EG UK Theory and Practice of Computer Graphics (2006)
M. McDerby, L. Lever (Editors)

Curvature-Based Segmentation for Sketch Understanding

Wentao Zheng, Zhiyu Liu and Zhengxing Sun

State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, China

Abstract

Segmentation is known as an important and fundamental task in sketch understanding, by which free hand drawing is processed into primitive geometric units such as line segments, arcs, etc. Usually, two types of information are used in segmentation: drawing speed and curvature. This paper first analyzes the necessity of drawing speed and curvature, then concludes that curvature is better for segmentation, and drawing speed is harmful to later steps in sketch understanding. To provide a fast and robust method for segmentation, we device a window-controlled relative curvature calculation method which is immune to ink noise. Based on the method, our segmentation approach adopts a two-step strategy which shows satisfying experimental results.

Categories and Subject Descriptors (according to ACM CCS): D.2.2 [Design Tools and Techniques]: User interfaces H.5.2 [User Interface]: Input devices and strategies I.3.6 [Computer Graphics]: Interaction techniques J.6 [Computer Aided Engineering]: CAD

1. Introduction

Sketching with pen and paper is widely used in designing works. This is mainly because sketching is a natural, convenient way that a person expresses his intention. It simplifies conceptual design activities through abstract models and let designers focus on critical issues rather than intricate details. During the past years, some systems, such as [HL00], [LM01], [LNHL01], [LLG*03], have been proposed to provide users sketch-based interfaces for conceptual designing works. However, limitations still exist, such as the constrains of drawing habit these systems have made to users.

The difficulty of providing a natural sketch-based interface is handling the ambituity of sketch, e.g. a rectangle can be drawn by different styles (Figure 1). To resolve the ambiguity of sketch, the process of sketch understanding is usually divided into two phases: early processing and parsing [GKSS05] [KS04] [SSD01]. Early processing focuses on translating messy free hand drawing into certain kinds of primitive geometric units, such as line segments, arcs, and ovals etc. Parsing focuses on recognizing composite symbols from primitive geometric units.

Early processing usually contains the following steps:

- Ink sampling: sampling users' drawing

Figure 1: *Two types of rectangles*

- Noise reduction: reducing or eliminating noises in users' drawing.
- Ink segmentation: segmenting users' drawing into several fragments.
- Primitive symbols generation: converting segmented fragments into primitive symbols.

The major task in early processing is ink segmentation. During the past years, researchers have been working on this issue, and provide some practical solutions, such as [SSD01], [Sta04], [CD04]. Till now, solutions for parsing are still complex and fragile in practical use, which demands that ink segmentation should be solved in an easier, faster and robuster way.

This paper introduces a new method for ink segmentation in sketch understanding. We first analyze the necessity of using drawing speed and curvature in segmentation, and conclude that segmentation with drawing speed would have bad

effect on later steps in sketch understanding. In order to use curvature as a key to accomplish segmentation, a window-controlled method for calculating relative curvature is proposed, which is immune to ink noise. Based on the method, a two-step strategy is introduced as a solution to ink segmentation. The effect of our approach is shown in experiments.

Comparing to previous segmentation methods, our approach has the following advantages: (1) geometric characteristic (curvature) is used as a key to accomplish segmentation task, imitating human's perception of sketch; (2) our segmentation algorithm is noise immune; (3) the approach we proposed can be adopted into many situations, including those without drawing time sampling equipment; (4) our approach is easy to implement with low time complexity ($\Theta(n)$).

The rest of this paper is organized as follows: in Section 2, we first go through some related work done in sketch understanding; in Section 3, we describe the task of ink segmentation; in Section 4, we give an analysis on the necessity of drawing speed and curvature in segmentation; in Section 5, we introduce our approach of curvature-based segmentation; in Section 6, we show experimental results and analyze our method's advantages and weaknesses; Section 7 gives a conclusion to this paper and presents some future works.

