On-line Graphics Recognition

[1]Jin Xiangyu, [2]Liu Wenyin, [1]Sun Jianyong, [1]Zhengxing Sun
[1]State Key Lab for Novel Software Technology,
Nanjing University,
Nanjing, 210093, PR China
xiangyu_jin@yahoo.com

[2]Department of Computer Science,
City University of Hong Kong,
Kong Kong SAR, PR China
csliuwy@cityu.edu.hk

Abstract

A novel and fast shape classification and regularization algorithm for on-line sketchy graphics recognition is proposed. We divided the on-line graphics recognition process into four stages: preprocessing, shape classification, shape fitting, and regularization. The Attraction Force Model is proposed to progressively combine the vertices on the input sketchy stroke and reduce the total number of vertices before the type of shape can be determined. After that, the shape is fitted and gradually rectified to a regular one, thus the regularized shape fits the user-intended one precisely. Experimental results show that this algorithm can yield good recognition precision (averagely above 90%) and fine regularization effect with a fast speed. Consequently, it is especially suitable for weak computation environment such as PDAs, which solely depends on a pen-based user interface.

1. Introduction

Graphics is an important means of expression. The most natural and convenient way to input graphic objects is to draw sketches on a tablet using a pen, just like drawing on a real sheet of paper. Then the system analyzes the user's sketch and shows to the user the most similar but regular shape that is intended by the user. This process is specified as on-line graphics recognition [9], which can be specified as: given a sketchy stroke of a closed-shape, determine the shape that the user intended to input. This problem can be divided into three sub problems.

(1) Shape Classification

Input: the stroke of a closed-shape

Output: a basic shape class: such as triangle, quadrangle, pentagon, hexagon, or ellipse

Requirement: the output shape class is consistent with the user intention

(2) Shape Fitting

Input: the stroke of a closed-shape and its corresponding basic shape class

Output: the fitted shape

Requirement: the fitted shape, which is of the basic shape class, has the lowest average distance to the input stroke

(3) Shape Regularization

Input: the fitted shape

Output: the regularized shape

Requirement: the regularized shape is consistent with the user intention

Figure 1 gives such an example of shape classification, shape fitting, and shape regularization. The black dash line is the original stroke a user inputs. The red real line is the fitted or regularized one.

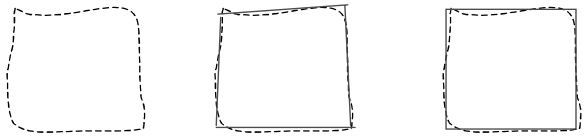


Figure 1. An example of Shape Classification, Fitting, and Regularization: (a) The input stroke of a square, (b) The fitted shape, and (c) The regularized shape.

There are already many works done for the shape classification problem. We mainly classified them into four groups. They are Off-line Approaches, Filter-based Approaches, Energy Minimization Approaches, and

Machine Learning based Approaches. Next, we present brief reviews of these approaches.

(1) Off-line Approaches

Revanka et al. [3] proposed a decision tree based approach for off-line shape classification. First, they extract all the line segments from the image of a hand drawn geometric line sketch. Then, they use intuitive and simple rules to build a decision tree, and each leaf of the tree represents a sub shape category. E.g., if there is a closed chain of three line segments, then it is a triangle. However, this naive approach is somewhat too simple and cannot distinguish visual ambiguity. Although they claim their approach can achieve good performance in an off-line system, we doubt that this approach is unsuitable for on-line shape recognition. This is because it is very difficult to break the stroke into line segments correctly without ambiguity.

(2) Filter-based Approaches

Apte et al. [1] have proposed another way to classify shapes based on filters. However, these filters are sensitive to orientation of the geometric objects. All the shapes, except for circles and lines, should be drawn in parallel with the X-axis and Y-axis. This precondition is somewhat too strict. Fonseca, Jorge, and their research group [2] have extended Apte et al.'s work by adding some new filters. First they calculate some global attributes of the given shape, such as Convex Hull [4], Largest-area Inscribing Triangle, Largest-area Inscribed Quadrilateral [5], and Enclosing Rectangle [6]. Then they form a group of filters based on these attributes to do shape classification, and fuzzy logic is also employed in their algorithm. Compared with Apte et al.'s work, their recognition approach is orientation independent. Instead of outputting a definite shape type, their approach introduced some uncertainty and can output several types in a ranked list. This is reasonable for ambiguous situations in on-line shape classification. However, the filters they used can hardly distinguish very ambiguous shapes such as pentagon and hexagon, and this ad hoc approach is not easily extensible.

