



Published in Image Processing On Line on 2023-01-17.  
 Submitted on 2022-04-25, accepted on 2022-12-21.  
 ISSN 2105-1232 © 2023 IPOL & the authors CC-BY-NC-SA  
 This article is available online with supplementary materials,  
 software, datasets and online demo at  
<https://doi.org/10.5201/ipol.2023.401>

# Binary Shape Vectorization by Affine Scale-space

Yuchen He<sup>1</sup>, Sung Ha Kang<sup>2</sup>, Jean-Michel Morel<sup>3</sup>

<sup>1</sup>Institute of Natural Sciences, Shanghai Jiao Tong University, China

<sup>2</sup>School of Mathematics, Georgia Institute of Technology, US

<sup>3</sup>Centre Borelli, École Normale Supérieure Paris-Saclay, France

([yuchenroy@sjtu.edu.cn](mailto:yuchenroy@sjtu.edu.cn), [kang@math.gatech.edu](mailto:kang@math.gatech.edu), [moreljeanmichel@gmail.com](mailto:moreljeanmichel@gmail.com))

*Communicated by* Pascal Monasse

*Demo edited by* Pascal Monasse

## Abstract

Binary shapes, or silhouettes, are building elements of logos, graphic symbols and fonts which require various forms of geometric editing without compromising the resolution. In this paper, we present an effective silhouette vectorization algorithm that extracts the outline of a 2D shape from a raster binary image and converts it to a combination of cubic Bézier polygons and perfect circles. Compared to state-of-the-art image vectorization software, this algorithm has demonstrated a superior reduction in the number of control points while maintaining high accuracy.

## Source Code

The reviewed source code and documentation for this algorithm are available at [the web page of this article](#)<sup>1</sup>. See `README.txt` for usage instructions in the archive.

**Keywords:** silhouette vectorization; affine scale-space; curvature extrema

## 1 Introduction

A silhouette is a subset of the plane traditionally obtained by copying on paper the shadow projected on a wall by a person placed in front of a point light source<sup>2</sup>. In modern computer graphics, a silhouette's boundary is described as a union of primitive components, such as line segments, circular arcs, and Bézier curves for their real-time rendering [18]. The boundaries of these shapes are encoded in the Scalable Vector Graphics (SVG) format<sup>3</sup> where each involved primitive element is specified by a small number of 2D vectors, called *control points*, and the encoded shape can be scaled independently from the resolution. Very often the silhouette is first given as a grayscale image. Thresholding this image yields a binary image, but this representation as a union of black

<sup>1</sup><https://doi.org/10.5201/ipol.2023.401>

<sup>2</sup><https://en.wikipedia.org/wiki/Silhouette>

<sup>3</sup><https://en.wikipedia.org/wiki/SVG>

and white pixels, is not scalable and can show traces of the pixel grid. It must be converted to an SVG format, which becomes scale independent, and can be zoomed in or deformed without creating blur or aliasing. The conversion from a pixel image to the SVG format is called *vectorization*. The geometric features captured by vectorization are also important in feature identification [19], remote sensing [12], and other applications [29, 28, 30].

Common silhouette vectorization methods [5, 24, 7, 16, 20, 29, 21, 27] consist of two steps: identification of control points on the shape’s boundary and approximation of the boundary curves connecting these control points. Ramer [24] proposed an iterative splitting scheme for identifying a set of control points on a polygonal line  $\mathcal{C}$  such that the Bézier polygon  $\hat{\mathcal{C}}$  defined by these vertices approximates  $\mathcal{C}$  in  $L^\infty$  norm. The Hausdorff distance between  $\hat{\mathcal{C}}$  and  $\mathcal{C}$  is constrained to stay below a predefined threshold, and the number of control points is suboptimal. More recently, Sarfraz [27] proposed an outline vectorization algorithm that splits the outline at corners which are identified without computing curvatures [6], then new control points are introduced to improve curve fitting.

We present here a detailed implementation of a novel vectorization approach fundamentally based on mathematical advances for their stability and sub-pixel accuracy. The method’s theory was published in [11]. It has three main steps that work together to find geometrically meaningful control points: it first identifies (i) curvature extrema of the outline computed at the sub-pixel level, by (ii) backpropagating control points detected as curvature extrema at coarser scale in the affine scale-space, then (iii) computing piecewise least-square cubic Bézier polygons joining these control points while fitting the smoothed outline with a predefined accuracy.

We organize the paper as follows. Section 2 presents the proposed algorithm, which consists of three steps. We explain the first step in Subsection 2.1, the level line extraction and sub-pixel curvature computation. In Subsection 2.2, the second step, the affine scale-space induced by the smooth bilinear outline and define the candidate control points. In Subsection 2.3, we describe the final step of an adaptive piecewise least-square Bézier polygon fitting, where the set of candidate points is modified to achieve a compact representation and to guarantee a predefined accuracy.

## 2 Silhouette Vectorization by Affine Scale-space

On a rectangular domain  $\Omega = [0, H] \times [0, W] \subset \mathbb{R}^2$  with  $H > 0$  and  $W > 0$ , a *silhouette* is a compact subset  $\mathcal{S} \subset \Omega$  whose topological boundary  $\partial\mathcal{S}$ , the *outline*, is a piecewise smooth curve. Suppose  $\mathcal{S}$  is shown in a *raster binary image*  $I : \Omega \cap \mathbb{N}^2 \rightarrow \{0, 255\}$ , that is, the set of black pixels

$$\overline{\mathcal{S}} = \{(i, j) \in \Omega \cap \mathbb{N}^2 \mid I(i, j) = 0\}$$

approximates  $\mathcal{S}$ . We assume  $\mathcal{S} \cap \partial\Omega = \emptyset$ , so that the level lines only consist of closed curves. The main objective of this paper is to find a cubic Bézier polygon close to  $\partial\mathcal{S}$  in the Hausdorff distance such that the vertices are geometrically meaningful. As a result, the proposed algorithm takes any binary raster image and converts it to an SVG file with compact size. In the following description, our algorithm will be applied independently to each connected component of the boundary of  $\mathcal{S}$ . Hence, without loss of generality, we assume that  $\partial\mathcal{S}$  is a single closed curve homeomorphic to a circle.

Figure 1 shows the overview of the proposed method. From the input image (a), which is a pixelized raster image, [Step 1] bilinear outlines are computed in (c), and [Step 2] affine scale-space is used to find the control points in (f). [Step 3] By cubic Bézier polygon, the vectorized result is presented in (g). Images (b) and (h) show the zoom-in of one of the corners, which illustrates a sharp representation of the given raster image in the vectorized form. Every step is designed to fully explore mathematically and geometrically meaningful features of the silhouette, utilizing the

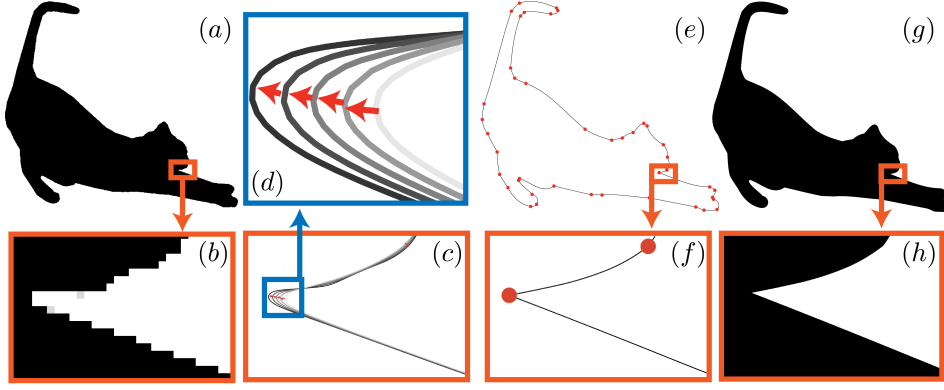


Figure 1: A flowchart of the proposed method. (a) A given raster image of a *cat*'s silhouette. (b) Zoom-in of (a). (c) Extracted bilinear outline of (a). (d) Inversely tracing the curvature extrema along the affine shortening flow. (e) The vectorized outline of (a) with control points marked as red dots. (f) Zoom-in of (e). (g) Vectorized result of silhouette (a) by the proposed method. (h) Zoom-in of (g). Notice the improvement from the given raster image (a) to the proposed method's result in (g), as well as the zoom of (b) and (h).

techniques which promote affine invariance. This approximation guarantees subpixel accuracy in reconstruction. In the following, we describe the three main steps.