2. Related Work

An earlier work in sketch understanding is done by Stahovich [Sta97]. He presents a program called SKETCHIT, that transforms a single sketch of a mechanical device into multiple families of new designs. To "interpret" a sketch, the program first determines how the sketched device should work, then derives constraints on the geometry to ensure it works that way. The program is based on qualitative configuration space (qc-space), a novel representation that captures mechanical behavior while abstracting away the particular geometry used to depict this behavior. The program employs a paradigm of abstraction and resynthesis: it abstracts the initial sketch into qc-space then maps from qc-space to new geometries.

Later then, Landay and Myers [LM01] presents another sketch-based designing system named as SILK. It is an informal sketching tool that combines many of the benefits of paper-based sketching with the merits of current electronic tools. With SILK, designers can quickly sketch an interface using electronic pad and stylus, and SILK recognize widgets and other interface elements as the designer draws them. Four primitive components — rectangle, squiggly line, straight line, and ellipse — can be recognized by their system.

It is the first time that Sezgin *et al.* [SSD01] emphasize the importance of early processing in sketch understanding. It introduces a system capable of using multiple sources of information, including drawing speed and curvature, to produce good approximations of freehand sketches. Users can sketch on an input device as if drawing on paper and have the computer detect the low level geometry, enabling a more natural interaction with the computer, as a first step toward more natural user interfaces generally, and toward earlier use of automated tools in the design cycle in particular.

Stahovich [Sta04] presents a technique for segmenting pen strokes into lines and arcs. The technique uses pen speed information to help infer the segmentation intended by the drawer. To begin, a set of candidate segment points is identified. This set includes speed minima below a threshold computed from the average pen speed. It also includes curvature maxima at which the pen speed is again below a threshold. The ink between each pair of consecutive segment points is then classified as either a line or arc, depending on which fits best. Finally, a feedback process is employed, and segments are judiciously merged and split as necessary to improve the quality of the segmentation. Formal user studies were conducted, and the system was observed to perform accurately, even for new users.

Cates and Davis [CD04] propose a graphical model based approach to early sketch processing. Small areas of a sketch corresponding to features such as corners and straight segments are considered individually, and a likely labeling for such features is found by incorporating some context in order to improve on labels computed with only local information. Results from applying their approach to the problem of detecting corners show an improvement.

Based on early processing, some high-level works have also been done to provide flexible systems for multi-domain sketch understanding. Alvarado *el al.* [AOD02] present an architecture to support the development of robust recognition systems across multiple domains. Their architecture maintains a separation between low-level shape information and high-level domain-specific context information, but uses the two sources of information together to improve recognition accuracy.

Hammond and Davis [HD03] create LADDER, the first language to describe how sketched diagrams in a domain are drawn, displayed, and edited. The language consists of predefined shapes, constraints, editing behaviors, and display methods, as well as a syntax for specifying a domain description sketch grammar and extending the language, ensuring that shapes and shape groups from many domains can be described. Shape groups describe how multiple domain shapes interact and can provide the sketch recognition system with information to be used in top-down recognition. Shape groups can also be used to describe "chain-reaction" editing commands that effect multiple shapes at once.

3. Task of Segmentation

Before describing the details of our work, we should introduce the definitions of some important concepts in sketch understanding.

Definition 1 A **stroke** S refers to a sequence of sampled points $p_1, p_2, \ldots, p_n$ between succesive pen-down and pen-up events.

Free hand drawing is usually sampled as a number of strokes.

Definition 2 The **segmentation** of a stroke $S = \{p_1, p_2, \ldots, p_n\}$ is a procedure `SEG` in which the points sequence $\{p_1, \ldots, p_n\}$ is divided into several fragments at some points $p_{f_1}, p_{f_2}, p_{f_k} (1 < f_i < n)$.

Definition 3 The **Feature points** refer to $p_{f_1}, p_{f_2}, p_{f_k} (1 < f_i < n)$ in Definition 2.

