1-1-2007

Center of gravity guided signature of planar shapes

Hina Jain
University of Nevada, Las Vegas

CENTER OF GRAVITY GUIDED SIGNATURE OF PLANAR SHAPES

by

Hina Jain

Bachelor of Engineering
Information Technology
B.M. Institute of Engineering and Technology
India
May 2005

A thesis submitted in fulfillment
of the requirements of the

Master of Science Degree in Computer Science
School of Computer Science
Howard R. Hughes College of Engineering

Graduate College
University of Nevada, Las Vegas
December 2007

UNLV
UNIVERSITY OF NEVADA LAS VEGAS

Thesis Approval

The Graduate College
University of Nevada, Las Vegas

NOVEMBER 14TH , 20 07

The Thesis prepared by

HINA JAIN

Entitled

CENTER OF GRAVITY GUIDED SIGNATURE OF PLANAR SHAPES

is approved in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCES

Examination Committee Chair

Dean of the Graduate College

Examination Committee Member

Examination Committee Member

Graduate College Faculty Representative

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

Center of Gravity Guided Signature of Planar Shape

by

Hina Jain

Dr. Laxmi P. Gewali, Examination Committee Chair
Professor of Computer Science
University of Nevada, Las Vegas

Measuring the similarities between two planar shapes is a complex problem. A notion of calculating the signature of a planar shape has been proposed. This signature is a unique feature of the planar shape that differentiates it from other planar shapes. Moreover, the comparison of signatures of two planar shapes helps in determining the degree of similarity between them. In past, researchers have tried to propose effective algorithms to compute the signature of the planar shapes. O'Rourke introduced the concept of signture of simple polygons for measuring similarities between two dimensional shapes. We propose to model a generalized notion of signature by considering the center of gravity of polygons. Standard signature is determined by considering the half plane through the edges of the polygon. In the generalized model, we propose to measure signature by considring half plane through the center of gravity of polygons and parallel boundary edges.

ACKNOWLEDGEMENT

I would like to express my gratitude for many individuals whose assistance, support and suggestions has helped me in completing this thesis. I would like to express my utmost gratefulness and appreciation to my advisor and mentor, Dr. Laxmi P. Gewali for his unmatchable assistance, encouragement, patience and motivation. I would also like to thank Dr. David Pinelle, Dr. Yoohwan Kim and Dr. Rama Venkat for their continued cooperation and encouragement on my thesis work. Last but not least, a very special thanks to my brother, Varun Jain and my family for their guidance, sacrifices and patience throughout my course of study. Without them this thesis would not have been possible.

CHAPTER 1

INTRODUCTION

Finding similarities between shapes is a common problem in image processing, pattern recognition, robotics, and environmental science. Often satellite images are analyzed for searching pattern of interest such as forest cover, water shades, fault-lines, erosion, etc. In medical science, shape similarity techniques are used for processing X-ray and NMR-images. In robotics, shape recognition is used by a robot to traverse a path in the presence of obstacles.

Shape recognition algorithms have been considered mainly by (i) image processing and (ii) computational geometry research communities[1,4,7,8]. In image processing a shape consists of pixels and voxels and the similarity measurement is done by seeking global shape properties such as size, perimeter, elongation, and compactness. Additional preprocessing techniques such as filtering, smoothing etc., are considered for shape analysis. While the shapes of two dimensional images are pre-processed for extracting the boundary, the exact spatial model of such shapes is prominently considered in computational geometry. One of the simplest model of 2-D shapes in computational geometry is the polygon, which is described by listing the coordinates of its boundary in the order of their occurrences on the boundary. Polygonal models of shapes have been considered by many researchers for comparing shape similarity[1,4]. One of the early works in the shape similarity of polygonal models is the notion of signature introduced by O'Rourke

[4]. The notion of the signature of polygon can be extended in a straightforward manner to open curves in two dimensions. Intuitively, the signatures of polygons are much simpler than their boundaries, and at the same time they retain some structural properties of the polygon. It has been found that signatures of polygons can be used with some success in capturing the similarity between hand written characters [4].

