Computers & Graphics (2018)

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

# A Delaunay triangulation based approach for cleaning rough sketches

ARTICLE INFO

*Article history*:
Received March 12, 2018

*Keywords:* Delaunay triangulation, Sketch Simplification, Sketch segmentation, Vectorization

ABSTRACT

Given a set of rough strokes drawn by an artist (either in pen-paper medium or in digital medium) in raster format, the objective is to group them meaningfully and represent the group with simple most appropriate curves. In this paper, a Delaunay triangulation based algorithm is proposed for grouping strokes. The grouping procedure is capable of identifying open curves and reconstructing broken strokes. The proposed algorithm is capable of helping the user in masking misinterpreted regions. We also introduce a shape aware skeleton smoothing procedure which best approximates the shape by taking input raster sketch as a reference to create final vector output. The user can also control the final output. The proposed algorithm combines the techniques in computational geometry as well as in image processing to utilize the power of both.



## 1. Introduction

From an artist's perspective, it is easier to create a rough drawing which usually has multiple strokes to represent a simple shape. The main reason behind this is that, it gives the artist more flexibility to correct the mistakes that happen during drawing. Also, artists usually draw a complicated sketch starting from a set of vague simple shapes which he/she overdraws to create the original object. This rough sketch making is the initial step of story telling in applications like model creation, image generation etc. On the other hand, for developing applications such as 3D modeling, character drawing etc., developers prefer vector drawings. Vector drawing helps application developer to easily edit the shape by manipulating a few control points. It also helps to create sharp and clean drawings with less storage requirement. To handle this gap between artists and developers comfortness, a rough sketch to vector drawing conversion is necessary.

It is very difficult to draw a sketch perfectly by both expert and novice users. Keeping this in mind, various sketch beautification and sketch simplification systems have been introduced. The aim of sketch beautification systems is to help users by modifying user drawn strokes based on the geometric and proximity relationship between strokes. The sketch simplification systems simplify the user drawn sketch by removing unwanted information from the stroke. In this paper, we introduce a Delaunay triangulation based stroke simplification algorithm and a shape aware skeleton simplification algorithm for rough raster sketch to vector drawing conversion.

The main contributions of the proposed approach are:

- A Delaunay triangulation based stroke grouping algorithm which helps in a better identification of regions and hence features.
- A shape aware skeleton smoothing algorithm for generating a better approximation of the shape.
- User control for masking of regions in the sketch and control of accuracy of the vectorized output.

### 1.1. Related Works

Sketch simplification tools mainly concentrate on representing the entire sketch with a set of few meaningful strokes. Most of the approaches can be divided into one of the two classes: stroke reduction and stroke grouping. In stroke reduction based techniques, the input strokes are classified according to their relevance and are removed one by one until the sketch is simplified. Work introduced by Preim et al. [1] uses parameters like length of line, distance and density, to prioritize strokes. Density information is used in the work introduced by Grabli et al. [2] to select relevant strokes from a set of cluttered line drawings. The 2D properties along with information from three

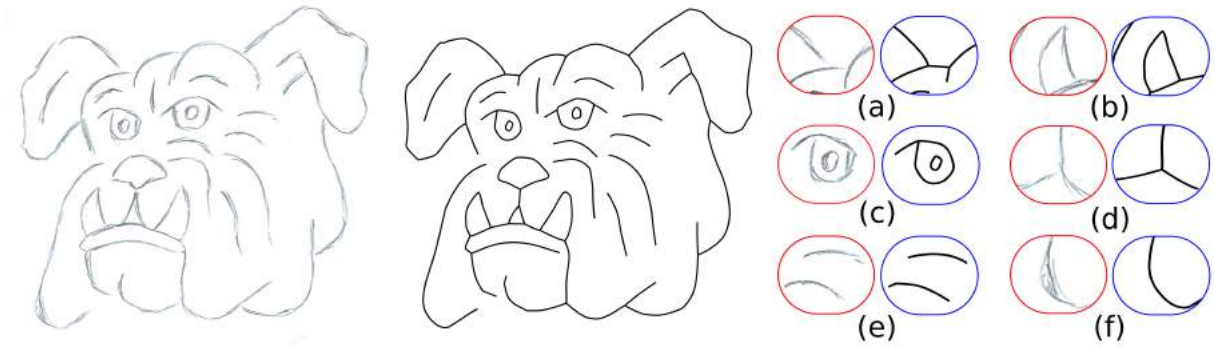



Fig. 1: Left to Right:: Input rough sketch (scanned sketch), Output of our algorithm (vector drawing) and blown up parts of various features (in blue boxes) ((a): Broken strokes, (b): Sharp corners, (c): Small regions, (d): Junctions, (e): Open, disconnected curves, (f): A group of strokes) captured by our algorithm along with input sketch (in red boxes).

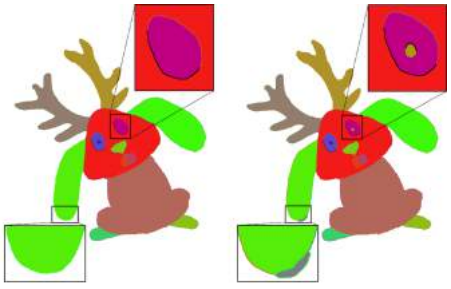


Fig. 2: Result of trapping ball segmentation for two different ball radii, blown up part shows the error occured during segmentation.

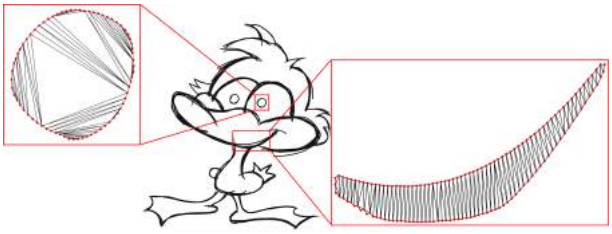


Fig. 3: Delaunay triangulations inside a region and between two adjacent strokes.

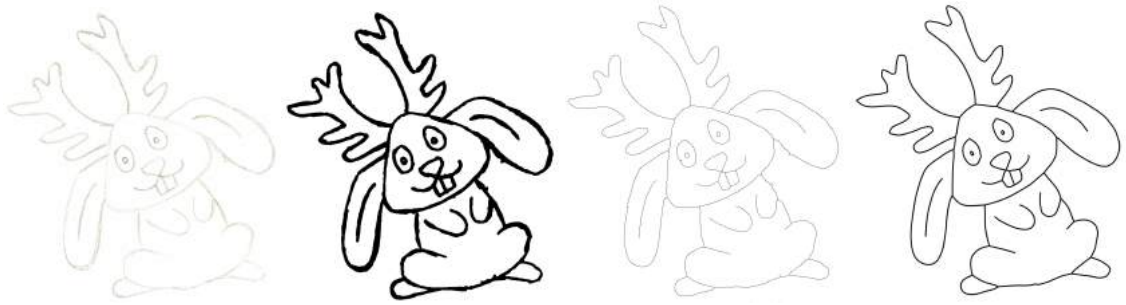
dimensional inputs are used in works introduced by Wilson et al. [3] and Deussen et al. [4]. The main drawback with these kind of systems is that, since it is simplifying strokes by iteratively removing irrelevant strokes. The simplified stroke is always a subset of input strokes, whereas, artists usually use a set of strokes for representing a long curve [5].

In stroke grouping based methods, the input strokes are grouped together and replaced by a simple stroke. The strokes are grouped based on continuation, parallelism and proximity in works like Rosin et al. [6]. A selection assistance tool for perceptual grouping and suggesting potential extensions is introduced by Lindlbauer et al. [7]. A greedy algorithm based on the concept of $\epsilon$-lines is introduced by Barla et al. [8][9] for grouping and is further used for simplification. Learning based stroke grouping is used in works by Orbay et al. [10], Simo-Serra et al. [11] and Ogawa et al. [12]. Noris et al. [13] use stroke-to-stroke and scribble-to-stroke relationship for grouping strokes in a digitally drawn sketches. Gestalt phenomenon of closure is utilized by [5] to interpret regions which helps to group strokes. The recent work introduced by Favreau et al. [14] concentrates on generating a vector drawing from a rough sketch to approximate the shape and is composed of a small number of curves which makes it easily editable.

Works like interactive beautification [15], suggestive interface for 3d drawing [16], exploring sketch beautification techniques [17] and ShipShape [18] follow the sketch beautification systems. The main disadvantage of this kind of system is that the input should be given in a progressive manner which makes it unsuitable for applications where the input is a scanned image drawn by an artist.