(3) Energy Minimization Approaches

The shape classification approach reported by Arvo and Novins [7] continuously morphs the sketchy curve to the guessed shape while the user is drawing the curve. Their recognition approach only handles two simplest classes of shapes (circles and rectangles) drawn in single strokes. Liu et al. [8] proposed another Hypothesis-and-Test approach for shape classification. It is also a model-based approach. They first hypothesize the input shape as a triangle, a rectangle, or an ellipse. Then they calculate the difference between each hypothesized shape and the normalized original sketchy stroke. In order to evaluate all possibilities, each model should be scaled and rotated many times for such comparisons. The energy is the average distance of each vertex of the normalized stroke to the hypothesized shape. The model with the smallest energy is selected as the shape classification result. Unfortunately, the model with smallest energy is only the most similar one to the user sketch, not the most similar to the one that the user has in his/her mind.

(4) Machine Learning based Approaches

In the work reported by Liu et al. [9], two statistical machine-leaning based approaches, which are based on neural networks (NN) and support vector machines (SVM), are employed in shape recognition. Although this approach can yield a good performance (averagely 95.8%) [9], it has high computational cost and not suitable for handled devices, which has low computational power and less memory, such as PDAs.

Unlike the shape classification problem, very few related work has been done for the shape fitting and regularization/rectification problems. Usually these problems are regarded as parts of shape classification problem. In Revanka et al.'s shape classification approach [3], each identified shape is further classified into more detailed shape classes. E.g., a triangle is further classified as right triangle, equilateral triangle, isosceles triangle, or other triangle. As we have illustrated before, this approach is mainly for off-line shape recognition purpose and there is no separated process for shape fitting. The rectification is line-based, hence each rectification error will be propagated and accumulated, and might cause distortion of the overall shape.

In this paper, a novel and fast shape recognition and regularization algorithm is proposed. This algorithm can first classify a sketchy, closed shape into one of the primitive shape types (including triangles, quadrangles, pentangles, hexagons, and ellipses), and then regularize

it to the user-intended ones The whole approach includes the following four sub-processes: Preprocessing, Shape Classification, Shape Fitting, and Shape Regularization, as illustrated in Figure 2.

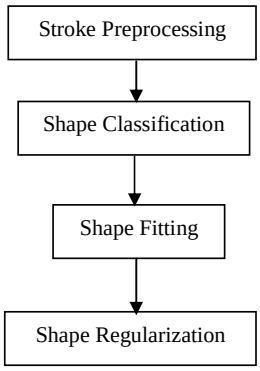


Figure 2. The Proposed Shape Classification and Regularization Approach

These four sub-processes of our proposed algorithm are presented in the next four consecutive sections. Experiments and evaluations are presented in Section 5. Finally, in Section 6, we present our concluding remarks.

2. Preprocessing

Prior to shape classification, pre-processing is done to reduce all kinds of noises from the input stroke, which undergoes the following four processes: Polygonal Approximation, Agglomerate Points Filtering, End Points Refinement, and Convex Hull Calculation.

2.1. Polygonal Approximation

Many intermediate points on the sketchy line of the input stroke are redundant because they lie (approximately) on the straight-line segment formed by connecting their neighbours. These points can be removed from the chain so that the sketchy line can be approximately represented by a polyline (an open polygon) with much fewer critical vertexes. The extent to which the polyline maintains the original shape is controlled by a parameter ε. If the distance of a point to the straight-line segment formed by connecting its neighbours is smaller than ε, this point is non-critical and should be removed from the polyline. As the ε value gets smaller, more vertices are left, but the edge accuracy is improved. Usually ε can be set to be 1-2 pixels to preserve the original shape as much as possible.