A general method of segmentation can be described in Figure 2. The fundamental problem in segmentation is how to find out the feature points where the stroke should be segmented.

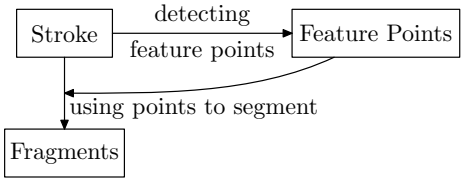


Figure 2: *Process of segmentation*

4. Curvature or Speed

We have mentioned that the major problem in segmentation is how to find out the feature points. To solve this problem, two types of information are frequently used: drawing speed and curvature.

Using drawing speed to find feature points is based on the observation that people usually slow down their drawing speed when drawing through turning area, and keep a constant speed when drawing through straight area. The works in [SSD01] and [Sta04] both use drawing speed information to find feature points. Using curvature to find feature points is based on the observation that points with larger curvature usually separate the ink into several fragments.

Drawing speed represents temporal information of sketch, curvature represents spatial information of sketch. Feature points can be determined by either of them, or both of them. However, which of them is more important to sketch understanding? Do we need both of them to detect feature points? To answer these questions, we should first look into the process of sketch understanding.

Sketch understanding is known as a process that makes computer recognize all the information contained in sketch. The information in sketch is usually expressed by a number of symbols with particular shapes which make them easily distinguished by human. A person could understand the meaning of sketch because he recognizes all the contained symbols. And the reason why he is able to recognize contained symbols is because he knows the geometric structures of them. Therefore, to make computers understand sketch as human do, we consider curvature to be more important than drawing speed, because curvature has more to do with the geometric structure of a symbol than drawing speed does.

In addition, some other reasons also support our choice of using curvature in detecting feature points:

- Drawing speed is unreliable. It could vary tremendously when drawing a straight line. Even in the situation of drawing a right angle, the drawing speed may stay unchanged. Another factor that makes drawing speed unreliable is different drawing habit of different people.
  Figure 3 shows some situations when speed-based segmentation fails to find feature points correctly. The left graph shows an example of losing a feature point; the right graph shows an example of extra feature points found.

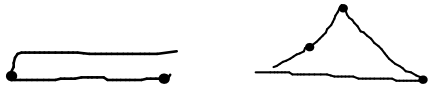


Figure 3: *Failures of using drawing speed to find feature points*

- Using drawing speed is less adaptable than using curvature. To calculate the drawing speed at one point, we should record the time when this point is being drawn. It makes speed-based segmentation method only usable in only on-line sketch understanding, where ink is sampled instantly. While curvature-based segmentation can be used in not only on-line sketch understanding, but also off-line sketch understanding.

The challenge in curvature-based segmentation is how to eliminate noises in strokes. Some works such as [SSD01] suggest using both drawing speed and curvature in segmentation, then merging the results of speed-based segmentation and curvature-based segmentation. In our approach, only curvature is used because we have developed a relative curvature calculation which can eliminate the negative effect of noises.

5. Approach

In this section, we introduce our curvature-based method for ink segmentation.

5.1. Noise-Immune Relative Curvature

In curvature-based segmentation, how to calculate the curvature of a sampled point is the most fundamental work.

In sketch understanding, as a result of the complexity of parsing, algorithms in segmentation should be fast, robust, and effective. Therefore, curvature calculation should possess the following characteristics:

- Fast: it works with low time complexity.
- Effective: it is able to distinguish feature points from normal points easily.
- Robust: ink noise should have little effect on the result of curvature calculation.

