Experimental Evaluation of an on-line Scribble Recognizer

Manuel J. Fonseca and Joaquim A. Jorge

Departamento de Engenharia Informática
IST/UTL, Av. Rovisco Pais, 1
1049-001 Lisboa
Portugal
email: mjf@ist.utl.pt, jorgej@acm.org

Abstract: *We present a fast, simple and compact approach to recognize Scribbles (multi-stroke geometric shapes) drawn with a stylus on a digitizing tablet. Our method is able to identify shapes of different sizes and rotated at arbitrary angles, drawn with dashed, continuous strokes or overlapping lines. We use temporal adjacency to allow users to input the most common shapes in drawing such as triangles, lines, rectangles, circles, diamonds and ellipses, using multiple strokes. We have further extended this approach to identify useful shapes such as arrows, crossing lines and uni-stroke gestural commands and have developed a library of software components to make this software generally available. The recognition algorithm uses Fuzzy Logic and geometric features, combined with an extensible set of heuristics to classify scribbles. Most recent evaluation results show recognition rates over 97%.*

Keywords: *On-line Symbol Recognition, Tools and Techniques, Fuzzy Logic, Multi-stroke Shapes, Experimental Evaluation*

1. INTRODUCTION

User Interfaces are developing away from the classical WIMP (Windows, Icons, Mouse and Pointing) paradigm. Among the new generation of intelligent systems there is a broad class based on new modalities such as sketching, hand drawing and stylus input gestures, combined optionally with speech, which we call *Calligraphic Interfaces*. People use diagrams to think, reason about and discuss designs in domains as diverse as architectural engineering and music. People tend to draw diagrams to represent complex models and ideas, arguably a problem-solving strategy.

However, despite the "G" in the current generation of Graphical User Interfaces, these are characterized by discrete interaction modalities such as selecting items from menus, pointing to things on the screen and pressing buttons. The drawing paradigm, so powerful and so useful for humans goes largely underutilized.

Our aim is to develop simple components to help people use computers in more engaging and "natural" modes. To this end, we have developed a simple recognizer capable of identifying the most common constructs and gestures people use to create drawings. In this paper we review summarily the state of the art in on-line graphics recognition, describe the geometric features used and examine the experimental results from evaluating our approach. While the method uses very simple concepts and a reduced set of geometric features, it is surprisingly robust and extensible, making it usable in a wide variety of practical environments.

2. RELATED WORK

The idea of calligraphic interfaces is not new, even if we discount the fact that sketches and planar diagrams as a way of communication precede the invention of writing by more than 30 centuries [9]. In 1963, Sutherland presented Sketchpad [13] the first interactive system that used one light pen to draw diagrams directly over the screen surface. The main limitation of this system resided in the recognition capabilities, limited resources and high cost of the computer used.

Nevertheless, due in part to ergonomic problems with the light pen and the invention of mouse in 1964, today graphical interfaces relegated pen based interfaces to specific CAD applications until the appearance of the first pen computers in 1991. Tappert [14], Provides an excellent survey of the work developed in the area of non-interactive graphics recognition for the introduction of data and diagrams in vectorial CAD systems.

Even though handwriting recognition is not the main target of our work, we will shortly analyze the Graffiti [7] system. This system recognizes characters drawn with a single stroke. While this makes the recognition process easier, on the other hand it forces users to learn a new vocabulary. The Graffiti system, as is typical with most handwriting recognition systems, uses local features, such as drawing speed, while our method uses global geometric information to characterize drawings. As a result, our recognition process uses simpler features and analyzes complex shapes as a whole, regardless of the way individual strokes were drawn. Further, the vocabulary of geometric shapes is considerably smaller than that of Graffitti, thus requiring a much smaller number of features.

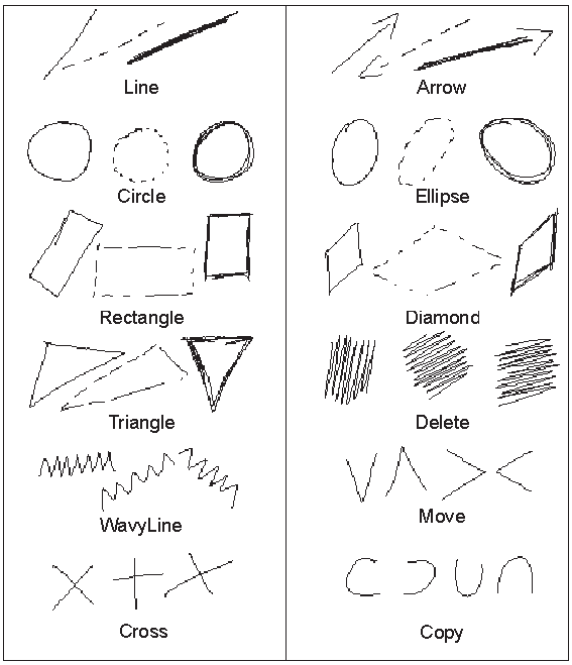

Figure 1. Multi-stroke and uni-stroke shapes.