## 2.1 Sub-pixel Curvature Extrema Localization

For [Step 1], we extract level lines considering the sub-pixel curvature via the bilinear interpolation [8]  $u : \Omega \rightarrow [0, 255]$  such that

$$u(i + 1/2, j + 1/2) = I(i, j), \quad (i, j) \in \Omega \cap \mathbb{N}^2.$$

For any  $\lambda \in (0, 255)$ , the level line of  $u$  corresponding to  $\lambda$  is defined as  $\mathcal{C}_\lambda = \{(x, y) \in \Omega \mid u(x, y) = \lambda\}$ . It approximates the discrete outline as a piecewise  $C^2$  Jordan curve, except at finitely many points such as image saddle points [4]. Fixing any non-integer  $\lambda^* \in (0, 255) \setminus \mathbb{N}$ ,  $\mathcal{C}_{\lambda^*}$  is either piecewise linear (horizontal or vertical) or a part of a hyperbola whose asymptotes are horizontal and vertical [8].

Due to pixelization,  $\mathcal{C}_{\lambda^*}$  shows strong staircase effects [3], which causes unstable curvature computation. Such oscillatory behavior is effectively reduced by the affine shortening flow [3, 26] by evolving the noisy curve  $\mathcal{C}$  by the following time-dependent PDE

$$\frac{\partial \mathcal{C}(s, t)}{\partial t} = \kappa^{1/3}(s, t) \mathbf{N}(s, t), \quad \mathcal{C}(s, 0) = \mathcal{C}(s), \quad t \geq 0 \quad (1)$$

till some short time  $T_0 \geq 0$ . Here each curve  $\mathcal{C}(\cdot, t)$  is arc-length parameterized by  $s \in [0, \text{Length}(\mathcal{C}(\cdot, t))]$  for any  $t$ ,  $\kappa$  denotes the signed scalar curvature, and  $\mathbf{N}$  is the inward unit normal at  $\mathcal{C}(s, t)$ . This process is independent from the viewpoint on the shape [14, 17]. For the curve evolution we use the level line extraction scheme which delivers a polygonal curve with fine subpixel sampling and Moisan's affine curve evolution scheme which processes this polygonal line [15], as implemented in the IPOL article [9].

Denoting the smooth bilinear outline by  $\Gamma_{\lambda^*}$ , at any vertex  $P \in \Gamma_{\lambda^*}$ , its unit normal direction  $N(P)$  is computed by central difference, and its curvature  $\kappa(P)$  is approximated by the curvature of the circumcircle that passes through three consecutive vertices on  $\Gamma_{\lambda^*}$  [9]. The discrete curvature values can be obtained at arbitrary resolution based on the sampling frequency applied to the bilinear outline  $\mathcal{C}_{\lambda^*}$ , and the curvature extrema are localized via Algorithm 1. To stabilize the process of identifying extrema, we smooth the curvature values via a simple moving average approach, whose kernel is designed heuristically to achieve good experimental results.

---

**Algorithm 1** Curvature extrema localization
 

---

**Require:** Sequence of curvature values  $\{\kappa(P_i^k)\}_{i=0}^{N^k}$  of the curve  $\Sigma_{\lambda^*}^k$  at scale  $k\Delta\sigma$ .

- 1: Process  $\{\kappa(P_i^k)\}_{i=0}^{N^k}$  by repeatedly applying the filter (1/18, 4/18, 8/18, 4/18, 1/18) with periodic boundary condition for 20 times.
- 2: Based on the filtered data  $\{\tilde{\kappa}(P_i^k)\}_{i=0}^{N^k}$ ,  $P_i^k$  is a curvature extremum if

$$|\tilde{\kappa}(P_i^k)| > |\tilde{\kappa}(P_j^k)|, \text{ for } j = i \pm 1, i \pm 2.$$

and

$$|\tilde{\kappa}(P_i^k)| > 0.001.$$

- 3: **return** Sequence of curvature extrema  $\{X_i^k\}_{i=1}^{S^k}$ .
- 

## 2.2 Affine Scale-space Control Points Identification

In [Step 2], we filter the control points by incorporating varying geometric scale of the outline based on the affine scale-space.

The set of solutions of (1) at different time  $t \geq 0$ , i.e.,  $\{\mathcal{C}(\cdot, t)\}_{t \geq 0}$  defines an affine scale-space [26], and the non-negative parameter  $t$  is called *scale*. This parametric space satisfies the causality [26] that every curvature extremum on the curve at a coarser scale, i.e., at larger  $t$ , is the continuation of at least one of the extrema at a finer scale, i.e., at smaller  $t$ . The lack of one-to-one correspondence is due to the possibility of multiple extrema (e.g., two maxima and one minimum) merging to a single one during the evolution. By tracing curvature extrema from the coarser scales to the finer scales, the resulting extrema are robust to noise and help capture prominent corners.

We define the control points as the curvature extrema on  $\Gamma_{\lambda^*}$  which persist across different scales in its affine scale-space. Given a sequence of discrete scales  $t_0 = 0 < t_1 < \dots < t_K$  for some positive integer  $K$ , we obtain the curve  $\mathcal{C}(\cdot, t_n)$  at scale  $t_n$  by the affine shortening flow (1) for  $n = 0, 1, \dots, K$ . For any  $1 \leq n \leq K$ , the affine shortening flow (1) is approximated as

$$\frac{\mathcal{C}(s, t_n) - \mathcal{C}(s, t_{n-1})}{t_n - t_{n-1}} = (\kappa^n(s))^{1/3} \mathbf{N}^n(s) + \mathbf{r}(s), \quad (2)$$

where  $\kappa^n$  and  $\mathbf{N}^n$  denote the curvature and normal at the scale  $t_n$ , and  $\mathbf{r}$  is a remainder such that  $\|\mathbf{r}(s)\| = O(t_n - t_{n-1})$ . Rearranging (2) gives

$$\mathcal{C}(s, t_{n-1}) = \mathcal{C}(s, t_n) - (t_n - t_{n-1})(\kappa^n(s))^{1/3} \mathbf{N}^n(s) - (t_n - t_{n-1})\mathbf{r}(s).$$

This expression shows that, if  $t_n - t_{n-1}$  is sufficiently small, by following the opposite direction of the affine shortening flow at  $\mathcal{C}(s, t_n)$ , that is,  $-\text{sign}(\kappa^n(s))\mathbf{N}^n(s)$ , we can find  $\mathcal{C}(s, t_{n-1})$  nearby. Here  $\text{sign}(r)$  denotes the sign function which gives +1 if  $r > 0$ , -1 if  $r < 0$  and 0 if  $r = 0$ .

