Recognizing Simple Geometric Figures Using Fuzzy Expert System

Zhang Linghao
13307130225@fudan.edu.cn

1 Introduction

This is the report for the project of ***Introduction to Artificial Intelligence*** *course (2015 Fall) at Fudan University.*

Our task is to build a rule-based expert system to recognize simple geometric figures, such as circles, triangle, rectangles, etc. In this report, we describe the design and implementation of a fuzzy expert system for solving the task, evaluate its performance, and finally discuss its limitations.

For the ease of narration, this report will use *we* as the first person pronoun and present tense with all verbs.

2 Image Generation

2.1 Overview

Training images are vital to our task. Computer generated images are too standard to contain enough noise, making our model vulnerable to possible noisy test images; hand-drawn sketches are too noisy, on the other hand, raising challenges for image feature extraction and complicating our rules. Both senarios could diminish the generalization ability of our model. Therefore we use the following three different kinds of images:

First in the system prototyping stage, we use a painting tool to draw several images of **circles, ellipses, triangles, rectangles and squares** as the **prototyping set**. Note that these images are not generated in vector format on purpose, thus introducing small noises at a pixel level. We hope that by analyzing these quasi-standard images, we can derive the basic framework of our system.

Then we use image processing techniques to distort and rotate the standard figures, obtaining a great amount of images and split them into **training set, developing set and testing set**. Now we can use the training set to determine the fuzzy sets and the developing set to tune the fuzzy sets and rules based on error analysis.

Finally in the testing stage, we use the testing set to evaluate our system. We also ask different people to **draw sketches with a digitized tablet** for measuring performance. We decide not to use mouse because that would produce overly noisy images.

Note that for the convenience of development, **all images we use only contain one shape except for the sketch images**. In other words the final system is intended to perform well with sketches containing multiple (non-overlapping) figures.

2.2 Details

2.2.1 Computer-generated Images

Images are generated as follows:

First we generate a random image of the target shape. Then we **distort the contour** of the figure by applying a Sine function on each row and column of the image, with a given factor to control the distorting effect. At last image is **rotated** by a random angle.

Finally we obtain 2000 images for each shape, and we split them into **training set**, **developing set** and **testing set** with a 8 : 1 : 1 ratio. The images look like this. (Figure 1)

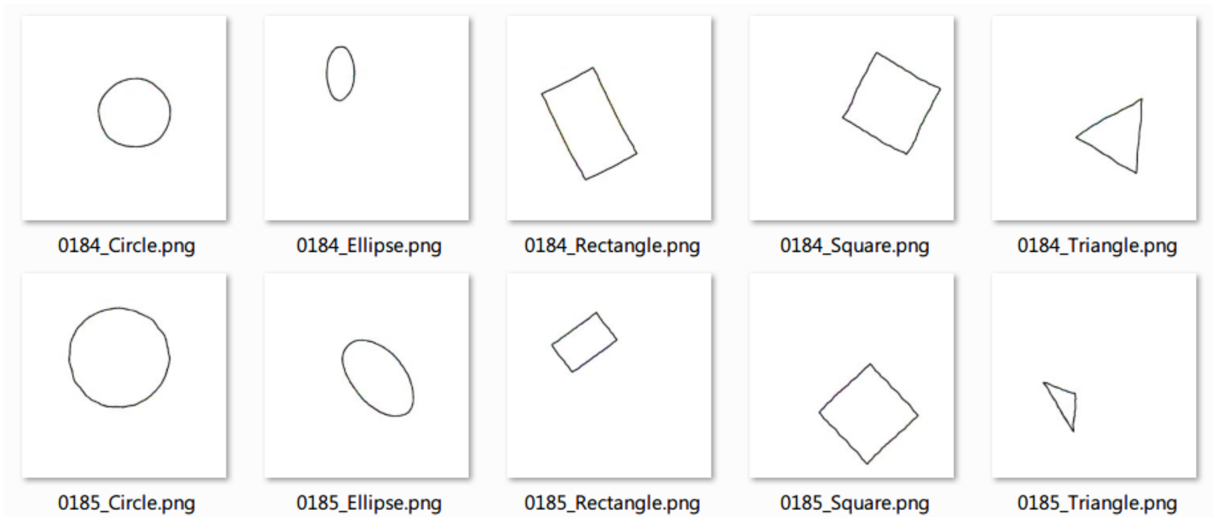


Figure 1: Computer-generated Images

2.2.2 Hand-drawn Sketches

The sketches are drawn by two difference people. They look like this. (Figure 2)