The Newton system [1], one of the first hand-held pen based computers, incorporates a hand written recognizer, a shape recognizer and a gesture recognizer. The shape recognizer only recognizes solid shapes and requires the shapes to be drawn with just one stroke and aligned with the axes. The gesture recognizer is more suitable for text editing than for schematic drawing.

Gross [8] describes a system fundamentally based in sketches that are partially interpreted for use in architecture drawings, using a recognizer substantially more simple and less robust than ours.

Other authors have proposed more complex methods, involving neural networks [15] using a procedure that might prove more robust than ours, although such claims are not made explicit and substantiated by experimental results.

3. THE RECOGNITION ALGORITHM

The recognition method is based on three main ideas. First, it uses entirely global geometric properties extracted from input shapes. Since we are mainly interested in identifying geometric entities, the recognizer relies mainly on geometry information. Second, to enhance recognition performance, we use a set of filters either to identify shapes or to filter out unwanted shapes using distinctive criteria. Third, to overcome uncertainty and imprecision in shape sketches, we use fuzzy logic [4] to associate degrees of certainty to recognized shapes, thereby handling ambiguities naturally.

This algorithm recognizes elementary geometric shapes, such as `Triangles`, `Rectangles`, `Diamonds`, `Circles`, `Ellipses`, `Lines` and `Arrows`, and five gestural commands, `Delete`, `Cross`, `WavyLine`, `Move` and `Copy`, as

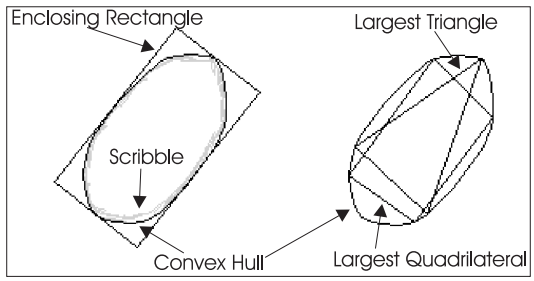

Figure 2. Polygons used to estimate features.

depicted in Figure 1. Shapes are recognized independently of changes in rotation, size or number of individual strokes. Commands are shapes drawn using a single stroke, except the `Cross` that is built of two strokes. The recognizer works by looking up values of specific features in fuzzy sets associated to each shape and command. This process yields a list of plausible shapes ordered by degree of certainty.

This approach extends and improves Kimura's work [3], which did not distinguish rotated, open/closed, dashed and bold shapes. The present is an evolution of previous work [10], which did recognize less shapes and used a decision tree to prune out incorrect classifications. We found out that using classification rules alone, provides more flexibility and extensibility without sacrificing robustness.

3.1. Geometric Features

We start the recognition process by collecting data points using a digitizing tablet, from the first pen-down event until a set timeout value after the last pen-up. Next, we compute the convex hull (ch) of the input points thus collected, using Graham's scan [12]. We then use this convex hull to compute three special polygons. The first two are the largest area triangle (lt) and quadrilateral (lq) inscribed in the convex hull [5]. The third is the smallest area enclosing rectangle (er) [6] (see Figure 2).

A *Stroke* is the set of points from pen-down to pen-up. A *Scribble* is a sequence of strokes collected within the timeout value. A *Shape* is a recognized scribble. It relates the input points to a class and a set of geometric attributes, such as, start- and end-points, bounding box.

The set of shapes selected and presented in Figure 1 are the basic elements to allow the construction of technical diagrams such as electric or logic circuits, flowcharts or architectural sketches. These diagrams also require distinguishing between solid, dash and bold depictions of shapes in the same family. Typically, architects will use multiple overlapping strokes to embolden lines in sketches, a mechanism commonly used in drawing packages. We are therefore interested in recognizing different renderings of a given shape as illustrated in Figure 1. We are just interested in collecting qualitative differences, not in obtaining precise values for attributes, without regard to different linestyles and widths.

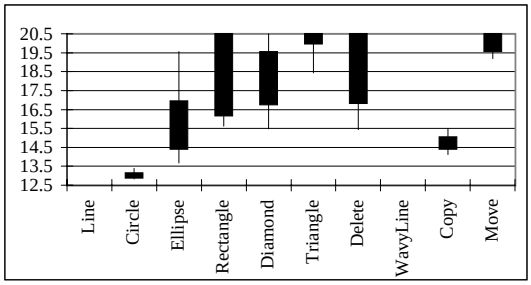

Figure 3. Percentiles for the P_{ch}^2/A_{ch} ratio.