A number of methods are introduced for vectorizing an input drawing [19], [20], [21], [22]. The main idea behind these algorithms is to find junction points from a 1-pixel width skeleton of the drawing. The curves in between junction points are further replaced by simple primitives like lines, arcs and curves. Different from these existing algorithms, the proposed algorithm uses the grouped strokes for determining the curve to be kept between junction points. A recent work by Bessmeltsev et al. [23] makes use of vector fields to place curves. Donati et al. [24] made use of Pearson's cross correlation and introduced a new thinning algorithm with improved capability to handle varying stroke widths and scribbles.

The stroke grouping methods try to cluster similar strokes into same group. Most of the algorithms in this category rely on a region identification procedure (like trapped ball segmentation) to group the strokes. Even though trapped-ball segmentation [25] used by various related works (works by Liu et al. [5], Falvreau et al. [14] etc.) can do region identification moderately fine, the control of user over the segmentation result is less. Figure 2 shows the result of trapped ball segmentation. It can be noted that, as we try to decrease the ball radius to detect iris, unwanted regions are getting detected as well. In this paper, we introduce a Delaunay triangulation based region identification algorithm that provides better control over the identified regions. Though Delaunay triangulation has been used for piecewise-linear reconstruction from 2D point-sets [26] (which



(a) (b) (c) (d)

Fig. 4: Overall steps in the proposed algorithm. (a) Input rough sketch, (b) Result after grouping similar strokes, (c) Result of skeletonization, (d) Our result.

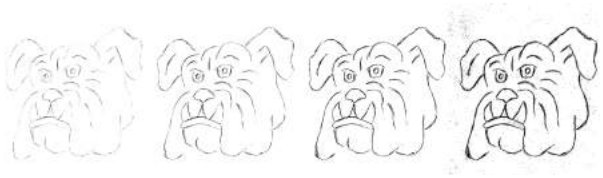


Fig. 5: Effect of varying threshold parameter, Left to right: Results of thresholding for values 0.7, 0.8, 0.9 and 1 respectively.

can be used for a noisy point set similar to sketch strokes), to the best of our knowledge, it has not been used yet for region identification from a sketch input.

Similarly, works in image vectorization field usually simplify the 1-pixel width skeleton for finding vector representation of the sketch. The vector drawing can further be simplified by reducing the points for representing curves. This direct simplification of skeleton typically results in loss of information. To avoid this, while simplifying, the skeleton can be matched with the original rough sketch to ensure that the curve always follow the structure defined by rough sketch and hence the artist's imagination. Hence, different from other stroke grouping approaches, in our approach, the similar strokes are grouped based on a Delaunay triangulation based region identification algorithm, and furthermore, is used to generate a vector representation of the sketch. We exploit the triangle shape/structure and their connectivity present in DT to identify regions. This enables the user to mask the pseudo regions (the gap between strokes which looks like a region by itself) generated from the input sketch.

Rest of the paper is arranged as follows: Section 2 gives the overall framework of the proposed system. The algorithm is explained in Section 3 and the results and discussion are given in Section 4. Finally, the paper is concluded along with possible future works in Section 5.

## 2. Overview

The main motivation behind the work is based on the observation that the Delaunay triangulation inside a region has fat triangles (acute triangles) as shown in Figure 3. On the other hand, the Delaunay triangles between the adjacent strokes (which can be combined into a single group) has thin triangles (obtuse triangles). The observation helped us to realize that capturing the fat triangles and their associated ones lead to the identification of regions.

Based on the observation, our overall framework is as shown in Figure 4. The input to our system is a pencil drawn rough sketch as shown in Figure 4(a) (can also be a set of digital strokes made by digital drawing tools). Image thresholding is applied on the sketch (if the input is a pencil sketch) and the result is fed to our region identification and skeleton generation procedure. The adjacent strokes are combined using a Delaunay triangulation based algorithm (as shown in Figure 4(b)), from which an approximate shape is generated (as in Figure 4(c)). This is further represented using a set of Bézier curves to create a vector drawing (as shown in Figure 4(d)). The user is given the flexibility to alter the number of control points representing the shape, which helps in easier editing of the sketch. The result of our procedure is in ready-to-edit simplified vector format. The user can edit the control points using any existing tools like Inkscape to alter the shape.

## 3. Algorithm

Our algorithm can take a scanned pencil drawn paper sketches or digitally drawn sketch as input. If the input is scanned pencil drawn paper sketch, a binary thresholding is applied on it. After several experiments, we fixed the thresholding parameter as 0.9 (in which all pixels which has intensity lower than 0.9*255 are retained). Effect of varying the threshold parameter is shown in Figure 5. As shown in the figure, many useful stroke components are missing for low thresholding values and unwanted strokes come to play for larger threshold values. After thresholding, the procedure mainly consists of two steps: (a) Region identification and skeleton generation (to identify regions and create an unpolished simplified version of the rough sketch) (b) Shape aware skeleton smoothing (to smooth and create a best shape from the approximation).

### *3.1. Region identification and skeleton generation*

**DEFINITION 1.** ***Exterior triangle:*** *A triangle (in a set of triangles) having an edge that is not shared with any other triangle (edge in 'red' in Figure 6) and has length more than a user given parameter len.*

**DEFINITION 2.** ***Fat seed triangle:*** *A fat triangle whose circumcenter lies inside itself (green triangle in Figure 7a) and having edge lengths greater than len.*

Given a sketch, this step creates an approximate simplified shape in the form of skeleton. The Delaunay triangulation is used for creating an approximate shape in this step. Since the input sketch is in pixel format and Delaunay triangulation is defined over a point set, a mapping from pixels in the image to points in Euclidean space has been made (by keeping points in the center of each pixel) for further processing. Starting from the Delaunay triangulation of the mapped point set, a fat seed triangle (similar to high persistent cells in [27]) is found and a growing procedure is initiated to identify a region. From a fat seed triangle (which represents a region), the algorithm grows by recursively merging the adjacent triangles (if the edge shared between them has length more than a user given parameter *len*). The growing procedure is continued until no more triangles can be merged to the region. Once the region cannot grow further, the procedure continues to find the next region, starting from an unprocessed fat seed triangle. The entire procedure is continued until no more fat seed triangles can be found. Figure 7 shows the illustration of our region finding algorithm on the sketch of a Jackalope. Starting from a fat seed triangle (Figure 7(a)), the algorithm recursively merges adjacent triangles to the region (Figure 7(b)) until it cannot grow further (since the length of edge is less than the parameter, Figure 7(c)). Finally, the triangles are merged to create a region (Figure 7(d)). Once a region is found, the region growing procedure continues by finding the next fat seed triangle (Figure 7(e) - 7(l)). The procedure terminates when the growing procedure is done for all fat triangles (whose all edge lengths are greater than *len*). Figure 7(m) shows the regions identified using our region growing algorithm. The *len* parameter tells how much apart the strokes of the same group can be, and is easily found by visually tuning the sketch and checking whether unwanted strokes are getting grouped together.

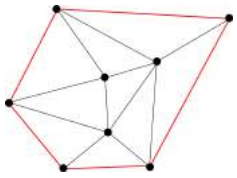


Fig. 6: A sample Delaunay triangulation with exterior edges shown in red color.

All unvisited triangles (which are not visited during the growing procedure) are filled with black color (as in [20]) which gives an approximation of the shape by combining similar strokes (Figure 7(n)). The filling procedure is applied for combining small triangles between strokes. However, the resulting filled shape has varying thickness depending on the strokes in the input sketch (as shown in blown up part of Figure 7(n)). It is difficult to convert this filled shape to a set of simple curves. In order to tackle this problem, the filled shape is mapped back into an image and a skeleton generation [28] algorithm is applied to make its width uniform (Figure 7(o)). For further explanation, the result after filling the triangles are called as filled-shape.

A fat seed triangle might lie outside the shape as well, which leads to the detection of regions outside. In order to stop the algorithm from taking such fat seed triangles, a Delaunay sculpting [29] is employed before region identification procedure. The Delaunay sculpting procedure is as follows:

- On the Delaunay triangulation *DT* of the point set, Exterior edges are found (edges which is shared with only one triangle)
- Exterior edges having length greater than *len* are removed, and the exterior edges are updated until the length of all exterior edges are less than *len*

Figure 8 shows the Delaunay triangulation of the point set, and the remaining triangles respectively. It can be seen that the Delaunay sculpting procedure with appropriate *len* removes all triangles lying outside the shape and hence stops the procedure from selecting fat seed triangles from outside the shape. The *len* parameter depends on the distance between two adjacent strokes in same group.

An interesting property we observed while experimenting is that, the number of regions identified can be controlled by restricting the number of fat seed triangles being processed. In order to process large regions before processing small ones, the fat seed triangles are processed in the decreasing order of their circumradii. Figure 9 shows the results of our algorithm for different number of fat seed triangles that are getting processed. It can be seen that the nostrils are getting completely masked in the beginning and started appearing one by one as we increase the number of fat seed triangles (masking value) to be processed.

