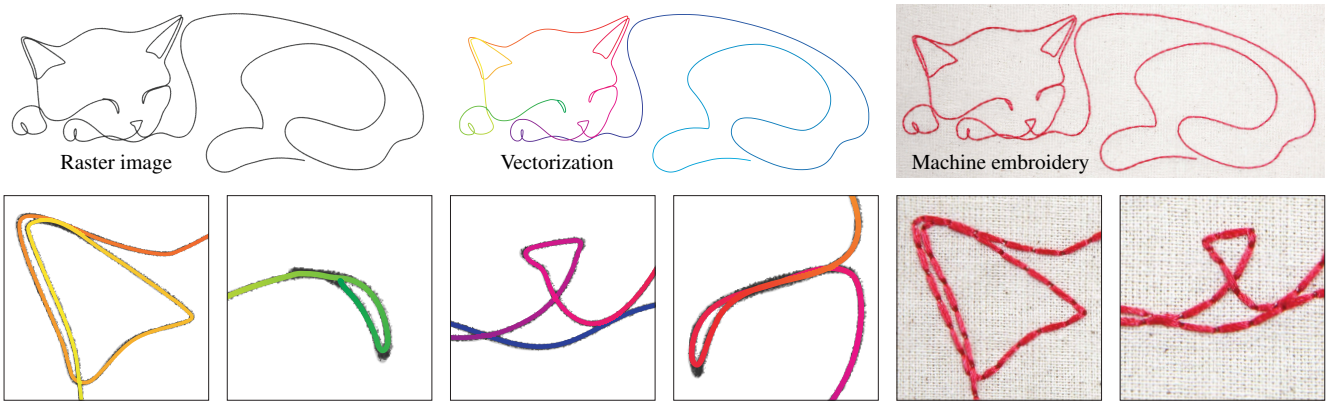


**Figure 1:** *Left and middle: A raster single-line drawing and the parameterized vectorization computed with our method. The insets show the vectorization overlaying on top of the raster image. The color gradient encodes the parameterization of the curve, showing the plausible drawing order produced by our method. Right: Since our vectorization is correctly parameterized and composed of one curve, it easily enables applications such as machine embroidery, where the stitches are sequentially sewn along the curve.*

**Abstract**

*Vectorizing line drawings is a repetitive, yet necessary task that professional creatives must perform to obtain an easily editable and scalable digital representation of a raster sketch. State-of-the-art automatic methods in this domain can create series of curves that closely fit the appearance of the drawing. However, they often neglect the line parameterization. Thus, their vector representation cannot be edited naturally by following the drawing order. We present a novel method for single-line drawing vectorization that addresses this issue. Single-line drawings consist of a single stroke, where the line can intersect itself multiple times, making the drawing order non-trivial to recover. Our method fits a* single *parametric curve, represented as a Bézier spline, to approximate the stroke in the input raster image. To this end, we produce a graph representation of the input and employ geometric priors and a specially trained neural network to correctly capture and classify curve intersections and their traversal configuration. Our method is easily extended to drawings containing multiple strokes while preserving their integrity and order. We compare our vectorized results with the work of several artists, showing that our stroke order is similar to the one artists employ naturally. Our vectorization method achieves state-of-the-art results in terms of similarity with the original drawing and quality of the vectorization on a benchmark of single-line drawings. Our method's results can be refined interactively, making it easy to integrate into professional workflows. Our code and results are available at https://github.com/tanguymagne/SLD-Vectorization.*

**CCS Concepts**

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

## 1. Introduction

Line drawings are often used by artists to quickly sketch rough ideas, and can be the basis for future, more refined versions of an art piece. At the same time, as line drawings offer a stylized, expressive and aesthetically pleasing representation, many artists also employ them as the final version of their artwork. Creating line art demands understanding of the semantically meaningful aspects

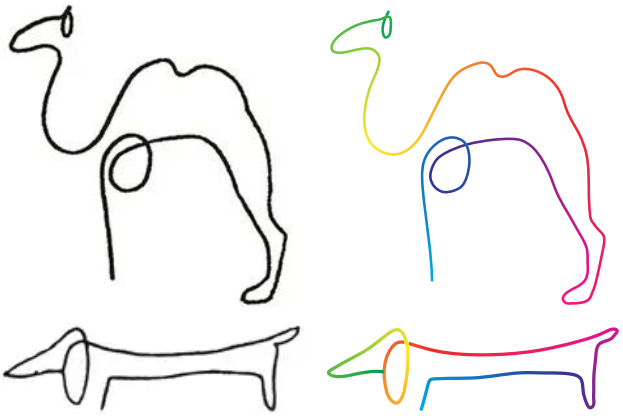

**Figure 2:** *Picasso's single-line drawings. Left: Input drawing scan. Right: Our correctly parameterized vectorized output.*

of the represented subject and abstraction skills to distill the essence of what is being depicted. *Single-line* drawings, sometimes referred to as continuous-line drawings, push this concept to its limits, since a single, continuous stroke is used to represent a whole object or scene. Picasso's one-line drawings (Fig. 2) are perfect examples of such pieces, where the essence of the portrayed animals is captured within the confines of a single stroke. These drawings provided a way for Picasso and many other artists to experiment and train their artistic skills.

Line drawings are usually created on paper, or sometimes using graphics tablets. Even in the latter case, the result is often a raster image, since raster graphics allows drawing more freely. However, in subsequent applications, vector graphics are generally required because they can be easily manipulated and edited. Regular vectorization tools such as Potrace [Sel03] are not well suited for line drawings, since they produce regions covering the strokes instead of lines. This means that moving a control point changes the shape of the covered region, which does not maintain the thickness and style of the curve. Several methods have been proposed to address this issue [BS19, MSG*21, YLA*24], which allow to vectorize line drawings using multiple strokes represented by spline curves. Established vector graphics software, such as Adobe Illustrator [Ill24] and Inkscape [Ink24], offer their versions of such vectorization methods. While producing better results than region-based vectorization tools, these approaches often fail to properly connect the strokes due to ambiguous line continuity at intersections. This causes difficulties for subsequent editing of the vectorized drawing.

Recovering a valid vectorized representation is especially challenging for single-line drawings, since in this case, self-intersections form an essential part of the expressive power. Correctly vectorizing them is crucially important for further editing, as they are key components of the representation. Notably, single-line drawings contain many crossings of two curves, but few T-junctions or Y-junctions, which are more frequently studied when vectorizing regular artistic or industrial line drawings. To the best of our knowledge, we are the first to tackle single-line drawing vectorization specifically, though some work tried to reduce the number of strokes in the vectorized output [ILB15, FLB16].

We introduce a method specifically designed for vectorizing single-line drawings. Our method takes into account the special nature of such drawings and outputs a correctly parameterized curve, in the sense that it follows a common drawing order. The starting point of our algorithm is a medial axis representation of the underlying shape. This representation has been used in the past for vectorizing line drawings, but it has gone out of favor due to its sensitivity to noise and erroneous representation of intersections. We propose a way to correctly simplify the medial axis graph to obtain a valid representation of the intersections by exploiting geometric priors. In particular, we obtain a graph whose node valences appropriately reflect the number of corresponding crossing strokes.

When strokes cross or touch each other, the line parameterization is challenging to recover, as the connectivity can be ambiguous (Fig. 5). Our method tackles this challenge by using a small convolutional neural network (CNN) to classify each intersection. The network is trained on a synthetic dataset generated procedurally and achieves a high classification accuracy. Thanks to its small size, the CNN is fast at inference time and runs on any consumer-grade device. The CNN has a local context, allowing to efficiently process large inputs.

We integrate the graph based representation and intersection classification into a fully automatic vectorization pipeline. While this allows for fast results, we believe that retaining control of the output is essential in artistic work. Thus, we also develop an interactive framework that allows users to easily modify the output of the algorithm at any stage in the pipeline. We show that our automatic vectorization pipeline significantly outperforms other line drawing vectorization methods on a benchmark dataset of single-line drawings. Furthermore, our interactive interface allows guiding the algorithm and editing its output.

Our method, which relies on a new geometric approach to simplify the medial axis representation and a small classifier CNN to traverse intersections correctly, can be used to convert a relatively clean raster image of a single line drawing to a vector representation. It can also extract the underlying curve of a vector graphics line drawing represented by the contour of the shape, a representation often found on stock websites. The output of our algorithm can be used in several applications, such as backstitch embroidery (Fig. 1, right), accelerated laser engraving or wire art [WZY*24].

## 2. Related work

Vectorizing raster graphics is a well developed field of study, but single line drawings have not been explicitly addressed in previous works. Most methods can be categorized into region-based or stroke-based. The former are not well suited for vectorizing line drawings, since they produce areas bounded by curves and can at best generate long regions representing the stroke outline. We thus limit our discussion to stroke-based approaches. We assume the input image represents a *clean* line drawing and omit methods that simplify rough sketches, as they usually output raster images.