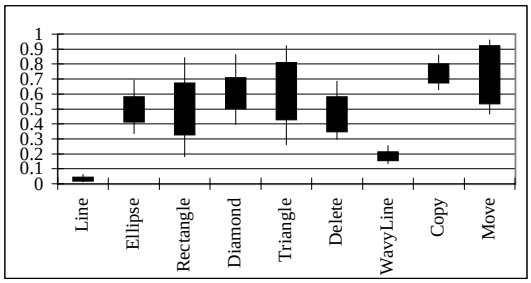

Figure 4. Percentiles for the H_{er}/W_{er} ratio.

In order to select the features that best identify a given shape, we built percentile graphics for each feature. These graphics illustrate the statistical distribution of feature values over the different classes, extracted from sample drawings. For each shape, the solid bar spans the 25% to 75% percentiles, while the line extends from 10% to 90% of all observed values of a given feature.

Our initial selection of features takes into account specific properties of shapes to identify. Associated with these features we infer fuzzy sets from training data, which express the allowable values for a given shape. Usually one feature alone is not enough to distinguish shapes, yielding incorrect classifications. We then add extra features (with corresponding fuzzy sets) to weed-out unwanted classifications, yielding more complex rules as needed.

To distinguish `Circles` from other shapes we use the *Thinness* ratio (P_{ch}^2/A_{ch}), where A_{ch} is the area of the convex hull and P_{ch}^2 is its perimeter squared. The thinness of a circle is minimal, since it is the planar figure with smallest perimeter enclosing a given area, yielding a value near 4π (see Figure 3). In this figure the thinness values for `Lines` and `WavyLines` lie outside the range of values indicated. We chose not to indicate these values in order to make the range of values for `Circles` more visible.

We identify `Lines` using another thinness ratio, which compares the height of the (non-aligned) enclosing rectangle (H_{er}) with its width (W_{er}). The H_{er}/W_{er} ratio will have values near zero for lines and bigger values for other shapes (see Figure 4). This feature is more reliable for distinguishing bold lines than P_{ch}^2.

In order to identify `Rectangles` we use two ratios. One relates the convex hull to the enclosing rectangle while the other measures the largest quadrilateral that fits inside the convex hull against the enclosing rectangle. For rectangular shapes the area of the convex hull will be very close to that of the enclosing rectangle (A_{er}) and this one will be very close to the largest quadrilateral's (A_{lq}). The A_{ch}/A_{er} and A_{lq}/A_{er} ratios will have values near unity for rectangles (see Figure 5).

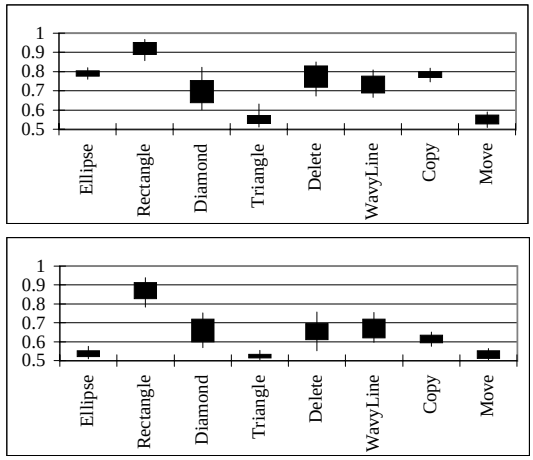

Figure 5. Percentiles for A_{ch}/A_{er} (top) and A_{lq}/A_{er} (bottom) ratios.

We identify `Ellipses` by comparing the area of the largest quadrilateral to that of the convex hull. The A_{lq}/A_{ch} ratio will have values near 0.7 for ellipses and bigger values for other shapes (see Figure 6). This is more robust than using A_{ch}/A_{er} which is too ambiguous.

Since this recognizer is intended for interactive use, we are interested in detecting gestures such as a set of zigzag strokes drawn in succession to signify erasing objects underneath. Using our geometry approach, we detect this pattern by comparing the total length of strokes (T_l) drawn to the perimeter of the convex hull (P_{ch}). The T_l/P_{ch} ratio has large values for the `Delete` command,

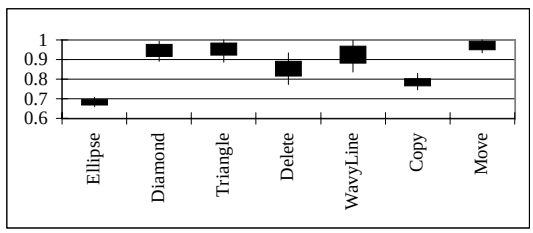

Figure 6. Percentiles for the A_{lq}/A_{ch} ratio.

