Recognizing Multistroke Geometric Shapes: An Experimental Evaluation

Ajay Apte, Van Vo and Takayuki Dan Kimura

Laboratory for Pen-Based Silicon Paper Technology (PenLab)
Department of Computer Science
Washington University
St. Louis, MO 63130, USA
(314) 935-6122, tdk@wucs1.wustl.edu

ABSTRACT
A fast, simple algorithm for recognizing hand drawn geometric shapes is presented and evaluated. The algorithm recognizes (without regard to size) rectangles, ellipses, circles, diamonds, triangles, and lines. Each shape may consist of multiple strokes as long as they are entered without pauses. A pen-based graphic diagram editor employing this algorithm was developed on GO's PenPoint operating system. The editor will be part of a pen-based notebook system for teaching math to school children. The recognition algorithm was evaluated by ten subjects drawing a total of 200 shapes. It achieved a recognition rate of up to 98%.

KEYWORDS:
Shape Recognition, Pen User Interface

INTRODUCTION: THE PEN USER INTERFACE
Historically user interfaces evolved from Textual (TUI) to Graphical (GUI); now the Pen User Interface (PUI) is emerging [6,8,11]. The direction of the evolution has been from discrete media to continuous media of communication. The input interface technology moved from typing to (mouse) clicking and dragging, and now moves to drawing and gesturing by pen. Typing and clicking are discrete actions; dragging, drawing, and gesturing are continuous actions. Though dragging is a continuous action, it is not comparable to drawing; drawing by dragging is possible but not practical.

Textual representation of information is generally precise though abstract. Visual or graphical representation may be imprecise, but is intuitive and concrete. Visual representation may be unstructured, e.g., icons and pictures, or structured, e.g., diagrams. Examples of diagrams commonly used in computer science are dataflow diagrams, state diagrams, Petri nets, and flowcharts. The process of transforming a concept (idea) in the user's mind into a visual representation should be as direct as possible.

Consider the processes involved in different user interface paradigms to construct a geometric shape such as a rectangle. Entering a rectangle into a computer requires a specification of at least three parameters; type (shape), size, and location.

In a TUI a textual description language is used to define the three parameters. Examples of such language are the pict command in Unix and the resource file in the Macintosh OS. The user must know the syntax and semantics of the description language, and there is no resemblance between the description and what is described; the representation is highly indirect.

In a typical mouse-based GUI the construction process consists of two phases. In the first phase, the type is entered by clicking on a menu item; the other two parameters are entered in the second phase by dragging, continuously tracking the mouse location. In the second phase, the user gets direct feedback of the visual construction (WYSIWYG), even though the mouse movement (the diagonal traverse of the rectangle) does not directly reflect the shape. Rubine [12] proposes a variation of this two-phase operation: The first phase is entered by a gesture; the second manipulation phase starts after a short pause in which the gesture is interpreted, and the mouse begins to drag the object. The merit of Rubine's proposal is that one continuous mouse move is sufficient for entering the three parameters. The drawback is that gesture recognition is neither simple nor perfect [15].

This two-phase approach, either menu-based or gesture-based, is also available in a PUI. However, free-hand drawing of a rectangle is more direct and natural (intuitive) than any other method; the type (shape), size, and location are simultaneously entered without any intermediate representations. What you draw is what you get. The cost of this directness is the complexity of shape recognition. Different users have different styles of drawing; some draw a rectangle with a single stroke, i.e., with a single pen-down and a single pen-up event. Others use multiple strokes. In correcting an incomplete rectangle, some may even use more than four strokes.

ShapeExpert, the drawing component of NoteTaker 1.0.3 of InkWare [5] contains a pen-based shape recognizer which assumes that every shape is entered by a single stroke. Recognizing multistroke shapes is harder than recognizing single-stroke shapes.

Problems in Pen Input Shape Recognition

Most of the research in shape recognition has been related to the field of image recognition. Some of the techniques used for shape recognition include polygon matching based on turning functions[2],convexity of cellular blobs[13];theory of minimum-perimeter polygonal application[14], shape-oriented similarity measure[9], and shape recognition using string-matching[10]. All of these techniques are primarily for the large problems of machine vision and recognition. Our research deals with pen-based input. The complexity of the above techniques are significantly beyond the requirements to recognize simple geometric shapes. In pen user interface, only a small vocabulary of shapes is required to be recognized. The algorithm presented in this paper solves the problem quickly and with minimal complexity.