*Medial axis methods.* Early work on line drawing vectorization relies heavily on medial axis representation, or 1-skeletons. Many

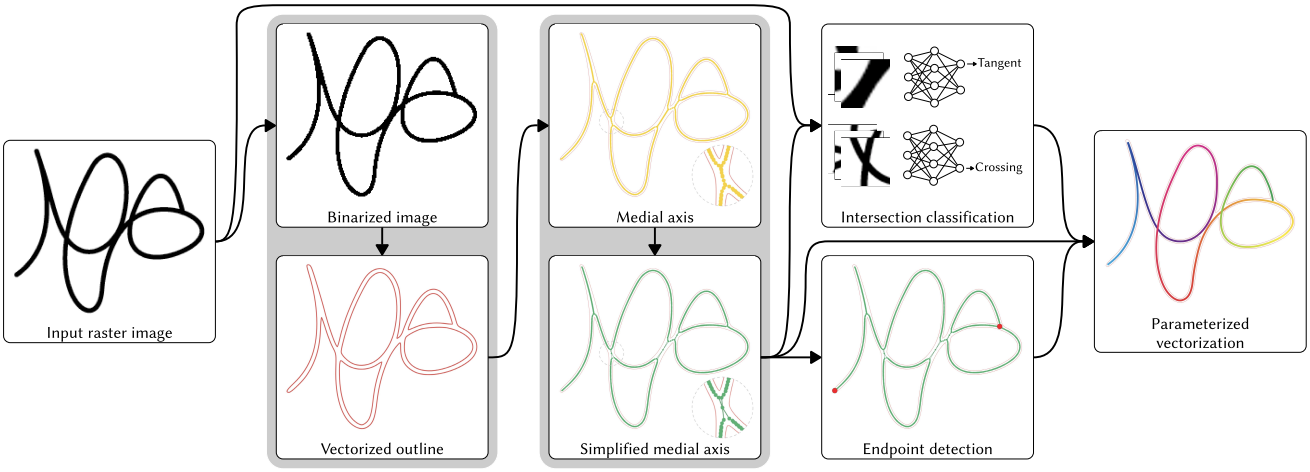


**Figure 3:** *Overview of our vectorization pipeline.*

focus on technical drawings and can only produce straight line segments and arcs [JV97,HT06]. Another interesting domain employing the medial axis, and where line parameterization is important, is handwritten character recognition [HY95, Jag96, KY00, QNY06]. However, the heuristics used for handwriting do not transfer well to freeform drawings.

Several improvements over simple skeletonization methods have been proposed in order to handle complex freeform drawings, mostly sharing a common approach: First, a graph is extracted from the drawing, then intersections are resolved and curves are fitted. Noris et al. [NHS*13] rely on the gradients of the image colors to move points towards the centerline. From these points, a graph is constructed and filtered through a minimum spanning tree algorithm with topology preserving features. Favreau et al. [FLB16] also rely on a 1-skeleton, but they compute it using the contour of a trapped-ball segmentation [ZCZ*09] and simple morphological thinning for the open curve. The obtained graph is then simplified to reduce the number of curves and their degree, but long strokes are still split into several parts, especially around junctions. Bo et al. [BLW16] also compute a skeleton from an input image, but rely on a larger scope of information to disentangle intersections. Curve boundary information can be utilized to improve the skeleton thinning [DCP19, ZLL*22]. While these methods show better results than simple medial axis extraction, they often fail to correctly handle complex curve junctions. In this line of work, the WrapIt method from Iarussi et al. [ILB15] stands out. It relies on simple morphological thinning to create the graph and vectorize the drawing, and uses simulated annealing to optimize a loss function that includes both the number of strokes and the continuity of those strokes. This allows to reduce the number of strokes. However, their simple method for graph creation may lead to poor vectorization, and, unlike our method, WrapIt does not utilize the image to disambiguate the intersections, leading to some errors.

The method of Zou and Yan [ZY01] relies on Delaunay triangulation to remove intersection artifacts of a skeletonization. However, it fails to correctly simplify the medial axis around long and thin intersections, which occur when two curves touch tangentially. In contrast, our method correctly captures such configurations.

*Frame field methods.* Bessmeltsev and Solomon [BS19] propose to vectorize line drawings by fitting frame fields to raster inputs, optimized to align with the curve tangents. Lines are traced in the frame field and grouped to create a graph, which is then simplified following drawing topology and converted into a vector representation. Puhachov et al. [PNCB21] extends this framework by detecting key points of the drawing with a neural network and integrating them as constraints in the graph representing the drawing. Guțan et al. [GHB*23] improves the frame field computation, removing all its singularities with an optimal transport based matching. Instead of tracing the frame fields, Stanko et al. [SBBB20] compute a grid based parameterization guided by the frame field and extract its isolines. Bao and Fu [BF23] rely on a similar approach by combining tangent and gradient vector fields to extract a graph, where junctions are optimized. While these methods usually produce satisfactory results, the involved optimization is computationally costly. Additionally, tracing is prone to error and often leads to discontinuous strokes, especially around high curvature areas or intersections.

*Deep learning methods.* Several works propose deep learning approaches to line art vectorization. Due to the sequential nature of strokes, many methods rely on recurrent neural networks (RNN) [HE17, DYH*21], involving a global latent representation, and thus tend to miss details in the drawing. To circumvent this issue, Liu et al. [LLLW22] use one latent embedding per stroke. Mo et al. [MSG*21] employ an RNN, but preserve the details better by leveraging local embeddings produced by a small convolutional neural network. Notably, they use direct image supervision through differentiable rasterization [LLGR20] and a regularization loss that encourages longer strokes. While improving the results, this still cannot prevent tiny strokes, and the method does not tackle curve intersection connectivity.

Other deep learning methods produce outputs that aid subsequent vectorization. Kim et al. [KWÖG18] predict a semantic segmentation of the line drawing into strokes. Each segment is then vector-

ized into regions with Potrace [Sel03]. Guo et al. [GZH*19] use a convolutional neural network (CNN) to predict a centerline and a junction image. Then, similarly to our approach, they use another CNN to recover the intersection connectivity. However, their CNN outputs images representing the different curves, while ours directly predicts the connectivity, making our method more efficient. Yan et al. [YLA*24] employ a CNN to predict an unsigned distance field to the center line, as well as other helper maps. The unsigned distance field is processed through a variation of Neural Dual Contouring [CTFZ22] and the result is refined with the other predicted maps to produce the vectorized output.

The above learning based approaches rely on datasets for training. Our method relies on a small CNN to locally disambiguate intersection connectivity, but it does not require training on a dataset of whole drawings, only on a synthetic dataset of curve intersections. In addition, our method relies on explainable geometric priors and uses a data-driven approach only when those are not enough.

**Single-line drawings.** While vectorization of single-line drawings has not been explicitly tackled in previous works, single-line drawings have been studied for the task of image stylization and generation. Early works [BH04, KB05] sample points based on the intensity of the input image and then connect them by finding a path through all the points. More advanced methods involving optimal transport [LdKW19], segmentation of the input image [WT13a, LHM17] or the contour of a tree extracted from the image [LM14] have been proposed. All these methods generate intersection-free single-line drawings. Wong and Takahashi [WT11] propose a method that can generate self-intersecting single-line drawings based on edges detected in the image. Other methods can merge an input set of single-line drawings into one complete single-line drawing [WT13b]. All these works are generative and are not capable of vectorizing existing raster representations of single-line drawings.

Another related line of work is the reconstruction of wire art from multiple images of different views [LCL*17]. The method from Van Mossel et al. [VMLV*21] allows to simplify overdrawn vector drawings containing clusters of strokes. While related to our work, both methods differ in the kind of input they accept.

## 3. Method

We assume the input to be a raster representation of a *single-line* drawing. Our method vectorizes the input with a correct line parameterization, i.e., it outputs a single parametric curve that follows the order artists commonly use to draw strokes. We show in Sec. 4.5 that our method can be easily extended to line drawings containing multiple strokes. We assume that the drawing contains crossings of no more than two lines, since higher-order crossings are rare in single-line drawings.

The overview of our vectorization pipeline is shown in Fig. 3. The rasterized image is first preprocessed: it is binarized and vectorized using a region-based method (Sec. 3.1). The medial axis graph of the resulting shape is then extracted. The obtained graph contains noisy branches and other artifacts, and must be simplified (Sec. 3.2). This simplified graph is not a 1-manifold, as self-intersections in the original drawing yield nodes with degrees higher than 2. To generate a continuous curve from it, we recover the drawing sequence using a neural network to predict valence-4 nodes' configurations (Sec. 3.3). At the end of this step, the single-line drawing is represented by a list of points (a polyline). We apply curve fitting to compute the final cubic Bézier spline representation (Sec. 3.4).

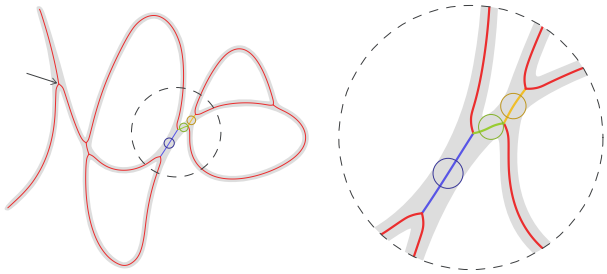

**Figure 4:** *Left: A single-line drawing with its medial axis. The node at the tip of the arrow does not need to be merged. Right: Two pairs of degree-3 nodes need to be merged. The green branch is the shorter one, but the blue and yellow branches have larger distances to the shape outline (larger circle), so these branches should be contracted.*

### 3.1. Preprocessing

We binarize the input by converting RGB images to grayscale and then using the thresholding method of Li and Lee [LL93], computed using the iterative algorithm of Li and Tam [LT98]. By default, the image is not blurred before binarization, but the user can choose to apply a Gaussian filter (Sec. 4.3), which removes noise and improves the binarization of pencil-drawn strokes. Subsequently, we vectorize the image using Potrace [Sel03], a region-based vectorization tool. The obtained vector shape is used to compute the medial axis of the line drawing. While it could be done directly on a binary image [ZS84], we find that a vectorized outline of the stroke leads to more precise medial axis results.

### 3.2. Medial axis computation

The vectorized outline of the input single-line drawing is represented by a series of curves. We compute the medial axis of this shape with the Voronoi diagram method [BA92]. The medial axis is then pruned using the vanishing angle [RJ23], a method that is particularly well suited for line drawings, since it preserves tubular shapes while removing noisy branches.