**Algorithm 1:** Grow_Triangle()

**Input:** *Triangulation DT*, *Triangle T*, *Length len*, *Region R*
**Output:** A region *R*
**if** *T is already visited* **then**
  **return**
**end**
$R = R \bigcup T$
Let $p_1$, $p_2$, $p_3$ be three vertices of $T$
**if** *(Euclidean_distance(*$p_1$, $p_2$*) >len )* **then**
  Grow_Triangle(DT, Neighbor(DT, T, $p_1$, $p_2$), len, R)
**end**
**if** *Euclidean_distance(*$p_2$, $p_3$*) >len* **then**
  Grow_Triangle(DT, Neighbor(DT, T, $p_2$, $p_3$), len, R)
**end**
**if** *Euclidean_distance(*$p_3$, $p_1$*) >len* **then**
  Grow_Triangle(DT, Neighbor(DT, T, $p_3$, $p_1$), len, R)
**end**
**return** *R*

Figure 10 (first row) shows a small blown up part of a sketch after thresholding, its pixels, generated point set (by placing points in the center of each pixel whose color is black), Delaunay triangulation and the result after filling. Second row of Figure 10 shows the image after thresholding, triangles after growing procedure, regions and filled shape.

The pseudocode given in Algorithm 1 shows the region growing procedure. The function Euclidean_distance(a, b) finds the

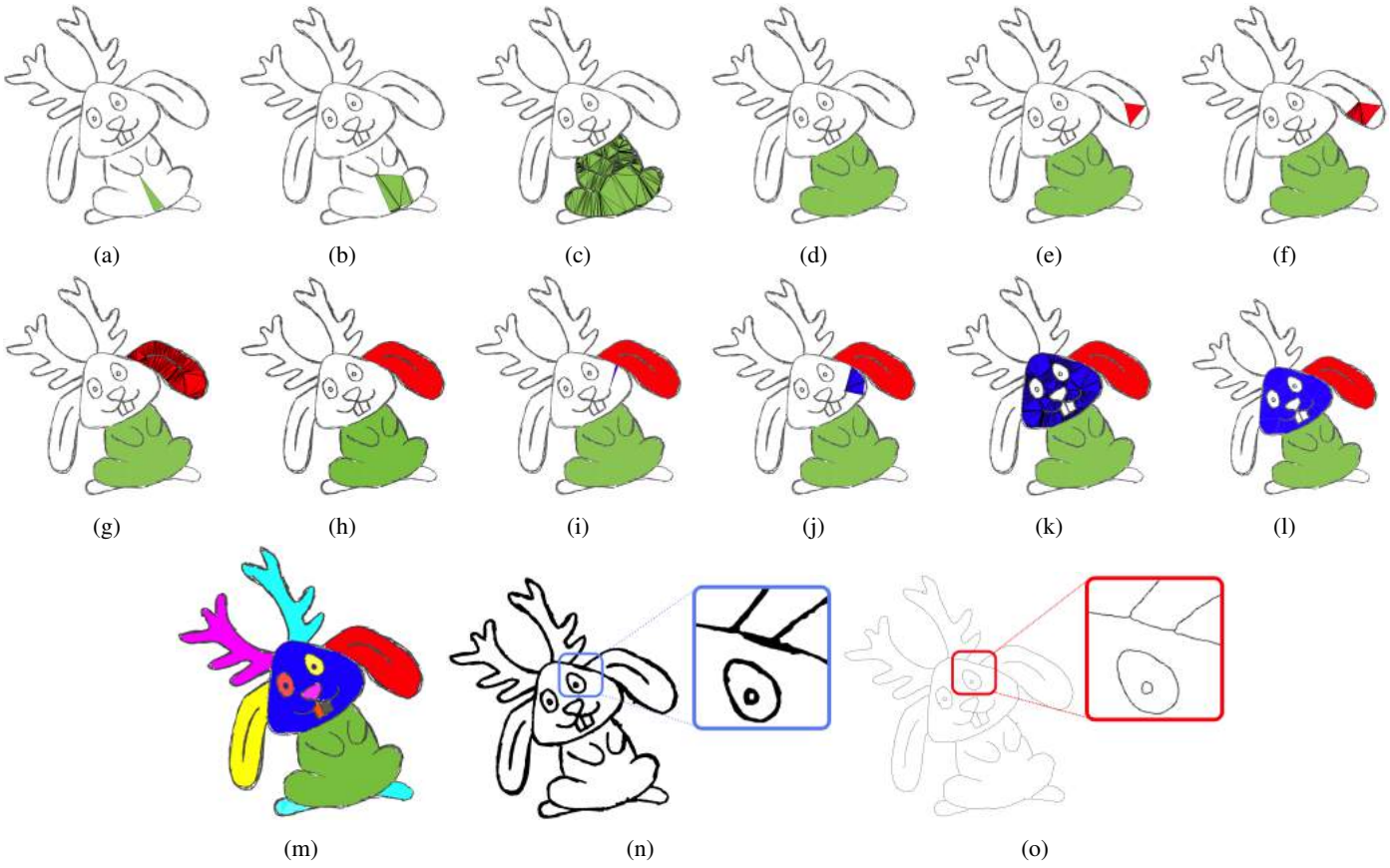



Fig. 7: Illustration of region finding procedure: (a), (e) and (i) show the beginning of region growing algorithm with green, red and blue triangles as the fat seed triangles, (b), (f) and (j) show the intermediate steps of region growing procedure where a cluster of triangles are merged to the intial fat seed triangle, (c), (g) and (k) show the termination of region growing algorithm for corresponding fat seed triangles, (d), (h) and (l) show the respective identified regions, (m) final set of regions found, (n) filled shape, (o) skeleton of the filled shape.

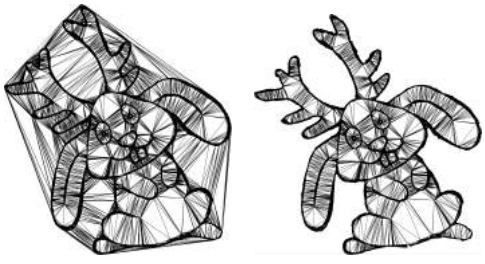



Fig. 8: (a) DT of a rough sketch, (b) Triangles after Delaunay sculpting.

Euclidean distance between two points a and b. Neighbor($DT$, $T$, $a$, $b$) function returns the triangle from $DT$ which shares edge $(a, b)$ with $T$, if any. The algorithm is called for each fat seed triangle to divide the rough sketch into regions. The complete region identification procedure is shown in Algorithm 2, where Delaunay_Triangulation(PS) function computes the Delaunay triangulation of the point set $PS$. The Delaunay_Sculpting(DT, $len$) function performs Delaunay sculpting on $DT$ until all exterior edge lengths are less than $len$, and returns the remaining triangles.

At the end of Algorithm 2, since skeletonization results in images, further processing is done on pixels. Small protrusions are removed from skeleton (in the following sections, we will be using the term skeleton to refer to the skeleton of filled shape) to ignore the branches made by varying thickness of the filled

**Algorithm 2:** Algorithm for skeleton generation

**Input:** *Thresholded image I*, *len*
**Output:** *Skeleton of grouped strokes*
DT=Delaunay_Triangulation(point set representation of $I$)
DST= Delaunay_Sculpt(DT, *len*)
**for** *each fat seed triangle T in DST sorted in decreasing order of circumradius* **do**
  **if** *number of identified regions is less than masking value* **then**
    $R = \phi$
    $R$=Grow_Triangle($DST$, T, len, R)
    Remove all triangles in $R$ from DST and classify their union as a region
  **end**
**end**
Fill each triangle in $DST$ with black color and create a new image $NI$
Apply skeleton thinning on $NI$
**return** $NI$

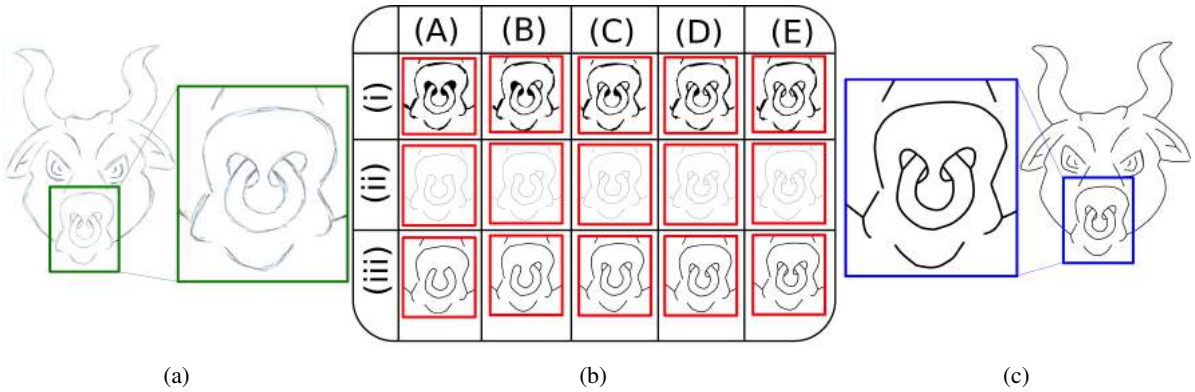



Fig. 9: (a) Input rough sketch, (b) Result of our algorithm with various masking values. Rows (i), (ii) and (iii) show filled shape, skeleton and vectorized results respectively. Columns (A)-(E) show result for masking values 1,2,3,4,5 respectively. As the masking value increases, the number of regions identified are increasing one by one as shown in Row (iii), (c) Final result of our algorithm.