There are two difficulties involved in multistroke shape recognition: segmentation and incomplete specification. Since the number of strokes used to construct one object is not known a priori, grouping of strokes is difficult when the user enters more than one object in succession as in Figure 1. The segmentation problem is to identify which strokes belong to which objects. The other problem is that every stroke in a multistroke object may not be connected to the other strokes of the object; the shape is not closed — it is incompletely specified, as in Figure 2.

We solved the segmentation problem by introducing a variable length time-out after each stroke. The user pauses before starting the first stroke of the next object. If the time-out threshold is too long, the user has to wait unnecessarily. If it is too short, the user is forced to rush

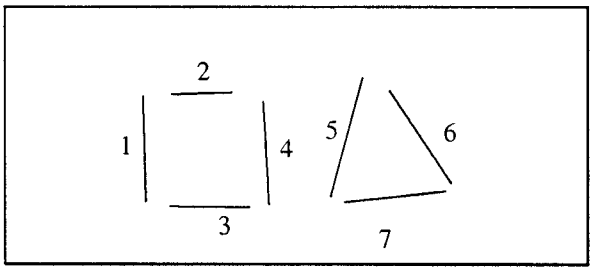


Figure 1: A segmentation problem.

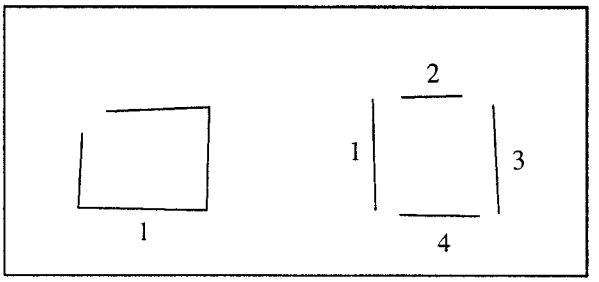


Figure 2: Examples of incomplete specifications.

for the next stroke within an object. This is particularly troublesome in large objects, where the user may have to move the pen some distance to start the next stroke. Our solution to this time-out problem is to make the time-out duration proportional to the length (size) of the preceding stroke. The incomplete specification problem is solved by computing the convex hull of the collected strokes after sorting them in a counterclockwise direction.

The multistroke shape recognition algorithm is incorporated into our Graphic Diagram Editor (GDE) implemented on GO's PenPoint operating system [3]. The editor is a part of a pen-based notebook computer system being developed to teach math to school children [7]. We evaluated the recognition algorithm by asking ten subjects to draw a total of 200 shapes. The algorithm recognized 98% of the shapes.

Graphic Diagram Editor

Our Graphic Diagram Editor (GDE) supports pen-based entry of geometric diagrams [1]. It runs on GO's PenPoint operating system. The objects can be drawn, selected, resized, deleted, copied and pasted. The current version of GDE recognizes rectangles, ellipses, circles, triangles, diamonds, lines and polylines (joined sequence of line segments). Polylines can be drawn using the the editor, however, they do not utilize recognition. The rationale for recognizing these particular shapes is based on the needs of user interfaces and drawing applications. Users want to draw these shapes "cleanly" more often than any other shape. As seen on

any mouse based drawing application, the two most commonly used functions are "draw rectangle" and "draw oval". To the more general problems of machine vision and object recognition, this will probably be less useful.

Figure 3 illustrates sample shapes that can and cannot be recognized by GDE. Note that only lines and circles are rotation independent. All the other shapes must be oriented in parallel with X-axis and Y-axis. GDE also provides gesture support for editing geometric shapes. It also loads and saves the bitmaps (of user-drawn strokes) and the corresponding shapes as recognized.

Multistroke Recognition

The most important feature of GDE is its ability to recognize geometric shapes drawn with multiple strokes. Users may draw objects with as many strokes (in any order) as they like. Only static data points (i.e. XY coordinates) are used for shape recognition. A dotted circle and a solid circle are identically recognized.

The recognition procedure is triggered by a time-out event which is initialized every time the pen is taken out of proximity of the screen. The timer is reset when the pen retouches the screen. Successive strokes of the same object and the terminating stroke are distinguished by the time-out. In the current version, the time-out (time-out=500+4n where n is the number of data points collected) is linearly proportional to the number of data points collected so far so that larger objects will have a longer time-out. The shortest time-out value (500ms) has been set to permit the drawing of very small objects with multiple strokes.

