

A Domain-Independent System for Sketch Recognition

Bo Yu, Shijie Cai

State Key Laboratory of Software Novel Technology, Nanjing University, China
mikeyucn@hotmail.com

Abstract

Freehand sketching is a natural and powerful means of interpersonal communication. But to date, it still cannot be supported effectively by human-computer interface. In this paper, we describe a domain-independent system for sketch recognition. Our system allows users to draw sketches as naturally as how they do on paper, and it recognizes the drawing through imprecise stroke approximation which is implemented in a unified and incremental procedure. This method can handle smooth curves and hybrid shapes as gracefully as it does to polylines. With a feature-area verification mechanism and the intelligent adjustment in the post-process, the system can produce user-intended results. Moreover, the output is organized in a hierarchical structure which includes syntactic and semantic information as well as raw data. Our system mainly utilizes low-level geometric features and does not rely on any domain-specific knowledge. Therefore, it will serve as a general and solid foundation for future high-level applications.

Keywords: Sketch Recognition, Graphics Recognition, HCI, Multimodal Interface

1. Introduction

Freehand sketching is a very natural and powerful means of interpersonal communication, and people frequently use it to convey information efficiently. It is also a very promising technology of human-computer interface which will release users from a maze of menus, toolbars and many complicated commands. Sketch recognition can be broadly used in the fields of engineering or software design, multimodal user interface, multimedia database retrieval, and so on.

The research on sketch recognition and interpretation has a long history. In [6], Rubine describes a trainable gesture recognizer which uses a linear evaluation function to discriminate gestures. But it can handle only unistroke symbols which must be drawn in a pre-specified manner.

The Design Rationale Group of MIT Artificial Intelligence Laboratory has done much work on sketch recognition [7] and some high-level applications based on it (e.g. [4], [5]). In [7], Sezgin described a scale space approach to analyze curvature and speed data of a stroke for vertex detection. The method avoids priori threshold settings and can achieve good results in some very noisy situations. However it treats vertex detection as a separate stage of process, which does not accommodate interior exchange of information between stages. Consequently, the procedure of vertex detection cannot utilize the result of shape approximation. In addition, his approach cannot handle smooth curves as gracefully as it does to polylines.

In [1], Arvo and Novins described a new sketching interface. Different from traditional ways of recognizing sketches, their method continuously morphs raw input strokes into ideal geometric shapes, which can apprise users the recognition progress and the appearance of final shapes. However, their method can handle only circles and rectangles drawn with one stroke. This is not adequate for a practical system.

Igarashi, Matsuoka, Kawachiya and Tanaka [8] created an interactive beautification technique which could turn imprecise sketches into a rigorous drawing just as being drafted with a CAD system. Their method infers any possible geometric constraints such as perpendicularity, congruence, symmetry etc, generates multiple beautification candidates, and then chooses the most plausible one. However, the restriction that each line must be drawn with a separate stroke may bother users.

Fonseca and Jorge [3] presented a method, based on fuzzy logic, for recognizing multi-stroke sketches. Their system processes one or more strokes, which are collected within a timeout value, as a whole and classifies it into one of several predefined shape kinds. However, their system cannot identify the constituent parts of a shape. And moreover, because it relies entirely on global geometric properties such as area and perimeter, it may have difficulties to distinguish between similar shapes.

From the above reviews, we can find two main deficiencies of the existing methods for sketch recognition:

- Strict restrictions on the drawing manner which do not allow users to sketch naturally.
- Lack of the ability to analyze hybrid or smooth curves and decompose them into primitive shapes.

In this paper, we will describe a domain-independent system for sketch recognition. It is designed to recognize freehand sketches and represent them with a hierarchy of primitive shapes and simple objects. In addition to the above two problems, it aims at generality, and all its operations are based on low-level geometric features. Therefore, it can be integrated conveniently with other high-level and domain-specific applications.

2. System Overview

We think that a practical sketch recognition system must have the following attributes:

- It must allow users to draw in a natural way just as how they do on paper.
- It should produce a consistent recognition result, i.e. if a user wants to sketch a line segment, no matter it is drawn with a single stroke, or more than one stroke, or a compound stroke which contains other primitive elements, the system should interpret it as a line segment.
- It should understand the hierarchical relations among the recognized graphic objects. For example, a stroke which is composed of several primitive shapes may serve as a constituent part of some more complex object.
- It should try to guess what a user intends to draw and turn the imprecise strokes into regular objects. Although domain-specific knowledge may be very helpful here, some simple decisions can still be made according to low-level geometric features only.