We apply the algorithm developed by Sklansky and Gonzalez [10] to implement polygonal approximation in our approach. The complexity of this process is O(n), where n is the number of points. Some results of polygonal approximation are shown in Figure 3. The grey lines are before polygonal approximation and the black lines are the results of polygonal approximation. In order to keep less critical points but preserve the original shape well, we apply two passes polygonal approximation, with ε=1.0 in the first pass and ε=5.0 in the second pass.

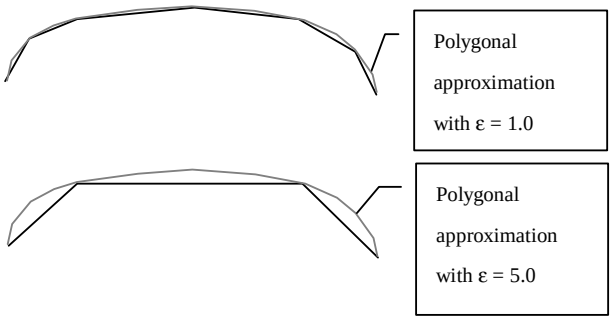


Figure 3. Illustration of the polygonal approximation process.

2.2. Agglomerate Points Filtering

Due to the shaky operations caused when the pen-tip touches the tablet and when it is lifted up, there are often some hooklet-like segments at the ends of the sketchy lines. There might also be some circlets at the turning corners of the sketchy line (cf. Figure 4). These noises usually remain after polygonal approximation. Agglomerate Points Filtering is employed to reduce these noises. Polyline segments, which have a hooklet or circlet, usually have much higher point densities than the average value of the whole polyline. The task of agglomerate points filtering is to find such segments and use fewer points to represent the segment.

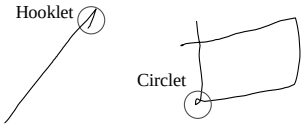


Figure 4. A hooklet and a circlet.

Denote *L* as the length of the whole polyline and *N* as the number of its points, denote *l* as the length of a

polyline segment and n as the number of its point, we define $\eta=(Ln)/(lN)$. Given any part of the polyline, if its η is above a threshold, we replace these points of the segment with their gravity centre. The strategy to select the polyline segment is "shortest first", in order that the change of the polyline could be maintained at the minimum extent. The computational complexity of this algorithm is $O(n^2)$. An example is illustrated in Figure 5.

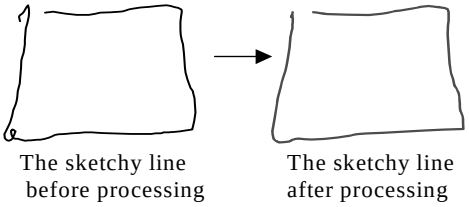


Figure 5. An example of Agglomerate Points Filtering.

2.3. End Points Refinement

Because it is difficult for users to draw perfectly closed shapes, a sketchy stroke is usually not closed or forms a cross near its endpoints (cf. Figure 6). In other words, it has improper endpoints. These improper endpoints are great barriers for both shape classification and regularization. For a self-crossed stroke, we delete its extra points to make it properly closed. For an open stoke, we extend its endpoints along its end directions and make it closed. After that, it can undergo other processing as if it were previously closed (cf. Figure 7).

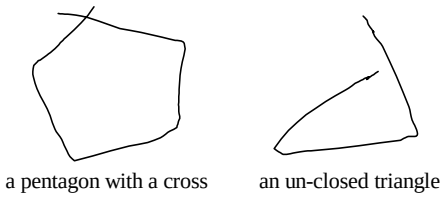


Figure 6. Examples of shapes with improper endpoints.

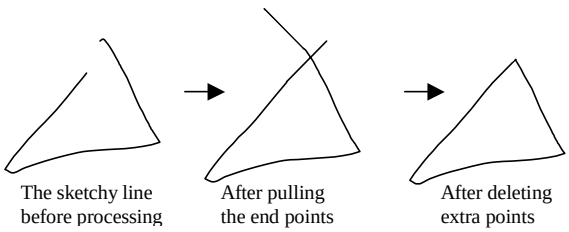


Figure 7. Examples of endpoint refinement.

2.4. Convex Hull Calculation

The sketchy line the user draws is often very cursive, and might also be concave. These noises have strong influence on later-stage processing. We employ the classical algorithm developed by Graham [11] to obtain the convex hull of the polyline, which is used to represent its original line, and therefore remove those noises. The complexity of this process is $O(nlgn)$, where n is the number of points of the stroke.