We device a window-controlled relative curvature calculation fomula which meet the forementioned three requirements. Given a stroke $S = \{p_1, p_2, \ldots, p_n\}$, our formula can calculate the relative curvature at each point $p_i (1 \leq i \leq n)$ by giving a parameter k:

$$\rho_r(p_i, k) = \frac{\sum_{i-k}^{i+k} \mathrm{Dis}(p_j, l_{i-k,i+k})}{\mathrm{Len}(l_{i-k,i+k})} \tag{1}$$

In the formula 1, k refers to window size, $l_{i,j}$ refers to the line segment between points p_i and p_j, the geometric meaning of functions Dis and Len are shown in Figure 4.

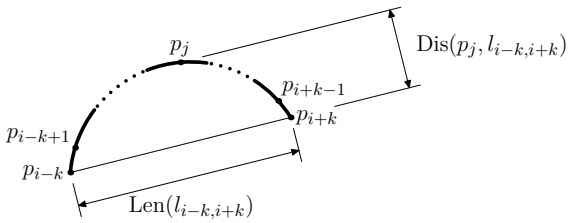


Figure 4: *Illustration of formula* ρ_r

Figure 5 illustrates the comparisons between our relative curvature and approximate curvature for some strokes (arrows indicate where these strokes start). In each subgraph (Figure 5(a), 5(b), 5(c)), the left graph is relative curvature for each point p_i calculated by formula 1 with $k = 3$, the right graph is approximate curvature calculated by MATLAB.

From the comparisons, we conclude that:

- The time complexity of calculating $\rho_r(p_i, k)$ for a stroke is about $\Theta(k \cdot n)$. If k is unrelated to n, the time complexity is linear. Apparently, our window-controlled relative curvature calculation is fast.
- In the turning area, $\rho_r(p_i, k)$ is relative larger than in the straight area, which ensures the determination of feature points. Therefore, our window-controlled relative curvature calculation is effective.
- From the graphs of approximate curvature, we see that ink noise will effect the value of curvature, which results too much feature points detected. However, in the graphs of $\rho_r(p_i, k)$, noise is smoothed out. It shows that our window-controlled relative curvature calculation is robust.

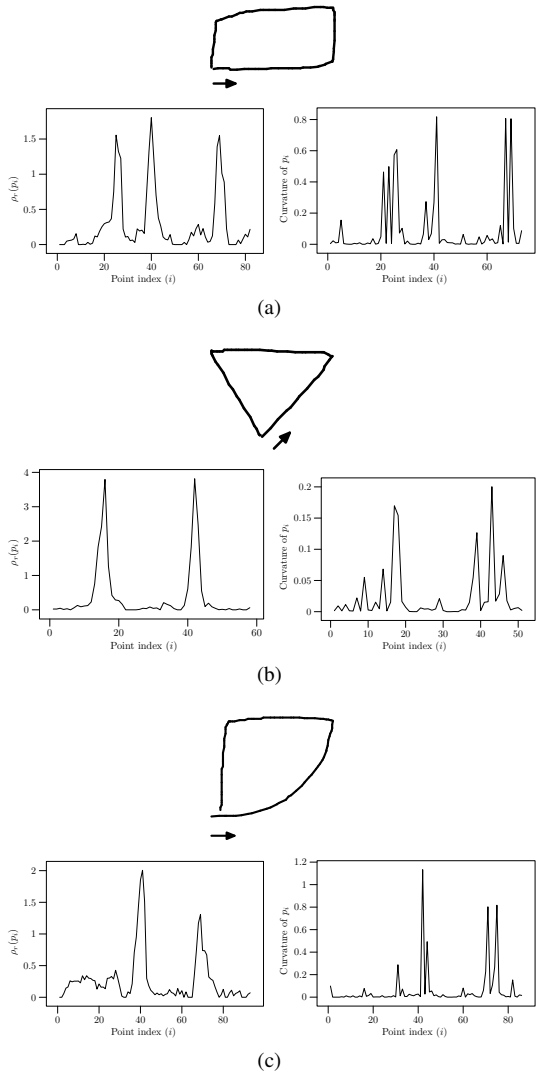


Figure 5: *Comparison between relative curvature (with* $k = 3$*) and approximate curvature*

5.2. Finding Feature Points

With relative curvature ρ_r, we can find feature points by specifying a threshold h_k (k is the window size). In Figure 6, the points whose ρ_r is larger than threshold are considered as feature points.