*Medial axis simplification.* The medial axis graph faithfully represents the centerline of the simpler parts of the line drawing, but it often fails to capture intersections correctly. Where the graph should have a node of degree 4, the medial axis tends to have clusters of degree-3 nodes (see Fig. 4). This happens because our inputs have organic, irregular shapes, such that it is unlikely for the medial axis computation to find an isolated point at the curve intersection that is exactly equidistant to 4 points on the curve outline. We assume that the line drawing contains intersections of a maximum of two lines, hence we need to find pairs of degree-3 nodes in the medial axis that stem from the same intersection of two lines and should

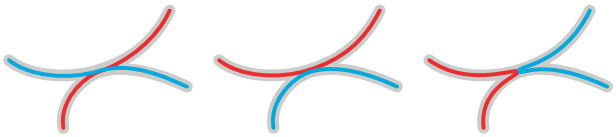

**Figure 5:** *Two curves can intersect in 3 different configurations. Left: crossing intersection. Middle and right: touching configuration.*

**Figure 6:** *Example of generated samples used to train the classification neural network. Left: intersecting samples. Right: tangential samples.*

be merged into a degree-4 node. A naive approach would be to merge a node of degree 3 with its nearest neighbor, either based on Euclidean or graph distance. However, this can lead to erroneous connectivity, since two true intersections might be closer to each other than the distance between two degree-3 nodes belonging to the same intersection (see Fig. 4). We observe that around intersections, the width tends to be thicker compared to the rest of the strokes due to multiple passes of drawing. We use this insight to identify the right pairs of degree-3 nodes to merge. We traverse the three outgoing branches (i.e., sequences of degree-2 nodes) of each degree-3 node and record the smallest distance to the shape outline along each branch, meaning, the width $w$ of the narrowest spot. The width of the narrowest spot is computed as the smallest medial-axis radius along the branch. We pick the branch whose $w$ value is the largest and mark its end node as the selected candidate to merge with our node. We prioritize the merging of node pairs where both nodes selected each other in this procedure, and then proceed to the remaining degree-3 nodes.

Note that some degree-3 nodes do not need to be merged, because they correctly represent the curve configuration along a sharp feature (see Fig. 4, left). We call a degree-3 node a *Y-junction* if its foot ends in a degree-1 node. The *foot* of a degree-3 node is the branch opposite of the smallest angle formed by two branch tangents. At the tip of a branch, the tangent is computed as the average of the first $k = 20$ edges. We preserve Y-junctions as they represent sharp features of the curve and must be traversed twice.

### 3.3. Graph ordering

The core step of our method is determining the correct traversal sequence on the simplified medial axis graph to obtain the curve parameterization, represented as a polyline. The traversal starts at one of the detected endpoints and proceeds along degree-2 nodes where the order is unambiguous; each traversed graph node is marked as such. Whenever a degree 1, 3 or 4 node is encountered, our algorithm must choose the next outgoing edge, the most challenging configuration being a degree-4 node. The goal is to traverse the entire graph and walk on each node to form a sequence of neighboring vertices that can be interpreted as a 1-manifold. We describe the components of our traversal algorithm below.

*Endpoint detection.* We rely on a set of simple rules to detect the endpoints in the simplified medial axis graph. Endpoints are necessarily odd degree nodes. All degree-1 nodes are endpoints, except the ones that are at the feet of Y-junctions, since these degree-1 nodes represent sharp features of the curve, and their branches are traversed twice. If a node has valence 3, we consider it an endpoint if none of its branches lead to a degree-1 node. Nodes of valence 5 and higher cannot be created by our method and are therefore not handled. Since we assume the drawing to be a single-line drawing, there should be either two or zero endpoints. If there are no endpoints detected, the drawing is a closed curve, and we can start the traversal from any node.

#### 3.3.1. Intersection classification.

Each degree-4 node can be traversed in three ways: crossing or touching in two different configurations, see Fig. 5. Determining the correct option is quite challenging, as the curve tangents that can be extracted from the medial axis are noisy and unreliable. We train a small neural network to predict whether the two curves forming the intersection are *crossing* or *tangent*. Specifically, we use an efficient fully convolutional neural network, ResNet50 [HZRS16], which can predict two classes.

*Training dataset.* The network is trained on a fully synthetic dataset that we create procedurally. Each sample is composed of two curves that either intersect or touch tangentially. Each curve is a single cubic Bézier curve, and we rasterize them with various brushes using Blender's Grease Pencil tool [Ble24]. Some samples from the training dataset are shown in Fig. 6. We generate 100,000 samples for each of the two classes.

*Training.* The network is trained on a single RTX 4090 GPU for 30 epochs using the Adam optimizer [KB15]. The learning rate is initialized at 0.001 and is multiplied by 0.2 every 5 epochs. We apply random data augmentations including color jitter, blur, noise, vertical and horizontal flips, rotations and perspective to the input of the network.

*Performance.* Similarly to the training set, we also create a validation set containing 5,000 samples of each class, as well as two small benchmark test sets, each containing 50 samples. The first test set consists of curves drawn on paper and then scanned, and the second benchmark contains curves drawn using a tablet. After training, our model achieves 95% accuracy on the synthetic validation dataset, 88% accuracy on the tablet benchmark and 79% accuracy on the pen-and-paper benchmark. We suspect that the gap between synthetic and real data is caused by the curves themselves and not by their rasterization. Indeed, to evaluate the model on challenging examples, our validation datasets include ambiguous cases: curves crossing at very shallow angles that may appear tangential, and tangent curves that resemble crossings ones.

*Classifying an intersection.* At inference time, to classify the type of a degree-4 node, we crop a small part of the input image around the intersection. To further improve the robustness of the model, for every intersection, we augment the initial crop with multiple images of it with rotations and other appearance augmentations. We then use

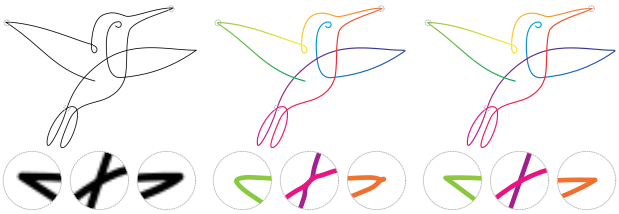

**Figure 7:** *Effect of node filtering on the result. From left to right: input image, curve fitted before node filtering, curve fitted after node filtering. Original hummingbird design © Felix Hornung.*

the network to classify the intersection on each of these images, and take the most frequent prediction. The frequency of that prediction defines the confidence of the model. If the intersection is classified as *crossing*, there is no ambiguity, but for *tangent* intersections, two different traversals are possible (see Fig. 5). We choose the one minimizing tangent discontinuity, measured as the angle between consecutive tangent vectors along the line.

#### 3.3.2. Graph traversal

Having determined the endpoints and classified each degree-4 node, we can traverse the graph. The goal is to obtain a sequence of nodes, with some repetitions, that can form a 1-manifold representing the drawing. We start from an endpoint: If it has valence 1, then its only neighbor is the next node in the traversal; otherwise, it is a degree-3 node, and we go in the direction opposite to the largest angle formed by two branch tangents, as the other two branches should be traversed sequentially since they are the most continuous ones. We continue traversing the graph, applying the rules listed below, depending on the degree of the current node. For degree-2 nodes, we move along the edge not yet visited. For degree-4 nodes, we continue according to the prediction of our model. The rules for degree-3 nodes are somewhat more complex, since we need to pick among two outgoing edges. Remaining degree-3 nodes in the simplified medial axis are either Y-junctions or endpoints. To traverse Y-junctions, we first go in the direction of the degree-1 node if the node has not yet been visited; if not, we go in the only unvisited direction. Otherwise the node is an endpoint, and we stop there if all connected nodes have been traversed, or continue in the direction of the closest tangent. Once an endpoint is reached, the stroke is stopped. A new one begins if not all nodes have been traversed. If all endpoints have been used, we start the traversal in both directions from a random unvisited degree-2 node.

At the end of this process, we have a list of nodes and their positions representing the stroke. If all nodes have been visited with a single stroke, we terminate. Otherwise, it means that some degree-4 nodes, common to two strokes, have been traversed incorrectly. In that case we flip the intersection type predicted by the model for the degree-4 node with the lowest confidence, and re-traverse the graph. We repeat this process until all nodes have been traversed with a single curve or all degree-4 nodes predictions with a confidence score lower than 1 have been switched.

### 3.4. Curve fitting

The previous step leaves us with a polyline representing the drawing. We could stop there, but in order to get a more appealing result, we fit a cubic Bézier spline to approximate the polyline. We first filter out nodes that lie in regions of high uncertainty, such as intersections, and may not represent the underlying stroke well. We remove all the points that are too close to an intersection, i.e., within a given Euclidean distance from a neighbor of a degree-4 node. The distance used is the medial axis radius. We keep the degree-4 nodes themselves only if the intersection is a *crossing* and not too close to its neighbors (see the middle inset of Fig. 7 for the effect of this). We also remove all the points on the foot of a Y-junction, except the one at the very end of the branch, since those are nodes traversed twice, which are part of a sharp feature of the curve. This allows the curve to traverse the branch more freely; the effect of this filtering can be seen in the right inset of Fig. 7. Finally, instead of fitting a single smooth curve, we split the polyline into several segments at degree-1 nodes, since those nodes represent sharp features that should be preserved (see the left inset of Fig. 7). We then fit a cubic B-spline to each segment of the polyline with the `splprep` function of the `scipy` library [VGO*20]. We fit the curve tightly for computing the metrics (Table 1) and more loosely to have smoother curves for all the figures in this paper. The cubic B-spline is then converted to a series of cubic Bézier curves using the `insert` function of `scipy`, which is based on the FITPACK Fortan routine [Die95]. Converting the B-spline to Bézier curves allows for easier manipulation and storage as an SVG file. Fig. 7 demonstrates the effect of the node filtering on curve fitting, and shows the final result of our method.

## 4. Results and discussion

We implement our method in Python and C++ and test it on a 32-cores Intel i9-13900K @ 5.5GHz with 64GB RAM machine with an NVIDIA GeForce RTX 4090 GPU for the CNN training and predictions. It takes around 10 seconds to automatically vectorize a raster image of a line drawing of 1000 × 1000 resolution. Our source code is available at https://github.com/tanguymagne/SLD-Vectorization.