Future experiments will use more complex time-out formulas. For instance, a bell-shaped distribution could be used. At one end, users who make small shapes do not need much time to move the pen between strokes. At the upper end, because there is a limit on the number of points collectable, as that limit is approached, the user is more likely to be finished and would thus desire a shorter time-out.

Shape Editing

The user may choose whether recognized objects will be rendered in bold or normal thickness. Lines and polylines may be optionally be drawn with arrowheads. An object is selected by single tap gesture. A selected object is recognized by the selection handles drawn at its corners. It can be resized, moved, deleted, or copied to the buffer and pasted. A circular gesture copies the selected shape into the buffer. A circular gesture followed by a single tap copies and pastes the selected shape. Thus all editing operations except 'delete' can be performed without accessing a menu. The size or orientation of gestures does not matter. Recognition occurs only when no items are selected.

THE RECOGNITION ALGORITHM

The shape recognition algorithm is based on two ideas. First, because shapes are geometric entities, they should be recognizable solely by their geometry. No stroke information (i.e., order of input data points, speed of input) is used for analysis; only data points are used. Users may draw shapes with as many strokes as they desire. Second, recognition can be achieved by filtering. Each filter categorizes input according to certain criteria. Passing data through filters (three in this case) provides an increasingly refined description of the data.

Setup

Using the input data points, we derive the minimum spanning convex polygon, or convex hull. We then compute the perimeter and area of this polygon for use by some of the filters.

Filters

Each shape employs various filters. Each filter is weighted according to its success in recognizing that shape. For triangles and diamonds, the area-ratio and triangle-diamond filters are used. For circles and lines, only the P^2/A ratio filter is used. And for rectangles and ellipses, the area-ratio and P^2/A ratio filters are used. The weighted outputs are combined to select the final shape.

Area-Ratio Filter

Let A_c be the area of the convex hull defined by the collected points. Let A_r be the area of the coordinate-oriented bounding rectangle. The ratio of the areas is A_c / A_r. Because both triangles and diamonds take up about 50% of the area of the bounding rectangle, an area ratio close to 50% means the object is either a triangle or diamond. Ellipses and rectangles can also be distinguished with this filter; the ratio for rectangles is close to 100%, and the ratio for ellipses is approximately 80%.

Triangle-Diamond Filter

If the shape is a triangle or a diamond, we check the relation of the left and right corners to the rest of the shape by averaging the Y values of those corners. In a triangle, those corners are near the bottom of the shape. In a diamond, those corners are near the middle of the shape. This filter can also detect upside-down triangles since the corners are near the top of the shape. Left-pointing and right-pointing triangles can be recognized by checking the X values of the top and bottom