Another technique from computational geometry that is used for finding similarity between polygonal shapes is the notion of turning function[1]. In the method of turning function, the boundary of the polygon is mapped to a step-function called its "turning function". Like signatures, turning functions are simpler to compare than the boundary of a polygon. At the same time, the step-function contains significant global features for comparing shape properties.

Use of polygonal models for classifying contours of breast tumors is reported in [8]. For this process, turning function is used to extract some special features from the tumors [8]. Application of polygon similarity measure has been used for the retrieval of vertebral images [7].

In this thesis, we introduce a variation of the standard signature technique, which we have termed as *CG Guided Signature*. Unlike the technique of the standard signature, the proposed method is effective in distinguishing convex polygons of different shapes. An overview of the thesis is as follows. In Chapter 2, we present a brief review of computational geometric techniques for measuring shape similarity. One of the techniques we review is the method of signatures of planar shapes. The other technique we review is the method of turning function. In Chapter 3, we formally introduce the notion of cg-guided signature for polygons. We develop the notion by considering the center of gravity of the polygon for defining the signature of a shape. The cg-guided notion can be viewed as

a variation of the standard notion of signature introduced in [4]. It is found that the cg-guided signatures are effective in distinguishing convex polygons of different shapes. We present an $O(n^2)$ time algorithm for computing the cg-guided signature of polygons. We also present a linear time algorithm for filtering noise edges from a polygon which can be taken as a pre-processing step for signature computation. In Chapter 4, we present an implementation and experimental results of the proposed cg-guided signature. In Chapter 5, we discuss the findings of the experiment, possible extensions of the proposed technique, and the scope for future research.

CHAPTER 2

PRELIMINARIES AND LITERATURE REVIEWS

2.1 Introduction

A shape in two dimensions can be represented by a simple polygon. From computational geometry, it is known that a simple polygon is formed by connecting line segments (edges) such that no two edges intersect in the interior of the polygon. Comparing shapes in two dimensions is a complex problem. Over the years, researchers have tried to come up with an acceptable metric for comparing planar shapes. Geometric techniques that include signature[1] and turning function [4] have been proposed to calculate shape properties and their comparison.

While comparing shapes, it is necessary to develop the notion of "distance" between two shapes. The distance should be defined in such a way that between very similar shapes it is very small and between shapes that do not have any similarity it is very large. As observed in [1] the distance function $d(.,.)$ should satisfy several desirable properties that include:

a. It should be a metric such that the distance between two identical shape should be zero i.e., $d(A,A) = 0$.

b. The distance between any two shapes A and B should not be negative i.e. $d(A,B) > 0$.

c. It should satisfy symmetric property. In other words, the distance between A and B should be same as the distance between B and A i.e. $d(A,B) = d(B,A)$.

d. It should also satisfy the triangle inequality property i.e. for three shapes A, B and C, the relation $d(A,B) + d(B,C) \geq d(A,C)$ should be satisfied.

e. It should be invariant under translation, rotation, and the change of scale.

One of the seminal works on comparing similarity between polygonal shapes was proposed by O'Rourke by developing the notion of the signature of plane curves [4]. A related geometric technique called the turning function was later proposed by Arkin et.al. [1].

2.2 Signature of Polygons

A brief overview of the algorithm for computing the signature of a polygon as proposed in [4] can be described as follows. The signature of a polygon is determined in term of the signature of its edges. To understand the working of the algorithm, the polygon shown in Figure 2.1 is used as a running example. Consider the line L_3 passing through the edge (v_3, v_4) of the polygon. In the figure, line L_3 is drawn with dashed strokes. The total length of the perimeter or the portion of the edges lying on or to the left of L_3 is defined as the **signature** of edge (v_3, v_4). It is emphasized that the length of the edge through which line L_3 passes also accounts in its signature.