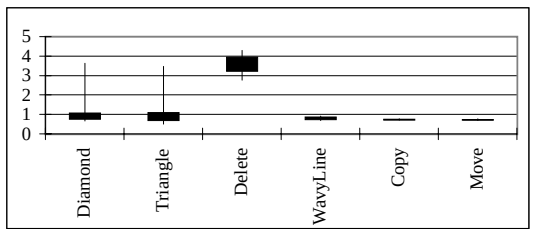

Figure 7. Percentiles for the T_l/P_{ch} ratio.

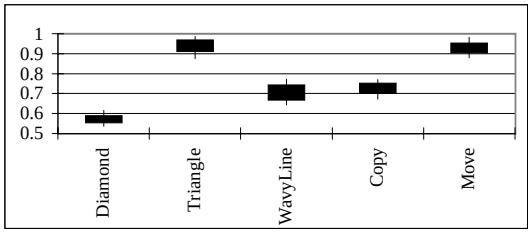

Figure 8. Percentiles for the A_{lt}/A_{lq} ratio.

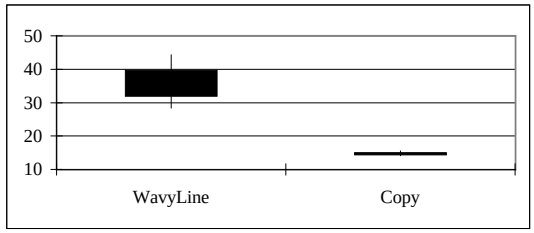

Figure 10. Percentiles for the P_{ch}^2/A_{ch} ratio.

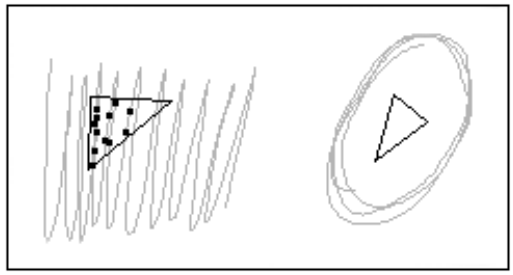

Figure 11. Hollowness of `Delete` and `Ellipse`.

and smaller values for other shapes (see Figure 7).

To distinguish `Diamonds` from other shapes we divide the area of the largest triangle that fits inside the convex hull (A_{lt}) by the area of the largest quadrilateral. The A_{lt}/A_{lq} ratio has values between 0.5 and 0.6 for diamonds and bigger ones for other shapes (see Figure 8).

To identify `Triangles` and `Move` gestures we compare the area and perimeter of the largest triangle to that of the convex hull. A_{lt}/A_{ch} and P_{lt}/P_{ch} will have values near unity for triangles and moves, and smaller values for other shapes (see Figure 9). We distinguish these two shapes using the openness of the move command.

The `Copy` command is recognized using the thinness ratio P_{ch}^2/A_{ch} (see Figure 10). It has values near 15 while others have bigger values.

The `WavyLine` gesture has two attributes that allow the distinction from other shapes. First it is a little "fatter" than lines, so the H_{er}/W_{er} ratio has bigger values (see Figure 4). Second, its total length is smaller than the total length of the `Delete` command, so the T_l/P_{ch} ratio has smaller values (see Figure 7).

To distinguish `Dashed` from solid lines we use the ratio $P_{ch}*N_s/T_l$ where N_s is the number of strokes in the scribble. Because dashed shapes have a large number of strokes, with a small total length, this ratio exhibits large values for dashed shapes and smaller values for closed shapes.

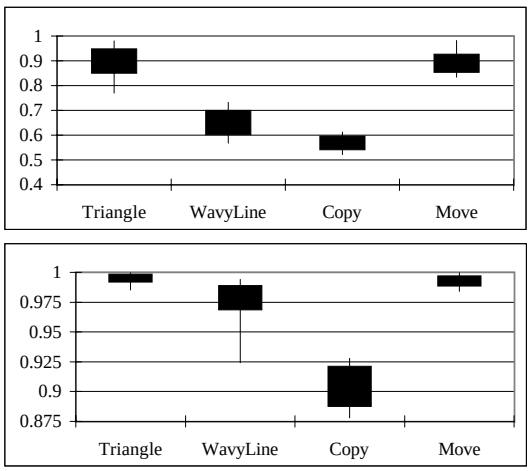

Figure 9. Percentiles for A_{lt}/A_{ch} (top) and P_{lt}/P_{ch} (bottom) ratios.

The T_l/P_{ch} ratio separates `Bold` from solid lines. Values bigger than 1.5 indicate redundant strokes (i.e. Bold lines).