### 4.1. Quantitative evaluation

We compare our results to commercial software [Ill24, Ink24], and to state-of-the-art line drawing vectorization methods, such as frame-field methods [BS19, PNCB21, GHB*23], deep-learning approaches [MSG*21, YLA*24] and to WrapIt [ILB15]. We also applied the WrapIt algorithm to our simplified graph and compared its output against our approach. Note that Inkscape's line drawing vectorization is based on Autotrace [Web24]. We were unable to compare to the deep learning based method of Guo et al. [GZH*19] because their code is not publicly available.

*Benchmark.* We compile a dataset of 56 *vector* single-line drawings, created by artists or taken from stock websites. This way we can compute metrics by directly comparing the vector representation of the input images and the output results. We rasterize the line drawings using unstylized strokes in a resolution of 1000 pixels in the longest direction.

*Evaluation metrics.* We follow Yan et al. [YLA*24] to measure

**Table 1:** *Quantitative comparisons of vectorization performance on the single line drawing benchmark, given as averages of the measured metrics. Gold, silver and bronze background indicates best, second-best and third-best scores, respectively. The first two metrics are measured on the rasterized vector graphics:* SSIM *stands for structural similarity index measure [WBSS04],* PSNR *is peak signal-to-noise ratio. We also measure the quality of the vectorized representation using* Length ratio*–the relative total stroke length difference between the ground truth and vectorized result, as well as stroke count (which ideally should be 1). Our approach stands out by creating significantly fewer strokes, with median stroke count of 1. We report the sensitivity and specificity of intersection detection in the* Undetected *and* Hallucinated *columns. Our method provides the best balance between these metrics, with the lowest rate of missing intersections. Adobe Illustrator shows fewer hallucinated intersections only because it produces almost no intersections–its average rate of missing intersections is 91.73%. Refer to Sec. 4.1 for more details.*

| | Image | | Stroke | | Intersections | |
|---|---|---|---|---|---|---|
| Method | SSIM ↑ | PSNR ↑ | Length ratio ↓ | Count ↓ | Undetected ↓ | Hallucinated ↓ |
| Adobe Illustrator [Ill24] | 0.958 | 19.1 | 4.71% | 19.7 | 91.73% | 3.57% |
| Autotrace [Ink24] | 0.969 | 21.1 | 2.73% | 19.7 | 88.43% | 36.41% |
| Bessmeltsev and Solomon [BS19] | 0.986 | 26.6 | 0.85% | 30.9 | 26.73% | 18.42% |
| Puhachov et al. [PNCB21] | 0.985 | 26.2 | 7.89% | 43.0 | 64.64% | 83.90% |
| Mo et al. [MSG*21] | 0.975 | 23.3 | 12.78% | 55.7 | 40.10% | 76.76% |
| Guțan et al. [GHB*23] | 0.984 | 25.5 | 0.93% | 25.3 | 32.48% | 20.61% |
| Yan et al. [YLA*24] | 0.983 | 26.3 | 1.04% | 10.5 | 16.90% | 19.39% |
| Iarussi et al. [ILB15] | 0.969 | 21.4 | 3.01% | 8.4 | 62.34% | 29.82% |
| Iarussi et al. [ILB15] with our graph | 0.983 | 25.2 | 2.78% | 2.5 | 37.01% | 9.87% |
| Ours | 0.987 | 27.8 | 0.99% | 1.2 | 6.42% | 8.46% |

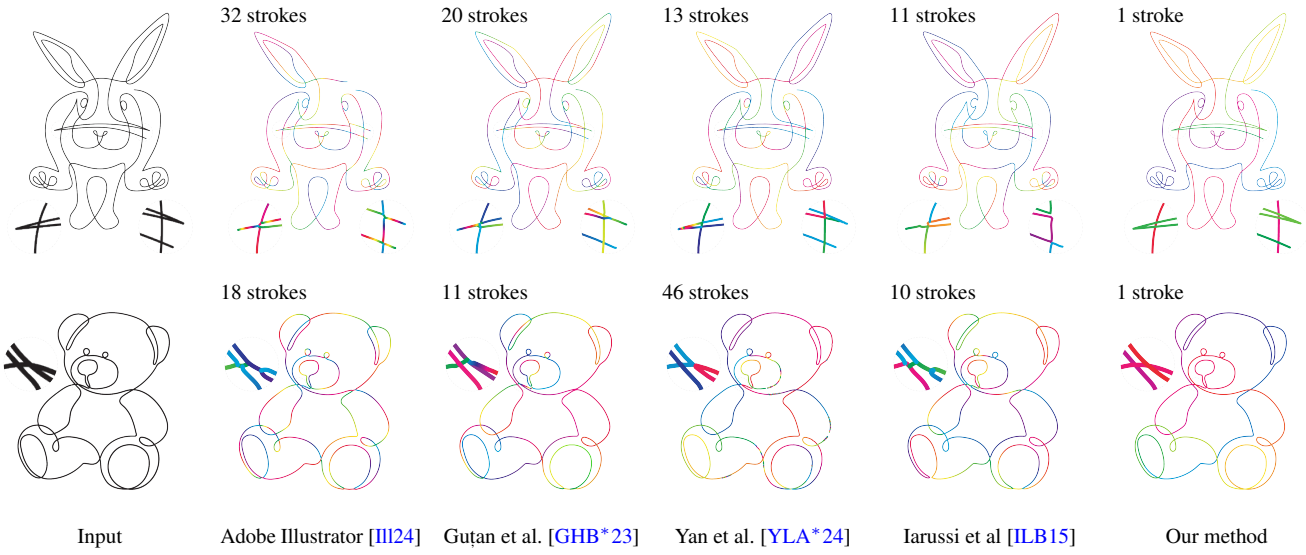


**Figure 8:** *Qualitative comparison of various line drawing vectorization methods. The top example is an open line, while the bottom is a drawing with a closed curve. The color gradients represent stroke parameterization. The ground truth parameterization can be found by following the line in the input image; our method reproduces the ground truth parameterization in these examples. The vector graphics version of this figure is available at https://github.com/tanguymagne/SLD-Vectorization. Original rabbit design © Felix Hornung.*

the vectorization quality directly in the vector domain. To account for repeated strokes (or lack thereof) we measure the relative total stroke length difference between the ground truth and vectorized result. Since we are interested in correctly vectorizing each stroke, we also count the number of strokes. To measure how well intersections are captured, we compute all intersections between curves, including self-intersections, in both the ground truth and the output vector representations. An intersection is counted as common between ground truth and output if the corresponding points lie within a 5-pixel distance threshold. We report the ratio of intersections that are present in the ground truth but not in the output, termed *undetected* intersections, and the ratio of intersections found in the

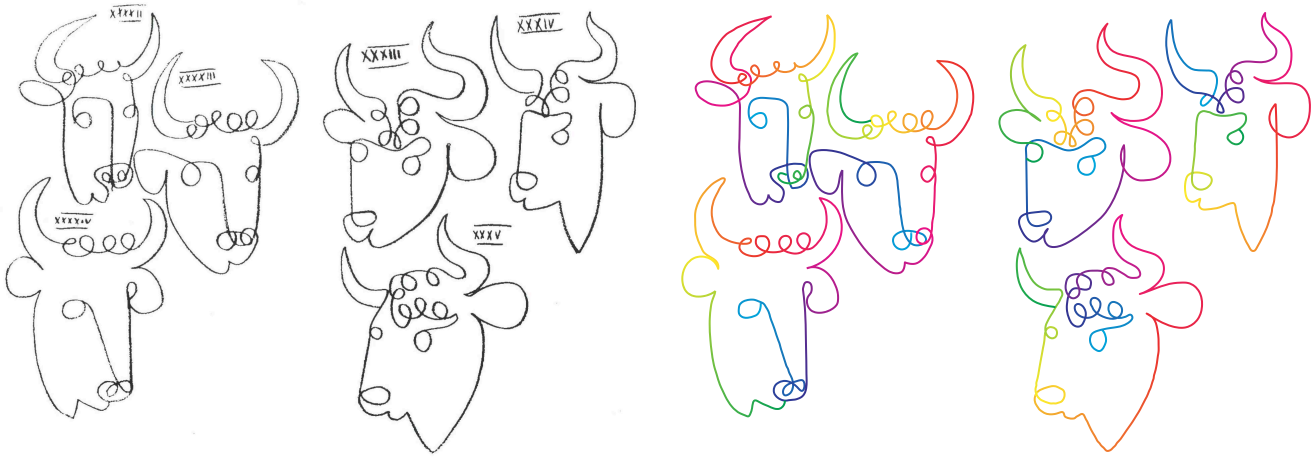

**Figure 9:** *Vectorization of Picasso's bulls. The input line drawing (left) is quite rough and complex. Our method, enhanced with some manual input, creates a clean vectorization (right).*

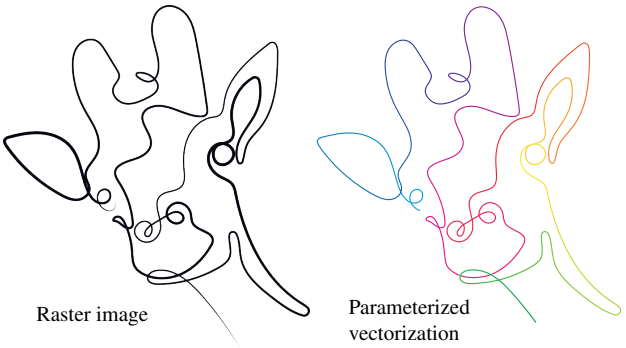


**Figure 10:** *Example of a single-line drawing with varying width. Left: input raster image. Right: our algorithm recovers the stroke's parameterization.*

output representation but not in the ground truth, termed *hallucinated* intersections. To measure the visual similarity between the input image and the obtained vector representation, we rasterize the output vector result using the same resolution and stroke width as the input. We compare the two images using standard image similarity metrics: PSNR (peak signal-to-noise ratio) and SSIM (structural similarity index measure) [WBSS04].