The **signature of a point** q on the boundary of a polygon can be defined in term of the tangent of the polygon through that point. This means the signature of a point q lying on the interior of an edge e_i of the polygon is the same as the signature of e_i. Thus the signature of all points on the interior of an edge are identical. It may be noted that the signatures of vertices are not defined. The signature of all edges of the polygon put together gives the

signature of the polygon. Since the signature of interior points on an edge are identical, the signature of the entire polygon consists of a sequence of horizontal line segments. A plot of the signature of an example polygon is shown in Figure 2.1. To plot the signature we need to pick a starting point on the boundary which we take, without loss of generality, as the point next to a vertex in counterclockwise direction. To plot the signature of a point q, we take x-coordinate as the length of the portion of the boundary from the start point to q and the y-coordinate is taken as the signature of point q. The signature has discontinuity at the vertices of the polygon which are shown by dashed vertical line segments in the figure. It can be observed that the signature function is a step function.

The signature of the edge can be plotted by choosing some starting point on the boundary of the polygon. To plot the signature, the length of the boundary from the starting point to the candidate edge is taken as the x-coordinate and the signature of the candidate edge is taken as the y-coordinate. For edge(v_3, v_4), the starting point is v_0, the x-coordinate is the sum of the length of the edges e(v_0, v_1), e(v_1, v_2), e(v_2, v_3) and the y-coordinate is the calculated signature. The plotted signature is shown in Figure 2.1 b.

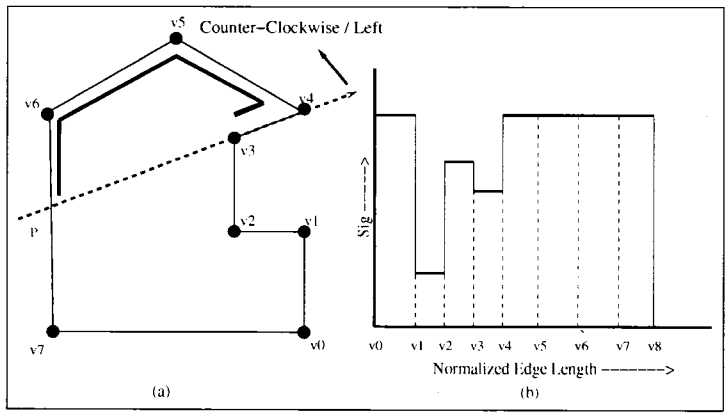


Figure 2.1a Polygon P Figure 2.1b O'Rourke's Signature of Polygon P.

2.2.1 Properties of the Signature

The signature of a polygon has some significant structural properties. For example the signature of a polygon is a step function which has a look of orthogonal chain. Some interesting properties mentioned in [4] can be listed as follows.

Property 1: (Convexity) The signature function of a polygon ignores the differences in the convexity of polygons. This means that whether the polygon is a square, rectangle, convex pentagon, or any convex polygon, the signatures are same. Hence the signatures of all convex polygons are identical.

Property 2: (Invariance) The signature of a polygon is invariant under translation, rotation and scale change of the polygon. It is also observed that symmetric curves have similar signature but not vice versa. As observed in [2], the signature of a polygon is nearly invariant with respect to a slanting transformation. In a slanting transformation only the y-axis is rotated by a small angle to obtain the transformed polygon.

Property 3:(Inversion) It is interesting to ask whether the original polygon can be reproduced from its signature. It turns out that many polygons can have the same signature. This means it is not possible to reconstruct the polygon from its signature. Surprisingly, it has been proved [4] that if the polygon is orthogonal then it can be precisely reconstructed from its signature.