The algorithm of finding feature points is described in Algorithm 1. For convenience, it is named as procedure `FindFP`, satisfying $P = \texttt{FindFP}(S, k, h_k)$, in which P is the set of feature points, $S = \{p_1, p_2 \ldots, p_n\}$. The procedure from line 1.7 to line 1.11 ensures that if a sequence of continuous points from p_u to p_v are all feature points, only the central point $p_{[(u+v)/2]}$ is treated as feature point. The reason is obvious: after specifying threshold h_k, there could be

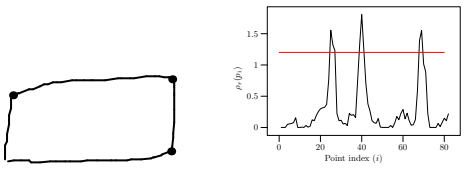


Figure 6: *Filtering feature points by specifying a threshold (red line)*

more than one point whose ρ_r exceeds h_k at the area where the segmentation should be operated. An example is shown in Figure 7.

input : Sampled points $\{p_1, p_2, \ldots, p_n\}$, window size k, threshold h_k
output: Feature points set P

$P \leftarrow \phi$;
$P_c \leftarrow \phi$;
for *each* $p_i (n-k < i < k)$ *in* S **do**
  if $\rho(p_i) > h_k$ **then**
    $P_c \leftarrow P_c \cup \{p_i\}$;
  else
    if $P_c \neq \phi$ **then**
      now $P_c = \{p_u, \ldots, p_v\}$;
      $P \leftarrow P \cup \{p_{[(u+v)/2]}\}$;
      $P_c \leftarrow \phi$;
    end
  end
end

Algorithm 1: Finding feature points in a stroke

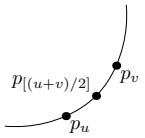
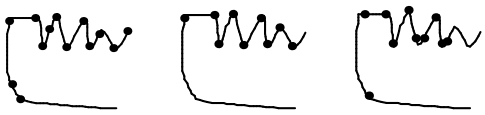


Figure 7: *Points satisfying $\rho_r > h_k$ at a segmented area*

In function $P = \texttt{FindFP}(S, k, h_k)$, the output feature points set P depends on two parameters: window size k, and threshold h_k. We analyze how k and h_k effect the output P.

- The window size k determines how many points are included to calculate the relative curvature of a point p_i. The $2 \times k$ points (including former k points and later k points) compose a context where $\rho_r(p_i, k)$ is calculated. If $k = 1$, then some noisy points may have bad effect on `FindFP` (see Figure 8(a)); If k is very large, some strokes such as Figure 8(c) can not be segmented correctly, because the relative curvature is based on a large context which is not sensitive to small area.



(a) $k = 1$ (b) $k = 3$ (c) $k = 9$

Figure 8: *How k effects `FindFP`*

- The threshold h_k is used to determine which point should be chosen as a feature point. To make the result effective, h_k should be assigned according to window size k. In experiments, we have found that the average value of $\rho_r(p_i, k)$ of a stroke becomes larger when k grows. Therefore, if h_k is not assigned large as k grows, the result feature points may be more than we expect. Figure 9 shows how k effects the average value of $\rho_r(p_i, k)$.

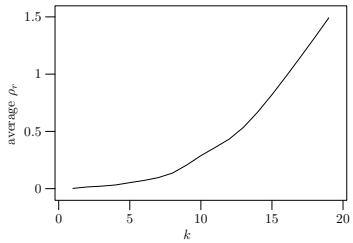


Figure 9: *How k effects average value of $\rho_r(p_i, k)$*

From the above analysis, we find that function $P = \texttt{FindFP}(S, k, h_k)$ has the following properties:

- If k is small, the returning P contains points at relative sharp angles; if k is large, the returning P contains points at round angle.
- To find out feature point effectively, h_k should be assigned according to the value of k. If k grows larger, h_k should be assigned larger.