*Quantitative results.* We present the comparison results in Table 1. Our method achieves higher image similarity metrics than commercial software and comparable but better image similarity metrics than other state-of-the-art line drawing vectorization methods. In terms of vector quality, the four best methods, including ours, achieve similar results in terms of relative length difference: around 1%. However, we use a much smaller number of strokes to reach this performance. On the dataset, our method produces an average of 1.2 strokes per drawing. In comparison, all the other methods produce more than 5 times more strokes. Only WrapIt [ILB15] when applied on our simplified graph produces a reasonable amount of strokes, but still more than our method, showing that relying on the input image is important to disambiguate intersections. Our method produces a single-line representation of the input drawing for 46 out of the 56 drawings and never outputs more than 3 strokes per drawing. In terms of intersection handling, our method achieves the best balance between undetected and hallucinated intersections and the lowest ratio of missed intersections. Only Adobe Illustrator [Ill24] produces fewer hallucinated intersections, but this is only because it does not create almost any intersections at all (almost all ground truth intersections are undetected).

### 4.2. Qualitative evaluation

Figures 8, 15 and 16 present side-by-side comparisons of the results produced by multiple line drawing vectorization methods on several single-line drawings with open and closed curves. The entire set of results on our benchmark dataset is provided in the supplementary. The first row of Fig. 15 (the seahorse) shows that even for curves without self-intersections, state-of-the-art methods can produce multiple disconnected strokes. While this could be easily resolved by stitching nearby stroke endpoints, it shows that these methods are not well suited to vectorize long strokes. The other examples and their insets show multiple cases of ambiguous curves, such as crossing and tangential intersections, or sharp features correctly vectorized by our method, but wrongly represented in the outputs of other methods.

*Varying width.* Although our method relies on the distance from the medial axis to the border of the stroke to simplify the medial axis, it can still handle drawings with varying stroke width. An example of such a case is presented in Fig. 10. All samples in the benchmark dataset are single-line drawings of constant stroke width. This is because SVG, the most standard vector graphics format, does not support varying stroke width, and to represent such strokes in this format, one must represent the line contours. This makes the evaluation of all vector quality metrics impossible.

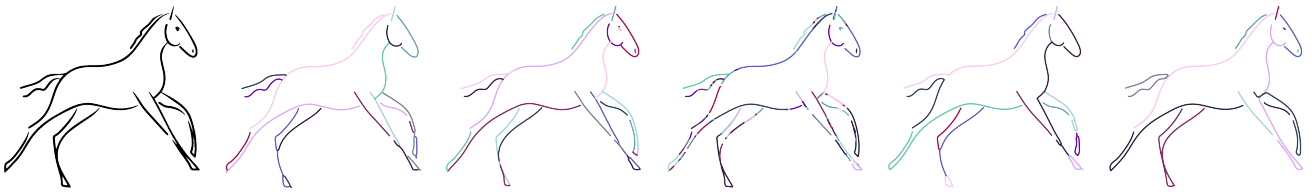


**Figure 11:** *Qualitative comparison of various line drawing vectorization methods on a simple multiple-line drawing. The multiple-line drawing was taken from the Adobe Stock website. Each stroke is represented with a different color. The vector graphics version of this figure is available at https://github.com/tanguymagne/SLD-Vectorization.*

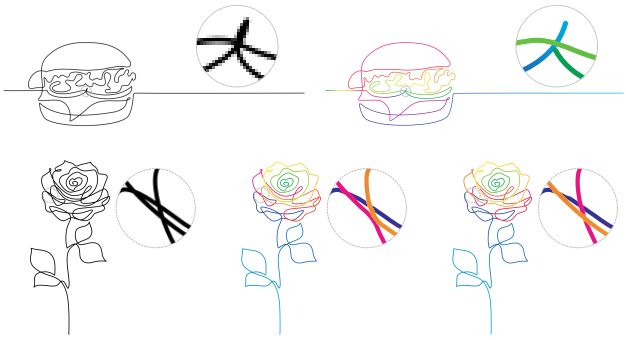

**Figure 12:** *Two examples of failure cases of our algorithm. Top: An intersection with 5 branches is not correctly simplified in the medial axis graph, thus 2 curves are created. Bottom: The intersection classification neural network incorrectly predicts one of the intersections (middle), but a simple click in our interactive refinement interface allows one to fix it (right).*

### 4.3. Manual editing

In addition to our automatic pipeline, we create a graphical user interface (GUI) that enables the user to easily update the output vectorization. In the preprocessing stage, the user can enable image blurring and manually select the binarization threshold. This gives the user control over the actual input of the vectorization. In particular, it can be used to manipulate the topology of the vectorization, as, the threshold can influence the number of connected components. The user can also modify the simplified medial axis. Any degree-4 node in the graph comes from the merge of two degree-3 nodes; the user can select a degree-4 node and split it, restoring the degree-3 nodes if they were wrongly merged. The user can also collapse a branch of the medial axis by selecting it, effectively merging two degree-3 nodes and forming a new degree-4 node. It is also possible to manually change the classification of the neural network to choose the connectivity of ambiguous intersections. Clicking on an intersection switches the intersection type between *tangent* and *crossing*. Finally, the result can be easily exported as an SVG file, and the curve can be further modified in any vector graphics editor. Refer to the accompanying video and code for a demonstration. These editing methods have been used to correctly vectorize Picasso's bulls presented in Fig. 9. A simple click also allows to fix the result at the bottom of Fig. 12, as explained in the next section.

### 4.4. Limitations

In some cases, our algorithm may fail to output the correct stroke topology (stroke number and parameterization). As it is not designed to handle intersections with more than two intersecting curves, our method may struggle to correctly vectorize such cases. This is demonstrated in the first row of Fig. 12: a node of degree 5 should be created in the simplified medial-axis graph, but our algorithm can only create nodes of degree 4 at most. This causes our algorithm to detect an endpoint and thus create two curves. Another failure mode is due to prediction error of the neural network when classifying intersection, demonstrated in the bottom row of Fig. 12. This type of error does not always lead to the creation of multiple strokes, but rather to a stroke that does not follow the expected drawing order. Fortunately, this is a rare occurrence and is easy to fix using our simple GUI. Other errors may also occur. For instance, an incorrect endpoint might be detected due to the rather simpler heuristics we employ for this task. In rare cases, two degree-3 nodes can be incorrectly merged. This mostly happens when the input resolution is too low, and the thickness of the line is thus not well captured in the raster image.

### 4.5. Extension: multiple lines drawings

While the primary target of our method is single-line drawings, it can easily be extended to tackle drawings containing multiple strokes, with simple modifications of the algorithm. First, we choose a slightly lower pruning threshold for the medial axis, as multiple line drawings contain smaller strokes. We also add degree-3 nodes that have a T-shape to the list of endpoints. These are degree-3 nodes where the largest angle between the branches' tangents is close to $180^\circ$ and the remaining two angles are close to each other. Finally, once all nodes have been traversed in the medial axis graph, if multiple strokes have been created, we skip the step that attempts to switch low-confidence nodes' intersection type. Using this algorithm, we can vectorize clean line drawings containing multiple strokes. Compared to other state-of-the-art methods, our method produces longer strokes that more faithfully represent the intended lines drawn by the artist. See Figures 11 and 17 for some comparative results. Our algorithm produces better results than other methods on these simple examples. However, on more complex examples, our algorithm can yield worse results, because our heuristics for endpoint detection are simple, and our method tries to create long strokes to minimize their amount, which does not always fit well with complex drawings.

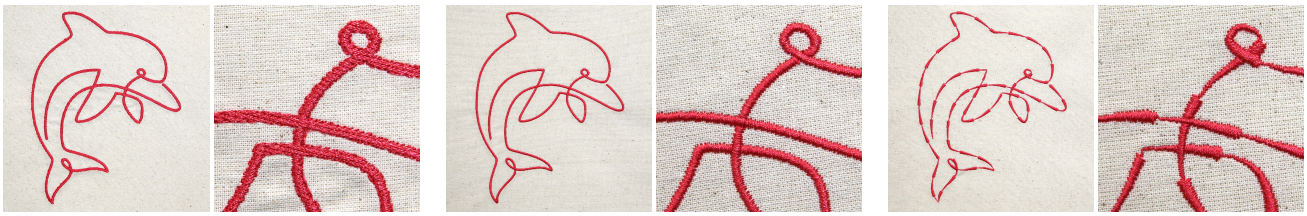

**Figure 13:** *Machine embroidery of a line drawing. Provided with a raster image as input, the machine randomly fills it with stitches, resulting in untidy appearance (left). In contrast, our parameterized vectorization yields crisp results that follow the curve, e.g. using perpendicular stitches (middle) or stylized arrows (right). Original dolphin design © Felix Hornung.*

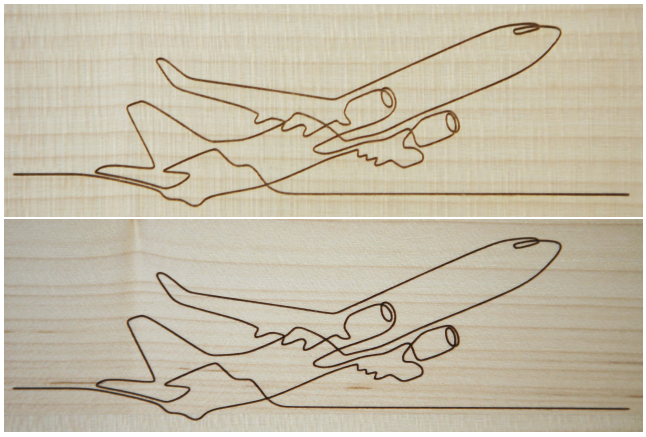

**Figure 14:** *Single line drawing engraved on wood. Top: engraved from a raster representation, taking 193 seconds. Bottom: engraved from a vector representation, taking only 20 seconds and showing a crisper result.*

### 4.6. Application to fabrication

*Embroidery.* Since our vectorized outputs are correctly parameterized, they are easy to automatically machine-embroider on fabric using multiple styles, such as backstitch (Fig. 1, right), zigzag stitches orthogonal to the curve or variable textured stitching along the curve (Fig. 13), yielding aesthetic, crisp results. In contrast, when using a raster image as input, the sewing machine first uses a region based vectorization method and then simply fills the shape with stitches, not respecting the tangent directions of the curve and resulting in a blurry appearance. Using the output of other vectorization methods would produce results containing several artifacts due to the line being represented by multiple strokes. For instance the stylized arrows on the right of Fig. 13 would start over for every stroke.