A very critical question is how to measure the distance between two signatures. In the method suggested in [4], the distance between two signatures is determined by applying a "dynamic programming" algorithm that seeks edge by edge(s) matching. During matching,

the pairing of signatures need not to be one-to-one. In this scheme, a signature sequence of size m could be compared with a signature of size k, where k is not necessarily equal to m. To compare polygon P with polygon Q, the algorithm matches the signature edge $P.e_i$ with one of the signatures of (i) $Q.e_j$, (ii) $Q.e_{j-1}$ and $Q.e_j$ or (iii) $Q.e_j$ and $Q.e_{j+1}$. This mechanism of pairing signature of one edge of P with the signature(s) of Q is termed as matching by warping. An example of matching with warping is shown in Figure 2.2 a and 2.2 b below.

The author also proposes "angular distance" measure between two polygons. The elements matched by the algorithm are the angles at each vertex of the polygon. In measuring the angle, left turn represents a positive angle and a right turn represents a negative angle. If α_i from polygon P is matched with β_j from polygon Q then the element to element distance measure is given by:

$$D(i,j) = |sin\alpha_i - sin\beta_j| + |cos\alpha_i - cos\beta_j| \quad (1)$$

It is noted that in this distance measure, the signature does not play any role. Only the angles on the vertices of candidate polygons are taken into account.

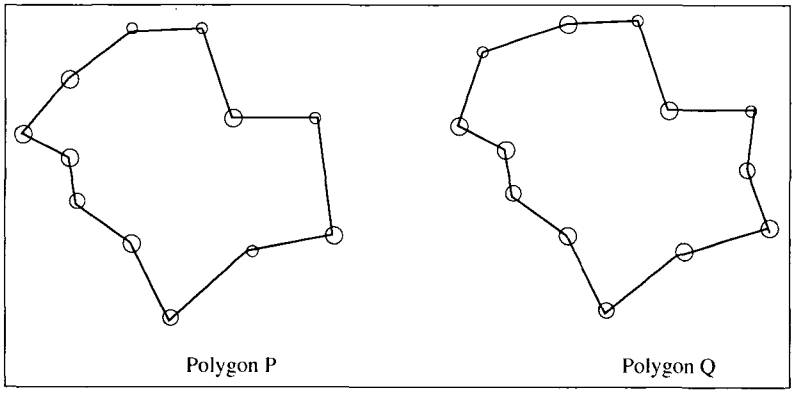


Figure 2.2 a: Polygons P and Q.

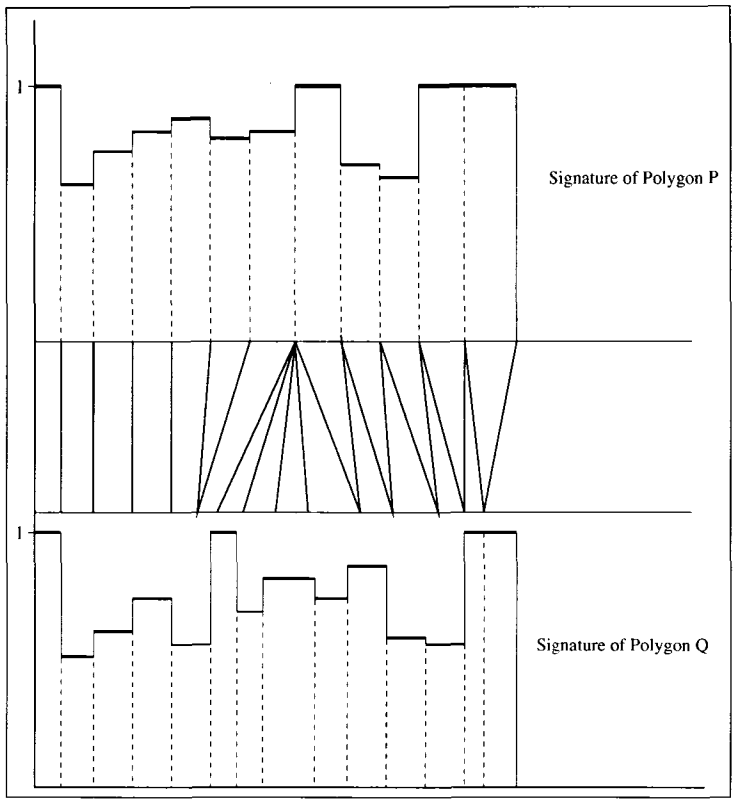


Figure 2.2 b: Illustrating warping of signature of P and Q.