Starting from  $K$ , for any curvature extremum  $X^K$  on  $\mathcal{C}_K = \mathcal{C}(\cdot, t_K)$ , we set up the following constrained optimization problem to find a curvature extremum  $X^{K-1}$  on  $\mathcal{C}_{K-1}$  at scale  $t_{K-1}$

$$\begin{aligned} & \max_{X \in \mathcal{C}_{K-1}} \frac{\langle X - X^K, -\text{sign}(\kappa^K)\mathbf{N}^K \rangle}{\|X - X^K\|} \\ \text{s.t. } & \frac{\langle X - X^K, -\text{sign}(\kappa^n)\mathbf{N}^n \rangle}{\|X - X^K\|} > \alpha, \|X - X^K\| < D, \text{ and } X \text{ is a curvature extremum on } \mathcal{C}_{K-1}, \end{aligned} \quad (3)$$

where  $\|\cdot\|$  denotes the Euclidean norm,  $D > 0$  is a positive parameter that controls the closeness between  $X$  and  $X^K$ , and  $\alpha$  enforces that the direction of  $X - X^K$  is similar to that of the inverse affine

shortening flow. The problem (3) looks for the curvature extremum on  $\mathcal{C}_{K-1}$  in the  $D$ -neighborhood of  $X^K$  that is the nearest to the line passing  $X^K$  in the direction of the inverse affine shortening flow. When  $D$  and  $\alpha$  are properly chosen, if (3) has one solution, we define it to be  $X^{K-1}$ . If (3) has multiple solutions, we choose the one with the shortest distance from  $X^K$  to be  $X^{K-1}$ . In case multiple solutions are having the same shortest distance from  $X^K$ , we arbitrarily select one to be  $X^{K-1}$ . In practice, if (3) has a solution, it is almost always unique.

We repeat (3) for decreasing  $K - 1, K - 2, \dots, 1$ . Either the solutions always exist until the scale  $t_0$ , or there exists some  $m \geq 1$ , such that (3) at  $t_m$  does not have any solution. In the first case, we call  $X^K$  a *complete* point, and in the second case, we call it *incomplete*. For each curvature extremum  $X^K$  on  $\mathcal{C}_K$ , we construct a sequence of points  $\mathcal{L}$  that contains the solutions of (3) for  $K, K - 1, K - 2, \dots$ , starting at  $X^K$  in a scale-decreasing order. If  $X^K$  is complete, then  $\mathcal{L}$  has exactly  $K + 1$  elements, and we call the sequence complete; otherwise, the size of  $\mathcal{L}$  is strictly smaller than  $K + 1$ , and we call the sequence incomplete.

If there is at least one complete sequence, we define the last elements of the complete sequences as the candidate control points, and denote them as  $\{O_i(t_K)\}_{i=1}^{M(t_K)}$ , where  $M(t_K) \geq 1$  is the number of complete sequences. These points are ordered following the orientation of  $\Gamma_{\lambda^*}$ . The parameter  $t_K$  in the parenthesis indicates that the candidate control points are associated with the curvature extrema identified at the scale  $t_K$ . When the scale  $t_K$  is fixed, we simply write  $\{O_i\}_{i=1}^M$ .

This inverse affine scale-space approach prioritizes the curvature extrema, which persist across different affine shortening flow scales. This step is essential in keeping geometrically meaningful control points and reducing the total number of control points. However, if all the sequences are incomplete, i.e., there is no curvature extrema identified by tracing backward the scale space, we encounter the degenerate case, which calls for a different procedure.

**Degenerate Case:** When there is no complete sequence, either  $\mathcal{S}$  is a disk, or  $\partial\mathcal{S}$  has curvature extrema with small curvature, which failed to be numerically identified. We propose to address these two scenarios separately, since if  $\mathcal{S}$  is a disk, the vectorization only requires its center and radius, which simplifies the following curve approximation. We use the isoperimetric inequality to determine if  $\Gamma_{\lambda^*}$  represents a circle: for any closed plane curve with area  $A$  and perimeter  $L$ , we have  $4\pi A \leq L^2$  and the equality holds if and only if the curve is a circle. In practice, we decide that  $\Gamma_{\lambda^*}$  is a circle only if the corresponding ratio  $1 - 4\pi A/L^2$  is sufficiently small. By this criterion, if  $\Gamma_{\lambda^*}$  is classified as a circle, its center and radius are easily computed by arbitrarily three distinct points on  $\Gamma_{\lambda^*}$ . For numerical stability, we take three outline points that are equidistant from each other. Otherwise, we insert a pair of most distant points on  $\Gamma_{\lambda^*}$  to be the candidate control points. An efficient approach for finding these points is to combine a convex hull algorithm, e.g., the monotone chain method [1], which takes  $O(N \log N)$  time, with the rotating calipers [23], which takes  $O(N)$  time. Here  $N$  is the number of vertices of the polygonal line  $\Gamma_{\lambda^*}$ . The subroutine for dealing with degenerate cases is described in Algorithm 2. Note that in a degenerate case,  $\mathcal{S}$  is close to a convex shape, thus the area computation is simplified to summing areas of a sequence of triangles. Moreover, the cross product of 2D vectors is computed as they are embedded in  $\mathbb{R}^3$  with the third component set to be 0.

## 2.3 Adaptive Cubic Bézier Polygon Approximation

After the control points are identified from the affine scale-space,  $H := \{O_i\}_i^M$ , we adjust  $H$  by deleting non-salient sub-pixel curvature extrema and inserting new control points for guaranteeing a predefined accuracy. In [Step 3], this adaptive approach yields a cubic Bézier polygon  $\mathcal{B}(H)$  whose vertices are points in  $H$  and edges are cubic Bézier curves computed by least-square fittings.

A cubic Bézier curve is specified by four points  $B_0, B_1, B_2$ , and  $B_3$ . Its parametric form is

$$B(s) = (1 - s)^3 B_0 + 3(1 - s)^2 s B_1 + 3(1 - s) s^2 B_2 + s^3 B_3,$$

---

**Algorithm 2 Subroutine for the degenerate case**


---

**Require:** closed polygonal curve  $\Sigma = \{P_0, \dots, P_N\}$  with  $P_N = P_0$  and there is no complete sequence after tracing back the affine-scale space (3).

1: Compute the area  $A$  and perimeter  $L$  of  $\Sigma$  as follows:

$$A = \frac{1}{2} \sum_{i=1}^{N-2} \|(P_i - P_0) \times (P_{i+1} - P_0)\|, \quad L = \sum_{i=0}^{N-2} \|P_{i+1} - P_i\|.$$

2: **if**  $1 - 4\pi A/L^2 < 0.005$  **then**

3:   Mark  $\Sigma$  as a circle.

4:   Take three equidistant points on  $\Sigma_{\lambda^*}$  to compute the center  $O$  and radius  $r$ .

5:   **return**  $O, r$ .

6: **else**

7:   // $\Sigma$  is not a circle, yet no corner is identified by tracing back the affine-scale space.

8:   Find the most distant pair of points on  $\Sigma$ :  $O_1, O_2$ . Set  $H = \{O_1, O_2\}$ .