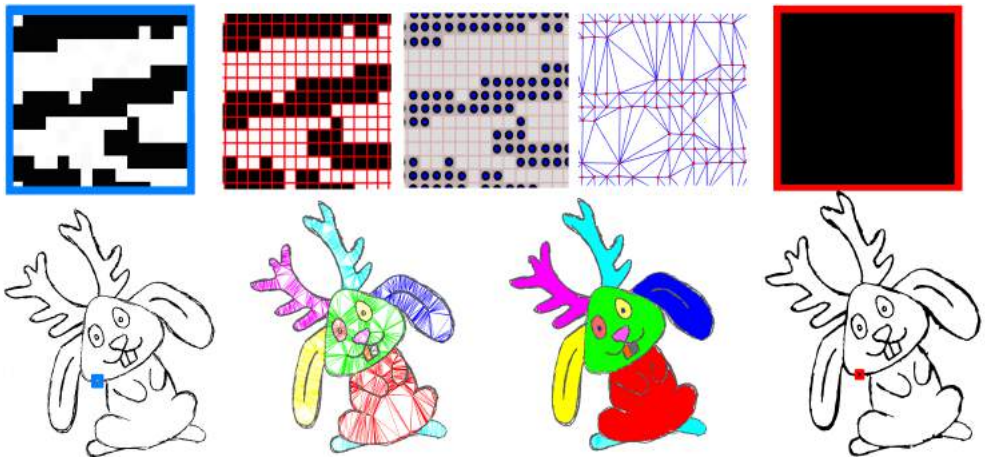


Fig. 10: First row: (Left to Right): Blown up image part, its pixels as squares, point set generated, Delaunay triangulation, filled part, Second row: (Left to Right): Thresholded sketch input, Clustered triangles, Identified regions, Filled shape.

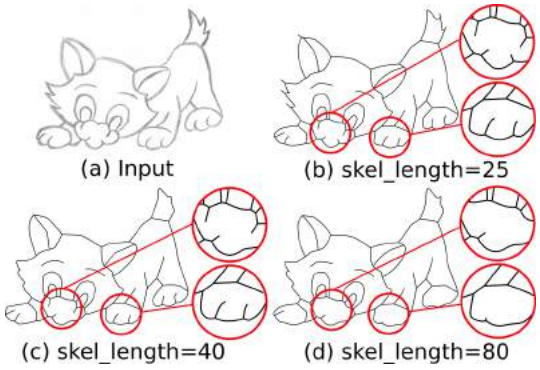



Fig. 11: A sample rough sketch along with results after increasing the skeleton pruning parameter.

shape. For skeleton pruning, a parameter *skel_length* is used, and all protrusions which contain less than the number of pixels given by *skel_length* are removed. Figure 11 shows the effect of *skel_length* parameter on a sample rough sketch. The *skel_length* is the minimum length, an open curve should have in order to be retained. For example, a high random value can be given for *skel_length* parameter, if the shape does not have any open curve. It can also be easily tuned by checking whether the parameter value removes all unnecessary open curves. After applying skeletonization procedure, we noticed that sharp corners are converted to junction pixels (pixels which has more than two neighbors in its 8-neighborhood) with three edges, out of which one is an open curve. To restore such sharp corners, similar to biasing effect removal in [24], we made use of a simple heuristic. If the angle between non-open curves in a junction pixel is lesser than that of the one with open curve, then it is considered as a sharp corner and restored by moving the intersection point to the end point. Figure 12 shows a sample sketch, whose skeletonization has both sharp and non-sharp corners, and how the heuristic is handling such cases.

### *3.2. Shape aware skeleton smoothing*

As shown in Figure 7(o), the generated skeleton is not smooth or in vector format. To alleviate this problem, a shape aware skeleton smoothing algorithm is introduced. In this step, the skeleton is replaced by a smooth shape that preserves the shape drawn by an artist.

Figure 13(a), shows a filled shape along with its skeleton. Directly, the skeleton pixels can be considered as anchor points and a cubic Bézier curve can be fitted, but the resulting output

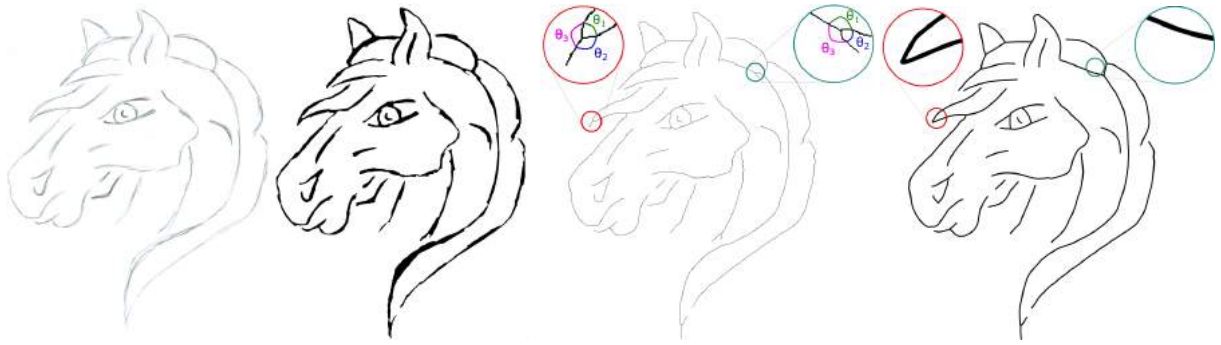



Fig. 12: Heuristic for restoring sharp corners, Left to Right: Input rough sketch, Filled shape, Skeleton (sharp corner and non-sharp corner in blown up circles), Our result with restored sharp corner.

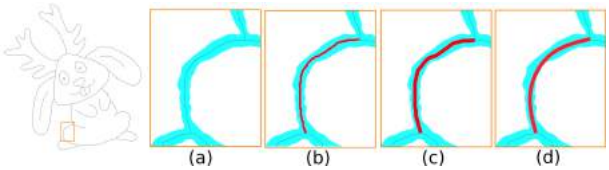



Fig. 13: Smoothing skeleton by skipping its intermediate pixel. A blown up part of Jackalope's leg is taken for reference (a) Filled shape (blue color) along with skeleton (black color), (b) Filled shape along with Bézier curve representation by taking all pixels of skeleton into consideration, (c) Filled shape along with its Bézier curve representation by skipping 25 intermediate pixels, (d) Filled shape along with its Bézier curve representation by skipping 65 intermediate pixels.

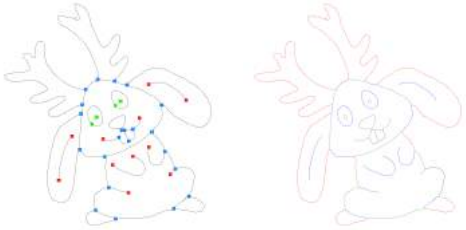



Fig. 14: Representative pixels shown inside boxes (Left), Skeleton with segments shown in different colors (Right).

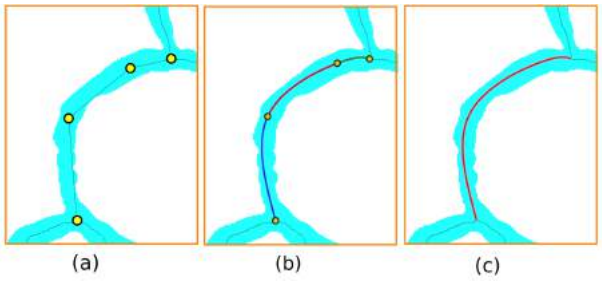



Fig. 15: Shape aware smoothing. (a) Filled shape along with new polyline $NP$ (vertices shown in yellow color), (b) Filled shape along with its Cubic Bézier curve fitted using vertices of $NP$, (c) Smooth curve lying inside the filled shape.