2.3 Turning Function

The turning function method proposed in [1] defines turning angle at each point on the interior of edge of the polygon. A simple polygon P is represented by a turning function $\theta_P(s)$. The turning function is defined at all points on the boundary of the polygon except at the vertices. To properly define the turning function, a point O on the boundary of the polygon is taken as the origin. The polygon is normalized so that the total length of its boundary is 1. The turning function $\theta_P(s)$ maps $l(s)$ to $T(s)$, where $l(s)$ is the length of the boundary path from origin O to s and $T(s)$ is the angle that the tangent at s makes with the reference direction. The reference direction is take as the x-axis. In measuring turning angle, left-turn is taken as positive and right-turn is taken as negative. Thus, $\theta_P(s)$ is the

turn angle $T(s)$. Figure 2.3 shows a simple polygon with its turning function.

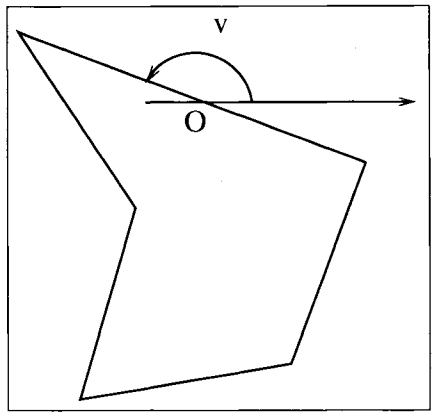


Figure 2.3: Turning function v, where 'O' is the reference point.

It is observed that the turning angle along an edge is constant and the turning angle at the vertices of the polygon is not defined. Hence, the turning function is a step-function. For convex polygons, the turning function is monotonically increasing. Usually, turning function's domain is the interval [0, 1]. However, the turning function domain can be extended to the entire real line R by accumulating angles as the boundary is traversed more than one time. By this extension, the turn angle at boundary point s after completing k turns is $T(s)+2\Pi k$.

To compute the distance between two turning functions $\theta_p(s)$ and $\theta_Q(s)$ the algorithm presented in [1] uses Euclidean distance between the turning functions. The distance between the turning functions depends on the choice of the origin point in the polygon P and Q. Consider the overlay of the turning functions of P and Q as shown in Figure 2.4.

What needed is the vertical and horizontal shifting of the turning function so that the area between the two step functions is minimized. An algorithm to obtain this minimized area is reported in [1]. The time complexity of the algorithm is $O(mn\log mn)$, where m

and n are the numbers of vertices in polygons P and Q respectively.