9:   **return**  $H$

---

for  $s \in [0, 1]$ . Here, (i)  $B_0$  and  $B_3$  are the two endpoints for  $B(s)$ ; and (ii)  $B_1 - B_0$  is the right tangent of  $B(s)$  at  $B_0$ , and  $B_2 - B_3$  is the left tangent at  $B_3$ . To approximate a polygonal line segment  $\Sigma = \{P_0, P_1, \dots, P_N\}$ , we find a cubic Bézier curve that is determined by  $B_0 = P_0$ ,  $B_1$ ,  $B_2$ , and  $B_3 = P_N$  such that the squared fitting error

$$\tilde{S} = \sum_{i=1}^N \|P_i - ((1 - \tilde{s}_i)^3 B_0 + 3(1 - \tilde{s}_i)^2 \tilde{s}_i B_1 + 3(1 - \tilde{s}_i) \tilde{s}_i^2 B_2 + \tilde{s}_i^3 B_3)\|^2 \quad (4)$$

is minimized. Here  $\tilde{s}_i = (\sum_{k=1}^i \|P_k - P_{k-1}\|) / (\sum_{k=1}^N \|P_k - P_{k-1}\|)$  is the chord-length parameter for  $P_i$  with  $i = 1, \dots, N$ . We note that (4) is used to initialize an iterative algorithm in [22] for a more accurate Bézier fitting. The benefit of this approximating setup is that we have closed-form formulae [16] for the minimizing  $B_j$ ,  $j = 1, 2$ . See Algorithm 3.

### 2.3.1 Control Point Refinement: Deletion of Sub-pixel Extrema

The candidate control points  $H = \{O_i\}_{i=1}^M$  are curvature extrema at sub-pixel level, and they may not reflect salient corners of the silhouette. To remove spurious sub-pixel extrema from  $H$ , we compare the left tangent and right tangent at each candidate control point. In particular, we take advantage of the second property of cubic Bézier curves above. For  $i = 1, \dots, M$ , we fit a cubic Bézier to the polygonal line segment whose set of vertices is

$$\{O_i = P_{j(i)}, P_{j(i)+1}, \dots, P_{j(i+1)} = O_{i+1}\},$$

where  $j(i)$  denotes the index of the vertex of the polygonal line segment that corresponds to the  $i$ -th control point in  $H$ . We take  $O_{M+1} = O_1$ , and obtain the estimated defining points  $B_{i,1}$  and  $B_{i,2}$  for the Bézier curve. The left and right tangent at  $O_i$  are computed as

$$T_i^- = B_{i-1,2} - O_i, \quad T_i^+ = B_{i,1} - O_i, \quad (5)$$

respectively, where  $B_{-1,2} = B_{M,2}$ . These tangent vectors are associated with all the points between neighboring candidate control points. The angle formed by  $T_i^-$  and  $T_i^+$  measures the sharpness of



**Algorithm 3 Subroutine for Bézier fitting****Require:** A polygonal curve segment  $\{P_0, P_1, \dots, P_N\}$ 

- 1: Set  $B_0 = P_0$  and  $B_3 = P_N$ .
- 2: **if**  $N = 1$  **then**
- 3:    $B_1 = B_2 = 0.5(P_0 + P_1)$
- 4: **else if**  $N = 2$  **then**
- 5:    $B_1 = B_2 = P_1$
- 6: **else if**  $N > 2$  **then**
- 7:   Set

$$B_1 = (a_2 C_1 - a_{12} C_2) / (a_1 a_2 - a_{12}^2), \quad B_2 = (a_1 C_2 - a_{12} C_1) / (a_1 a_2 - a_{12}^2)$$

where

$$a_1 = 9 \sum_{i=1}^N \tilde{s}_i^2 (1 - \tilde{s}_i)^4, \quad a_2 = 9 \sum_{i=1}^N \tilde{s}_i^4 (1 - \tilde{s}_i)^2, \quad a_{12} = 9 \sum_{i=1}^N \tilde{s}_i^3 (1 - \tilde{s}_i)^3$$

$$C_1 = \sum_{i=1}^N 3\tilde{s}_i (1 - \tilde{s}_i)^2 [P_i - (1 - \tilde{s}_i)^3 P_0 - \tilde{s}_i^3 P_3], \quad C_2 = \sum_{i=1}^N 3\tilde{s}_i^2 (1 - \tilde{s}_i) [P_i - (1 - \tilde{s}_i)^3 P_0 - \tilde{s}_i^3 P_3]$$

and  $\tilde{s}_i = (\sum_{k=1}^i \|P_k - P_{k-1}\|) / (\sum_{k=1}^N \|P_k - P_{k-1}\|)$ .

- 8: Compute the error  $e$  by (6).
- 9: **return** The Bézier curve defined by  $B_0, B_1, B_2, B_3$ , and error  $e$ .

$\Gamma_{\lambda^*}$  at  $O_i$  from a more global perspective. We delete  $O_i$  from the set of candidate control points  $H$  if the angle between  $T_i^+$  and  $T_i^-$  is close to  $\pi$ . The set  $H$  is updated with the remaining control points.

When all the candidate control points  $\{O_i\}_{i=1}^M$  are removed after this procedure, we encounter a degenerate case. If the underlying outline is a circle, we compute the center and radius; if it is not, we take the most distant pair of outline points to update  $H$ .

**2.3.2 Control Point Refinement: Insertion for Accuracy**

The candidate control points in  $H$  split the outline  $\Gamma_{\lambda^*}$  into polygonal line segments, each of which is approximated by a cubic Bézier using least square fitting. We obtain a Bézier polygon that approximates  $\Gamma_{\lambda^*}$ , denoted by  $\mathcal{B}(H)$ . A natural measure for the error of approximating  $\Gamma_{\lambda^*}$  using the Bézier polygon  $\mathcal{B}(H)$  is

$$e = \max_{P_i \in \Gamma_{\lambda^*}} \text{dist}(P_i, \mathcal{B}(H)), \quad (6)$$

where  $\text{dist}(P_i, \mathcal{B}(H)) = \inf_{P \in \mathcal{B}(H)} \|P_i - P\|$  is the distance from  $P_i$  to the curve  $\mathcal{B}(H)$ . It is desirable that the user can specify the threshold for the error,  $\tau_e > 0$ . To guarantee that  $e \leq \tau_e$ , we apply the splitting strategy [24] which inserts  $P_{\text{new}} \in \Gamma_{\lambda^*}$  to  $H$  as a new control point if

$$\text{dist}(P_{\text{new}}, \mathcal{B}(H)) > \tau_e, \quad (7)$$

and among those points on  $\Gamma_{\lambda^*}$  satisfying (7), the distance from  $P_{\text{new}}$  to  $\mathcal{B}(H)$  is the largest. After the insertion, we fit  $\Gamma_{\lambda^*}$  using a Bézier polygon based on the new set of control points in  $H$ . If the error of the newly fitted Bézier polygon is still greater than  $\tau_e$ , we insert another point based on the same criterion. This series of insertions terminates once the condition  $e \leq \tau_e$  is met.

Finally,  $\mathcal{B}(H)$  with the updated set of control points  $H$  gives a Bézier polygon that approximates the outline  $\partial\mathcal{S}$ . With its interior filled with black, we obtain the vectorized silhouette for  $\mathcal{S}$  from

the raster image  $I$ . Note that the interior and exterior of the boundary curve (Jordan curve) is determined by the ray-algorithm (evenodd fill-rule) embedded in the SVG format.

## 2.4 Summary of the Algorithm

Algorithm 4 presents the pseudo-code of the proposed method. It requires any gray-scale raster image as the input and efficiently converts the pixel silhouette to a scalable vector graphic. There are mainly two parameters adjustable by the user: the error threshold  $\tau_e$  and the smoothness parameter  $\sigma_0$ . For a more accurate representation of the original silhouette, smaller values of  $\tau_e$  should be chosen; and for smoother boundary curves of the silhouette, larger  $\sigma_0$  should be considered. The effects of these parameters will be explained in the numerical section.

## 3 Numerical Experiments

After obtaining the SVGs from SVG SILH<sup>4</sup>, we rasterized them as PNG images, which were used as inputs in the following experiments. The inputs were either binary or gray-scale. We extracted the level line for  $\lambda^* = 127.5$  to approximate the outlines throughout the experiments.