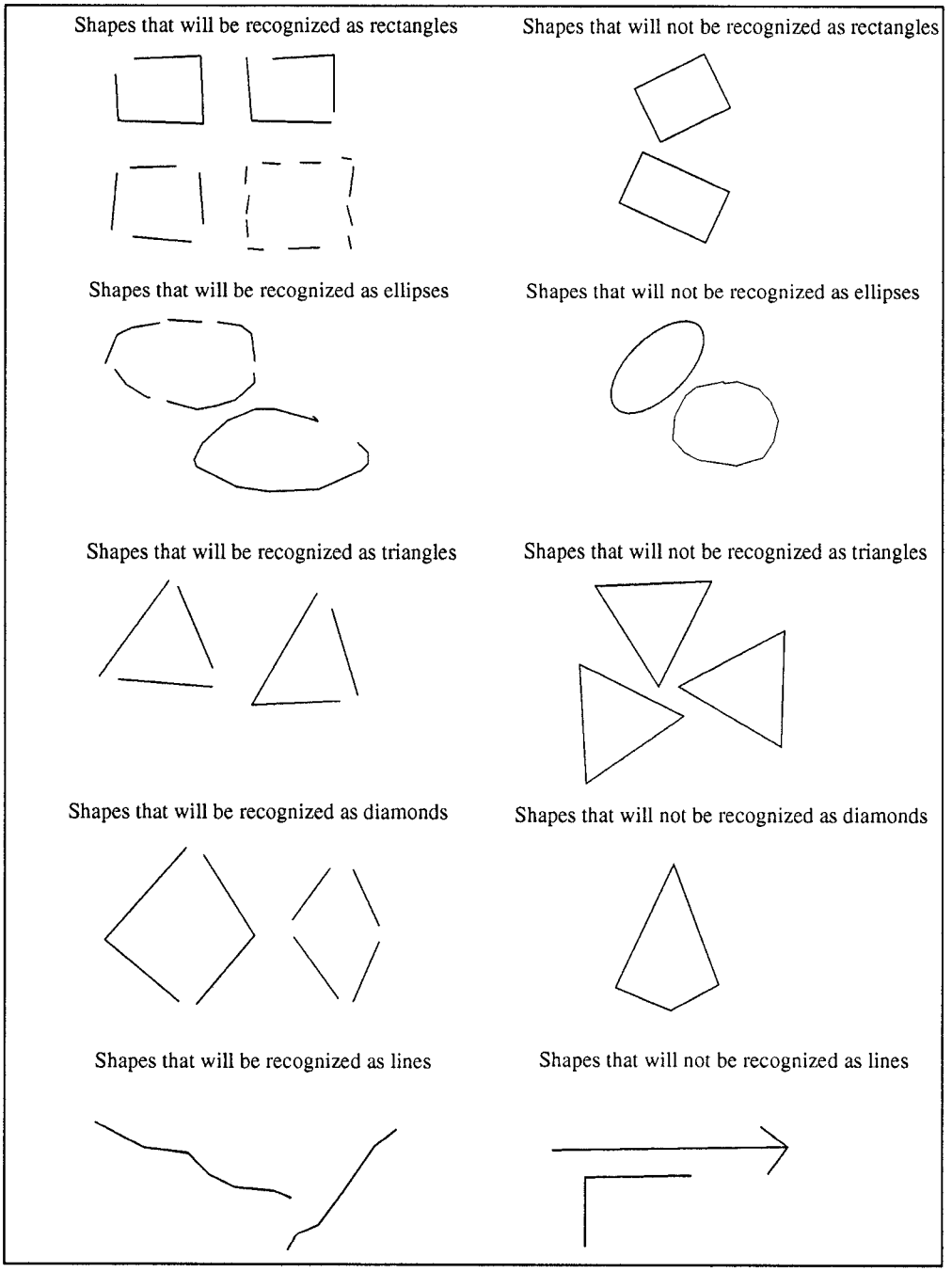


Figure 3: Sample diagrams of various shapes that are and are not recognized by the recognizer

extreme. Currently GDE supports only right-side-up triangles.

P^2/A Ratio Filter

The perimeter (P) and area (A) partly define the shape of an object. These may be related by the ratio P^2/A (also known as compactness[4]). This ratio is a scalar for any shape. For instance, $P^2/A = 16$ for squares of any size. Size independence is important to generalization. Let s = width / height for the bounding rectangle of a given shape. We can extend the P^2/A ratio (referred to as Π) to shapes with $s \neq 1$. Π as a function of s for perfectly drawn shapes is shown in Figure 4. The ratio for lines is theoretically infinite. In practice, we found that lines had ratios greater than 55.

To integrate this filter into the algorithm, we first use the perimeter and area values found during the setup to compute Π_{in} of the input shape. We compute the bounding rectangle of the input shape and then compute s = w / h. We then compute all the theoretical $\Pi_x(s)$ for each shape X (i.e. Π_r, Π_e, Π_d, etc.). The shape of the input object is Shape X, where $|\Pi_{in}-\Pi_x(s)|$ is minimum. For a step-through of the filter, see Figure 5.

Filtering Order

First we check for circles. If $\Pi_x(s)$ is close to 4p then it is a circle because $\Pi_c=4\pi$, which is the smallest Π can