*Engraving.* Our method benefits laser engraving along curved paths. When engraving a drawing on e.g. a piece of wood from a raster drawing, the machine proceeds layer by layer, because it has no awareness of the line parameterization. Using our algorithm to first vectorize the single line drawing and then using the vector representation as input allows the machine to follow the line order and directly engrave the parts of interest. This makes the engraving process much faster. For instance, the drawing in Fig. 14 was engraved in 193 seconds using the raster image as input and in 20 seconds using the image vectorized with our method. Using the output of other vectorization methods would also take more time, since the laser head would need to move to the start of each new stroke and could not proceed in one go. In addition, the engraving is slightly cleaner and more pronounced when using vector input.

## 5. Conclusion

We presented a new method for line drawing vectorization, specifically designed to vectorize *single-line* drawings with the correct parameterization. It relies on computing the medial axis, which is then simplified through a geometric approach and traced. We leverage a small neural network to handle the inherent ambiguity of intersection traversal, since the features that distinguish between crossing and touching curves can be too subtle to encode in explicit rules. Our method proves effective at disentangling such difficult intersections on single-line drawings and outperforms state-of-the-art line drawing vectorization algorithms both quantitatively and qualitatively on a benchmark of such drawings. It also succeeds on simple line drawings containing multiple strokes, with only minimal extension of the technique.

Our preprocessing steps are quite rudimentary, in particular, blurring is invoked manually, and the thresholding may on rare occasions need to be manually adjusted as well. We believe that these tasks can be automated and improved by a specially tailored segmentation model. Our method is based on some heuristics to detect the endpoints of the curves. While this works well in our setting, we expect endpoint detection to benefit from a learning approach that could enhance the performance of our method on more complex multi-component line drawings.

## Acknowledgements

We thank the anonymous reviewers for their detailed feedback and helpful suggestions. We express our gratitude to Marcel Padilla and Loïc Magne for their insightful comments and to Danielle Luterbacher for her help with the embroidery process. Open access publishing facilitated by Eidgenossische Technische Hochschule Zurich, as part of the Wiley - Eidgenossische Technische Hochschule Zurich agreement via the Consortium Of Swiss Academic Libraries.

## References


[BA92] BRANDT J. W., ALGAZI V. R.: Continuous skeleton computation by voronoi diagram. *CVGIP: Image Understanding 55*, 3 (May 1992), 329–338. URL: https://www.sciencedirect.com/science/article/pii/1049966092900307, doi:10.1016/1049-9660(92)90030-7. 4

[BF23] BAO B., FU H.: Line drawing vectorization via coarse-to-fine curve network optimization. *Computer Graphics Forum 42*, 6 (2023), e14787. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14787, doi:10.1111/cgf.14787. 3

[BH04] BOSCH R., HERMAN A.: Continuous line drawings via the traveling salesman problem. *Oper. Res. Lett. 32*, 4 (July 2004), 302–303. doi:10.1016/j.orl.2003.10.001. 4

[Ble24] BLENDER: Blender. Blender Foundation, 2024. URL: http://www.blender.org. 5

[BLW16] BO P., LUO G., WANG K.: A graph-based method for fitting planar b-spline curves with intersections. *Journal of Computational Design and Engineering 3*, 1 (Jan. 2016), 14–23. URL: https://www.sciencedirect.com/science/article/pii/S2288430015000408, doi:10.1016/j.jcde.2015.05.001. 3

[BS19] BESSMELTSEV M., SOLOMON J.: Vectorization of line drawings via polyvector fields. *ACM Transactions on Graphics 38*, 1 (Jan. 2019), 9:1–9:12. URL: https://dl.acm.org/doi/10.1145/3202661, doi:10.1145/3202661. 2, 3, 6, 7

[CTFZ22] CHEN Z., TAGLIASACCHI A., FUNKHOUSER T., ZHANG H.: Neural dual contouring. *ACM Transactions on Graphics 41*, 4 (July 2022), 104:1–104:13. URL: https://dl.acm.org/doi/10.1145/3528223.3530108, doi:10.1145/3528223.3530108. 4

[DCP19] DONATI L., CESANO S., PRATI A.: A complete hand-drawn sketch vectorization framework. *Multimedia Tools and Applications 78*, 14 (July 2019), 19083–19113. URL: https://doi.org/10.1007/s11042-019-7311-3, doi:10.1007/s11042-019-7311-3. 3

[Die95] DIERCKX P.: *Curve and Surface Fitting with Splines*. Numerical Mathematics and Scientific Computation. Oxford University Press, Oxford, New York, Apr. 1995. 6

[DYH*21] DAS A., YANG Y., HOSPEDALES T. M., XIANG T., SONG Y.-Z.: Cloud2curve: Generation and vectorization of parametric sketches. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2021), CVF, pp. 7088–7097. URL: https://ieeexplore.ieee.org/document/9577952, doi:10.1109/CVPR46437.2021.00701. 3

[FLB16] FAVREAU J.-D., LAFARGE F., BOUSSEAU A.: Fidelity vs. simplicity: A global approach to line drawing vectorization. *ACM Transactions on Graphics 35*, 4 (July 2016), 120:1–120:10. URL: https://dl.acm.org/doi/10.1145/2897824.2925946, doi:10.1145/2897824.2925946. 2, 3

[GHB*23] GUȚAN O., HEGDE S., BERUMEN E. J., BESSMELTSEV M., CHIEN E.: Singularity-free frame fields for line drawing vectorization. *Computer Graphics Forum 42*, 5 (2023). URL: https://diglib.eg.org:443/xmlui/handle/10.1111/cgf14901, doi:10.1111/cgf.14901. 3, 6, 7, 9, 13, 14

[GZH*19] GUO Y., ZHANG Z., HAN C., HU W., LI C., WONG T.-T.: Deep line drawing vectorization via line subdivision and topology reconstruction. *Computer Graphics Forum 38*, 7 (2019), 81–90. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13818, doi:10.1111/cgf.13818. 4, 6

[HE17] HA D., ECK D.: A neural representation of sketch drawings, May 2017. URL: http://arxiv.org/abs/1704.03477, arXiv:1704.03477, doi:10.48550/arXiv.1704.03477. 3

[HT06] HILAIRE X., TOMBRE K.: Robust and accurate vectorization of line drawings. *IEEE Transactions on Pattern Analysis and Machine Intelligence 28*, 6 (June 2006), 890–904. URL: https://ieeexplore.ieee.org/document/1624354, doi:10.1109/TPAMI.2006.127. 3

[HY95] HUANG T., YASUHARA M.: Recovery of information on the drawing order of single-stroke cursive handwritten characters from their 2D images. *Transactions of Information Processing Society of Japan 36*, 9 (Sept. 1995), 2132–2143. URL: https://ipsj.ixsq.nii.ac.jp/ej/index.php?active_action=repository_view_main_item_detail&page_id=13&block_id=8&item_id=13823&item_no=1. 3

[HZRS16] HE K., ZHANG X., REN S., SUN J.: Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 770–778. URL: https://openaccess.thecvf.com/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html. 5

[ILB15] IARUSSI E., LI W., BOUSSEAU A.: WrapIt: Computer-assisted crafting of wire wrapped jewelry. *ACM Trans. Graph. 34*, 6 (Nov. 2015), 221:1–221:8. doi:10.1145/2816795.2818118. 2, 3, 6, 7, 8, 9, 13, 14

[Ill24] ILLUSTRATOR A.: Adobe illustrator. Adobe Inc., 2024. URL: https://adobe.com/products/illustrator. 2, 6, 7, 8, 9, 13, 14

[Ink24] INKSCAPE: Inkscape. The Inkscape Project, 2024. URL: https://inkscape.org/. 2, 6, 7

[Jag96] JAGER S.: Recovering writing traces in off-line handwriting recognition: Using a global optimization technique. In *Proceedings of 13th International Conference on Pattern Recognition* (Aug. 1996), vol. 3, pp. 150–154 vol.3. URL: https://ieeexplore.ieee.org/document/546812, doi:10.1109/ICPR.1996.546812. 3

[JV97] JANSSEN R. D. T., VOSSEPOEL A. M.: Adaptive vectorization of line drawing images. *Computer Vision and Image Understanding 65*, 1 (Jan. 1997), 38–56. URL: https://www.sciencedirect.com/science/article/pii/S1077314296904841, doi:10.1006/cviu.1996.0484. 3

[KB05] KAPLAN C. S., BOSCH R.: TSP Art. In *Renaissance Banff: Mathematics, Music, Art, Culture* (July 2005), pp. 301–308. 4

[KB15] KINGMA D. P., BA J.: Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (2015), Bengio Y., LeCun Y., (Eds.). URL: http://arxiv.org/abs/1412.6980. 5

[KWÖG18] KIM B., WANG O., ÖZTIRELI A. C., GROSS M.: Semantic segmentation for line drawing vectorization using neural networks. *Computer Graphics Forum 37*, 2 (2018), 329–338. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13365, doi:10.1111/cgf.13365. 3

[KY00] KATO Y., YASUHARA M.: Recovery of drawing order from single-stroke handwriting images. *IEEE Transactions on Pattern Analysis and Machine Intelligence 22*, 9 (Sept. 2000), 938–949. URL: https://ieeexplore.ieee.org/document/877517, doi:10.1109/34.877517. 3