To solve (1), we apply the fully consistent geometric scheme [15] which is independent of grid discretization. Consequently, the scale parameter  $t$  is conveniently replaced by a chord-area parameter  $\sigma$ . The scale  $T_0$  for the initial smoothing (Section 2.1) required for curvature computation thus corresponds to some *smoothness parameter*  $\sigma_0$ . The computed discrete curvatures are filtered by moving average with periodic boundary condition to reduce the noise. A curvature extremum is identified only if it has absolute value greater than its neighbors and above 0.001. For the parameters in (3), we fixed  $D = 10$  and  $\alpha = 0.9$ . During the inverse tracing  $K = 4$ , and since the sequence of scales  $\{t_k\}_{k=1}^K$  can be replaced by chord-area parameters, the curvature extrema were traced for scales corresponding to chord-areas  $k\Delta\sigma$ ,  $k = 1, 2, 3, 4$  respectively, where  $\Delta\sigma = 0.5$ . The threshold for the degenerate case (Section 2.2) is set to be 0.005.

By default, we set the error threshold  $\tau_e = 1$ , so that the vectorized outline was guaranteed to have sub-pixel accuracy; and the smoothness parameter  $\sigma_0 = 1$ . Table 1 collectively displays the silhouettes used in the following experiments. They are all downloadable from <https://svgsilh.com>, which are released under Creative Commons CC0.

### 3.1 General Performance

We present some results of our proposed algorithm in Figure 2. In (a), we have a silhouette of a *cat*. It has a single outline curve that contains multiple sharp corners on the tail, near the neck, and around the paws. These features provide informative visual cues for silhouette recognition, and our algorithm identifies them as control points for the silhouette vectorization shown as the red dots in (b). The outline of a *butterfly* in (c) has multiple connected components. In addition to the control points corresponding to corners, we observe in (d) some others on smooth segments of the outline. They are inserted during the refinement step of our algorithm, where a single Bézier cubic is inadequate to guarantee the accuracy specified by the error threshold  $\tau_e = 1$ . In (e), we show a tessellation of words, and (f) presents the vectorized result. The input is a PNG image of dimension  $1934 \times 1332$  and takes 346 KB in the storage. In contrast, its silhouette vectorization, saved as an SVG file, has 2683 control points and takes 68 KB if the coordinates are stored in float. In this example, our algorithm provides a compression ratio of about 80.35%. The total computational time for this case only takes 0.83 seconds.

<sup>4</sup><https://svgsilh.com>. All contents are released under Creative Commons CC0.



**Algorithm 4 Shape Vectorization by Affine Scale-space**


---

**Require:**    •  $I$ : gray-scale raster image with intensity variation concentrating around the contour.

    •  $\tau_e$ : error threshold.

    •  $\sigma_0$ : smoothness parameter.

    • Fixed parameters  $\lambda^* = 127.5$ ,  $\Delta\sigma = 0.5$ ,  $K = 4$ ,  $D = \sqrt{10}$ ,  $\alpha = 0.9$ ,  $\varepsilon = 0.1$ .

**Ensure:** Scalable vector graphic.

---

**I. Curvature extrema in varying scales**

1: Extract the bilinear level line  $\mathcal{C}_{\lambda^*}$ .

2: By Moisan's scheme [15], smooth  $\mathcal{C}_{\lambda^*}$  (a Jordan curve; see the explanation before the algorithm) via (1) up to scale  $\sigma_0$  yielding a sub-pixel smooth contour  $\Sigma_{\lambda^*} = \{P_i\}_{i=0}^N$ .

3: **for**  $k = 1, 2, \dots, K$  **do**

4:    Evolve  $\Sigma_{\lambda^*}$  up to scale  $k\Delta\sigma$ , denoted by  $\Sigma_{\lambda^*}^k = \{P_i^k\}_{i=0}^{N^k}$ .

5:    **for**  $i = 0, \dots, N^k$  **do**

6:     Compute curvature  $\kappa_i^k$  of  $\Sigma_{\lambda^*}^k$  at  $P_i^k$  by

$$\kappa_i^k = \frac{-2 \det \begin{bmatrix} x_{i-1}^k - x_i^k & x_{i+1}^k - x_i^k \\ y_{i-1}^k - y_i^k & y_{i+1}^k - y_i^k \end{bmatrix}}{\|P_{i-1}^k P_i^k\| \|P_i^k P_{i+1}^k\| \|P_{i-1}^k P_{i+1}^k\|}.$$

    where  $(x_i^k, y_i^k)$  is the coordinate for  $P_i^k$ .

7:    Locate curvature extrema  $\{X_i^k\}_{i=1}^{S^k} \subseteq \Sigma_{\lambda^*}^k$ . (See Algorithm 1)

**II. Inverse affine shortening flow**

8: **if**  $S^K \geq 1$  **then**

9:    **for**  $i = 1, \dots, S^K$  **do**

10:     Initialize a sequence  $\mathcal{L}_i = \{X_i^K\}$ , and set  $X_i^{(K)} = X_i^K$ .

11:     **for**  $k = K, K-1, \dots, 1$  **do**

12:       Solve the problem (3) associated with  $X_i^{(k)}$

13:       **if** (3) has a solution  $X_i^{(k-1)}$  **then**

14:         Append  $X_i^{(k-1)}$  into  $\mathcal{L}_i$ .

15:       **else**

16:         **break**

17:   Collect the final elements of complete sequences as  $H$ , whose elements are denoted by  $O_i$ ,  $i = 1, \dots, \#H$ , where  $\#H$  is the total number of control points.

18: **else**

19:   Run Algorithm 2 for  $\Sigma_{\lambda^*}$ .

**III. Boundary fitting**

20: **if**  $\#H > 0$  **then**

21:   **for**  $i = 1, \dots, \#H$  **do**

22:     Given periodic boundary condition, run Algorithm 3 for the curve segment

$$[O_i, O_{i+1}] := \{O_i = P_{j(i)}, P_{j(i)+1}, \dots, P_{j(i+1)} = O_{i+1}\} \subset \Sigma_{\lambda^*}$$

23:   **for**  $i = 1, \dots, \#H$  **do**

24:     Obtain the right tangent  $T_i^+$  at  $O_i$  and left tangent  $T_{i+1}^-$  at  $O_{i+1}$  according to (5).

25:     **if**  $|\frac{\langle T_i^+, T_{i+1}^- \rangle}{\|T_i^+\| \|T_{i+1}^-\|} + 1| < \varepsilon$  **then**

26:       Remove  $O_i$  from  $H$ .

27:   **if**  $\#H = 0$  **then**

28:     Run Algorithm 2 for  $\Sigma_{\lambda^*}$ .

29: **else**

30:   **for**  $i = 1, \dots, \#H$  **do**

31:     Set  $O_L = O_i$  and  $O_R = O_{i+1}$ .

32:     **while true do**

33:       Run Algorithm 3 for  $[O_L, O_R]$  and obtain the approximation error  $e$ .

34:       **if**  $e > \tau_e$  **then**

35:         Find the point  $O^* \in [O_L, O_R]$  furthest from the fitting Bézier curve that satisfies (7).

36:         Update  $O_R = O^*$ .

37:       **else**

38:         **if**  $O_R \neq O_{i+1}$  **then**

39:         Insert  $O_R$  into  $H$ .

40:         Update  $O_L = O_R$  and  $O_R = O_{i+1}$ .

41:       **else**

42:         **break**

**IV. SVG formatting**

43: **if**  $\Sigma_{\lambda^*}$  is a circle **then**

44:   Provide the center and radius obtained from Algorithm 2.

45: **else**

46:   Provide the sequence of control points for the Bézier segments.