3. Shape Classification

After pre-processing, the input stroke for a closed-shape is represented as a convex polygon with n vertexes. The number of vertexes can be further reduced using the following approach and the number of remaining vertexes can be used to determine the type of the shape using intuitive rules (e.g., a triangle has three vertexes).

We can select m points from the original n vertexes to form a new m-polygon, so that the average distance of the unselected n-m points to the newly formed m-polygon is minimal. Denote this minimal value as *avd_min*. Obviously, the larger m is, the smaller *avd_min* is. The shape classification problem can then be specified as: Given an n-polygon, try to find a trade-off between m and *avd_min*, such that both m and *avd_min* are small enough. A straightforward and brute-force algorithm for a globally optimal solution to this problem has an exponential complexity. Hence we develop a locally optimal algorithm, which is based on the Attraction Force model, to combine the adjacent vertexes progressively until the maximal attraction force is under a threshold.

The Attraction Force model we proposed is as follows. We consider that a vertex A of a polygon is attracted by its neighbour B. Denote their inner angles (an inner angle of a vertex is the smaller angle between its two neighbouring edges) as α, β respectively, define $Dis(A, B)$ as 20 times normalized spatial distance between A and B, and $f(A, B)$ as the force that A is attracted by B. Obviously, the function $f(A, B)$ should satisfy the following rules:

- *The larger α is, the larger f is.*
- *The larger β is, the smaller f is.*

- *The larger Dis(A, B) is, the smaller f is.*

Therefore, we define:

$$f(A,B)=\frac{\alpha}{\beta Dis^{r}(A,B)},\quad Dis(A,B)=\frac{20\sqrt{(x_A-x_B)^2+(y_A-y_B)^2}}{length\_of\_stroke}$$

where *length_of_stroke* is the length of the entire stroke. The parameter τ is better to be 2 according to our experimental results. For a vertex, it is both attracted by its upper and lower neighbours. If the resultant attraction force is large enough, we combine it to the neighbour with the larger attraction force. The vertex combination process is illustrated in Figure 8.

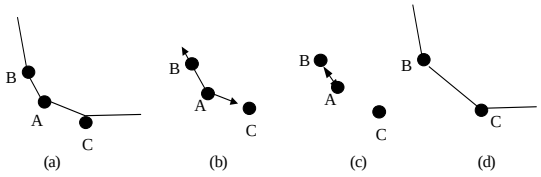


Figure 8. The process of vertex combination: (a) B and C are neighbours of A; (b) A is attracted by both B and C; (c) The resultant attraction force is toward B and larger than the threshold; (d) Combine A to B.

Denote the vertex set as *U*. Define $F(A)= f(A, A_{up\_neighbour}) - f(A, A_{low\_neighbour})$. We progressively combine the vertexes using the following algorithm.

Algorithm 1: Vertex Combination

Step 1. Select a vertex x from U whose |F(x)| is the largest.

Step 2. If |F(x)|< threshold_of_force, then ***Stop***

Step 3. Combine x to its upper neighbor (if F(x)>0) or its lower neighbor (if F(x)<0),

and then adjust the inner angles and edge accordingly.

Step 5. If |U|<4 then, ***Stop***

Step 6. Goto Step 1.

The computational complexity of this algorithm is only $O(n^2)$. Although this algorithm cannot achieve the best solution to this problem, it still yields pretty good results in our experiments. After vertex combination, some vertexes whose inner angles are larger than a given threshold (which is set as 3.0 in this paper) are filtered. Then we can determine the type of the shape from the number of the remaining vertexes, denoted as *V_num*. We classify the closed-shape according to the following intuitive rules:

1. *If V_num<=3 then type=triangle*
2. *If V_num=4 then type=quadrangle*
3. *If V_num=5 then type=pentagon*
4. *If V_num=6 then type=hexagon*
5. *If V_num>6 then* type=*ellipse*

4. Shape Fitting

The fitting process is employed to make the fitting shape similar to the sketchy one. There are independent fitting processes for polygons and ellipses.