Recognizing shapes drawn using overlapping strokes to signify `bold` linestyle conflicts with the `Delete` command, because we use the same feature (T_l/P_{ch}) in the same manner to identify both. To solve this conflict we introduce a heuristic feature to distinguish "filled" from "hollow" shapes. Hollow shapes do not have points inside near the barycenter. To compute *hollowness* we first calculate a triangle, whose dimensions are roughly 60% of the largest triangle that fits inside the convex hull and shares the same barycenter. We then count scribble points lying inside the smaller triangle. Shapes like `Triangles`, `Rectangles`, `Circles`, `Ellipses` or `Diamonds` shouldn't have any point inside, but `WavyLines` or `Delete` gestures must have (see Figure 11).

Figure 12 lists all shapes identified by the recognizer and the features used by each. As described earlier, conductive rules combine assertions or negations of these features as we shall see in the next section.

3.2. Re-segmentation

An approach based entirely on global geometric properties has some limitations. Even though `Arrows` or `Crosses` could not be recognized using this approach, it would be useful if the recognizer identified them. In order to recognize these shapes we must look out for new properties that characterize them. Those could be the small number of strokes, the existence of a stroke that uniquely identifies the shape, or a distinct spatial relation between strokes. For example `Arrows` are built of a set of strokes plus a last stroke that is a `Triangle` or a `Move` shape, as shown in Figure 13, and a `Cross` consists of two strokes

	Ach/Aer	Alq/Ach	Alt/Ach	Alq/Aer	Alt/Alq	Plt/Pch	Pch/Per	Plq/Pch	Nstrokes	Hollowness	Pch2/Ach	Her/Wer	Tl/Pch	Pch*Ns/Tl
Line												X		
Arrow									X					
Triangle			X		X	X	X			X				
Rectangle	X			X						X				
Circle										X	X			
Ellipse		X								X	X			
Diamond		X		X	X					X	X			
Delete									X	X		X	X	
WavyLine									X	X		X	X	
Copy			X		X			X	X		X		X	
Move			X			X			X		X		X	X
Cross		X						X	X					

Figure 12. Features used by each shape.

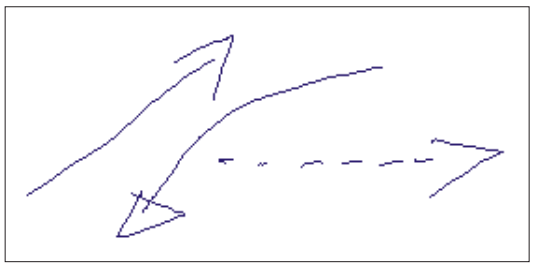

Figure 13. Different types of `Arrows`.

(that must be `Lines`), that cross each other.

The next two rules show how to classify these shapes. The first rule identifies `Arrows` while the second defines `Crosses`.

```
IF NumStrokes >= 2 AND
  (LastStroke IS LIKE Triangle OR
  LastStroke IS LIKE Move)
THEN
    Shape IS A Arrow

IF NumStrokes == 2 AND
  FirstStroke IS LIKE Line AND
  SecondStroke IS LIKE Line AND
  FirstStroke INTERSECT SecondStroke
THEN
    Shape IS A Cross
```

We perform the analysis of these new properties by re-segmenting the original scribble and by applying the recognition process to some specific strokes, e.g. the last stroke in the `Arrow`. Re-segmentation allows recognizing new gestures that could not be identified using just geometric properties.

4. The Fuzzy Recognition Method

The recognition method starts by collecting points from a digitizing tablet and computing some important polygons. Using these polygons area and perimeter we compute values for features selected as described in the previous section. We use fuzzy sets associated with each feature to classify the shape class(es) for a set of strokes (scribble). These sets were derived from percentile graphics like the ones shown in Figures 3 to 10. We identify shapes drawn by computing their degree of membership (dom) using the fuzzy sets associated with each feature. If several shapes are identified, the recognizer returns all classifications ordered by degree of certainty.

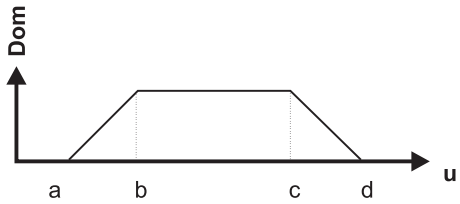

Figure 14. Definition of a fuzzy set.

The next two subsections describe how we derive fuzzy sets from experimental data and how to obtain the final classification results.

4.1. Deriving Fuzzy Sets from Training Data

To choose the "best" fuzzy sets describing each shape class we use a "training set" developed by three subjects, who drew each shape thirty times; ten times using solid lines, ten times using dashed lines and ten times using bold lines. Based on this training set we define several ratios, combining among others the area and perimeter of these three polygons described above.