Copyright © 2003 by the Association for Computing Machinery, Inc. Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions Dept, ACM Inc., fax +1 (212) 869-0481 or e-mail permissions@acm.org.

© 2003 ACM 1-58113-578-5/03/0002 \$5.00

- Besides a fairly high recognition rate, it should provide an efficient and convenient user interface for drawing modification as well.
- It can be easily integrated as an input module into other more complex systems.

Our system is easy to use. It does not put any particular constraint on the manner of drawing sketches. Users can use any pen-like input device or even a mouse to draw freely within the canvas area of the program. Our system will record the strokes, which are defined as a series of coordinates visited by the pen between a single pair of mouse-down and mouse-up events (such events can also be triggered by other pen-like input devices). Besides coordinates, each stroke point is associated with a time stamp, which mainly differentiates sketch recognition from other problems in image analysis such as OCR and vectorization. The time information is very important for extraction of geometric features.

The whole process of our system is divided into two stages: imprecise stroke approximation and post-process. The former will be performed immediately after each stroke is finished. Given a stroke as input, the recognition module tries to approximate it with one or a combination of primitive shapes out of several predefined kinds, i.e. lines, arcs, circles, ellipses and helixes. Then the approximation result will be displayed as a rapid feedback. This process will not interrupt the user, and he or she can keep on sketching other strokes just like drawing on paper. We do not use spline curves for approximation because we consider it is meaningless to generate an exact approximation to raw coordinates of input strokes which are imprecisely sketched. Furthermore, spline curves are of little use when high-level applications perform graph matching or some other similar tasks.

Post-process will be applied after the user finishes the whole drawing. Our system will analyze those primitive shapes obtained from stroke approximation, retrieve some simple relations among them, and try to represent them as what the user intends. Despite considerable distortions, our system can recognize some kinds of basic objects through several heuristic reasoning rules.

Our system provides a hierarchical output. The lowest level is the raw data which contain the original information of stroke points. The mid level is called the syntactic level where the vertexes and primitive shapes are stored. The highest level of the output is the semantic level. It provides the relation tables among the recognized primitive shapes and basic objects. Such information may aid the semantic analysis of high-level applications. Our system also provides a friendly user interface by which users can modify recognition results conveniently to obtain tidy drawings.

3. Imprecise Stroke Approximation

3.1. Definitions

For a given stroke, the *direction* and *curvature* (change in direction with respect to path length) for each point are computed as

$$d_n = \arctan\left(\frac{y_{n+1} - y_n}{x_{n+1} - x_n}\right)$$

and

$$c_n = \frac{\left| \sum_{i=n-k}^{n+k-1} \varphi(d_{i+1} - d_i) \right|}{D(n-k, n+k)}$$

where d_n and c_n represent the direction and curvature of the n -th stroke point respectively, k is a small integer defining the neighborhood size around the n -th point and $D(n-k, n+k)$ stands for the path length between the $(n-k)$ -th and $(n+k)$ -th stroke points.

We set k to 2 empirically as a tradeoff between the suppression of noise and the sensitivity of vertex detection. The function φ shifts its angle parameter to the range from $-\pi$ to π . Fig. 1 shows an input stroke together with its direction and curvature graphs.

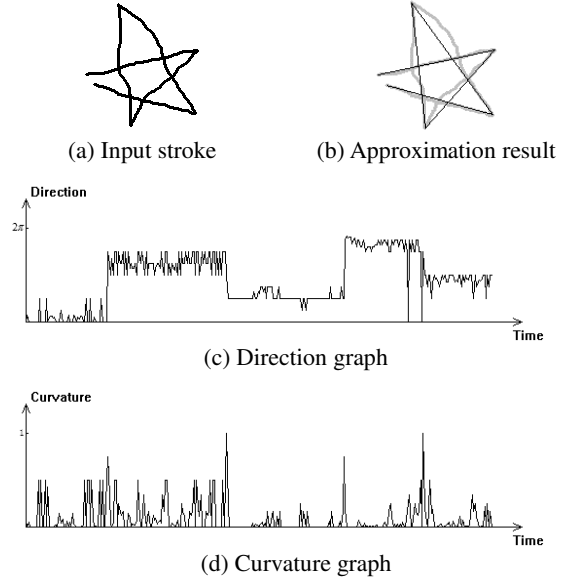


Figure 1: An example stroke with its direction graph and curvature graph.