4.1. Polygonal Fitting

Firstly, the remaining *n*-polygon after vertex combination (cf. Section 3) divides the original polygon into *n* pieces. Secondly, for each segment (piece), which is a polyline, we make a line (using the method of least squares) that can approximate the sampling points along the polyline segment at equi-length steps. Thirdly, we obtained the vertexes of the fitting polygon by calculating the intersection points of all segments' fitting lines. Finally, we adjust the position of these new vertexes so that their distance to their original stroke is no larger than a threshold. This process is illustrated in Figure 9.

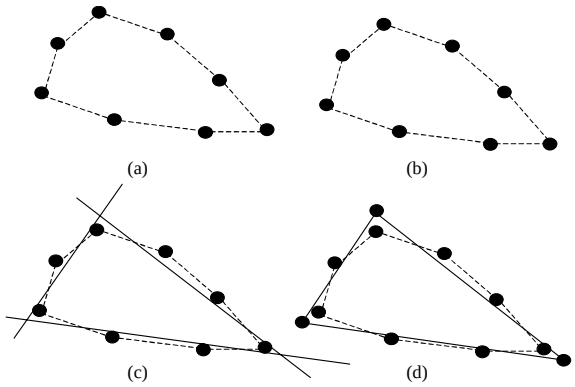


Figure 9. Polygon fitting: (a) The original polygon of a triangle; (b) After vertex combination, the remaining vertices, which are shown in thick black divide the stroke into 3 pieces; (c) Linear fitting for each piece; (d) Recalculating the vertexes and get the fitting triangle.

4.2. Ellipse Fitting

The parameters of an ellipse are its axis orientations, centre point, and axis lengths. First we determine its axis orientations by finding the eigenvectors of the

covariance matrix of the sampling points along the stroke at equi-length steps. We then obtain its centre point by calculating the gravity centre of all these sampling points, as illustrated in Figure 10(a). Secondly, we do coordinate transformation so that the centre of the ellipse becomes the origin of the new coordinate system and the two axes become x-axis and y-axis, cf. Figure 10(b). Thus the equation of the ellipse can be written as:

$$\frac{x^2}{a^2}+\frac{y^2}{b^2}=1\cdot$$

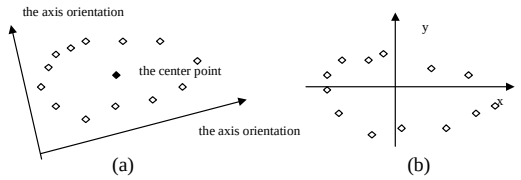


Figure 10. Finding the centre and axes of the ellipse.

Finally, we calculate the length of each axis. Denote the sampling points (in the new coordinate system) as (x_1, y_1), (x_2, y_2), …(x_n, y_n). We want to get a and b such that $\sum_{i=1}^{n}\left(\frac{x_i^2}{a^2}+\frac{y_i^2}{b^2}-1\right)^2$ gets its minimal value. To simply this problem, we convert it to a similar line fitting problem which can be specified as: Given n points (X_1, Y_1), (X_2, Y_2), …(X_n, Y_n), where $X_i=x_i^2$,$Y_i= y_i^2$, find a and b such that the line $Y = b^2 - \frac{b^2}{a^2}X$ fits these points best. Using the similar way we presented before we can get the value of a and b. The complexity of this fitting process is only $O(n)$.

5. Shape Regularization

The input shape that the user has drawn cannot precisely match the one that he/her really intends to input. The shape should undergo rectification such that the shape becomes a regular one and looks very similar to the one that the user has in his/her mind. Our approach is to rectify the fitted shapes to the most regular one, so that the overall distortion is under a given range. This process is currently rule based, and was divided into two sub-processes: Inner-shape Regularization and Inter-shape Regularization. Inner-shape regularization adjusts shapes only according to its own information. E.g., if a triangle is very similar to an isosceles triangle, it is then rectified into an isosceles one, as illustrated in Figure 11(a). Inter-shape regularization adjusts shapes according to its nearby shapes. E.g., if two adjacent rectangles are about the same size, they are then rectified into the same size. This is illustrated in Figure 11(b). The thinner (black) line is the fitted shape, and the thicker (red) line is the regularized one.