Each shape is defined by several fuzzy sets, some of them are used to identify the shape and others are used to avoid "wrong" results. We did not use the same number of features for all shapes because some of them are identified unambiguously with one or two features. In general we tend to avoid using "too many" fuzzy sets to characterize a shape. Each set requires careful data analysis and experimentation to find the "right values", that is, those yielding higher recognition rates and less misrecognitions (false positives). The simplest case is that of a line, which is distinguished by thinness.

As depicted in Figure 14 a fuzzy set is defined by four values. After the selection of the set of features for each shape, we compute these four values based on the percentiles. Values `b` and `c` correspond to the percentiles 10% and 90% respectively, and `a` and `d` to the "minimum" and "maximum" after we have identified the outliers in each distribution. The elimination of outliers minimizes confusion between different shape families. There is a tradeoff in designing fuzzy sets from statistical data. If we make `a` and `d` very wide apart we increase the recognition rate as well as the number of false positives. If we select "narrower" values we decrease the number of false classifications at the cost of a much lower recognition rate. Abe et. al.[2] discuss an automated procedure for collecting this information using activation and inhibition hyper-boxes. We chose to generate rules manually, since their approach is not sufficiently flexible for

our purposes.

4.2. Deriving Results from Fuzzy Sets

After collecting input points from the tablet and computing the special polygons from scribble data, the recognizer calculates the degree of membership for each shape class. This degree is the result of AND together degrees of membership for the relevant fuzzy sets as highlighted in Figure 12.

In the following paragraphs we exemplify how to build rules that classify shapes. The first rule identifies Lines while the second defines Diamonds.

```
IF Scribble IS VERY THIN
THEN
     Shape IS A Line
```

This is the simplest rule used in our recognizer. It ascertains that "very thin" scribbles should be classified as Lines.

A more complicated rule recognizes Diamonds:

IF A_{lt}/A_{lq} IS LIKE Diamond AND
A_{lq}/A_{ch} IS NOT LIKE Ellipse AND
A_{lq}/A_{er} IS NOT LIKE Bold Line AND
A_{lq}/A_{er} IS NOT LIKE Rectangle
THEN
Shape IS A Diamond

Where AND denotes the conjunction of fuzzy predicates $\mu_x(f \text{ AND } g) = \min(\mu_x(f), \mu_x(g))$ and NOT is defined by $\mu_x(\text{NOT } f) = 1 - \mu_x(f)$.

Figure 8 describes the statistical distribution for feature A_{lt}/A_{lq}. Figures 6 and 5(bottom) illustrate observed statistical distributions for the other two features respectively. We derive fuzzy sets for these features from the corresponding distributions as outlined previously.

The algorithm distinguishes between Solid, Dashed and Bold shapes after identifying "basic" shapes. This enables us to treat linestyles as attributes orthogonal to shape, making the design more modular which in turn will make it easier to recognize different shapes in the future.

4.3. Ambiguity

Taking into account the shapes identified by the recognizer, we present four special cases which can yield ambiguous results. Ambiguity exists between Lines and WavyLines, WavyLines and Deletes, Circles and Ellipses, Diamonds and Rectangles. These cases are presented in Figure 15.

Humans solve this natural ambiguity between geometric shapes identifying more than one shape and making the final distinction based on the context or in the feedback from others.

The recognizer described in this paper deals with ambiguity between shapes in a similar way, i.e. when it can not identify uniquely a geometric shape, returns a list of plausible candidates. The choice of the best candidate can then be made by the application using context information.

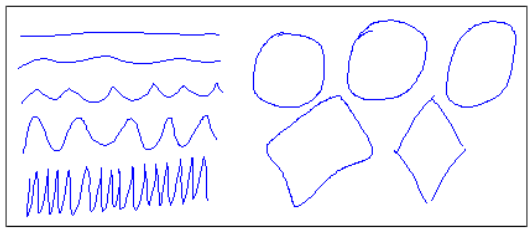

Figure 15. Ambiguity cases among shapes.

The ambiguities between shapes are modeled naturally using fuzzy logic to associate degrees of certainty to recognized shapes. Figure 16 illustrates corresponding fuzzy sets for the ambiguous cases shown in Figure 15.

5. An Example: Calligraphic Editor

The Calligraphic Editor presented in this section, uses the recognizer described above and allows the creation of "beautified" diagrams from "imperfect" sketches. With this editor users can create geometric shapes and invoke commands using hand drawn sketches. Shapes can be drawn, copied, moved and resized, using just the pen. We can delete an object or a set of objects by drawing the gestural command Delete, as illustrated in Figure 17.