[LCL*17] LIU L., CEYLAN D., LIN C., WANG W., MITRA N. J.: Image-based reconstruction of wire art. *ACM Transactions on Graphics 36*, 4 (July 2017), 63:1–63:11. doi:10.1145/3072959.3073682. 4

[LdKW19] LEBRAT L., DE GOURNAY F., KAHN J., WEISS P.: Optimal Transport Approximation of 2-Dimensional Measures. *SIAM Journal on Imaging Sciences 12*, 2 (Jan. 2019), 762–787. doi:10.1137/18M1193736. 4

[LHM17] LIU C., HODGINS J., MCCANN J.: Whole-cloth quilting patterns from photographs. In *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering* (New York, NY, USA, July 2017), NPAR '17, Association for Computing Machinery, pp. 1–8. doi:10.1145/3092919.3092925. 4

[LL93] LI C. H., LEE C. K.: Minimum cross entropy thresholding. *Pattern Recognition 26*, 4 (Apr. 1993), 617–625. URL: https://www.sciencedirect.com/science/article/pii/003132039390115D, doi:10.1016/0031-3203(93)90115-D. 4

[LLGR20] LI T.-M., LUKÁČ M., GHARBI M., RAGAN-KELLEY J.: Differentiable vector graphics rasterization for editing and learning. *ACM Transactions on Graphics 39*, 6 (Nov. 2020), 193:1–193:15. URL: https://dl.acm.org/doi/10.1145/3414685.3417871, doi:10.1145/3414685.3417871. 3

[LLLW22] LIU H., LI C., LIU X., WONG T.-T.: End-to-end line drawing vectorization. *Proceedings of the AAAI Conference on Artificial Intelligence 36*, 4 (June 2022), 4559–4566. URL: https://ojs.aaai.org/index.php/AAAI/article/view/20379, doi:10.1609/aaai.v36i4.20379. 3

[LM14] LI H., MOULD D.: Continuous Line Drawings and Designs. *International Journal of Creative Interfaces and Computer Graphics (IJCICG) 5*, 2 (July 2014), 16–39. doi:10.4018/ijcicg.2014070102. 4

[LT98] LI C. H., TAM P. K. S.: An iterative algorithm for minimum cross entropy thresholding. *Pattern Recognition Letters 19*, 8 (June 1998), 771–776. URL: https://www.sciencedirect.com/science/article/pii/S0167865598000579, doi:10.1016/S0167-8655(98)00057-9. 4

[MSG*21] MO H., SIMO-SERRA E., GAO C., ZOU C., WANG R.: General virtual sketching framework for vector line art. *ACM Transactions on Graphics 40*, 4 (July 2021), 51:1–51:14. URL: https://dl.acm.org/doi/10.1145/3450626.3459833, doi:10.1145/3450626.3459833. 2, 3, 6, 7

[NHS*13] NORIS G., HORNUNG A., SUMNER R. W., SIMMONS M., GROSS M.: Topology-driven vectorization of clean line drawings. *ACM Transactions on Graphics 32*, 1 (Feb. 2013), 4:1–4:11. URL: https://dl.acm.org/doi/10.1145/2421636.2421640, doi:10.1145/2421636.2421640. 3

[PNCB21] PUHACHOV I., NEVEU W., CHIEN E., BESSMELTSEV M.: Keypoint-driven line drawing vectorization via polyvector flow. *ACM Transactions on Graphics 40*, 6 (Dec. 2021), 266:1–266:17. URL: https://dl.acm.org/doi/10.1145/3478513.3480529, doi:10.1145/3478513.3480529. 3, 6, 7

[QNY06] QIAO Y., NISHIARA M., YASUHARA M.: A framework toward restoration of writing order from single-stroked handwriting image. *IEEE Transactions on Pattern Analysis and Machine Intelligence 28*, 11 (Nov. 2006), 1724–1737. URL: https://ieeexplore.ieee.org/document/1704830, doi:10.1109/TPAMI.2006.216. 3

[RJ23] RONG P., JU T.: Variational pruning of medial axes of planar shapes. *Computer Graphics Forum 42*, 5 (2023). URL: https://diglib.eg.org:443/xmlui/handle/10.1111/cgf14902, doi:10.1111/cgf.14902. 4

[SBBB20] STANKO T., BESSMELTSEV M., BOMMES D., BOUSSEAU A.: Integer-grid sketch simplification and vectorization. *Computer Graphics Forum 39*, 5 (2020), 149–161. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14075, doi:10.1111/cgf.14075. 3

[Sel03] SELINGER P.: Potrace : A polygon-based tracing algorithm, 2003. URL: https://potrace.sourceforge.net/potrace.pdf. 2, 4

[VGO*20] VIRTANEN P., GOMMERS R., OLIPHANT T. E., HABERLAND M., REDDY T., COURNAPEAU D., BUROVSKI E., PETERSON P., WECKESSER W., BRIGHT J., VAN DER WALT S. J., BRETT M., WILSON J., MILLMAN K. J., MAYOROV N., NELSON A. R. J., JONES E., KERN R., LARSON E., CAREY C. J., POLAT İ., FENG Y., MOORE E. W., VANDERPLAS J., LAXALDE D., PERKTOLD J., CIMRMAN R., HENRIKSEN I., QUINTERO E. A., HARRIS C. R., ARCHIBALD A. M., RIBEIRO A. H., PEDREGOSA F., VAN MULBREGT P.: Scipy 1.0: Fundamental algorithms for scientific computing in python. *Nature Methods 17*, 3 (Mar. 2020), 261–272. URL: https://www.nature.com/articles/s41592-019-0686-2, doi:10.1038/s41592-019-0686-2. 6

[VMLV*21] VAN MOSSEL D. P., LIU C., VINING N., BESSMELTSEV M., SHEFFER A.: StrokeStrip: Joint parameterization and fitting of stroke clusters. *ACM Transactions on Graphics 40*, 4 (July 2021), 50:1–50:18. doi:10.1145/3450626.3459777. 4

[WBSS04] WANG Z., BOVIK A., SHEIKH H., SIMONCELLI E.: Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing 13*, 4 (Apr. 2004), 600–612. URL: https://ieeexplore.ieee.org/document/1284395, doi:10.1109/TIP.2003.819861. 7, 8

[Web24] WEBER M.: Autotrace, 2024. URL: https://github.com/autotrace/autotrace. 6

[WT11] WONG F. J., TAKAHASHI S.: A Graph-based Approach to Continuous Line Illustrations with Variable Levels of Detail. *Computer Graphics Forum 30*, 7 (2011), 1931–1939. doi:10.1111/j.1467-8659.2011.02040.x. 4

[WT13a] WONG F. J., TAKAHASHI S.: Abstracting images into continuous-line artistic styles. *The Visual Computer 29*, 6 (June 2013), 729–738. doi:10.1007/s00371-013-0809-1. 4

[WT13b] WONG F. J., TAKAHASHI S.: Hierarchical Design of Continuous Line Illustrations. In *International Conference on Computer Graphics Theory and Applications* (Feb. 2013), vol. 2, SCITEPRESS, pp. 131–138. doi:10.5220/0004232701310138. 4

[WZY*24] WU Q., ZHANG Z., YAN X., ZHONG F., ZHU Y., LU X., XUE R., LI R., TU C., ZHAO H.: Tune-it: Optimizing wire reconfiguration for sculpture manufacturing. In *SIGGRAPH Asia 2024 Conference Papers* (New York, NY, USA, Dec. 2024), SA '24, Association for Computing Machinery, pp. 1–11. URL: https://dl.acm.org/doi/10.1145/3680528.3687588, doi:10.1145/3680528.3687588. 2

[YLA*24] YAN C., LI Y., ANEJA D., FISHER M., SIMO-SERRA E., GINGOLD Y.: Deep sketch vectorization via implicit surface extraction. *ACM Trans. Graph. 43*, 4 (July 2024), 37:1–37:13. URL: https://dl.acm.org/doi/10.1145/3658197, doi:10.1145/3658197. 2, 4, 6, 7, 9, 13, 14

[ZCZ*09] ZHANG S.-H., CHEN T., ZHANG Y.-F., HU S.-M., MARTIN R. R.: Vectorizing cartoon animations. *IEEE Transactions on Visualization and Computer Graphics 15*, 4 (July 2009), 618–629. URL: https://ieeexplore.ieee.org/document/4745633, doi:10.1109/TVCG.2009.9. 3

[ZLL*22] ZHANG Z., LIU X., LI C., WU H., WEN Z.: Vectorizing line drawings of arbitrary thickness via boundary-based topology reconstruction. *Computer Graphics Forum 41*, 2 (2022), 433–445. URL: https://doi.org/10.1111/cgf.14485. 3

[ZS84] ZHANG T. Y., SUEN C. Y.: A fast parallel algorithm for thinning digital patterns. *Communications of the ACM 27*, 3 (Mar. 1984), 236–239. URL: https://dl.acm.org/doi/10.1145/357994.358023, doi:10.1145/357994.358023. 4

[ZY01] ZOU J. J., YAN H.: Cartoon image vectorization based on shape subdivision. In *Proceedings. Computer Graphics International 2001* (July 2001), pp. 225–231. URL: https://ieeexplore.ieee.org/document/934678, doi:10.1109/CGI.2001.934678. 3