(a) (b)

Figure 11. Examples of shape regularization: (a) rectifying a triangle into an isosceles triangle and (b) rectifying two rectangles into the same size.

5.1. Inner-shape Regularization

Inner-shape regularization includes the following rectification processes.

(1) Equilateral Rectification

Adjust the edges of polygons so that their lengths are equal if they are nearly so. Adjust the A-axis and B-axis of so that their lengths are equal if they are nearly so.

(2) Parallelism Rectification

Adjust the edges of polygons to parallel if they are nearly so.

(3) Special Angle Rectification

Rectify the inner angles of polygons if they are nearly 90º. If one side of an inner angle of a polygon is horizontal or vertical, the angle will be rectified to 30º (150º), 45º(135º) or 60º(120º) if they are nearly so.

(4) Horizontal/Vertical Rectification

Rectify the edges of a polygon to horizontal or vertical if they are nearly so. Rectify the axes of an ellipses to horizontal or vertical if they are nearly so. Rectify the diagonals of a diamond to horizontal or vertical if they are nearly so.

We employed the same approach as Revankar et al.'s [3] to judge if two line segments L_1, L_2 are equilateral or parallel. Equilateral, parallelism, and special angle rectifications are first done according to a group of prioritized rules. Then horizontal/vertical rectification is undertaken, still preserving the specific shape type obtained before. Now, the only question remains for inner shape regularization is how to define the priority

ordering of rules. The rules we have used are illustrated in Figure 12. The closed-shape is gradually rectified to a regular shape following the arrowheads in Figure 13, with rules on each node.

Rules for Triangle Regularization:

T1. Isosceles Triangle: *Two edges are approximately equally long.*
T2. Equilateral Triangle: *All edges are approximately equally long.*
T3. Right-angled Triangle: *One inner angle is around 90º.*
T4. Right-Isosceles Triangle: *Satisfy both T1 and T3.*
T5. 30º-right-angled Triangle: *Satisfy T3 and an inner angle is about 30º/60º.*

Rules for Quadrangle Regularization:

Q1. Trapezoid: *Only one pair of opposite sides are approximately parallel.*
Q2. Parallelogram: *Both pairs of opposite sides are approximately parallel.*
Q3. Diamond: *Satisfy Q2 and its consecutive sides are approximately equal.*
Q4. Rectangle: *Satisfy Q21and with one 90º inner angle.*
Q5. Isosceles Trapezoid: *Satisfy Q1 and the nonparallel pair of opposite sides are approximately equal.*
Q6. Right-Trapezium: *Satisfy Q1 and an inner angle is 90º.*
Q7. Square: *Satisfy Q4 and Q3.*

Rules for Ellipse Regularization:

C1.Circle: *Two Axes are approximately equally long.*

Rules for Pentagon Regularization:

P1. Correct-Pentagon: *All edges are approximately equal.*

Rules for Hexagon Regularization:

H1. Correct-Hexagon: *All edges are approximately equal.*

Figure 12. Rules for inner-shape regularization.

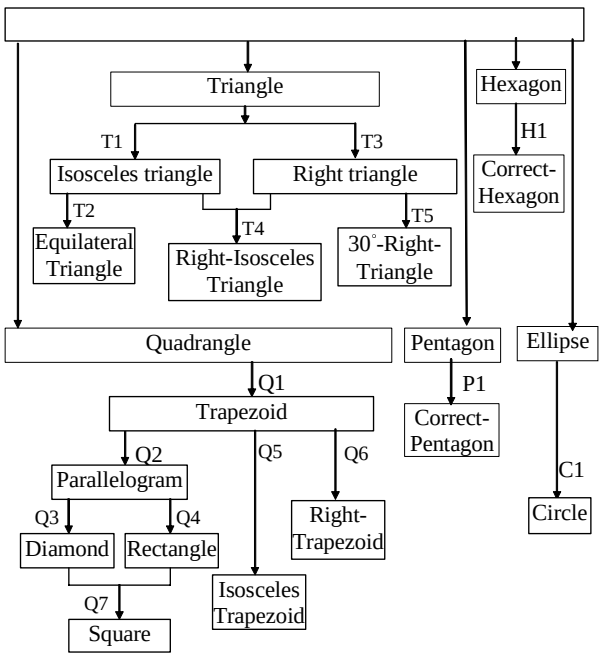


Figure 13. Rule priorities for inner-shape regularization.