Area is a kind of global geometric property which is much more resistant to noise than other ones. Considering its incapability of distinguishing between similar shapes, we use it only to guide recognition and verify results.

The *feature area* of a stroke is defined by its constituent points and another reference object which can be a line segment, an arc or a point, as illustrated by the hatched regions in Fig. 2. The feature area to a line is computed as the sum area of all the small quadrangles formed by two consecutive stroke points and their foot points on the line. And the feature area to an arc can be computed similarly. The feature area against a reference point equals to the sum area of all the small triangles formed by two consecutive stroke points and that reference point.

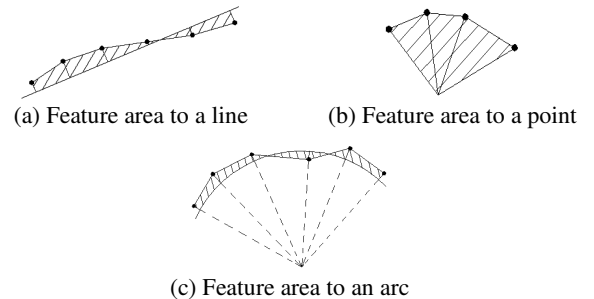


Figure 2: The examples of feature area.

3.2. Vertex Detection

Intuitively speaking, the vertexes of a stroke are usually the points whose curvatures are high enough. This serves as the basic idea for vertex detection. But freehand sketches are extremely different from carefully-drawn engineering diagrams, and we have to deal with considerable or even severe noise. The noise mainly comes from two sources. One is the inherent freewill and unpredictable

drawing manner of sketching. The other is the discrete sampling mechanism with which the computer tracks the movement of the pen. In Fig. 1d, although the noise obviously affects the shape of the curvature curve, it still cannot surpass the peaks corresponding to the true vertexes. But as to a smooth stroke without explicit vertexes, such as a circle, the noise will dominate the whole curve. Consequently, we need some mechanism that can identify true vertexes from the curvature curve, and will not be easily misled by noise.

Different from previous approaches used in [2] and [7], which treat vertex detection as a separate step, we combine vertex detection and primitive shape approximation into a unified and incremental procedure. This is one innovation of our system. As can be shown from Fig.1, there is usually severe noise in a stroke's curvature curve. Using curvature data alone for analysis is not reliable enough, and can be easily misled in some noisy situations. As direction and area features are taken into account in primitive shape approximation, and then the approximation result is used to guide vertex detection, our method fully utilizes the visual features of a stroke.

The procedure takes one stroke as input. It first checks whether the stroke can be approximated by any kind of primitive shape. The procedure stops if the approximation succeeds; otherwise, it chooses a dividing point corresponding to the highest peak in curvature curve, breaks the stroke into two children ones, and then invokes the procedure itself with these two children strokes as parameter recursively and respectively. Fig. 1b shows a approximation result of this procedure. Note that such a top-down procedure analyzes the structure of a hybrid-shape stroke incrementally and can preserve the hierarchical relations among primitive shapes, which is essentially different from the methods used in [6] and [3].

Here we avoid any empirical threshold or statistical calculation for judging vertexes, and whether this procedure can recognize strokes as intended and produce consistent results for similar strokes will mainly depend on the reliability of primitive shape approximation.

Although the speed data of a stroke can provide useful information sometimes, which are considered in [2] and [7], they will inevitably introduce more noise and thus counteract the positive effect. After all, visual features are the most important and reliable basis for stroke analysis. According to our experience, the system will not neglect true vertexes in most cases. Furthermore, through the powerful user interface for drawing modification, users of our system can easily correct the recognition errors.

3.3. Line Segment Approximation

Two kinds of clues may suggest an input stroke to be a line segment. One is from the direction graph, i.e. the direction curve can be fitted with a horizontal line. The other is whether the actual coordinates of the stroke points can be fitted by a line.