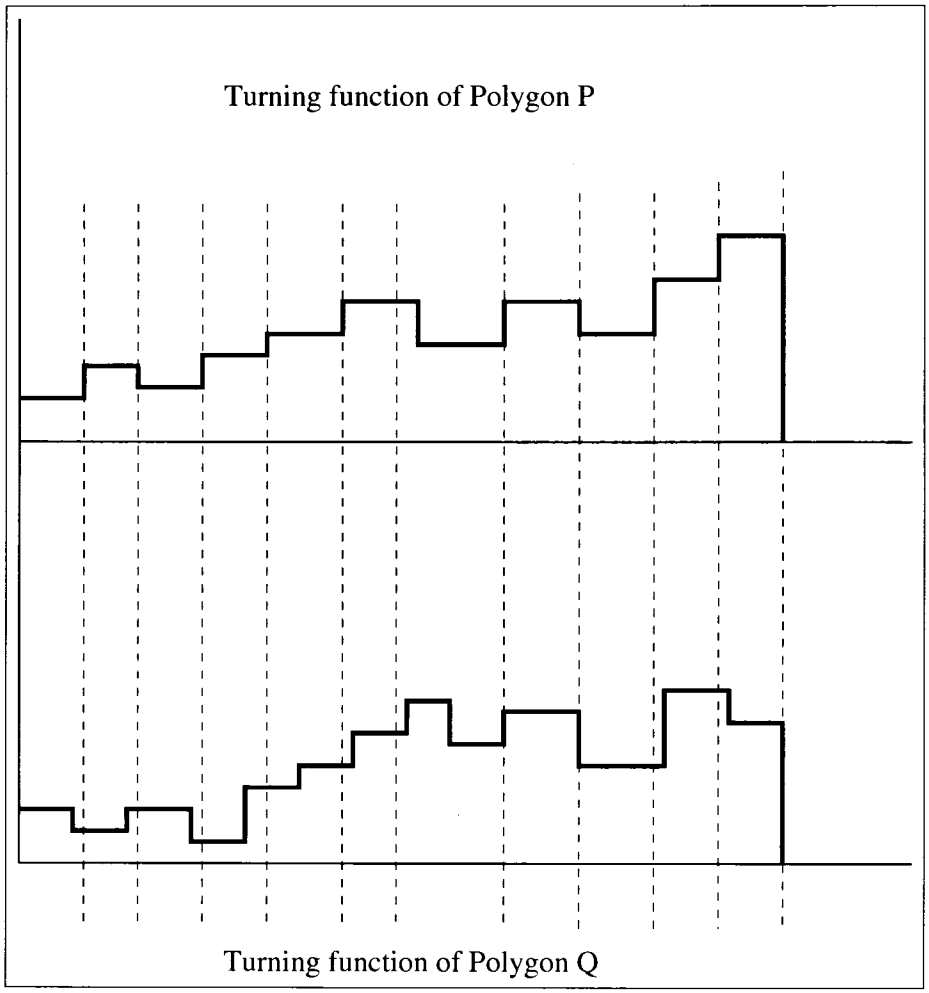


Figure 2.4: Illustrating overlay of turning functions of polygons P and Q.

It has been found in [1] that the turning function is reasonably effective in measuring shape similarity. However, a drawback with this method is that it does not work well in comparing shape similarity when the polygons have non-uniform noise.

CHAPTER 3

CENTER OF GRAVITY GUIDED SIGNATURE OF PLANAR SHAPES

3.1 Introduction

In this chapter we present the development of an efficient algorithm for finding the signature of a simple polygon, which is guided by its center of gravity. The proposed method produces the signatures of convex polygons that changes with the change in their shapes.

As mentioned in Chapter 2, a polygon is represented by an ordered circular sequence of vertices. Consecutive vertices in the sequence form edges. In addition to other information such as the name of the vertex and the angle at the vertex, a vertex record contains its x- and y-coordinates.

3.2 Center of Gravity of a Polygon

We begin with the definition of the center of gravity of a simple polygon. The **center of gravity** of a simple polygon is formally defined as the point where all the weights of the object can be considered to be concentrated. We can assume that the interior of a polygon is made of a material of same thickness and uniform density. Then, the center of gravity of the material is the center of gravity of the polygon. The formula for the computation of the center of gravity of the polygon is related to its area and the coordinates of its vertices. Let the vertices of the polygon be denoted by v_0, v_1, ..., v_{n-1}, where the vertex v_i has the

coordinates (x_i, y_i). The area A of the polygon is given by

$$A = \frac{1}{2} \sum_{i=0}^{N-1} (x_i y_{i+1} - x_{i+1} y_i) \tag{1}$$

The center of gravity of the polygon is related to the coordinate of the vertices and its area A as

$$c_x = \frac{1}{6A} \sum_{i=0}^{N-1} (x_i + x_{i+1})(x_i y_{i+1} - x_{i+1} y_i) \tag{2}$$

$$c_y = \frac{1}{6A} \sum_{i=0}^{N-1} (y_i + y_{i+1})(x_i y_{i+1} - x_{i+1} y_i) \tag{3}$$

Using these formulae, an algorithm for computing the center of gravity of the polygon can be sketched as follows:

Algorithm: Center of Gravity of a Polygon

Input: Coordinates of vertices of a polygon: (x_0, y_0), (x_1, y_1), ..., (x_{n-1}, y_{n-1}).