5.2. Inter-shape Regularization

Inter-shape regularization includes the following rectification processes.

(1) Size Rectification

If a group of adjacent primitive shapes are of the same type (the detailed type derived from inner-shape-regularization, e.g., a square) and are approximately of the same size, then scale them so that they are of the same size. This is illustrated in Figure 14(a).

(2) Position Rectification

If the edges of two adjacent polygons are nearly on the same horizontal/vertical line, then shift the two shapes so that they are on the same line. This is illustrated in Figure 14(a).

(3) Critical Points Rectification

The centers of ellipses, the vertexes of polygons, and the mid points of edges are called critical points. If two critical points of adjacent shapes are approximately of the same position, then rotate, shift, and/or scale the adjacent shapes when necessary, such that the two critical points can coincide (are of the same position). This processing should preserve the specific shape type obtained before. This is illustrated in Figure 14(b) and (c).

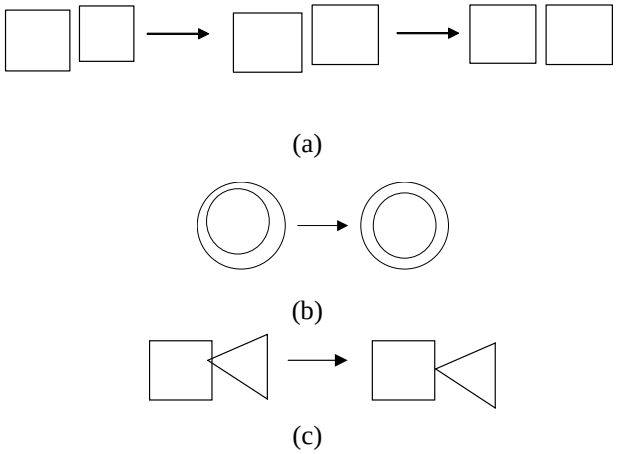


Figure 14. Inter-shape Regularization: (a) Scale the two adjacent squares so that they are of the same size and further align them; (b) Shift the two adjacent circles so that their centres are of the same position; (c) Shift the triangle and the square so that its leftmost vertex is on the centre of the rightmost edge of the square.

6. Experimental Results and Conclusions

6.1. Shape Classification Results

In our experiments, we asked five different subjects to draw closed shapes. There are 1367 sketchy shapes in total, including 156 ellipses, 360 triangles, 453 quadrangles, 240 pentagons, and 158 hexagons. Some shapes are very ambiguous, especially those pentagons and hexagons. We test the shape classification precision according to different thresholds of Force (which is set to be 0.2, 0.35, 0.5, 0.65, 0.75, 0.85, 1.0, and 1.5) and their average processing time (including pre-processing, classification, fitting, and regularization). All experiments are done on an Intel PIII 650 PC (with 128MB memory) running Microsoft Windows 2000.

The experimental result for User 1 is shown in Figure 15. As we have expected, the shape classification precision is relevant to the threshold. If we adjust the threshold smaller, which means it is easier for the vertexes to be combined, the classification precision will increase for small-vertex-number shape type, such as triangle, and will decrease for large-vertex-number shape type, such as ellipse. On the contrary, if we adjust the threshold larger, the classification precision will decrease for small-vertex-number shape type and will increase for large-vertex-number shape type. The average precision curve reaches its peak (>90%) at 0.75. The optimal shape classification precision and total processing time for each shape type at the best threshold of Force (0.75) is illustrated in Table 1, which shows that our approach can achieve good performance, with precision averagely being lager than 90%, with a low processing complexity (in a very short time). If we only include ellipses, triangles, and quadrangles in this experiment, the average precision will achieve 95.3%.

Table 1. Shape classification precision and processing time.