First, we compute the least-squares best fit line for either the direction curve or the coordinates of stroke points, and check whether the number of deviation points is small using a relatively loose threshold. Then, if succeeds, we hypothesize a line segment from the first stroke point to the last one and judge its validity through the corresponding feature area to that candidate line. Since we give a higher priority to line segments than arcs, if the ratio between the feature area and the candidate line's standard area, which equals to the product of its length and the input stroke's width, is less than a strict threshold (1.0 is chosen in our implementation), the system will accept the hypothesis at once and save that line segment. Otherwise, the final decision will be delayed after the same stroke has been tested by the arc recognizer.

We name the procedure of checking the ratio between feature and standard area "*feature-area verification*". As a judging criterion based on global geometric features, it serves as a complementary constraint to those local criteria based on direction and curvature information, and makes the recognition more reliable.

3.4. Curve Approximation

For a stroke with a circle or ellipse shape, although its curvature curve is dominated by noise and has little value for analysis, its direction graph has some recognizable pattern (as in Fig. 3c). Note that the direction angle values are shifted in order to make the data distribute continuously on the direction axis. From Fig. 3c, we can see that the direction curve can be approximated by a line with its slope close to $2\pi/n$, where n is the total number of the stroke points. This is because the stroke points are approximately distributed evenly over a range of 2π on the direction axis.

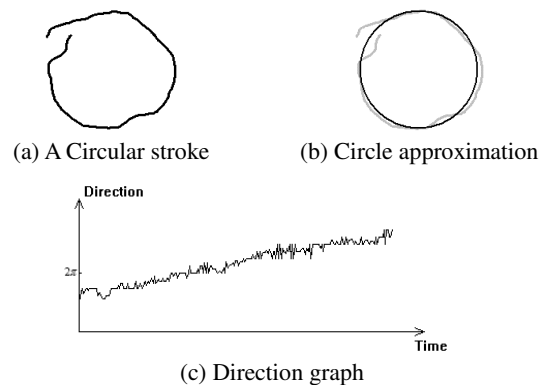


Figure 3: The direction graph of a circular stroke and the approximation result.

For circle approximation, we first check whether the direction graph of the input stroke can be approximated with a line mentioned above. If succeeds, we hypothesize a candidate circle which takes the center of the stroke's bounding box as its own center and the mean distance between the center and each stroke point as its radius. Then the feature-area verification is applied, i.e. we check whether the stroke's feature area to the center can match the actual area of that candidate circle. Ellipses can be handled similarly with one more constraint, i.e. the area should be nearly symmetrical about the major and minor axes respectively.

Given a stroke's direction graph, if the curve bears a line shape and its span projected on the direction axis can

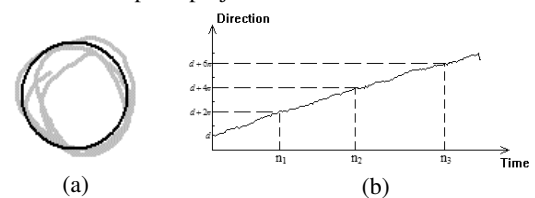


Figure 4: (a) An overtraced circle. (b) The selection of dividing points.

accommodate more than one segment whose length equals to 2π , the system will then break the stroke into several children ones accordingly and process them respectively. If the whole stroke seems to be formed by several circles or ellipses with similar size and position, they will be replaced by a "mean" one; if it seems to consist of a series of circles with ascending or descending radii, the system will accept it as a helix; or else, it will

be re-recognized through the common procedure (see section 3.2). Fig. 4 shows an example.

As for an arc stroke, the shape of its direction curve should also be near a line. But different from that of a circle stroke, the line's slope should be less than $2\pi/n$, since its span on direction axis is less than 2π . The algorithm for arc recognition is devised as follows:

- (1) Check whether the direction curve can be approximated with a line, and abort if fails.
- (2) Find the perpendicular bisector, denoted as M, of the first and last stroke points, calculate the intersection point A between line M and the stroke path, and then find the circle on which the first and last stroke points as well as point A reside. The circle's center is taken as the center of the candidate arc.
- (3) Take the mean distance from each stroke point to the center as the radius.
- (4) Compute the ratio between the corresponding feature area against the candidate arc and that arc's standard area. Check whether the ratio is less than a predefined threshold, and abort if fails.