Output: The coordinate of Center of Gravity of a polygon: (x_c, y_c).

Step 1: /* Find the area of the polygon.*/
i. Area = 0, $x_c = 0$, $y_c = 0$
ii. for(int i = 0; $i < n$; i++)
iii. Area = Area + $(x[i] * y[i+1] - x[i+1] * y[i])$
iv. Area = $\frac{1}{2}$ * Area;

Step 2: /* Make sign of the area positive. */
i. if(Area < 0)
ii. Area = (-1) * Area

Step 3: /* Compute Center of Gravity. */
i. for(int i = 0; $i < n$; i++)
ii. c_x = (x[i] + x[i+1]) * ((x[i] * y[i+1]) - (x[i+1] * y[i]))
iii. c_y = (y[i] + y[i+1]) * ((x[i] * y[i+1]) - (x[i+1] * y[i]))
iv. $x_c = \frac{1}{6A} * c_x$
v. $y_c = \frac{1}{6A} * c_y$

Step 4: Output x_c and y_c as the coordinates of the center of gravity.

An alternate method for computing the center of gravity of a polygon is to first triangulate the polygon [3]. After triangulating the polygon, the center of gravity of each triangle is computed. The centroid of a triangle is also known as the center of gravity point. The coordinates of the centroid of a triangle are given by the average of the coordinates of its vertices. If $(x_1, y_1), (x_2, y_2)$ and (x_3, y_3) are the coordinates of a triangle then the coordinates of its center of gravity (x_m, y_m) is given by

$$x_m = \frac{x_1 + x_2 + x_3}{3}, y_m = \frac{y_1 + y_2 + y_3}{3} \tag{4}$$

Once we have the center of gravity of all triangles, the center of gravity of the entire polygon can be found by the computing the weighted sum of the center of gravity of all the triangles. Let $T_1, T_2, ..., T_{n-2}$ be the triangles after triangulating the polygon. Let A_i denote the area of the triangle T_i. Also, suppose (x_i^c, y_i^c) denote the center of gravity of the triangle T_i. Then

$$x_c = \frac{\sum A_i * x_i^c}{\sum A_i} \tag{5}$$

$$y_c = \frac{\sum A_i * y_i^c}{\sum A_i} \tag{6}$$

3.3 Center of Gravity Guided Signature of a Polygon

The notion of center of gravity guided signature of a simple polygon is similar to the notion of the standard signature introduced by O'Rourke[2]. We recall that the standard signature of an edge e of a polygon is defined as the length of a portion of the perimeter of a polygon to the left of the line passing through e. The center of gravity guided signature is defined by considering a line passing through the center of gravity of a simple polygon as:

Definition 3.1: Consider an edge e of a simple polygon P. Let e' be the line passing through the center of gravity of P and parallel to e. The center of gravity guided signature of e is defined as the length of the portion of the permiter that lies to the left of e'. Figure 3.1 illustrates the center of gravity guided signature of an edge. The portion of the perimeter to the left of e' is drawn in dashed line.