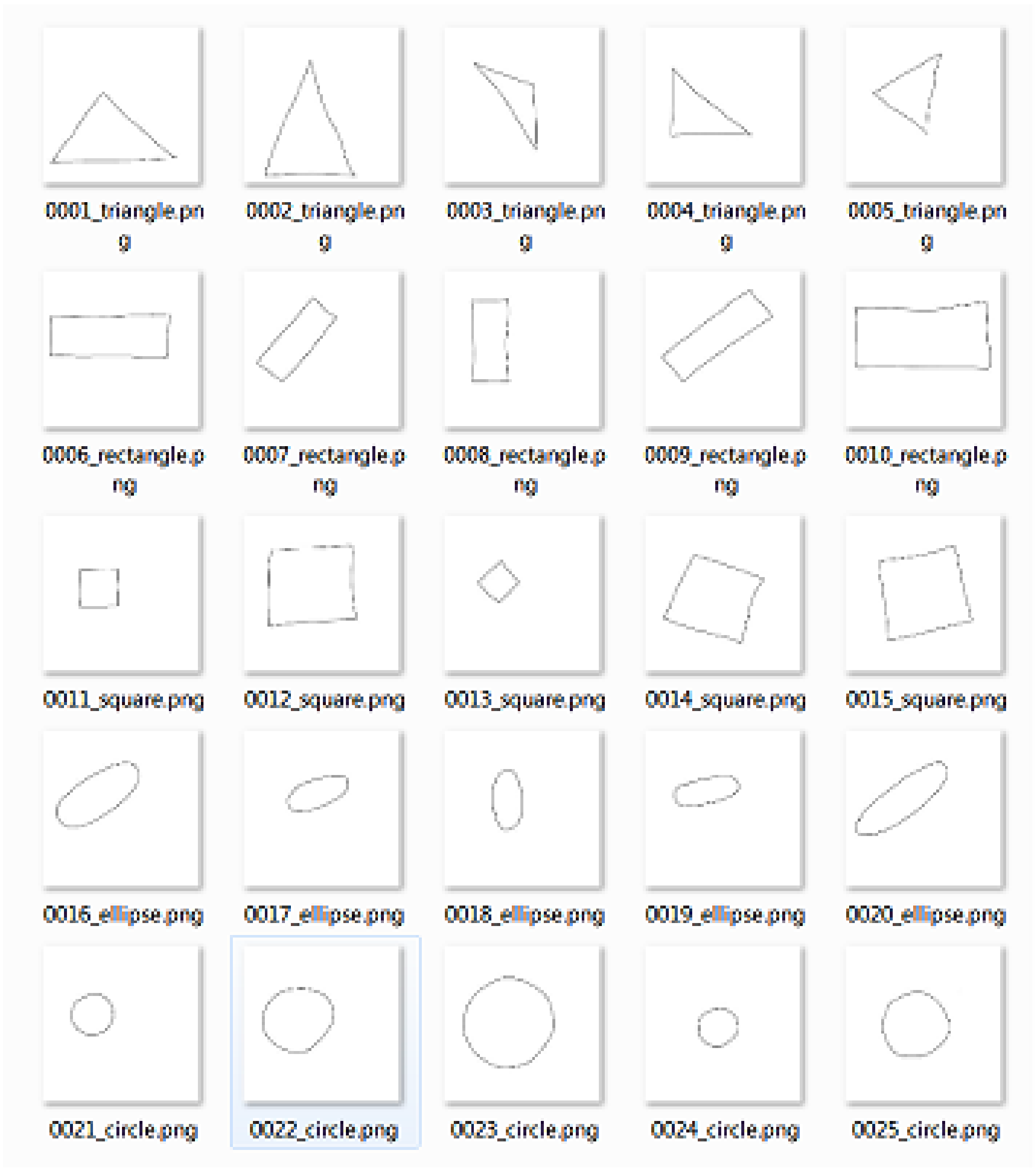


Figure 2: Hand-drawn Sketches

3 Feature Extraction

Feature extraction is mainly done with OpenCV.