As the slope continues to decrease, the arc becomes more flat, and the stroke looks more like a line segment. Here comes an inherent ambiguity in distinguishing between lines and arcs. Since there is no domain knowledge available here, we have to resort to feature area again to make a reasonable choice. During the approximation, there are two ratios recorded. One is between the feature area against the candidate line and that line's standard area, the other is between the feature area against the candidate arc and that arc's standard area. Our system will compare them and select the hypothesis with a smaller ratio value.

3.5. Handling Self-Intersection Strokes

Fig. 5 shows a self-intersection stroke. If we simply apply the analyzing process described above, we will probably not be able to recognize the main components, i.e. the two circles, since the curvature values are relatively low near the two self-intersection points and thus the stroke may not be divided there. On the other hand, we cannot blindly break any stroke which has self-intersection points, because sometimes they should definitely not be treated as dividing points (as in Fig. 1a).



Figure 5: An example of self-intersection stroke.

In order to resolve this kind of ambiguity and handle self-intersection strokes properly, we have revised the stroke approximation procedure as follows: Given a stroke which cannot be approximated by any of our predefined primitive shapes, if it has no self-intersection point, then follow the original procedure; otherwise, make two copies of that stroke, divide them using different strategies, i.e. from self-intersection points versus from the point with maximum curvature, then compare respective recognition result and choose the better one.

We have devised a function for comparing recognition results based on the following two heuristic rules: simpler is better which favors a smaller number of constituent primitive shapes, and more specific is better which prefers circles, ellipses and helices than lines and arcs. The fitting value of a child stroke is scaled by multiplying the length ratio to its father stroke. The fitting value of a compound stroke is the sum of all its children strokes' fitting values.

4. Post-Process

Contrary to the stroke approximation which is performed during the course of sketching, the post-process is applied after the whole drawing has been completed. It takes all the recognized primitive shapes as input, and the main tasks include:

- Eliminate the false and redundant elements introduced by noise.
- Modify the size and position of the elements, and adjust their layout in order to make them look as intended.
- Perform some basic, domain-independent object recognition.

Post-process includes three stages in order: simple relation retrieval, cleanup, and basic object recognition. After these tasks are finished, the hierarchical output will be generated.

4.1. Simple Relation Retrieval

Simple relation tables form an important portion of semantic output level. One key relation is connectivity. Besides the in-stroke connectivity that is preserved in stroke approximation, the system analyzes the cross-stroke connectivity as well. Temporal relation can be directly derived from the time stamp of each primitive element's start point. Another two important kinds of relations retrieved by our system are parallelism and perpendicularity which are necessary to subsequent operations. Note that the size and position of some graphic elements may be slightly changed accordingly.

4.2. Cleanup

Of all line segments and arcs, there may be some short meaningless ones, especially at the beginning or end of one stroke and some other positions where curvature changes abnormally. Removing these useless elements is one task of cleanup, and we must distinguish them from useful ones. Fig. 6 shows a narrow rectangle. We should not delete any of its shorter edges, otherwise the shape will degrade to a triangle or even two overlapped line segments. Therefore we introduce the following two rules:

Rule 1: If a line segment (arc) with at least one extreme point open, i.e. not connected to other elements, is very short compared with the mean length of all the elements in its original stroke, then remove this line segment (arc).

Rule 2: Given one line segment (arc) with each extreme point connected to some other element, if it is very short compared with its neighbor elements' length and these two ones are neither symmetrical nor parallel, then shorten this line segment (arc) to its midpoint.

Furthermore, we should consider merging primitive elements together under two circumstances: 1) when false vertexes caused



Figure 6: A narrow rectangle.

by noise disturbances break a continuous shape into parts; and 2) the user draws a primitive shape with more than one stroke. Rule 3 will handle these situations.

Rule 3: If two line segments (arcs) with similar angles (centers and radii) are connected or overlapped, then merge them into a new one.

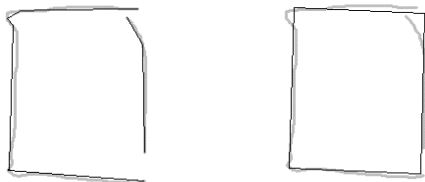
For drawings composed of only line segments and polylines, there are usually few meaningless elements left after the cleanup. For drawings composed of hybrid strokes, 60%-80%, on average,

of all the meaningless elements will be removed by the above rules.