Shape Type	Precision (%)	Processing Time (ms)
Ellipse	96.2	28
Triangle	95.4	26
Quadrangle	92.8	32
Pentagon	83.8	41
Hexagon	83.3	47
Average	91.2	33

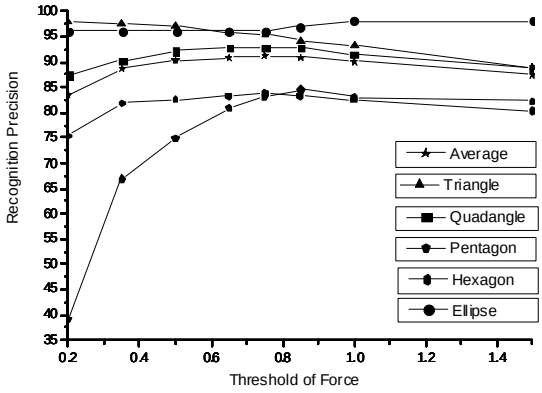
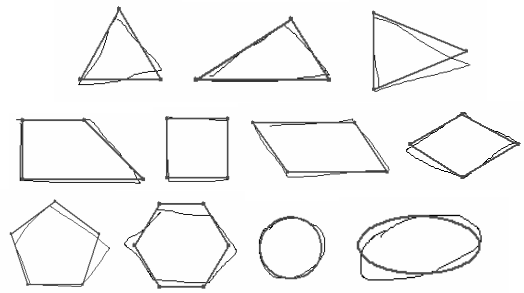


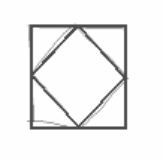
Figure 15. The shape classification precision is dependent on the threshold.

6.2. Shape Regularization Results

Figure 16(a) shows some results of inner-shape regularization. Figure 16(b) shows a result of inner- and inter- shape regularization. The original sketchy strokes are displayed in black (thinner) lines (for human vision evaluation) and the recognized graphics are displayed in red (thicker) lines. As we can see, the regularization effect is quite good.



(a)



(b)

Figure 16. Shape regularization results: (a) Inner-shape regularization results; (b) An Inner- and inter- shape regularization result.

7. Acknowledgement

The work described in this paper was partially supported by a grant from National Natural Science Foundation of China (Project No. 69903006) and by a grant from City University of Hong Kong (Project No. 7100247)

References

[1] Apte A, Vo V and Kimura TD, “Recognizing Multistroke Geometric Shapes: An Experimental Evaluation”, Proc. UIST'93, Atlanta, pp.121-128, 1993.

[2] Fonseca MJ and Jorge JA, “Using Fuzzy Logic to Recognize Geometric Shapes Interactively”, Proc. 9th IEEE Conf. on Fuzzy Systems, Vol. 1, pp.291-296, 2000.

[3] Revankar S and Yegnanarayana B, “Machine Recognition and Correction of Freehand Geometric Line Sketches”, Proc. of IEEE International Conference on Systems, Man, and Cybernetics, Vol. 1, pp.87 – 92, 1991.

[4] Joseph OR, “Computational Geometry”, Cambridge Univ. Press, 2nd edition, 1998.

[5] Boyce JE and Dobkin DP, Drysdale RL, and Guibas LJ, “Finding Extremal Polygons”, SIAM Journal on Computing, 14(1), pp.134-147, Feb. 1985.

[6] Freeman H and Shapira R, “Determining the Minimum-area Enclosing Rectangle for an Arbitrary Closed Curve”, Communication of the ACM, 18(7), pp.409-413, July 1975.

[7] Arvo J and Novins K, “Fluid Sketches: Continuous Recognition and Morphing of Simple Hand-Drawn Shapes”, Proc. 13th Annual ACM Symposium on User Interface Software and Technology, San Diego, California, November, 2000.

[8] Liu W, Qian W, Xiao R, and Jin X, “Smart Sketchpad—An On-line Graphics Recognition System”, Proc. ICDAR2001, pp. 1050-1054, Seattle, September 2001.

[9] Liu W, Jin X, Qian W, and Sun Z, “Inputting Composite Graphic Objects by Sketching A Few Constituent Simple Shapes”, Proc. GREC2001, pp.73-84, Kingston, Canada, 2001.

[10] Sklansky J and Gonzalez V, “Fast Polygonal Approximation of Digitized Curves”, Pattern Recognition, 12, pp. 327-331, 1980.

[11] Graham RL, “An Efficient Algorithm for Determining the Convex hull of A Finite Planar Set”, Information Processing Letters, 1(4), pp. 132-133, 1972.