3.1 Contours

Contours are probably the most starightforward feature that one can think of. With the help of OpenCV, once we've found the **contour** of the target figure, we can calcuate the following:

- Perimeter: apply cv2.arcLength on the contour gives us an approximation of the perimeter.
- Area: apply cv2.contourArea on the contour gives us an approximation of the area.

3.2 SIFT

SIFT allows us to easily find key points (corners) in the image. At first glance it seems to be a great feature, since it can distinguish circles / ellipses from triangles and rectangles / squares. However, the extraction of this feature is highly unstable. A little noise (e.g. a zigzag line) in the image could increase the number of key points by more than 20. Although it's possible to set some thresholds to tune out the undesired key points, still we couldn't put too much confidence on this feature.

3.3 Feature Expansion

In computer vision, when it comes to geometric figures, there are several frequently used features that we can easily obtain using the existing features. Some of them turn out to be useful for our task:

- $Thinness = \frac{P^2}{A}$: Thinness measures the degree to which the figure is "economical" in using its perimeter to enclose an area.
- $Extent = \frac{A}{A_{br}}$: Extent measures the how much space the figure takes up in its bounding rectangle (the rectangle of the smallest area that bounds the target figure).

3.4 Autonomous Feature Discovery

We will discuss the possibility of autonomous rule derivation in Section 7.

4 Rule Derivation

Rules derived are stored in *rule_base.txt*, together with the definitions of fuzzy sets.

4.1 Manual Rule Derivation

After observing a couple of patterns from data in the prototyping set and reasoning about their validities, we come up with several heuristics.

First we notice that *Extent* is a very good feature to start with: triangles have a *Extent* ratio close to 0.5, rectangles and squares have a *Extent* ratio close to 1, while circles and ellipses have a *Extent* ratio of around $0.75 \sim 0.85$.

Then we can use *Thinness* ratio to distinguish circles and ellipses, because circle has the smallest thinness ($= 4\pi$) among all 2D figures,.

Similarily, when rectangles have *Thinness* ratio close to 16, they become squares.

For the feature *Corners*, as mentioned in Section 3.3, it's a very shaky feature for our task. Therefore we decide not to use it.

Finally, we obtain 5 simple rules:

1. *IF Thinness IS LIKE Circle AND Extent IS LIKE ellipse THEN Shape IS Circle*
2. *IF Thinness IS NOT LIKE Circle AND Extent IS LIKE ellipse THEN Shape IS Ellipse*
3. *IF Extent IS LIKE Triangle THEN Shape IS Triangle*
4. *IF Thinness IS LIKE Square AND Extent IS LIKE rectangle THEN Shape IS Square*
5. *IF Thinness IS NOT LIKE Square AND Extent IS LIKE rectangle THEN Shape IS Rectangle*

4.2 Autonomous Rule Derivation

We will discuss the possibility of autonomous rule derivation in Section 7.

5 Fuzzy Inference

A simplified Sugeno-style fuzzy inference system is developed with functionalities for fuzzy rule / sets parsing, input fuzzification, autonomous inferencing and output defuzzification.

5.1 Determining Fuzzy sets

As mentioned in the previous section, the noises in the image inevitably introduces uncertainties to our system. Therefore the feature values referenced in our rules have to be represented in fuzzy sets.

To determine the **best fuzzy sets capturing the appropriate degree of uncertainties**, we use the statistics from the training set to find the optimal threshold.