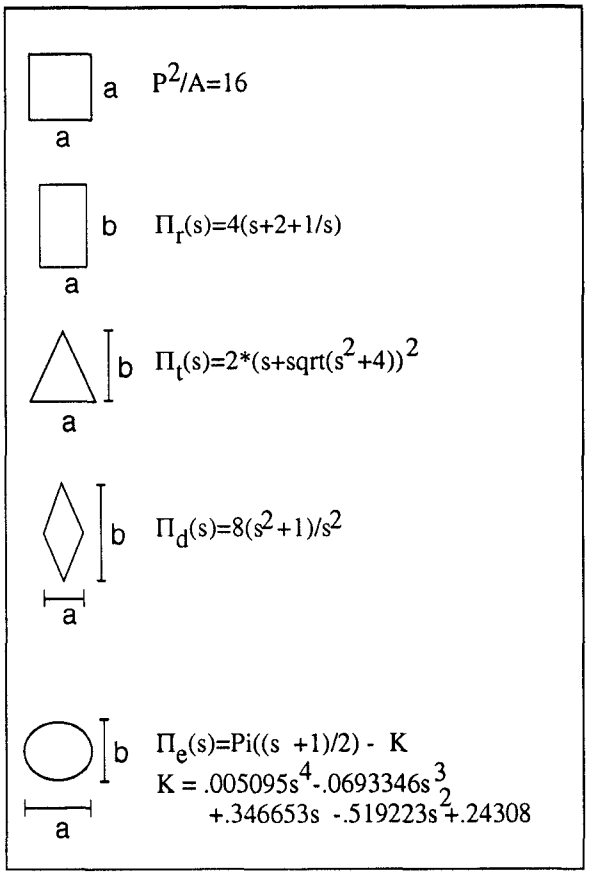


Figure 4: P^2/A ratios for perfect shapes.

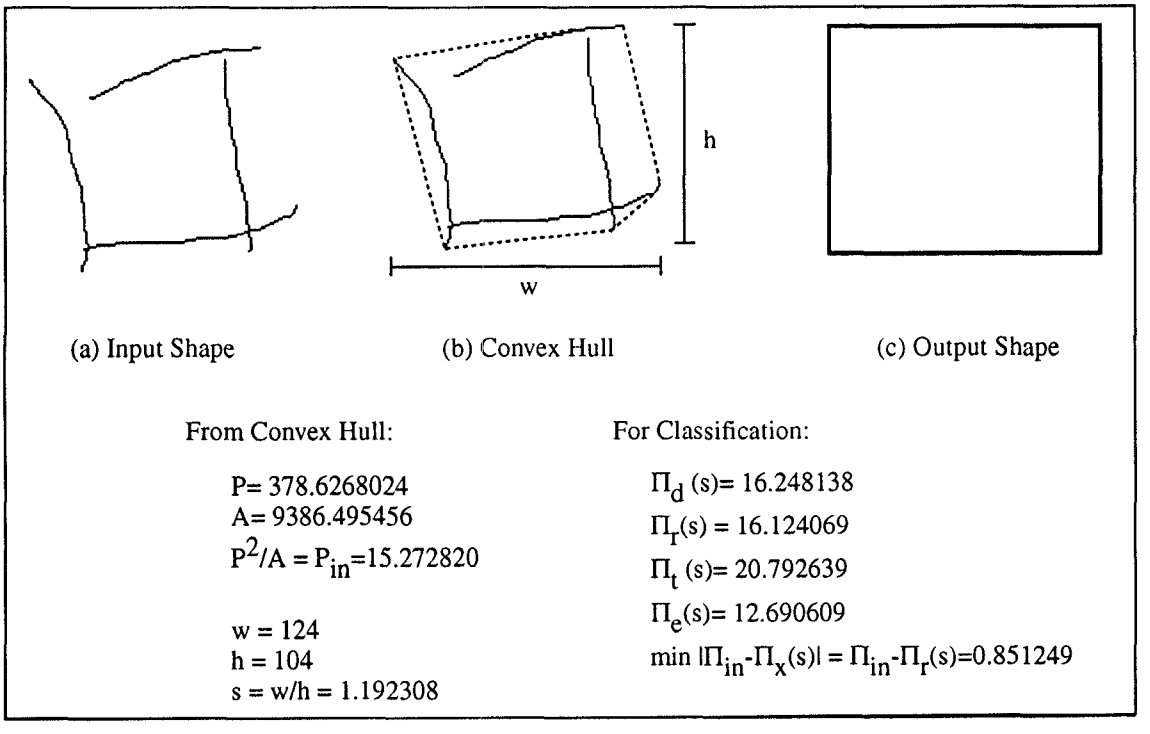


Figure 5: Recognition of a rectangle using P^2/A filter.

ever get both in theory and in practice. Next, if $\Pi_x(s)>55$, then the object is a line. Only very thin objects have Π above 55. We removed the range $35<\Pi_x(s)<70$ from consideration because we considered shapes in the range to be too thin to be a practical shape while too thick to be considered a line. Objects in this range were considered undefined.