This editor uses a simple way to deal with the ambiguity offered by the recognizer. Whenever the recognizer returns more than one shape, as the result of a sketch, the editor display a menu with all the possibilities, as shown in Figure 18, allowing the user to choose the correct one. This approach makes the creation of diagrams easier, because eliminates the need for redrawing the same sketch.

We are planning to try other solutions, like the use of dif-

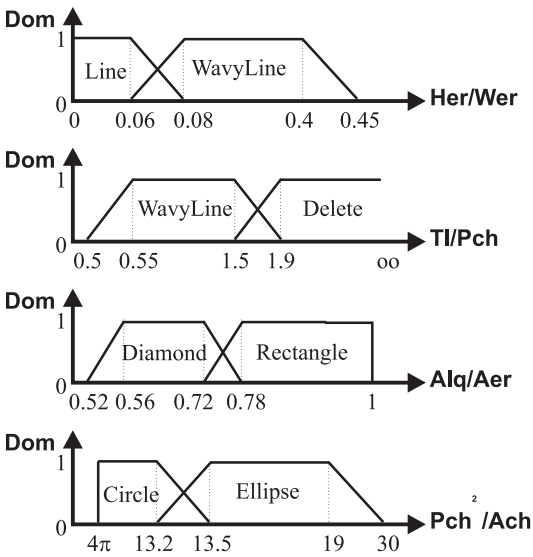

Figure 16. Fuzzy sets representing the ambiguity cases supported by the recognizer.

		Recognized												
	Shapes	Line	Arrow	Triangle	Rectangle	Diamond	Circle	Ellipse	Delete	WaviLine	Copy	Move	Cross	Unknown
Drawn	Line	**97.7**			0.5				1.4	0.2				0.2
	Arrow	0.5	**90.2**	0.3						0.3				8.7
	Triangle	0.3	0.8	**97.8**						0.6				0.6
	Rectangle	0.2	0.5		**96.9**	1.4		0.2						0.7
	Diamond		0.8		8.4	**87.8**		0.3	0.3	0.5				1.9
	Circle				0.3		**97.2**	2.5						
	Ellipse						1.1	**97.6**	0.5					0.8
	Delete					0.4			**98.9**					0.7
	WavyLine	2.0							2.7	**94.6**				0.7
	Copy										**99.6**	0.4		
	Move			3.0								**94.8**		2.2
	Cross												**97.0**	3.0

Figure 19. Confusion matrix (values in percentages).

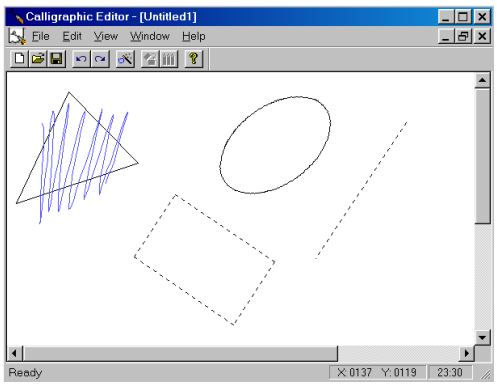

Figure 17. Deleting a shape.

ferent colors to communicate the number of recognized shapes or to denote the degree of certainty of the result. When the result of the recognition process is not unique we should offer an interaction technique to allow circulation among all returned shapes. Mankoff [11] describes some of these mechanisms to deal with ambiguity and errors present in hand writing recognition systems.

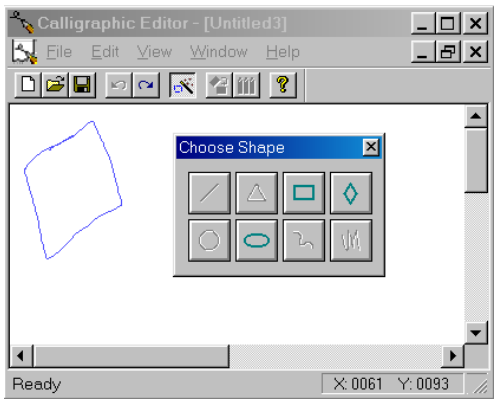

Figure 18. Menu showing the shapes returned.

6. Experimental Results

In order to evaluate the recognition algorithm, we asked nine subjects to draw each multi-stroke shape 40 times, using solid, dashed and bold lines, 30 times each uni-stroke shape and a simple Entity/Relationship diagram with 22 shapes. All these drawings yield a total of 4068 shapes. Subjects were told that the experiment was meant to test recognition, so they didn't try to draw "unnatural" shapes.

We used a Wacom LCD digitizing tablet PL-300 and a cordless stylus to draw the shapes. Two subjects were experts in using pen and tablet while the others had never used a digitizing tablet. We gave a brief description of the recognizer to the users, including the set of recognizable shapes. We also told them about the multi-stroke shape recognition capabilities, the independence of changes in rotation or size and about the set timeout value. Novice subjects had a short practice session in order to get used to the stylus/tablet combination. During the drawing session the recognizer was turned off to avoid interfering with data collection and to avoid any kind of adaptability from the user.