47: Select evenodd fill-rule to fill the shape.

---

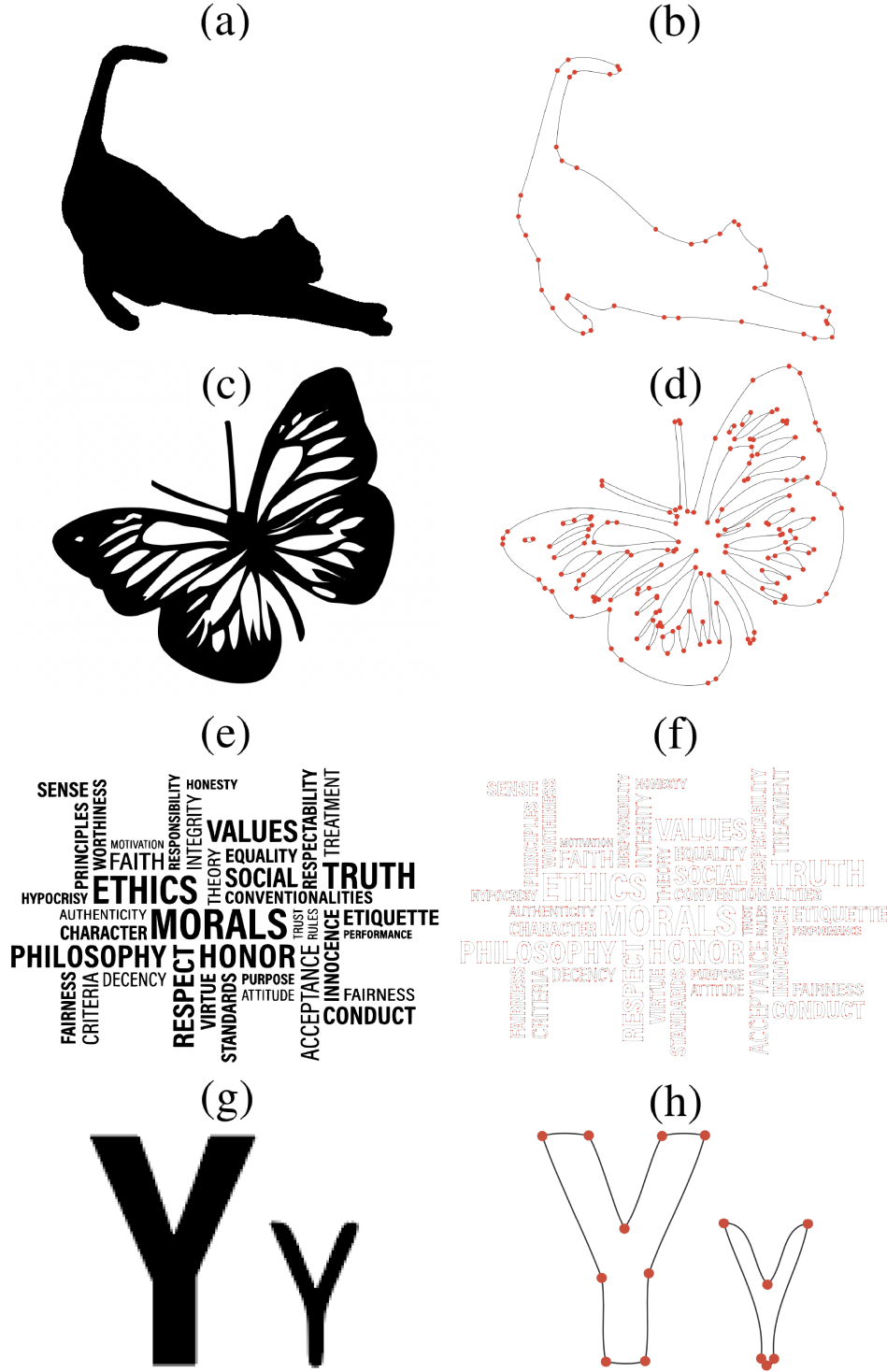


Figure 2: General performance. (a) *Cat* and (b) its vectorized outline (42 control points). (c) *Butterfly* and (d) its vectorized outline (158 control points). (e) Text design and its vectorized outline (2683 control points). Each red dot signifies the location of a control point. (g) Two letters exerted from (e) scaled up with the same magnitude. (h) Zoom-in of the vectorization (f) on the two letters in (g).

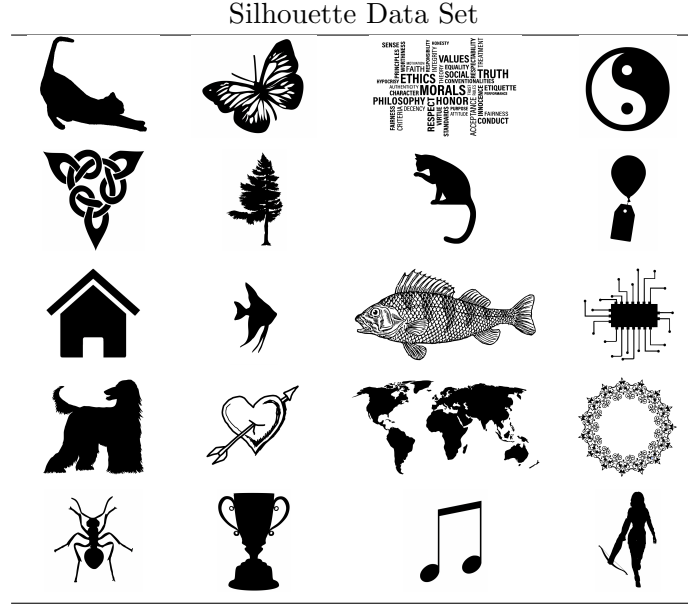


Table 1: Silhouette dataset used in the experiments. These silhouettes are chosen from <https://svgsilh.com>, and are released under Creative Commons CC0.

### 3.2 Qualitative Comparison with Feature Point Detectors

In Figure 3, we compare the distribution of these points with the results of some extensively applied feature point detectors: the Harris-Stephens corner detector [10], the features from Accelerated Segment Test (FAST) detector [25], the Speeded Up Robust Features (SURF) detector [2], and the Scale-Invariant Feature Transform (SIFT) [13]. The Harris-Stephens corner detector is a local auto-correlation based method. It locally filters the image with spatial difference operators and identifies corners based on the response. The FAST detector only considers the local configurations of pixel intensities; hence it is widely applied in real-time applications. From (b), we see that FAST identifies all the prominent corners the same as our method. Similarly to (a), there are no FAST points identified around the *balloon*. The SURF detector combines a fast Hessian measure computed via integral images and the distribution of local Haar-wavelet responses to identify feature points that are scale and translation invariant. The SURF points are marked over scales; hence we see most of the green crosses in (c) form sequences converging toward the outline. SIFT detects scale-invariant features of a given image. As shown in (d), SIFT successfully indicates the presence of corners and marks the *balloon*’s centers as well as the label, which are visually robust features of the silhouette.

### 3.3 Comparison with State-of-the-art Software

There are many software tools available for image vectorization, e.g., Vector Magic<sup>5</sup>, Inkspace<sup>6</sup>, and Adobe Illustrator 2020 (AI)<sup>7</sup>. We compare our method with these software tools using the number of control points generated for given silhouettes as a criterion. This quantity is equal to the number of curve segments used for approximating the outline, and a smaller value indicates a more compact silhouette representation.