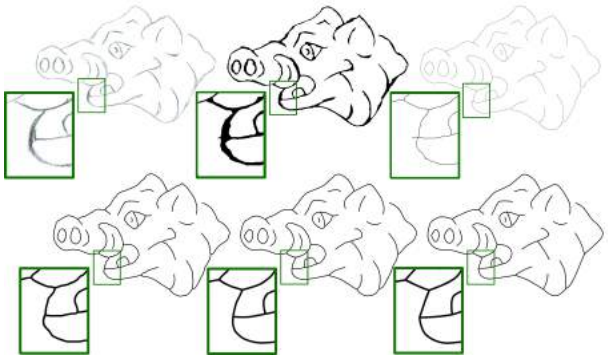


Fig. 16: Relevance of *length* parameter, First row (Left to Right): Input sketch, Filled shape, Skeleton, Second row (Left to Right): Output of our algorithm as the *lim_length* increases.

can have undue oscillations (Figure 13(b)). The problem can be tackled by skipping few intermediate points from the skeleton. Figures 13(c) and 13(d) show two examples by skipping intermediate 25 and 65 points (starting from a junction point, every 25th and 65th consecutive pixels are considered) respectively. Such a method raises two major concerns: (a) How many pixels should be skipped?, (b) Whether the shape will get distorted after skipping?

To address the problem, we introduce a shape aware skeleton smoothing procedure. Initially, a few representative pixels are selected from the skeleton as follows:

- A pixel which has more than two neighboring pixels in its 8-neighborhood (pixels inside blue colored boxes in Figure 14(a))
- A pixel which has only one neighbor in its 8-neighborhood

(pixels inside red colored boxes in Figure 14(a))

The skeleton is converted into a set of segments based on representative pixels (Figure 14(b) shows the skeleton segments shown in different colors). Each skeleton segment is converted into a polyline $P$ with each pixel considered as a vertex and two vertices $v_i$ and $v_j$ representing pixel $x_i$ and $x_j$ are connected with an edge if:

- If $x_i$ is in the 4-neighborhood of $x_j$
- If $x_i$ has only one/no pixel in its 4-neighborhood and $x_j$ lies in the 8-neighborhood of $x_i$

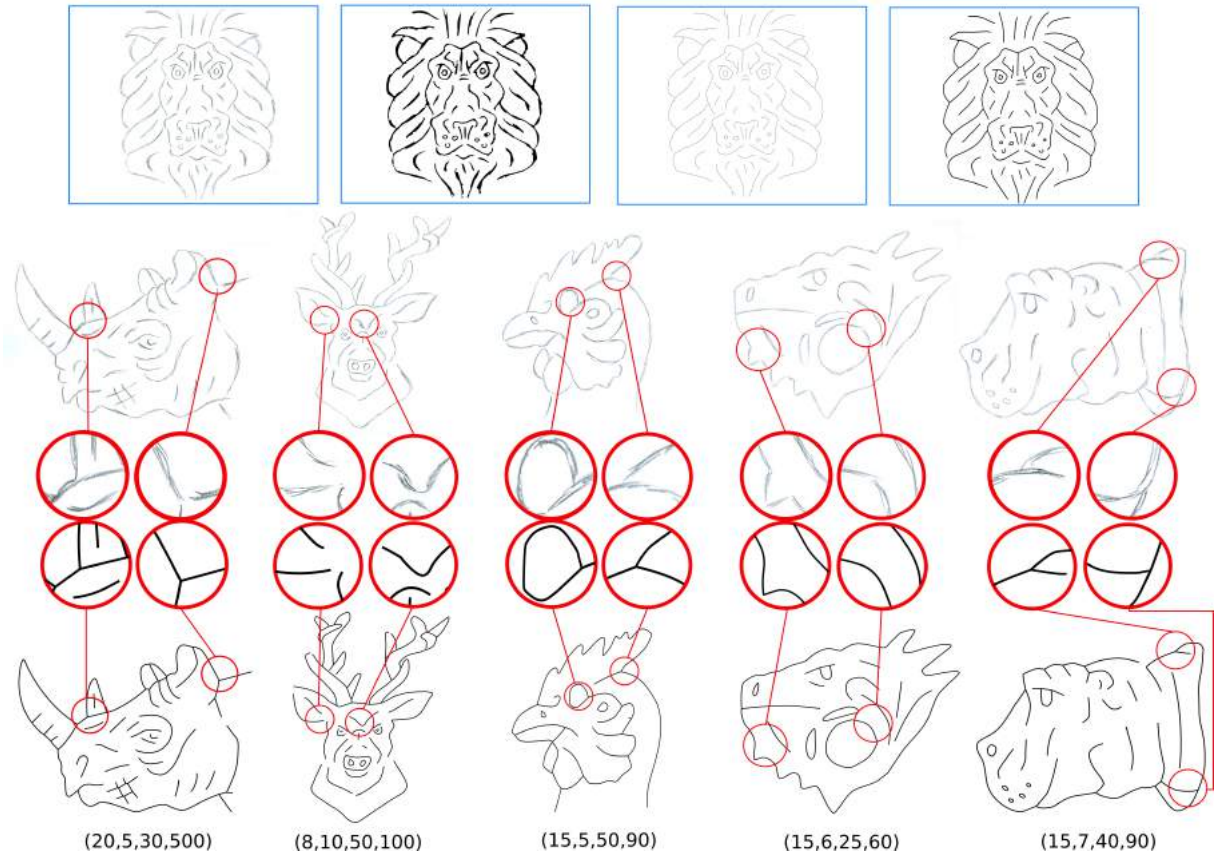



Fig. 17: Result of our algorithm for various inputs, First row shows the input sketch, filled shape, skeleton and result of our algorithm respectively for lion sketch. Second row shows the Input rough sketch (scaled down) and third row shows corresponding results. Parameters used are given in brackets (len, masking value, skel_length, lim_length) respectively.

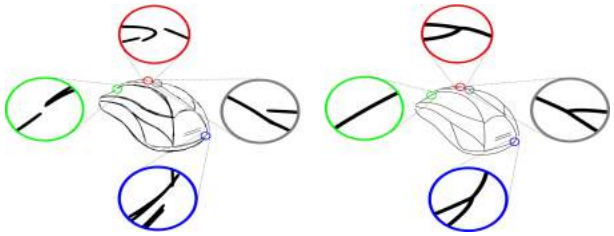


Fig. 18: Filling small gaps in input sketch: "Mouse" input and our output.

In order to represent part of the skeleton which is a simple closed curve (with all pixels have exactly two neighbors in its 8-neighborhood), and does not have any representative pixels, a random pixel is considered as the representative pixel to create a polyline (pixels shown inside green colored boxes in Figure 14(a)).

From the polyline $P = p_1, p_2, ..., p_n$, a new polyline $NP = np_1, np_2, ..., np_k$ with minimum possible number of vertices is created from $P$ satisfying the following conditions:

- Let the vertex set of $P$ and $NP$ be $X(P)$ and $X(NP)$, then $X(NP) \subseteq X(P)$.
- Let $np_i$,$np_j \in NP$ represents $p_k, p_m \in P$ respectively, then $\|np_i, np_j\| \leq \|p_k, p_m\|$.
- $|P| \geq |NP|$.
- Let $F$ be the filled shape, all edges $(np_i, np_j) \in NP$ lies interior to $F$.

The new polyline $NP$ is computed in a greedy fashion, in which, starting from an end point $p_1 \in P$, lines are computed to all succeeding vertices in $P$. The longest possible line which completely lies inside the filled shape is selected and the corresponding vertex $p_2$ is added to $NP$. Next, the lines are drawn from $p_2$, and the procedure continues until $p_2$ becomes the last vertex of $P$. By taking the vertices in $NP$ as anchor points, piecewise cubic Bézier curves can be fitted preserving tangent and second derivative continuity as in [14]. However, in this paper, we made use of a readily available SVG tool, Inkscape.

Figure 15(a) shows a new polyline $NP$ (yellow points represents vertices and black lines connecting them are the edges) along with a part of $F$ on both sides. Figure 15(b) shows the Bézier curve created using $NP$. Final Bézier curve along with its filled shape is shown in Figure 15(c).

The method of keeping straight lines inside the filled shape tells about how many pixels can be skipped between two anchor points. Results show that our algorithm is capable of generating vector output resembling their corresponding rough sketches.