The recognizer successfully identified 95.8% of the scribbles drawn considering just the first shape identified. It is fast: each scribble requires, on average, less than 50 ms (using a Pentium II @ 233 MHz) to be recognized, from feature vector computation to final classification.

A cursory analysis of the confusion matrix, shown in Figure 19, reveals that `Diamonds` are often confused with `Rectangles`, and have the lowest recognition rate. Other shape with a low recognition rate is the `Arrow`. The former is due to the ambiguity between `Rectangles` and `Diamonds` that favours `Rectangles` and the latter is due to incorrect drawing handling. We can also identify other cases of confusion between shapes, such as `Circles` with `Ellipses`, `Moves` with `Triangles` and finally `WavyLines` with `Lines` and with `Deletes`. In fact, the confusion between these shapes is both an acceptable and *intuitive* behavior.

Since ambiguity is one of the main characteristics of our recognizer, we prefer to consider the top three shapes identified instead of the first one. Based on this the recognition rate increases to 97%, showing a good improvement relatively to our previous approach [10] that had a recognition rate of 94% using less shapes and a more complex method.

7. CONCLUSIONS

We have described a simple and fast recognizer for elementary geometric shapes. Our intent was more to provide a means to support calligraphic interaction rather than a totally robust and "foolproof" approach to reject shapes outside the domain of interest. We improved on previous work [10] by increasing recognition rates using fuzzy rules instead of decision trees and introducing resegmentation to identify higher-level patterns such as arrows. We are working on a trainable version of this recognizer to allow us to easily add new shape classes to the core set presented in this paper. One idea is to automatically derive fuzzy sets from training data along the lines of [15] and performing principle component analysis to identify the features relevant to new shape classes.

The high recognition rates and fast response characteristic of this recognizer make it very usable in interactive applications.

REFERENCES

[1] http://www.paragraph.com.

[2] Shigeo Abe and Ming-Shong Lan. Efficient Methods for Fuzzy Rule Extraction From Numerical Data. In C. H. Chen, editor, *Fuzzy Logic And Neural Networks Handbook*, pages 7.1–7.33. IEEE Press, 1996.

[3] Ajay Apte, Van Vo, and Takayuki Dan Kimura. Recognizing Multistroke Geometric Shapes: An Experimental Evaluation. In *Proceedings of the ACM (UIST'93)*, pages 121–128, Atlanta, GA, 1993.

[4] James C. Bezdek and Sankar K. Pal. *Fuzzy Models for Pattern Recognition*. IEEE Press, 1992.

[5] J. E. Boyce and D. P. Dobkin. Finding Extremal Polygons. *SIAM Journal on Computing*, 14(1):134–147, February 1985.

[6] H. Freeman and R. Shapira. Determining the minimum-area encasing rectangle for an arbitrary closed curve. *Communications of the ACM*, 18(7):409–413, July 1975.

[7] D. Golberg and C. Richardson. Touch-typing with a stylus. In *Proceedings of the ACM (InterCHI'93)*, pages 80–87, Amsterdam, 1993.

[8] Mark D. Gross. The Electronic Cocktail Napkin - A computational environment for working with design diagrams. *Design Studies*, 17(1):53–69, 1996.

[9] J. B. Harley and D. Woodward, editors. *The History of Cartography*, volume 1. University of Chicago Press, Chicago, IL, 1987.

[10] Joaquim A. Jorge and Manuel J. Fonseca. A Simple Approach to Recognise Geometric Shapes Interactively. In *Proceedings of the Third Int. Workshop on Graphics Recognition (GREC'99)*, Jaipur, India, September 1999.

[11] Jennifer Mankoff and Gregory D. Abowd. Error Correction Techniques for Handwriting, Speech, and other ambiguous or error prone systems. Technical report, GVU TechReport GIT-GVU-99-18, June 1999.

[12] Joseph O'Rourke. *Computational geometry in C*. Cambridge University Press, 2nd edition, 1998.

[13] Ivan E. Sutherland. Sketchpad: A Man-Machine Graphical Communication System. In *Spring Joint Computer Conference*, pages 2–19. AFIPS Press, 1963.

[14] Charles C. Tappert, Ching Y. Suen, and Toru Wakahara. The state of the art in on-line handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(8):787–807, August 1990.

[15] F. Ulgen, A. Flavell, and N. Akamatsu. Geometric shape recognition with fuzzy filtered input to a backpropagation neural network. *IEEE Trans. Inf. & Syst.*, E788-D(2):174–183, February 1995.