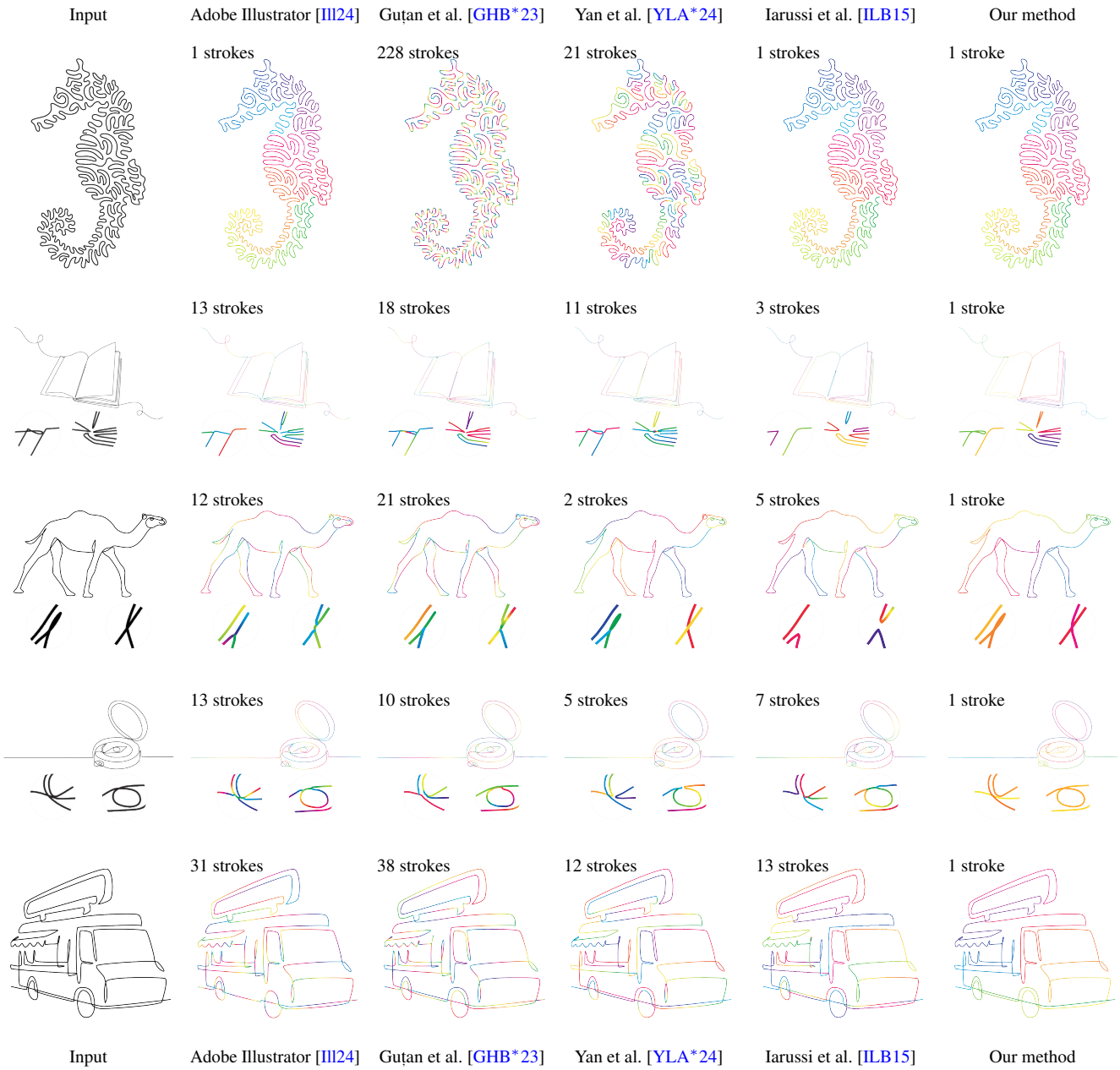


**Figure 15:** *Qualitative comparison of various line drawing vectorization methods on a set of single-line drawings taken from our benchmark dataset. The color gradients represent stroke parameterization. The ground truth parameterization can be found by following the line in the input image; our method reproduces the ground truth parameterization in these examples. The vector graphics version of this figure is available at https://github.com/tanguymagne/SLD-Vectorization.*

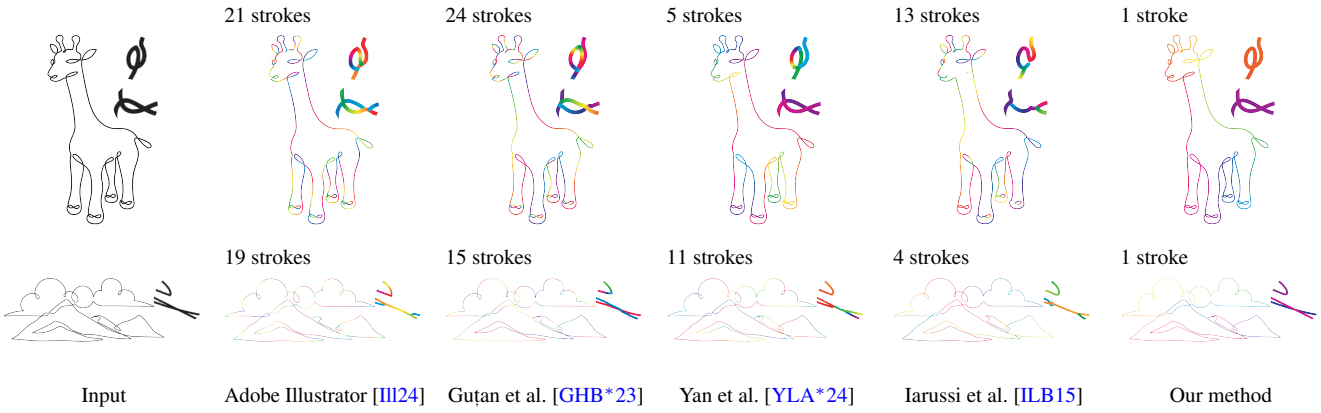


**Figure 16:** *Qualitative comparison of various line drawing vectorization methods on closed curve single line drawings. The color gradients represent stroke parameterization. The ground truth parameterization can be found by following the line in the input image; our method reproduces the ground truth parameterization in these examples. The vector graphics version of this figure is available at https://github.com/tanguymagne/SLD-Vectorization.*

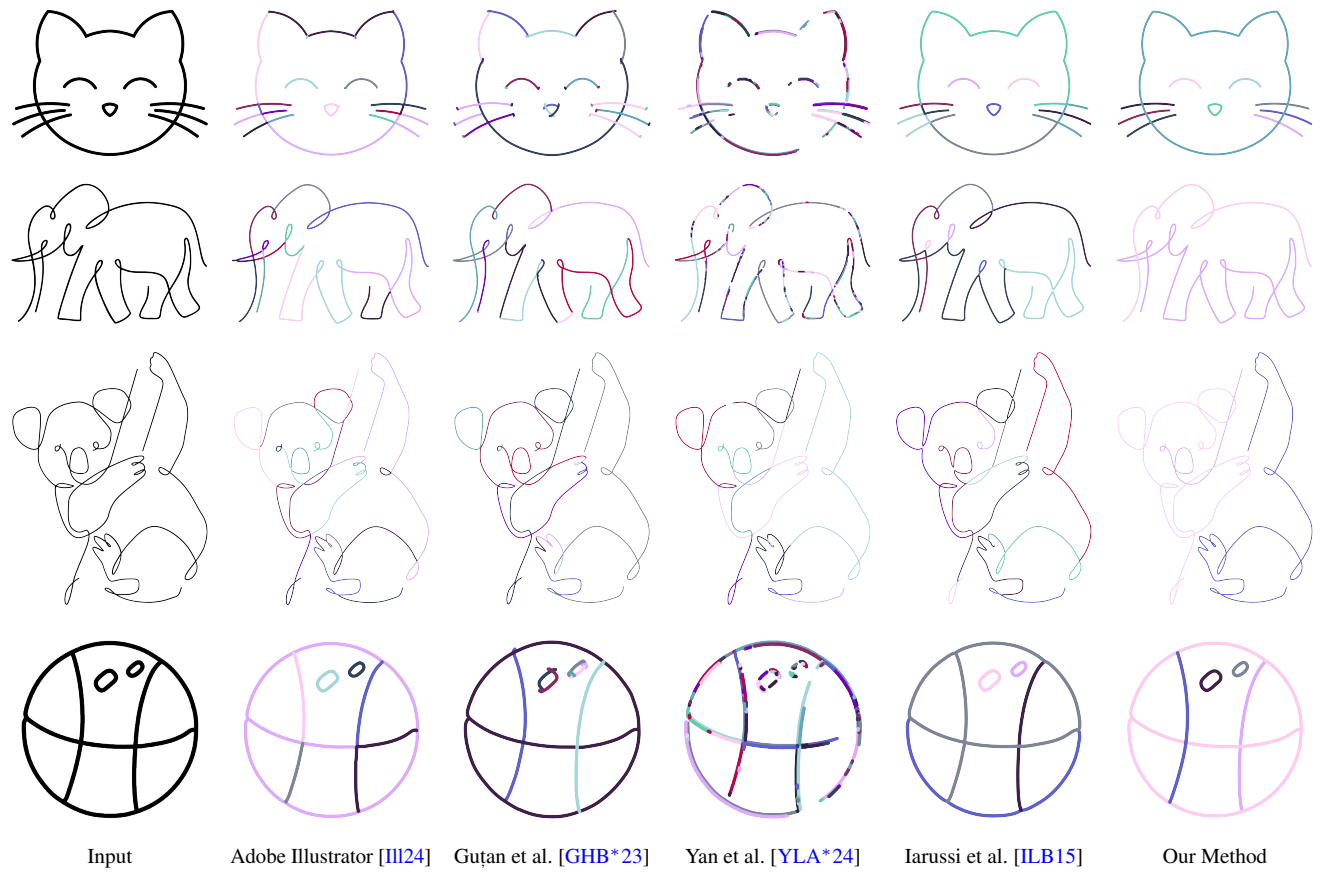


**Figure 17:** *Qualitative comparison of various line drawing vectorization methods on a set of multiple-line drawings. The multiple-line drawings were taken from the Adobe Stock website. Each stroke is represented with a different color. The vector graphics version of this figure is available at https://github.com/tanguymagne/SLD-Vectorization.*