To provide more control for user to decide on the accuracy of resulting shape, a limiting length parameter can be used. The role of limiting length parameter "lim_length" is to stop drawing of a line at a length given by the parameter. The parameter

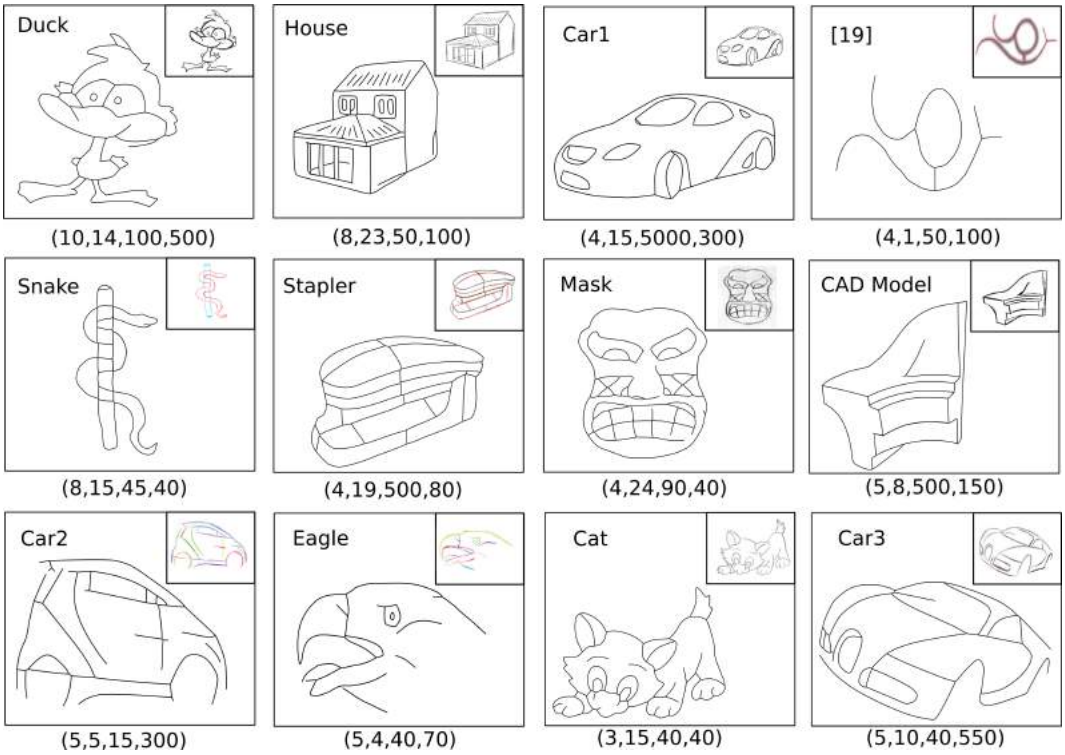



Fig. 19: Output of our algorithm for "Duck", "House", "Car" and inputs taken from [19], [13], [30], [11], [14] and [10]. Corresponding inputs are shown in top right corner. Parameters used are given in bracket (len, masking value, skel_length, lim_length) respectively.

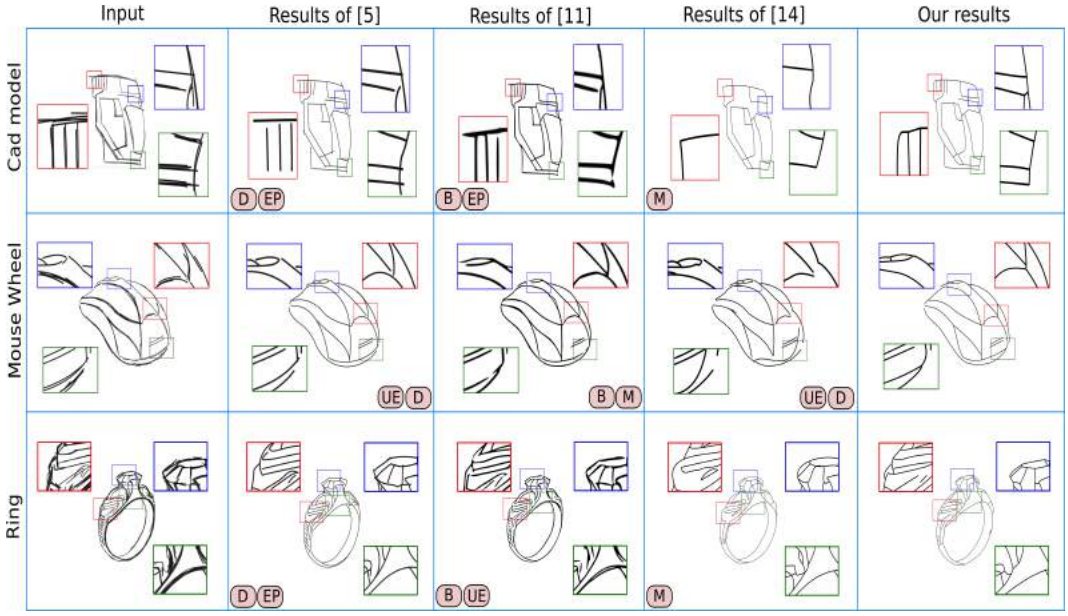
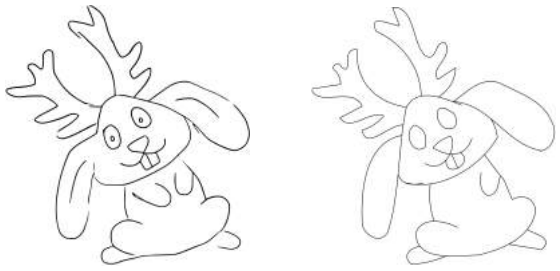



Fig. 20: Comparison of our results with the state of the art methods, Left to Right: Input, Output of [5], [11], [14] and Our result. Various problems in the outputs are denoted in bubbles, where each letter denotes, D: Disconnected, EP: Extra protrusions, B: Broken edges, UE: Unclean edges, M: Missing part.

stops the growing of the line in skeleton smoothing step either if a straight line cannot be drawn or the line under consideration exceeds the "lim_length". Increased value of the parameter decreases the number of points in $NP$ and hence the accuracy. On the other hand, decreasing the parameter leads to more number of points in $NP$ and hence the accuracy with respect to the skeleton. Figure 16 shows the result of our shape aware skeleton smoothing procedure for various values of "lim_length". The *lim_length* parameter is used to smoothen the resultant vector output (increasing the value increases the smoothness of the result) and can be increased/decreased until user gets a visually best result.



(a) (b)

Fig. 21: Results of [11] (Left) and [14] (Right) for "Jackalope" input.

## 4. Results & Discussion

Figure 17 shows the result of our algorithm for various inputs (which are scaled down appropriately) with different features. It can be seen that complex multiple strokes are represented using single simple stroke, all small protrusions coming out of strokes are removed and small details are retained. Experiments show that our algorithm is capable of capturing various features like: sharp corners, small regions, junctions, open disconnected curves etc.

Figure 18 shows a standard sample input in which the input sketch has various discontinuities. It can be seen that our algorithm is capable of dealing with such small gaps and gives nice results.

We generated results for the inputs taken from various papers ([10], [13], [19], [30], [11], [14]) and for standard inputs like "Duck", "House", "Car", and the outputs are shown in Figure 19. It can be seen that our algorithm is capable of generating good results for all the inputs. Irrespective of sketch type (pencil drawn or digital strokes), our algorithm generated good results. Our algorithm also captured original regions (duck), minute details (house), simple strokes from irregular sketch (car), self-intersections and junctions (snake, stapler), clean output from noisy paper sketch (mask), sharp corners (cad model) etc. Even though the very coarse nature of the input sketches resulted in a few unwanted strokes and gaps, it can be seen that our algorithm generated a visually better vector result.

Figure 20 shows the comparison of our algorithm with various existing methods on inputs which represents the general cases such as sketch with multiple unwanted protrusions (CAD model), disconnected components (mouse wheel) and very sketchy input with minute features (ring). It can be noted that the procedure by Liu et al. [5] is not able to handle broken data and many extra protrusions are retained as such. The learning based approach used in [11] has many discontinuities in the resulting shape, as they are ignoring various parts of the shape (mouse wheel) assuming that they are part of same stroke group. Favreau et al. [14] merges adjacent regions and so the important information in the shape is lost.

We have compared our "Jackalope" input with the most recent works ([11] and [14]) and Figure 21 shows the results of their algorithms. Results of Favreau et al. [14] is obtained from the authors, and it can be seen that it is not able to capture various open curves (curves which cannot be represented as a region). Also, it can be noted that the eyes are not captured well

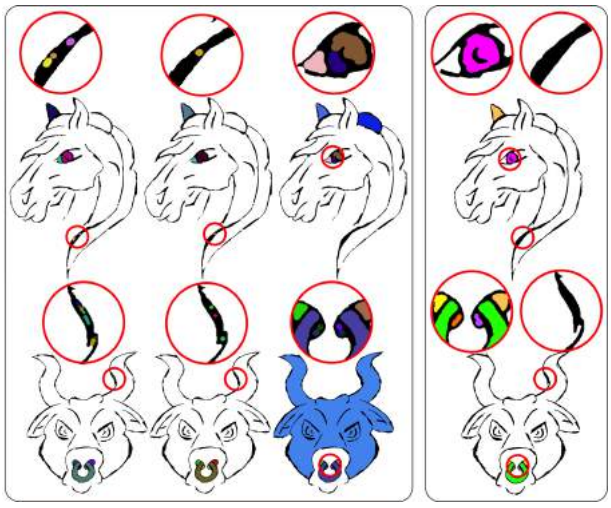


Fig. 22: Effect of the simple trapped ball segmentation on two sketches for varying ball sizes [(5,6,7) and (2,3,4) for horse and bull respectively] (Left), Result of our region identification algorithm (Right).