For *Thinness*, we use 3 fuzzy sets for ellipse-like, rectangle-like and triangle-like *Extent* ratio. For *Extent*, we use 2 fuzzy sets for circle-like and square-like *Thinness* ratio. The shapes and boundries of these fuzzy sets are determined based on the **distribution and percentiles of feature values** in the training set. We consider the area between 25th percentile and 75th percentile as the **high confidence range**, while the remaining area plus a small portion of area adjacent to the extreme values (to tolerate some outliners) are considered as the **low confidence range**. The corresponding polygons are then constructed based on these two ranges to represent the fuzzy sets. The final fuzzy sets are illustrated in Figure 3 and Figure 4 (Figures not drawn to scale).

5.2 Implementing Inference Engine

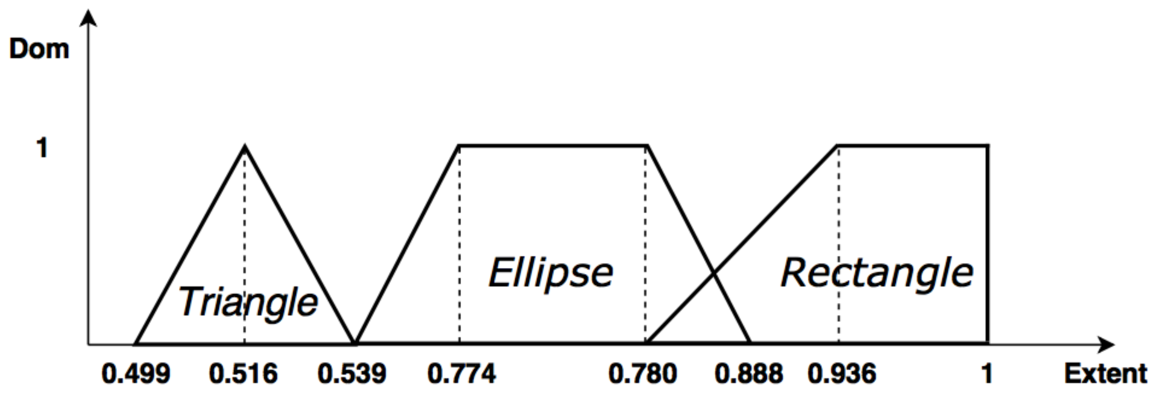
Now that the fuzzy sets are determined, we have to implement the inference engine. But the specific nature of our task requires us to make some adaptations to the the usual fuzzy inference engine.

Firstly, unlike the example from the textbook, the set of possible outputs in our task is **NOT** a set of linguistic values which can be represented by a series of contiguous intervals. We can't compute a COG-like output and decide which category it falls in, because we can't assign a reasonable order to the shapes.

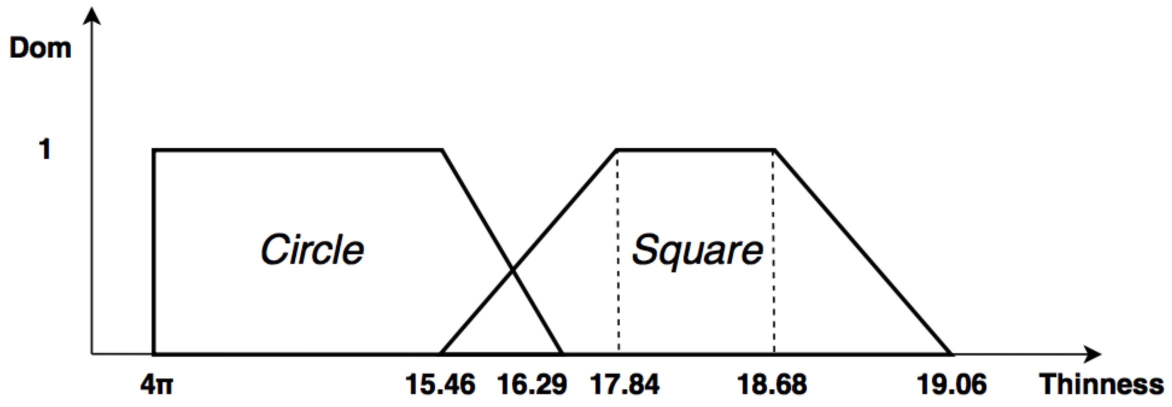
Secondly, rules derived in the previous section are all of the format:

IF X IS LIKE A AND Y IS NOT LIKE B THEN Shape IS C

Note that the consequent of each rule is an assertion about the shape of the input figure and that **each rule corresponds to the recognition process for one specific**



(a) Fuzzy Set of Extent



(b) Fuzzy Set of Thinness

Figure 3: Fuzzy Sets

shape. In fact a fuzzy inference system of this kind is essentially **a flattened decision tree**, with **each rule acting as a filter to calculate the possibility of one specific shape**. The only difference is that in our expert system rules are separated from inference, making it easily maintainable.