5.3. Segmentation

With the feature points set $P = \texttt{FindFP}(S, k, h_k)$, we can segment stroke S into $|P| + 1$ fragments by the points in P. This method is called **one-step segmentation**. However, based on the properties of `FindFP`, we conclude there is not such a window size k and its threshold h_k that enable all feature points be found. If k is assigned small, we could lost points at round angle; if k is assigned large, we could lost those at sharp angle. A practical compromise is doing segmentation twice, the first time with a smaller k and the second time with a larger k. The process of our segmentation strategy is described in Algorithm 2. We first use a smaller threshold k_1 to find all the feature points at shape angles, and use these points to separate the stroke into several candidate fragments; these candidate fragments are then re-segmented

using a larger threshold k_2 to find the feature points at round angles.

input : A stroke $S = \{p_1, p_2, \ldots, p_n\}$, window size k_1, k_2, threshold h_{k_1}, h_{k_2}
output: Segmented fragments $F = \{f_1, f_2, \ldots, f_m\}$

2.1 $F \leftarrow \phi$;
2.2 $P \leftarrow \texttt{FindFP}(S, k_1, h_{k_1})$;
2.3 *use P to segment S into several fragments* $F_0 = \{S_1, \ldots, S_t\}$;
2.4 **for** *each* S_i *in* F_0 **do**
2.5 $\quad P_i \leftarrow \texttt{FindFP}(S_i, k_2, h_{k_2})$;
2.6 $\quad$ *use* P_i *to segment* S_i *into several fragments* F_{1i};
2.7 $\quad F \leftarrow F \cup F_{1i}$;
2.8 **end**

Algorithm 2: Two-step segmentation

6. Experiments

We have tested three kinds of curvature-based segmentation on 9 hand drawn strokes (these 9 strokes are frequently used in many works on segmentation):

- the first experiment uses one-step method with $k = 3, h_k = 0.6$.
- the second experiment uses one-step method with $k = 9, h_k = 3$.
- the third experiment uses two-step method with $k_1 = 3, h_{k_1} = 0.6$ and $k_2 = 9, h_{k_2} = 3$.

We also introduce two statistic criteria to evaluate segmentation results:

- precision: the proportion between the number of expected feature points found and the number of all feature points found.
- recall: the proportion between the number of expected feature points found and the number of expected feature points.

Figure 10 shows a set of free hand drawing sketches, in which expected feature points are marked as a small oval manually. Figure 11–13 show the results of forementioned experiments, in which feature points are marked as red point. Table 1 gives a statistical analysis on the experimental results.

k_1	h_{k_1}	k_2	h_{k_2}	precision	recall
3	0.6	–	–	96.1% (73/76)	97.3% (73/75)
9	3	–	–	80.0% (44/62)	41.3% (31/75)
3	0.6	9	3	94.9% (75/79)	100% (75/75)

Table 1: *Statistical analysis*

From the figures and the table, we find that:

- Our two-step segmentation method do find out all the expected feature points (100% recall)