If the Area-Ratio value of the object is less than close to 0.5, then we classify it as either a triangle or diamond. The Triangle-Diamond filter makes the final classification.

If the object is still unclassified, then we see if the Area-Ratio is close to 1 or $\Pi_x(s)$ is closer to Π_r than Π_e, resulting in rectangle classification. Otherwise, it is considered an ellipse.

THE EXPERIMENTATION

An experiment was designed to evaluate the shape recognition algorithm. We asked ten subjects to draw two diagrams each with GDE. The subjects were not aware that the experiment intended to test recognition. Thus the results would not be influenced by the subjects' efforts to create unnaturally clear or clumsy shapes.

Object Selection

We used diagrams that included all the geometric shapes recognized by GDE. The two diagrams together included five instances each of diamonds, circles, ellipses, rectangles and triangles, and twenty instances of lines with arrowheads. The shapes were of different sizes. Some of the shapes were embedded inside others. The diagrams given to the subjects are shown in Figure 6.

Experiment Execution

The subjects were given a brief description of GDE, including the recognizable shapes. They were also told about the multistroke shape recognition capabilities of GDE and were encouraged to use them. They were made aware of the time-out feature of GDE encouraging them to draw more quickly. Finally the subjects were given a brief hands-on practice session.

The diagrams we used looked much like program flow charts because one of the applications the recognizer will be used for is a visual language program. For the experiment we wanted to avoid the influence of learning (i.e. one can draw the same shape more correctly a second time). Therefore, the diagrams were constructed to avoid adjacent shape types (i.e. triangles were not next to triangles). The subjects were instructed to draw the diagrams without concern for precision. They knew

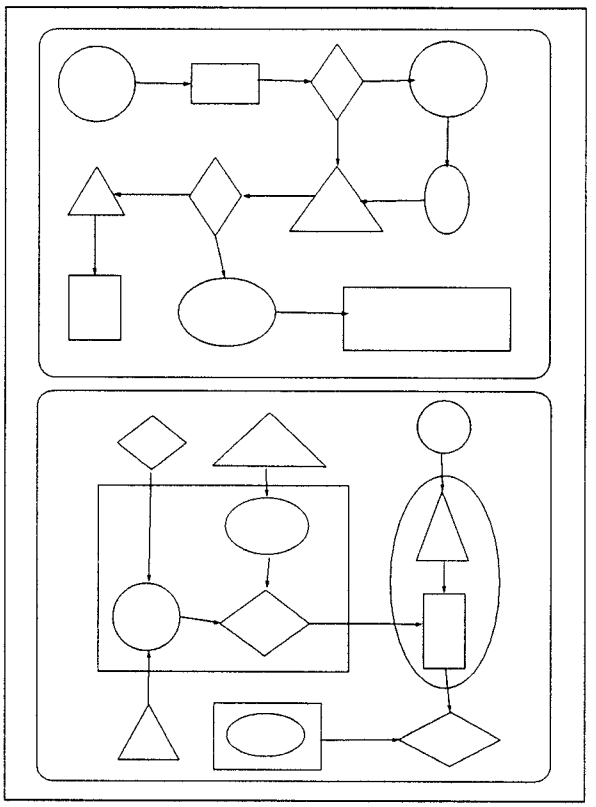


Figure 6: Object diagrams for the experiment.

that they were not being timed. They were also told not to worry about unrecognized or wrongly recognized shapes.At the end of each subject's session, the experimenter captured screen-shots of the strokes and the corresponding recognized shapes. A sample of such

	N	X	% Success
Rectangle	60	1	98
Ellipse	50	4	92
Circle	40	0	100
Diamond	50	0	100
Triangle	50	1	98
Line	200	6	97

Table 1: Recognition statistics by shape.

	N	X(Avg)	%Success
Diagram1	21	20.6	98.09
Diagram2	24	23.2	96.67

Table 2: Recognition statistics by diagram.

screen-shots obtained from two subjects is given in Figure 7.

RESULTS

The recognizer succeeded in 97.5% of the objects. The recognition rate of each shape type is summarized in Table 1 and Table 2. Almost all the subjects drew the shapes with multiple strokes. The poorest results were obtained for ellipses. In all the failures, an ellipse was recognized as a circle. Failure in line recognition was caused by the inclusion of arrowheads (contrary to instructions) which caused the line to be recognized as a diamond.