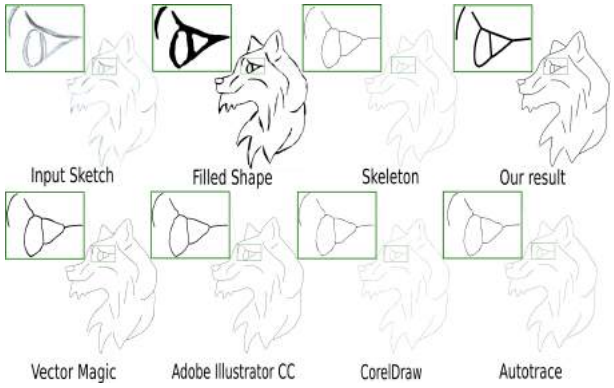



Fig. 23: Result of various vectorization algorithms, First row (Left to Right): Input image, Filled shape, Skeleton, Our result, Second row (Left to Right): Result of Vector Magic, Adobe illustrator CC, CorelDraw, Autotrace.

in the result since the ball radius used for region identification is high and as the ball radius decreases, small gaps between the adjacent strokes are considered as separate regions.

Result of learning based approach introduced in [11] is shown in Figure 21(b). We tuned the parameter and the best result is selected for comparison. It is clear that the result contains various disconnected curves. Many of the strokes are ignored (connection between ear) from the input sketch while generating the final result. Some strokes are misclassified as part of the final shape (small protrusion from the ear) and some other are not grouped appropriately (small gap between adjacent strokes in the hand). Different from learning based approaches, we do not require any training.

It can be noted that the [5] requires an input rough sketch in vector format and not as a raster image. Also, it requires additional cyclic refinements, once the outputs are generated. Figure 4(d) shows the vectorized result of our procedure for Jackalope input. For the failure case shown in [23], our algorithm captures the features better (House sketch in Figure 20). In general, the sharp corners seem to have been captured better by Bessmeltsev et al. [23].

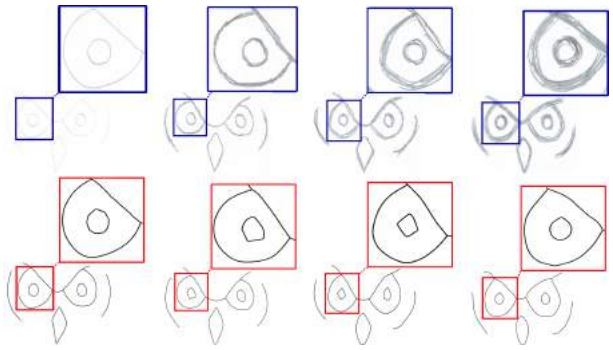


Fig. 24: Experimentation on varying complexity of sketchiness, showing the robustness of our method.

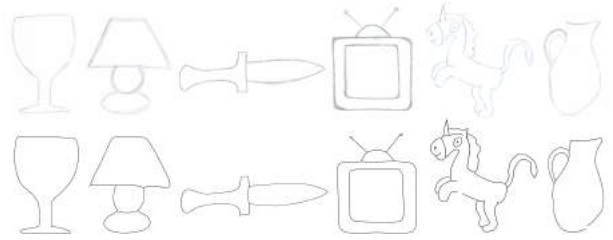


Fig. 25: Result of our algorithm on various sketches drawn during user study.

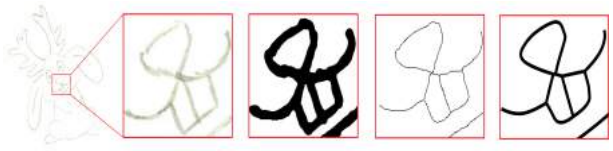


Fig. 26: Limitation: Case where sharp corner detection heuristic fails.

Figure 22(left) shows the result of trapped ball segmentation on two sample sketches for varying ball sizes. Result of our region identification algorithm for the sketches are shown in Figure 22(right). It can be seen that trapped ball segmentation results in small unwanted regions which are captured properly using our method.

We have also compared with other vectorization procedures to know how it differs from the shape aware skeleton smoothing. Figure 23 shows the result of various existing vectorization procedures. They give good vector results, but since our algorithm takes filled shape also under consideration, our results give shape with less zigzags where it is not required.

Since sketchy and thick lines are very challenging [14], we have conducted experiments on varying sketchiness (complexity). Figure 24 shows a few sample sketches with varying sketchiness, It can be seen that our algorithm produces consistent vector drawings irrespective of the complexity of the strokes.

We also conducted a user study in which six artists are asked to draw in multi-stroke fashion. Figure 25 shows the sketches drawn by user along with the vector results generated using our algorithm. The artists were surprised to see our algorithm transforming their sketchy input to an easily editable vector drawing. Also, all the artists agreed that the vector drawing captured all the important features that they kept in their mind while

Fig. 27: Few sketches in which our algorithm has potentially worse outputs along with our results.

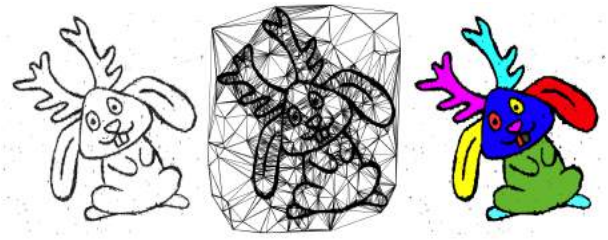


Fig. 28: The effect of outlier points in input sketch. Left to Right: Thresholded sketch with outliers, Its Delaunay triangulation, Our result.

drawing the sketch. Even though we require four parameters (whereas works like [14] requires more than five parameters) for generating a result, most of the parameters are intuitive in nature and hence easy to tune.

**Limitations:** First limitation is that the heuristic for finding sharp corners may not work in some cases. Figure 26 shows a sample in which our heuristic fails to identify sharp corners because of the absence of junction point with open branch. As shown in Figure 27, the second limitation is that there can be regions which has a similar structure of a gap between adjacent strokes which our algorithm may fail to capture. Also, outlier points in the input strokes has great effect on the Delaunay structure and even though it does not effect the region identification in greater scale, the outlier points group among themselves to create unwanted artifacts in the final result. Figure 28 shows an example sketch with outlier points.

## 5. Conclusion & Future Works

We introduced an algorithm for cleaning rough sketches which reduces the gap with artists who prefer ink and paper sketch and skillful vector artists. A Delaunay triangulation based approach is used to group adjacent strokes to identify regions which has the capability to mask regions to avoid misinterpretation. Our algorithm can also handle discontinuous and broken curves. Since it is easy to manipulate curves with lesser control points, we used a shape aware skeleton smoothing to vectorize the results. The user is given with the flexibility to control accuracy and simplicity in our algorithm. Irrespective of input format, pencil sketches or vector drawings, we can use our algorithm which facilitates the use of scanned drawings.

Modifying the shape aware skeleton smoothing step by taking the context of local segment into consideration will be an interesting path to look into. An improved heuristic for restoring sharp corners will increase the accuracy of the resultant vector result. Similarly, developing a method to separate open branches from irregularities made by strokes will increase the

accuracy of our algorithm. Also, a criterion to distinguish regions with shape similar to the gap between adjacent strokes from the gap between adjacent strokes will be useful. The left-overs of vague shapes which artists used for creating initial sketching cannot be handled with the current system (results in unwanted regions) and will be interesting to look into.

## Acknowledgments

The authors would like to thank Jean-Dominique FAVREAU (for running our inputs on their system), Xylia Xueting Liu (for giving permission to reuse results from their paper), Reviewers (for valuable comments to improve the manuscript), Ministry of Human Resource Development, Government of India (for supporting the work).

## References

[1] Preim, B, Strothotte, T. Tuning rendered line-drawings. In: In Proceedings of the 3rd Central European Conference on Computer Graphics. 1995, p. 227–237.

[2] Grabli, S, Durand, F, Sillion, FX. Density measure for line-drawing simplification. In: 12th Pacific Conference on Computer Graphics and Applications, 2004. PG 2004. Proceedings. 2004, p. 309–318. doi:10.1109/PCCGA.2004.1348362.