4.3. Basic Object Recognition

In order to facilitate the applications of high-level semantics analysis or domain-specific interpretation based on our system, a simple recognition module is devised to process the output of the cleanup module. It can identify some basic kinds of objects such as boundaries, triangles, rectangles, arrows, dashed lines, and etc. Here the word “boundary” means any set of line segments or arcs or both which are connected end to end and forms a closed path. These objects are very common in drawings across multiple domains, and we will not rely on any domain-specific knowledge to recognize them.

Because of the freewill drawing manner which is inherent to sketches, although these objects look very simple, we still have to face the dilemmas: What is the user’s intension by drawing the distorted strokes, for example, the sketches shown in Fig. 7a? Or more generally, to which extent can we recognize an irregular shape as what the user may intend? For those objects possessing areas, such as triangles and rectangles, the feature-area verification serves as a good criterion. For instance, the rectangle analyzer, taking a boundary as input, first tries to find the longest four line segments which satisfy the prerequisites that the opposite ones are nearly parallel and the neighboring ones can form an intersection angle bigger than 80 degree. Then these potential edges are rotated about their midpoints and extended to form a hypothesized rectangle. According to the center of this rectangle, the feature area of the input boundary will be calculated using the similar method applied in stroke approximation. Finally, the analyzer will accept the hypothesized rectangle if the ratio between the boundary’s feature area and the hypothesized rectangle’s standard area is close to 1. Fig. 7 shows the stroke approximation of a sketched rectangle together with its final recognition result. Note that in the data structure of rectangles and other basic objects, we will record their original boundaries or constituent strokes for possible later uses such as re-recognition guided by domain-specific knowledge. As to other basic objects without area, we recognize them with some simple templates.



(a) Stroke approximation (b) Final result

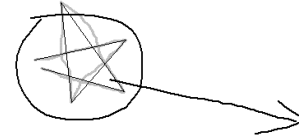
Figure 7: A sketched imprecise rectangle and its final recognition result.

A potential problem is that these recognition procedures are currently hard coded. We plan to devise some formal mechanism, which can be handled automatically by our system, to describe those basic objects and make the extension of recognizable object set much easier.

5. User Interface

In order to facilitate users to obtain their desired drawings, we provide a powerful user interface for drawing modification. Besides the traditional toolbox for the creation, modification and deletion of primitive shapes, it adopts some command symbols based on sketch recognition as well. Fig. 8 lists the supported ones.

When the system is in modification state, a clock will be set to check timeout after a first stroke is created. This stroke and the subsequent ones collected before timeout will be recognized together through the same process described in previous sections. They will either be identified as one command symbol or simply discarded.



(a) A circle is used to select objects. An arrow starting from one selected object means a copy operation to the pointed position.

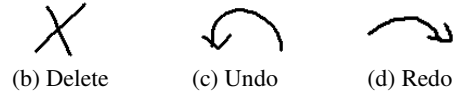


Figure 8: Command symbols.

When an object is selected, several control points determined by its shape will be shown and users can adjust the object’s geometric attributes through them. For example, a line segment has three control points, two end points which users can drag to change the line’s length and direction, and the third is the midpoint which users can drag to move the line to some other positions. The user interface also provides the snap mechanism similar to what is offered by AutoCAD, i.e. the cursor will be constrained to some exact locations, such as an intersection or a midpoint, on the objects.

6. Evaluation

We have conducted a user study to measure to which extent our system can achieve the desired features for a practical sketch recognition system listed at the beginning of section 2. There were totally ten students, who were chosen randomly from our university, participating in the study.

First, they all thought our system was very easy to use, and it took almost no time for them to learn how to use it.

The sample strokes for user study were divided into two categories: primitive shapes and hybrid shapes. Primitive shapes were just the shapes which can be approximated by our system in stroke approximation. A hybrid shape was defined as the composition of those primitive shapes. Note that for a hybrid shape, if the main and key structures were recognized correctly, we will accept it as a successful recognition.