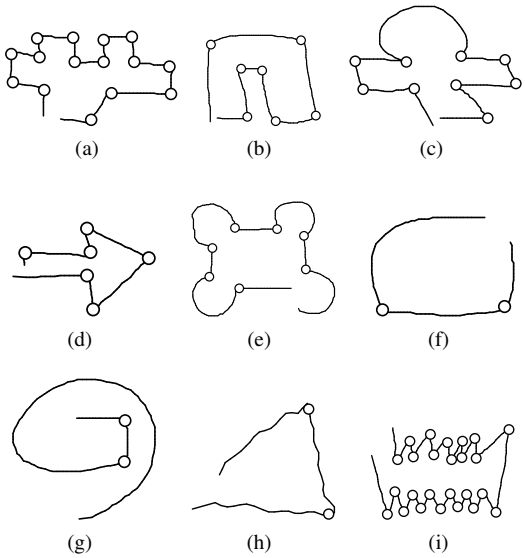


Figure 10: *Hand drawn strokes with marked 75 feature points*

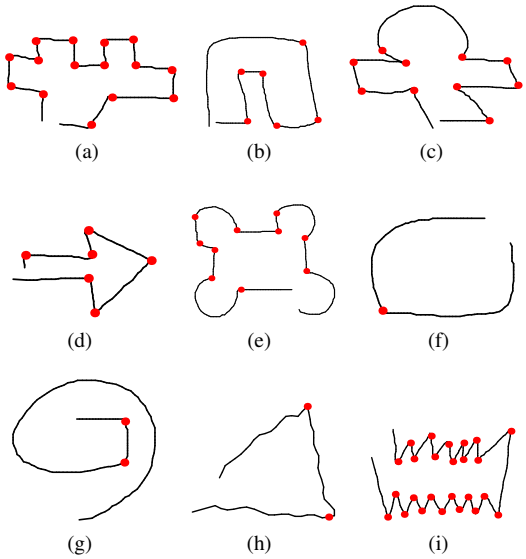


Figure 11: *Results of one step segmentation with* $k = 3, h_k = 0.6$ *(76 feature points found)*

- Our window-controlled relative curvature calculation is noise-immune (Figure 11(h), 12(h), 13(h)).
- One-step segmentation with larger window size $k = 9$ produces unacceptable result, which demonstrates that smaller window size should be used at first in our two-step segmentation method.

However, some limitations still exist:

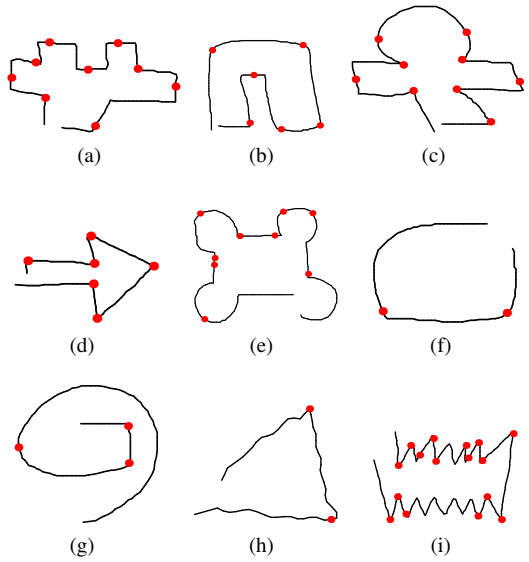


Figure 12: *Results of one step segmentation with* $k = 9, h_k = 3$ *(62 feature points found)*

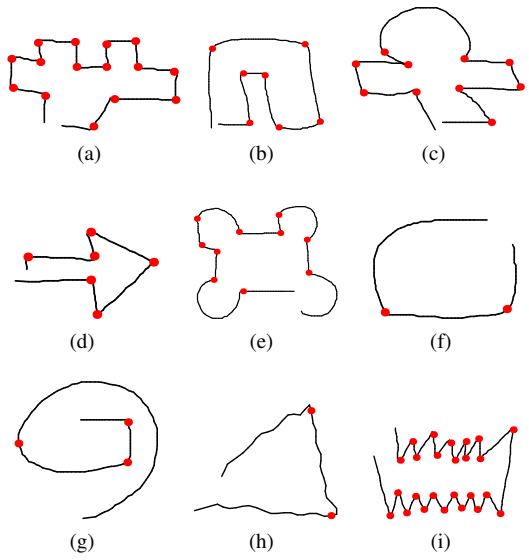


Figure 13: *Results of two-step segmentation with* $k_1 = 3, h_{k_1} = 0.6$ *and* $k_2 = 9, h_{k_2} = 3$ *(79 feature points found)*

- Two-step method has higher recall than one-step method, but the price is lower precision. It means that more non-feature points are found by two-step method (Figure 13(g)).
- Some global noises can not be eliminated by smaller window size (Figure 11(e)), but can be eliminated by larger window size (Figure 12(e)).