For comparison, after acquiring SVG files of various silhouettes, we rasterized them and used the PNG images as inputs. Table 2 summarizes the results. For Vector Magic, we tested three available settings: high, medium, and low for the vectorization quality. For AI, we chose the setting “Black

<sup>5</sup><https://vectormagic.com>

<sup>6</sup><https://inkscape.org>

<sup>7</sup><https://www.adobe.com/products/illustrator.html>

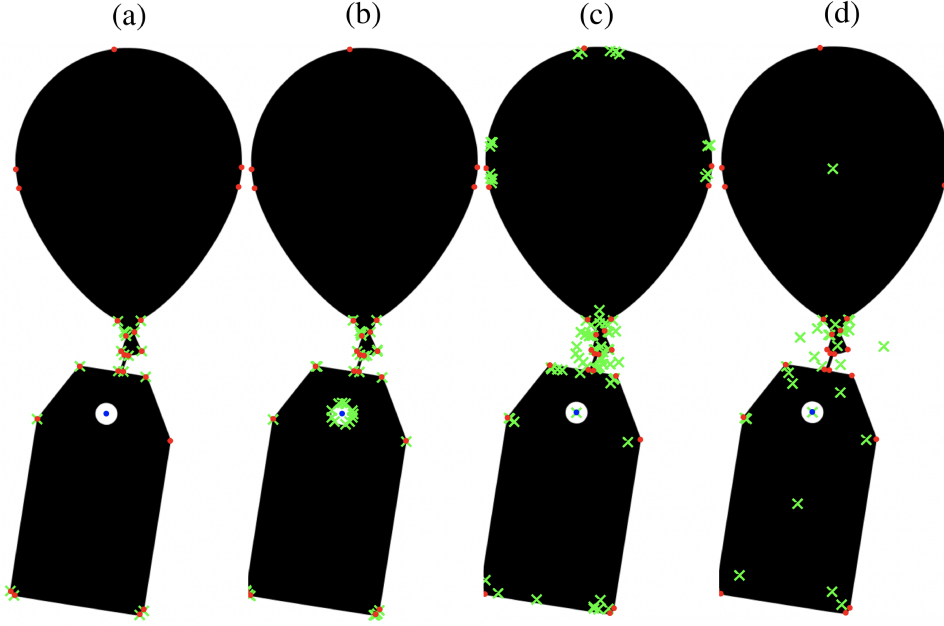


Figure 3: Comparison between the control points (red dots) plus the centers of circles (blue dots) produced by the proposed algorithm and other point feature detectors (green crosses). (a) Compared with the Harris corner detector [10]. (b) Compared with the FAST feature detector [25]. (c) Compared with the SURF detector [2]. (d) Compared with the SIFT detector [13]

and White Logo”, as it is suitable for the style of our inputs. We also include the results when the automatic simplification was used, which are marked by daggers. For InkSpace, we used the default parameter settings. As shown by the mean relative reduction values on the number of control points in the last row, our method produces the most compact vectorization results.

With such an effective reduction in the number of control points, it remains to verify that our method does not over-simplify the representation. We show a detailed comparison in Figure 4 between our proposed method and AI. In particular, we used AI without simplification and our method with two sets of parameters:  $\sigma_0 = 1$ ,  $\tau_e = 1$  and  $\sigma_0 = 0.1$ ,  $\tau_e = 0.5$ . We note that  $\sigma_0$  specifies the smoothness of the recovered outline, and  $\tau_e$  controls the accuracy. Notice that our method gives fewer control points under these settings, and our results preserve more details of the given silhouettes.

## 4 Conclusion

We described an efficient and effective algorithm for silhouette vectorization. The outline of the silhouette is interpolated bilinearly and uniformly sampled at a sub-pixel level. To reduce the oscillation due to pixelization, we apply the affine shortening to the bilinear outline. We then identify a set of candidate control points by tracing the curvature extrema across different scales along the well-defined inverse affine shortening flow. This set is then refined by deleting sub-pixel extrema that do not reflect salient corners, and inserting new points to guarantee any user-specified accuracy. We also designed special procedures to address the degenerate cases, such as disks, so that our algorithm adapts to arbitrary resolutions and offers better information compression. Our method provides a superior compression ratio by vectorizing the outlines. As shown in [11], this method is competitive compared to some well-established image vectorization software. It produces results with fewer control points for equally high accuracy.







Test Image	Number of Control Points ( $\#C$ )				
	Original	VM	IS	AI	Proposed
	405	248/256/245	330	280 (193 <sup>†</sup> )	168
	611	359/343/325	383	340 (293 <sup>†</sup> )	222
	682	296/294/263	272	211 (128 <sup>†</sup> )	120
	1434	915/828/715	932	698 (462 <sup>†</sup> )	379
	4434	2789/2582/2370	3292	2120 (1431 <sup>†</sup> )	1407
	6664	5470/5218/4955	6493	4870 (3441 <sup>†</sup> )	2810
MRR	—	37.97%/40.55%/45.01%	29.88%	45.79% (61.58% <sup>†</sup> )	67.38%

Table 2: Comparison with image vectorization software in terms of the number of control points. We compared with Vector Magic (VM), Inkspace (IS), and Adobe Illustrator 2020 (AI). For VM, we report the number of control points using three settings: High/Medium/Low. For AI, the values with dagger<sup>†</sup> indicate the numbers of control points produced by the automatic simplification. The input image dimensions are  $581 \times 564$ ,  $625 \times 598$ ,  $400 \times 390$ ,  $903 \times 499$ ,  $515 \times 529$ , and  $1356 \times 716$  from top to bottom. We also report the mean relative reduction (MRR) of the number of control points computed for the results above.

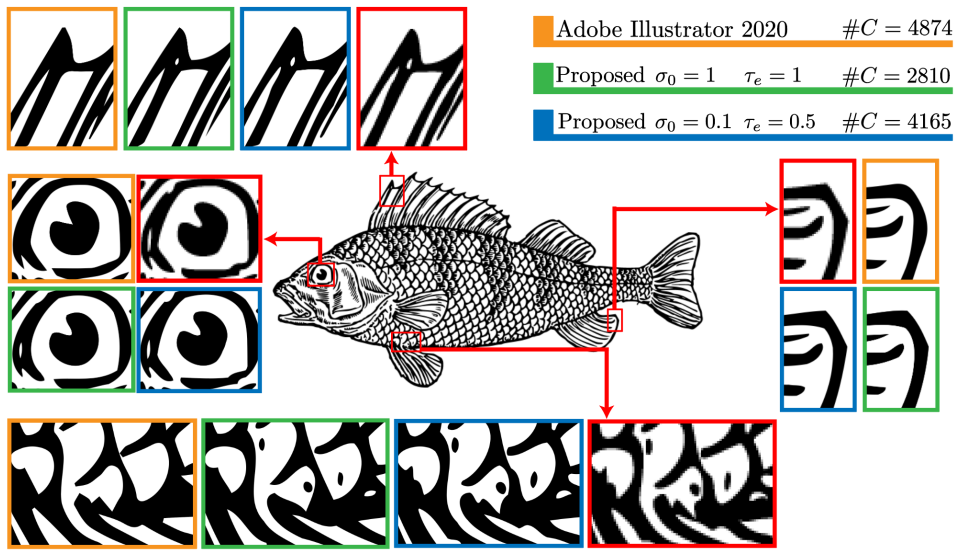


Figure 4: Comparison among the given raster image (red boxes), AI (orange boxes), the proposed with  $\sigma_0 = 1$ ,  $\tau_e = 1$  (green boxes), and the proposed with  $\sigma_0 = 0.1$ ,  $\tau_e = 0.5$  (blue boxes). With smaller numbers of control points ( $\#C$ ), our method preserves better the geometric details of the given silhouette.

## Image Credits

All silhouettes are from <https://svgsilh.com>, and are released under Creative Commons CC0.