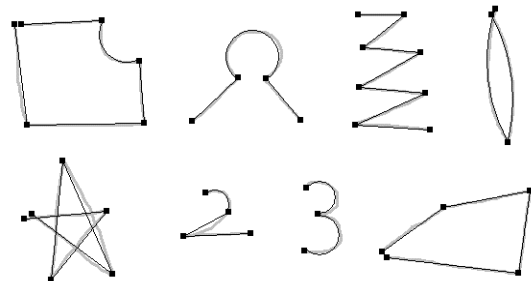


Figure 9: Examples of hybrid shapes with recognition results.

The overall correct rate for primitive shapes and polylines was near 98%. The correct rate for arcs alone was 94%, which was relatively low compared to other kinds of primitive shapes. This was because some coarsely sketched arcs were recognized as a series of small arcs and line segments. We felt that such a situation

could be best handled with the help of domain-specific knowledge. The hybrid shapes which contained smooth connections between lines and arcs were the most difficult to recognize, and the correct rate was only about 70%. It was because those smooth connections usually had a relatively low curvature. This problem severely influences the consistency of our system, and we have begun to deal with it. Other kinds of hybrid shapes all achieved correct rates of near or more than 90%. Fig. 9 lists some examples of hybrid shapes used in the user study. Although the figures of correct rate may fall behind the system developed by MIT AI Laboratory in some aspects, our system can recognize a broader range of primitive shapes and provide more syntactic and semantic information in the output, which will be very valuable to high-level applications built upon it.

As a low-level and domain-independent recognition system, there are usually few clues available for judging what the user intends. Our approach depends on only some visual features to resolve ambiguities and will inevitably make some mistakes sometimes. Thus when high-level applications doubt the validity of recognition results, the raw data of output will serve as the basis for correction or re-recognition.

The system can be easily integrated as an input module into other systems because the data flow through it is one-way and simple. Furthermore, the hierarchical output can satisfy different demands from high-level systems. We are now using this system as an input module to develop a query system for CAD drawings. Since some basic objects such as rectangles and arrows can be read directly from the semantic level of the hierarchical output, the query algorithm only needs to treat them just the same as lines and arcs, which is easy to implement. In addition, the parallelism and perpendicularity relations stored in semantic level can serve as spatial constraints to narrow down the search space and speed up the query process.

Another noticeable feature of our system is the convenient and efficient user interface for modifying recognition result. All the errors occurred in the user study were easily corrected by those participants.

7. Conclusion

We have built a domain-independent system for sketch recognition. It allows users to draw sketches as naturally as how they do on paper, and can recognize drawings efficiently and reliably. The recognition result is organized in a hierarchical structure which includes syntactic and semantic information as well as raw data. Our system mainly utilizes low-level geometric features and does not rely on any domain-specific knowledge. Therefore we consider it a powerful user interface, and we believe it will serve as a general and solid foundation for future high-level, domain-specific applications.

Reference

- [1] J. Arvo and K. Novins, "Fluid Sketches: Continuous Recognition and Morphing of Simple Hand-Drawn Shapes", *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology*, 2000.
- [2] C. Calhoun, T. F. Stahovich, T. Kurtoglu and L. B. Kara, "Recognizing Multi-Stroke Symbols", *2002 AAAI Spring Symposium on Sketch Understanding*.
- [3] M. J. Fonseca and J. A. Jorge, "Using Fuzzy Logic to Recognize Geometric Shapes Interactively", *Proceedings of the 9th International Conference on Fuzzy Systems*, 2000.
- [4] Tracy Hammond and Randall Davis, "Tahuti: A Sketch Recognition System for UML Class Diagrams", *2002 AAAI Spring Symposium on Sketch Understanding*.
- [5] Michael Oltmans and Randall Davis, "Naturally Conveyed Explanations of Device Behavior", *Proceedings of PUI-2001*, November 2001.
- [6] D. Rubine, "Specifying gestures by example", *Computer Graphics (SIGGRAPH '91 Proceedings)*, pages 329-337.
- [7] T. M. Sezgin, "Feature Point Detection and Curve Approximation for Early Processing of Free-Hand Sketches", *Master's thesis*, Massachusetts Institute of Technology.
- [8] T. Igarashi, S. Matsuoka, S. Kawachiya and H. Tanaka, "Interactive Beautification: A Technique for Rapid Geometric Design", *UIST'97*, pages 105-114.