[3] Wilson, B, Ma, KL. Rendering complexity in computer-generated pen-and-ink illustrations. In: Proceedings of the 3rd International Symposium on Non-photorealistic Animation and Rendering. NPAR '04; New York, NY, USA: ACM. ISBN 1-58113-887-3; 2004, p. 129–137. URL: http://doi.acm.org/10.1145/987657.987674. doi:10.1145/987657.987674.

[4] Deussen, O, Strothotte, T. Computer-generated pen-and-ink illustration of trees. In: Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques. SIGGRAPH '00; New York, NY, USA: ACM Press/Addison-Wesley Publishing Co. ISBN 1-58113-208-5; 2000, p. 13–18. URL: http://dx.doi.org/10.1145/344779.344792. doi:10.1145/344779.344792.

[5] Liu, X, Wong, TT, Heng, PA. Closure-aware sketch simplification. ACM Trans Graph 2015;34(6):168:1–168:10. URL: http://doi.acm.org/10.1145/2816795.2818067. doi:10.1145/2816795.2818067.

[6] Rosin, PL. Grouping curved lines. In: Proceedings of the British Machine Vision Conference. BMVA Press. ISBN 952-1898-1-X; 1994, p. 26.1–26.10. Doi:10.5244/C.8.26.

[7] Lindlbauer, D, Haller, M, Hancock, M, Scott, SD, Stuerzlinger, W. Perceptual grouping: Selection assistance for digital sketching. In: Proceedings of the 2013 ACM International Conference on Interactive Tabletops and Surfaces. ITS '13; New York, NY, USA: ACM. ISBN 978-1-4503-2271-3; 2013, p. 51–60. URL: http://doi.acm.org/10.1145/2512349.2512801. doi:10.1145/2512349.2512801.

[8] Barla, P, Thollot, J, Sillion, FX. Geometric clustering for line drawing simplification. In: ACM SIGGRAPH 2005 Sketches. SIGGRAPH '05; New York, NY, USA: ACM; 2005,URL: http://doi.acm.org/10.1145/1187112.1187227. doi:10.1145/1187112.1187227.

[9] Barla, P, Thollot, J, Sillion, FX. Geometric clustering for line drawing simplification. In: Proceedings of the Sixteenth Eurographics Conference on Rendering Techniques. EGSR '05; Aire-la-Ville, Switzerland, Switzerland: Eurographics Association. ISBN 3-905673-23-1; 2005, p. 183–192. URL: http://dx.doi.org/10.2312/EGWR/EGSR05/183-192. doi:10.2312/EGWR/EGSR05/183-192.

[10] Orbay, G, Kara, LB. Beautification of design sketches using trainable stroke clustering and curve fitting. IEEE Transactions on Visualization and Computer Graphics 2011;17(5):694–708. URL: http://dx.doi.org/10.1109/TVCG.2010.105. doi:10.1109/TVCG.2010.105.

[11] Simo-Serra, E, Iizuka, S, Sasaki, K, Ishikawa, H. Learning to simplify: Fully convolutional networks for rough sketch cleanup. ACM Trans Graph 2016;35(4):121:1–121:11. URL: http://doi.acm.org/10.1145/2897824.2925972. doi:10.1145/2897824.2925972.

[12] Ogawa, T, Matsui, Y, Yamasaki, T, Aizawa, K. Sketch simplification by classifying strokes. In: 2016 23rd International Conference on Pattern Recognition (ICPR). 2016, p. 1065–1070. doi:10.1109/ICPR.2016.7899777.

[13] Noris, G, Skora, D, Shamir, A, Coros, S, Whited, B, Simmons, M, et al. Smart scribbles for sketch segmentation. Computer Graphics Forum 2012;31(8):2516–2527. URL: http://dx.doi.org/10.1111/j.1467-8659.2012.03224.x. doi:10.1111/j.1467-8659.2012.03224.x.

[14] Favreau, JD, Lafarge, F, Bousseau, A. Fidelity vs. simplicity: A global approach to line drawing vectorization. ACM Trans Graph 2016;35(4):120:1–120:10. URL: http://doi.acm.org/10.1145/2897824.2925946. doi:10.1145/2897824.2925946.

[15] Igarashi, T, Matsuoka, S, Kawachiya, S, Tanaka, H. Interactive beautification: A technique for rapid geometric design. In: Proceedings of the 10th Annual ACM Symposium on User Interface Software and Technology. UIST '97; New York, NY, USA: ACM. ISBN 0-89791-881-9; 1997, p. 105–114. URL: http://doi.acm.org/10.1145/263407.263525. doi:10.1145/263407.263525.

[16] Igarashi, T, Hughes, JF. A suggestive interface for 3d drawing. In: Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology. UIST '01; New York, NY, USA: ACM. ISBN 1-58113-438-X; 2001, p. 173–181. URL: http://doi.acm.org/10.1145/502348.502379. doi:10.1145/502348.502379.

[17] Wang, B, Sun, J, Plimmer, B. Exploring sketch beautification techniques. In: Proceedings of the 6th ACM SIGCHI New Zealand Chapter's International Conference on Computer-human Interaction: Making CHI Natural. CHINZ '05; New York, NY, USA: ACM. ISBN 1-59593-036-1; 2005, p. 15–16. URL: http://doi.acm.org/10.1145/1073943.1073946. doi:10.1145/1073943.1073946.

[18] Fišer, J, Asente, P, Sýkora, D. ShipShape: A drawing beautification assistant. In: Proceedings of the Workshop on Sketch-Based Interfaces and Modeling. SBIM '15; Goslar Germany, Germany: Eurographics Association; 2015, p. 49–57. URL: http://dl.acm.org/citation.cfm?id=2810210.2810215.

[19] Noris G., G, Hornung, A, Sumner, RW, Simmons, M, Gross, M. Topology-driven vectorization of clean line drawings. ACM Trans Graph 2013;32(1):4:1–4:11. URL: http://doi.acm.org/10.1145/2421636.2421640. doi:10.1145/2421636.2421640.

[20] Bo, P, Luo, G, Wang, K. A graph-based method for fitting planar b-spline curves with intersections. Journal of Computational Design and Engineering 2016;3(1):14 – 23. URL: http://www.sciencedirect.com/science/article/pii/S2288430015000408. doi:http://dx.doi.org/10.1016/j.jcde.2015.05.001.

[21] Hilaire, X, Tombre, K. Robust and accurate vectorization of line drawings. IEEE Trans Pattern Anal Mach Intell 2006;28(6):890–904. URL: http://dx.doi.org/10.1109/TPAMI.2006.127. doi:10.1109/TPAMI.2006.127.

[22] Bao, B, Fu, H. Vectorizing line drawings with near-constant line width. In: 2012 19th IEEE International Conference on Image Processing. 2012, p. 805–808. doi:10.1109/ICIP.2012.6466982.

[23] Bessmeltsev, M, Solomon, J. Vectorization of line drawings via polyvector fields. CoRR 2018;abs/1801.01922. URL: http://arxiv.org/abs/1801.01922. arXiv:1801.01922.

[24] Donati, L, Cesano, S, Prati, A. A complete hand-drawn sketch vectorization framework. CoRR 2018;abs/1802.05902. URL: http://arxiv.org/abs/1802.05902. arXiv:1802.05902.

[25] Zhang, SH, Chen, T, Zhang, YF, Hu, SM, Martin, RR. Vectorizing cartoon animations. IEEE Transactions on Visualization and Computer Graphics 2009;15(4):618–629. doi:10.1109/TVCG.2009.9.

[26] de Goes, F, CohenSteiner, D, Alliez, P, Desbrun, M. An optimal transport approach to robust reconstruction and simplification of 2d shapes. Computer Graphics Forum 2011;30(5):1593–1602. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2011.02033.x. doi:10.1111/j.1467-8659.2011.02033.x.

[27] Sadri, B, Singh, K. Flow-complex-based shape reconstruction from 3d curves. ACM Trans Graph 2014;33(2):20:1–20:15. URL: http://doi.acm.org/10.1145/2560328. doi:10.1145/2560328.

[28] Zhang, TY, Suen, CY. A fast parallel algorithm for thinning digital patterns. Commun ACM 1984;27(3):236–239. URL: http://doi.acm.org/10.1145/357994.358023. doi:10.1145/357994.358023.

[29] Duckham, M, Kulik, L, Worboys, M, Galton, A. Ef-

ficient generation of simple polygons for characterizing the shape of a set of points in the plane. Pattern Recognition 2008;41(10):3224 – 3236. URL: http://www.sciencedirect.com/science/article/pii/S0031320308001180. doi:https://doi.org/10.1016/j.patcog.2008.03.023.

[30] Iarussi, E, Bommes, D, Bousseau, A. Bendfields: Regularized curvature fields from rough concept sketches. ACM Trans Graph 2015;34(3):24:1–24:16. URL: http://doi.acm.org/10.1145/2710026. doi:10.1145/2710026.