## References

- [1] A. M. ANDREW, *Another efficient algorithm for convex hulls in two dimensions*, Information Processing Letters, 9 (1979), pp. 216–219. [https://doi.org/10.1016/0020-0190\(79\)90072-3](https://doi.org/10.1016/0020-0190(79)90072-3).
- [2] H. BAY, T. TUYTELAARS, AND L. VAN GOOL, *SURF: Speeded up robust features*, in European Conference on Computer Vision (ECCV), Springer, 2006, pp. 404–417. [https://doi.org/10.1007/11744023\\_32](https://doi.org/10.1007/11744023_32).
- [3] F. CAO, *Geometric curve evolution and image processing*, Springer Science & Business Media, 2003. ISBN 9783540004028.
- [4] V. CASELLES AND P. MONASSE, *Geometric description of images as topographic maps*, Springer, 2009. ISBN 9783642046117.
- [5] H.-H. CHANG AND H. YAN, *Vectorization of hand-drawn image using piecewise cubic Bézier curves fitting*, Pattern Recognition, 31 (1998), pp. 1747–1755. [https://doi.org/10.1016/S0031-3203\(98\)00045-4](https://doi.org/10.1016/S0031-3203(98)00045-4).
- [6] D. CHETVERIKOV, *A simple and efficient algorithm for detection of high curvature points in planar curves*, in International Conference on Computer Analysis of Images and Patterns, Springer, 2003, pp. 746–753. [https://doi.org/10.1007/978-3-540-45179-2\\_91](https://doi.org/10.1007/978-3-540-45179-2_91).
- [7] L. CINQUE, S. LEVIALDI, AND A. MALIZIA, *Shape description using cubic polynomial Bézier curves*, Pattern Recognition Letters, 19 (1998), pp. 821–828. [https://doi.org/10.1016/S0167-8655\(98\)00069-5](https://doi.org/10.1016/S0167-8655(98)00069-5).
- [8] A. CIOMAGA, P. MONASSE, AND J.-M. MOREL, *Level lines shortening yields an image curvature microscope*, in International Conference on Image Processing (ICIP), IEEE, 2010, pp. 4129–4132. <https://doi.org/10.1109/ICIP.2010.5649850>.
- [9] —, *The image curvature microscope: Accurate curvature computation at subpixel resolution*, Image Processing On Line, 7 (2017), pp. 197–217. <https://doi.org/10.5201/ipol.2017.212>.
- [10] C. G. HARRIS AND M. STEPHENS, *A combined corner and edge detector*, in Alvey Vision Conference, vol. 15, Citeseer, 1988, pp. 10–5244. <https://doi.org/10.5244/C.2.23>.
- [11] Y. HE, S.-H. KANG, AND J.-M. MOREL, *Silhouette vectorization by affine scale-space*, Journal of Mathematical Imaging and Vision, (2021), pp. 1–16. <https://doi.org/10.1007/s10851-021-01053-z>.
- [12] A. KIRSANOV, A. VAVILIN, AND K. H. JO, *Contour-based algorithm for vectorization of satellite images*, in International Forum on Strategic Technology, IEEE, 2010, pp. 241–245. <https://doi.org/10.1109/IFOST.2010.5668109>.
- [13] D. LOWE, *Object recognition from local scale-invariant features*, in IEEE International Conference on Computer Vision (ICCV), vol. 2, IEEE, 1999, pp. 1150–1157. <https://doi.org/10.1109/ICCV.1999.790410>.



- [14] J. MATAS, O. CHUM, M. URBAN, AND T. PAJDLA, *Robust wide-baseline stereo from maximally stable extremal regions*, Image and Vision Computing, 22 (2004), pp. 761–767. <https://doi.org/10.1016/j.imavis.2004.02.006>.
- [15] L. MOISAN, *Affine plane curve evolution: A fully consistent scheme*, IEEE Transactions on Image Processing, 7 (1998), pp. 411–420. <https://doi.org/10.1109/83.661191>.
- [16] A. S. MONTERO AND J. LANG, *Skeleton pruning by contour approximation and the integer medial axis transform*, Computers & Graphics, 36 (2012), pp. 477–487. <https://doi.org/10.1016/j.cag.2012.03.029>.
- [17] J.-M. MOREL AND G. YU, *ASIFT: A new framework for fully affine invariant image comparison*, SIAM Journal on Imaging Sciences, 2 (2009), pp. 438–469. <https://doi.org/10.1137/080732730>.
- [18] M. MORTENSON, *Mathematics for computer graphics applications*, Industrial Press Inc., 1999. ISBN 9780831131111.
- [19] C. NADAL, R. LEGAULT, AND C. Y. SUEN, *Complementary algorithms for the recognition of totally unconstrained handwritten numerals*, in International Conference on Pattern Recognition (ICPR), vol. 1, IEEE, 1990, pp. 443–449. <https://doi.org/10.1109/ICPR.1990.118143>.
- [20] S. PAL, P. GANGULY, AND P. K. BISWAS, *Cubic Bézier approximation of a digitized curve*, Pattern Recognition, 40 (2007), pp. 2730–2741. <http://dx.doi.org/10.1016/j.patcog.2007.01.019>.
- [21] W. PAN, Z. LIAN, Y. TANG, AND J. XIAO, *Skeleton-guided vectorization of Chinese calligraphy images*, in International Workshop on Multimedia Signal Processing (MMSP), IEEE, 2014, pp. 1–6. <https://doi.org/10.1109/MMSP.2014.6958805>.
- [22] M. PLASS AND M. STONE, *Curve-fitting with piecewise parametric cubics*, in Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH), 1983, pp. 229–239. <https://doi.org/10.1145/800059.801153>.
- [23] F. P. PREPARATA AND M. I. SHAMOS, *Computational geometry: an introduction*, Springer Science & Business Media, 2012. ISBN 9781461210986.
- [24] U. RAMER, *An iterative procedure for the polygonal approximation of plane curves*, Computer Graphics and Image Processing, 1 (1972), pp. 244–256. [https://doi.org/10.1016/S0146-664X\(72\)80017-0](https://doi.org/10.1016/S0146-664X(72)80017-0).
- [25] E. ROSTEN AND T. DRUMMOND, *Fusing points and lines for high performance tracking*, in International Conference on Computer Vision (ICCV), vol. 2, Ieee, 2005, pp. 1508–1515. <https://doi.org/10.1109/ICCV.2005.104>.
- [26] G. SAPIRO AND A. TANNENBAUM, *Affine invariant scale-space*, International Journal of Computer Vision, 11 (1993), pp. 25–44. <https://doi.org/10.1007/BF01420591>.
- [27] M. SARFRAZ, *Vectorizing outlines of generic shapes by cubic spline using simulated annealing*, International Journal of Computer Mathematics, 87 (2010), pp. 1736–1751. <http://dx.doi.org/10.1080/00207160802452519>.

- [28] K. TOMBRE AND S. TABBONE, *Vectorization in graphics recognition: to thin or not to thin*, in International Conference on Pattern Recognition (ICPR), vol. 2, IEEE, 2000, pp. 91–96. <https://doi.org/10.1109/ICPR.2000.906024>.
- [29] H.-M. YANG, J.-J. LU, AND H.-J. LEE, *A Bézier curve-based approach to shape description for Chinese calligraphy characters*, in International Conference on Document Analysis and Recognition, IEEE, 2001, pp. 276–280. <https://doi.org/10.1109/ICDAR.2001.953798>.
- [30] J. ZOU AND H. YAN, *Cartoon image vectorization based on shape subdivision*, in Computer Graphics International, IEEE, 2001, pp. 225–231. <https://doi.org/10.1109/CGI.2001.934678>.