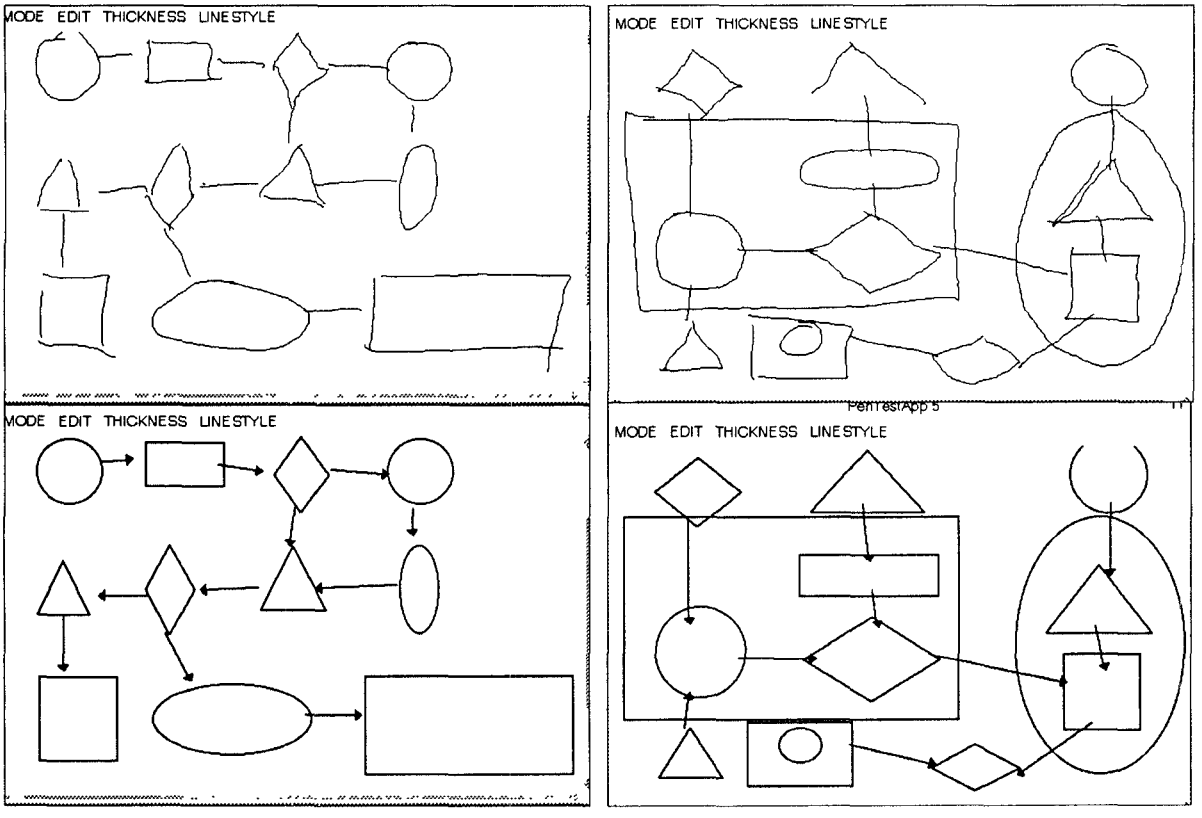
Failures for rectangles and triangles are attributed to the time-out. The time-out set up on GDE was simply too fast for some subjects. In such cases, the algorithm recognized the collected data as two shapes.

CONCLUSIONS

The recognition algorithm recognizes six types of geometric shapes. The same algorithm can be used to recognize regular polygons of any size. However, this feature is not implemented in the current version of GDE. Recognition of regular polygons works best with small dimension.

The Ratio-Area filter distinguishes triangles and diamonds from other shapes exceptionally well. Only rectangles rotated at least 31.5 degrees would fall into the 0.65 threshold for diamonds and triangles. However, this test does not handle rectangle/ellipse recognition as well. Just a 13.5 degree rotation of a rectangle marks it as an ellipse. Even rectangles with rounded edges are correctly recognized.