Thus, in order to gain the advantages brought by fuzzy inference without over-complicating our task, we choose to implement **a simplified Sugeno-style inference engine**. In the final defuzzification stage, we don't compute a weighted average of the rule outputs. Instead we examine all the possible values for the target variable *Shape* and **choose the one with the highest possibility as the output**.

6 Evaluation

The evaluation consists of two parts: one with computer-generated test set, one with hand-drawn sketches.

6.1 Computer-generated Test Set

6.1.1 Result on Developing Set

The confusion matrix for the evaluation on the development set is in Table 1. The developing set contains 1000 figures with 200 figures from each category.

Recognized / Label	Circle	Ellipse	Triangle	Square	Rectangle
Circle	**0.55**	0.03	0.42	0.00	0.00
Ellipse	0.12	**0.82**	0.06	0.00	0.00
Triangle	0.00	0.00	**0.99**	0.00	0.01
Square	0.00	0.00	0.03	**0.79**	0.19
Rectangle	0.00	0.00	0.04	0.39	**0.56**

Table 1: Confusion Matrix on Developing Set

6.1.2 Error Analysis and Tuning

First we see that nearly a half of circles are incorrectly recognized as triangles. Upon examination we find that these circles actually do not fall into any fuzzy sets because of their larger-than-expected *Thinness* ratio. Sorting on the list with each shape's score

= 0, a unexpected side effect happen that the letter “T” is alphabetically the largest. This problem can be fixed by making sure that **all fuzzy sets are interleaved and no blank area is left**, as what we do in this case of circles vs. squares.

Second we see that only 82% ellipses are correctly recognized. The issue here is similar to the first one, but slightly different with the solution. The *Extent* ratio for triangles and rectangles are quite centralized around 0.5 and 1.0. Therefore we can **safely expand the confidence range** of ellipses.

Finally We see that squares and rectangles are sometimes mixed up. This makes sense because some of rectangle images in our data do look like squares. If we are to deal with this type of error, we may consider using new features, such as identifying and comparing all the sides in the shape.

Besides, there are about 5% figures showing a *Extent* ratio of 1 due to image processing errors. When these figures happend to be non-rectangles, they will be recognized as triangles as explained above. The reason for the errors are currently unknown.

6.1.3 Result on Testing Set

After applying the necessary tweaks to the “parameters” (boundaries of fuzzy sets), we evaluate our system on the testing set, which also contains 1000 figures with 200 figures from each category. The confusion matrix is in Table 2.

Recognized / Label	Circle	Ellipse	Triangle	Square	Rectangle
Circle	**0.96**	0.01	0.03	0.00	0.00
Ellipse	0.08	**0.90**	0.02	0.00	0.00
Triangle	0.00	0.00	**0.99**	0.00	0.01
Square	0.00	0.00	0.00	**0.90**	0.10
Rectangle	0.00	0.01	0.03	0.39	**0.58**

Table 2: Confusion Matrix on Testing Set

Apart from the squares and rectangles mixing up, we notice that circles and ellipses also mix up due to similar reasons: some of ellipse images in our data do look like circles. But as this is largely due to the **inherent ambiguity in the images we use**, it would not be a serious issue when testing against hand-drawn sketches. Because **sketches are usually exaggerated in some ways to avoid double interpretation**, unless the person who draws them intentionally make them very ambiguous or such a distinction is no necessary.

6.2 Hand-drawn Sketches

As the final evaluation of our system, we feed about 55 sketches drawn by two people to the recognizer. The sketches include 1 $\sim$ 3 figures each, and contain in total 20 figures for every category. The confusion matrix is in Table 3.