These limitations might be caused by our naive strategy of merging smaller window size and larger window size segmentation — segmentation upon segmentation. A more complex merging strategy could produce better result.

7. Conclusion

This paper introduces a new method for ink segmentation in sketch understanding. We first argue that drawing speed is not necessary for segmentation, and it would have bad effect in some cases. Then we introduce our approach for segmentation, in which a window-controlled method is deviced for calculating relative curvature. Based on the method, we propose a two-step strategy for ink segmentation which is proved effective by experimental results.

In experiments, we discover that although our approach can find out all the expected feature points with 100% recall, precision is decreasing. We also notice some global noises can not be eliminated in our approach. We will work on these issues to provide a better solution for segmentation in sketch understanding.

References

[AOD02] ALVARADO C., OLTMANS M., DAVIS R.: A framework for multi-domain sketch recognition. *AAAI Spring Symposium on Sketch Understanding* (March 25-27 2002), 1–8.

[CD04] CATES S., DAVIS R.: New approach to early sketch processing. In *Making Pen-Based Interaction Intelligent and Natural* (Menlo Park, California, October 21-24 2004), AAAI Press, pp. 29–34.

[GKSS05] GENNARI L., KARA L. B., STAHOVICH T. F., SHIMADA K.: Combining geometry and domain knowledge to interpret hand-drawn diagrams. *Computers & Graphics 29*, 4 (2005), 547–562.

[HD03] HAMMOND T., DAVIS R.: LADDER: A language to describe drawing, display, and editing in sketch recognition. *Proceedings of the 2003 Internaltional Joint Conference on Artificial Intelligence (IJCAI)* (2003).

[HL00] HONG J. I., LANDAY J. A.: SATIN: A toolkit for informal ink-based applications. In *Proceedings of the ACM Symposium on User Interface Software and Technology* (2000), Toolkits and Techniques for Pen and Video, pp. 63–72.

[KS04] KARA L. B., STAHOVICH T. F.: Hierarchical parsing and recognition of hand-sketched diagrams. In *UIST* (2004), pp. 13–22.

[LLG*03] LI Y., LANDAY J. A., GUAN Z., REN X., DAI G.: Sketching informal presentations. In *Proceedings of the 5th International Conference on Multimodal Interfaces (ICMI-03)* (New York, Nov. 5–7 2003), ACM Press, pp. 234–241.

[LM01] LANDAY J. A., MYERS B. A.: Sketching interfaces: Toward more human interface design. *IEEE Computer 34*, 2 (2001), 56–64.

[LNHL01] LIN J., NEWMAN M. W., HONG J. I., LANDAY J. A.: DENIM: an informal tool for early stage web site design. In *Proceedings of ACM CHI 2001 Conference on Human Factors in Computing Systems* (2001), vol. 2 of *Interactive video posters*, pp. 205–206.

[SSD01] SEZGIN T. M., STAHOVICH T., DAVIS R.: Sketch based interfaces: Early processing for sketch understanding. *Workshop on Perceptive User Interfaces, Orlando FL* (2001).

[Sta97] STAHOVICH: Interpreting the engineer's sketch: A picture is worth a thousand constraints. In *Reasoning with Diagrammatic Representations II* (1997), AAAI Fall Symposium, pp. 31–38.

[Sta04] STAHOVICH T. F.: Segmentation of pen strokes using pen speed. In *Making Pen-Based Interaction Intelligent and Natural* (Menlo Park, California, October 21-24 2004), AAAI Press, pp. 1–7.