The P^2/A filter provides finer distinctions. It can easily be computed independently of orientation. However, the theoretical value is based on a width to height ratio which is limited to the screen orientation. Therefore, the orientation of the drawn shape is important. A rotated rectangle will have both a larger width and a larger height, providing a different width/height ratio to be used in finding the theoretical P^2/A ratio. However, if two shapes have theoretical values that are significantly different, then more distortion can be tolerated. Thus, orientation of the geometric object is critical in the recognition process; only circles and lines are orien-



(a) (b)

Figure 7: Snapshots of diagrams drawn by the subjects and the corresponding recognized shapes.

tation independent. Further investigation may help overcome this dependency.

The most important feature of our shape recognizer is multistroke recognition of geometric objects. Users naturally draw geometric shapes with multiple strokes. Our algorithm does not have an upper bound on the number of strokes used to draw the shapes. Also there is no relation between the number of strokes and the size of the geometric object.

The most sensitive factor in multistroke shape recognition is the computation of the time-out. In the current version of GDE, the time-out is linearly proportional to the number of data points collected in the previous strokes of the current shape. This equation failed only twice in our experiment. However, we feel that this is not necessarily the best algorithm for time-out. For cases where a large number of data points are collected (either due to bigger shapes or excessive scribbling by the users), the time-out is acceptably high. We are trying other time-out equations. For example, the time-out might be linearly or exponentially proportional to the number of data points collected, until a certain threshold is reached, after which it remains constant. We need more experimentation in this area.

Acknowledgement
We wish to thank all the subjects for participating in the experiment. We express our gratitude to Ed Hutchins for his ideas on the shape recognition algorithm and Tom Fuller for editing the manuscripts. Finally, we would also like to thank all the members of Kumon team for their co-operation.

References

1. Apte, A. and Kimura, T.D. "A Comparison Study of the Pen and the Mouse in Editing Graphic Diagrams," Submitted to VL'93, Bergen Norway, August 1993.

2. Arkin, Esther M., et. al. "An Efficiently Computable Metric for Comparing Polygonal Shapes," IEEE Transactions on Pattern Analysis and Machine Intelligence. Vol 13, No 3, March 1991. pp. 209-215.

3. Carr, R. and Shafer, D. The Power of PenPoint. Addison-Wesley, 1991.

4. Haralick, Robert M. and Linda G. Shapiro. "Glossary of Computer Vision Terms," Pattern Recognition. Vol 24, No 1, 1991. pp. 69-93.

5. InkWare NoteTaker User Guide, InkDevelopment Corporation, 1992.

6. Kimura, T.D. "Silicon Paper and A Visual Interface for Neural Networks," Proceedings of 1990 IEEE Workshop on Visual Languages, Chicago, IL, October 1990, pp. 241-246.

7. Kimura, T.D. "Learning Math with Silicon Paper," Technical Report WUCS-92-11, Department of Computer Science, Washington University, St. Louis, March 1992.

8. Kimura, T.D. "Potentials and Limitations of Pen-Based Computers," Panel session chiarman's position paper," Proceedings of 21st ACM Annual Computer Science Conference(CSC'93), Indianapolis, February 1993, pp. 536-537.

9. Lee, E. T. "The Shape-Oriented Dissimilarity of Polygons and Its Application To the Classification of Chromosome Images," Pattern Recognition. Vol 6, 1974. pp. 47-60.

10. Maes, Maurice. "Polygonal Shape Recognition Using String-Matching Techniques," Pattern Recognition. Vol 24, No 5, 1991. pp. 433-440.

11. Mahach, K.R. "A Comparison of Computer Input Devices: Linus Pen, Mouse, Cursor Keys and Keyboard," Proceedings of the 33rd Annual Meetings of the Human Factor Society, 1989, pp. 330-333.

12. Rubine, D.H. "The Automatic Recognition of Gestures," (PhD Thesis) Technical Report CMU-CS-91-202, School of Computer Science, Carnegie Mellon University, Pittsburgh, December 1991.

13. Sklanski, J. "Recognition of Convex Blobs," Pattern Recognition. Vol 2, 1970. pp. 3-10.

14. Sklanski, Jack and Victor Gonzalez. "Fast Polygonal Approximation of Digitized Curves," Pattern Recognition. Vol 12, 1980. pp. 327-331.

15. Taysi, B. "Gesture Systems for Graph Editor," (MS Thesis) Technical Report WUCS-92-35, Department of Computer Science, Washington University, St. Louis, December 1992.