Label \ Recognized	Circle	Ellipse	Triangle	Square	Rectangle
Circle	**20**	0	0	0	0
Ellipse	1	**19**	0	0	0
Triangle	0	0	**20**	0	0
Square	0	0	0	**18**	2
Rectangle	0	0	0	5	**15**

Table 3: Confusion Matrix on Sketches

We can see that performance is pretty good. There's only one problem left: sometimes squares are recognized as rectangles. The reason seems to be that value of $Thinness$ ratio is somewhat very sensitive to the input image.

7 Discussion

This section discusses the limitations (mainly about the expandability of the system) of our project and how it can be improved.

7.1 About Image Processing

Currently we do not support images which contains overlapping shapes due to the fact that dealing with overlapping shapes are very difficult.

Another thing is that for the feature extraction to work smoothly, we have several assumptions. Basically the contour lines in the image cannot be too thin, otherwise OpenCV would have problems finding the desired contour. While it is possible to apply some image transformations before feature extraction, it still remains a quite complicated (and out of the scope of this project) task to handle all kinds of visually valid images. Therefore we will **only use thick-lined images** in our project.

7.2 Limited Set of Shape Categories

Currently we can only recognize between 5 figures. To incorporate more figures into the system, we would have to go through the same workflow all over again. It would be tedious and new questions can arise. For instance we may find our current features are not sufficient to achieve a satisfactory performance. We may have to add new rules, change current rules, or reconstruct some (or even all) fuzzy sets. Although the separation of rule base and inference engine does help with system update, updating rule base alone would be quite time-consuming. Therefore it would be desirable to have **autonomous rule derivation** and **autonomous feature discovery**.

- Technically, we can generate more features based on current ones, as long as they are useful for in our task. Currently features are selected based on **geometric knowledge (maybe "common sense" is a better word) and observation of the training data**. It is entirely possible to apply some machine learning techniques to autonomously generate and select new features. Here we experiment with genetic algorithms.

Under development.

With the second problem, I believe we could utilized genetic algorithms. A new feature could be represented by the product of some old features raised to a certain power. So we can encode the exponets of each base feature into a chronosome. And the fitness could be computed by testing how much information gain (as in the decision tree) could be achieve by using the feature encoded in the chronosome.

7.2.1 Deriving New Rules

Since rules and inference are separated in our system, there's no doubt that if we obtain some new rules we could just add them to the system. Therefore the main challenge here is how to derive these rules. Currently, we don't have a fully automatic method to do this. Rules must be derived with human intervention. However, we could **make this process a lot easier by using a decision tree model to help identify features that have discriminative power**, as used for the fitness function of the genetic algorithm described in section 3.4.

7.2.2 Discovering New Features

Another problem is how to generate new features. Currently we use a subset of commonly used features. Of course we could use the whole set and select the useful ones, but that leaves us with two problems: 1. How to identify the most useful features? 2. How to integrate newly found features into our system automaticly?

With the the first problem, we experiment with genetic algorithms as a way to generate features outside the original feature set. Another (better?) approach is to use the whole feature along with all the derivatives and perform **feature selection** techniques (e.g. LASSO).

The second problem is actually the rule derivation problem paraphrased. It's very hard to entirely eliminate human intervention from this process. Either we **incorporate the feature into some existing rule** after validating its discriminative power, or we just **add a new rule**, then the inferencer would deal with multiple rules itself. Yet **how to combine outcomes of differnet rules** would become a new "parameter" that may need training and tuning.

8 References

1. Joaquim A Jorge and Manuel J Fonseca. A simple approach to recognise geometric shapes interactively. In *Graphics Recognition Recent Advances*, pages 266–274. Springer, 2000

2. Manuel J Fonseca, Joaquim Jorge, et al. Using fuzzy logic to recognize geometric shapes interactively. In *Fuzzy Systems, 2000. FUZZ IEEE 2000. The Ninth IEEE International Conference on*, volume 1, pages 291–296. IEEE, 2000