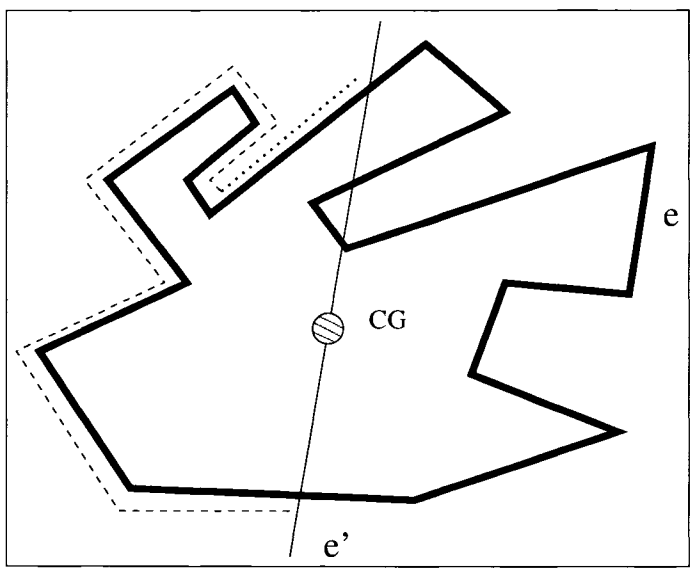


Figure 3.1: Illustrating center of gravity guided signature of an edge

Definition 3.2: The center of gravity guided signature of a simple polygon P is obtained

by considering the center of gravity guided signature of all its edges. We pick a point O on the boundary of the polygon as the origin. The center of gravity guided signature of a point (say p) on a boundary edge e is the same as the center of gravity guided signature of e. Corresponding to each boundary point q, we can identify two numbers: (i) its signature denoted by $sig(q)$ and (ii) its boundary distance $dist(q)$ from the origin point O. The cg-guided signature is displayed by plotting $dist(q)$ in x-axis and $sig(q)$ in y-axis for all boundary points of P. An example of the plot is shown in Figure 3.2.

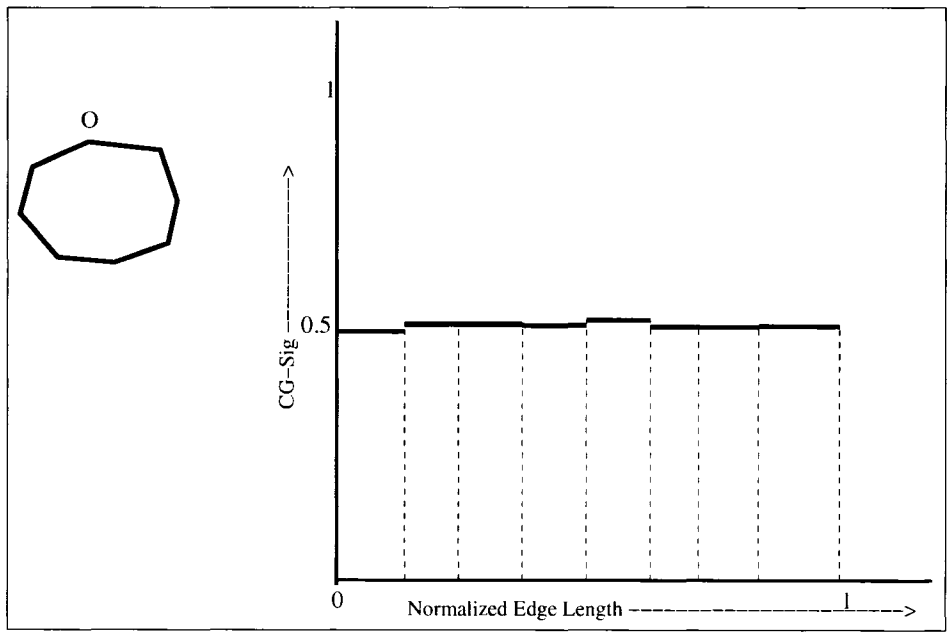


Figure 3.2: Illustrating center of gravity guided signature of a convex polygon.

It is interesting to note that unlike the standard signature, the center of gravity guided signature is not identical for all convex polygons. Figure 3.3 shows the center of gravity guided signatures of two different convex polygons, which are not same.

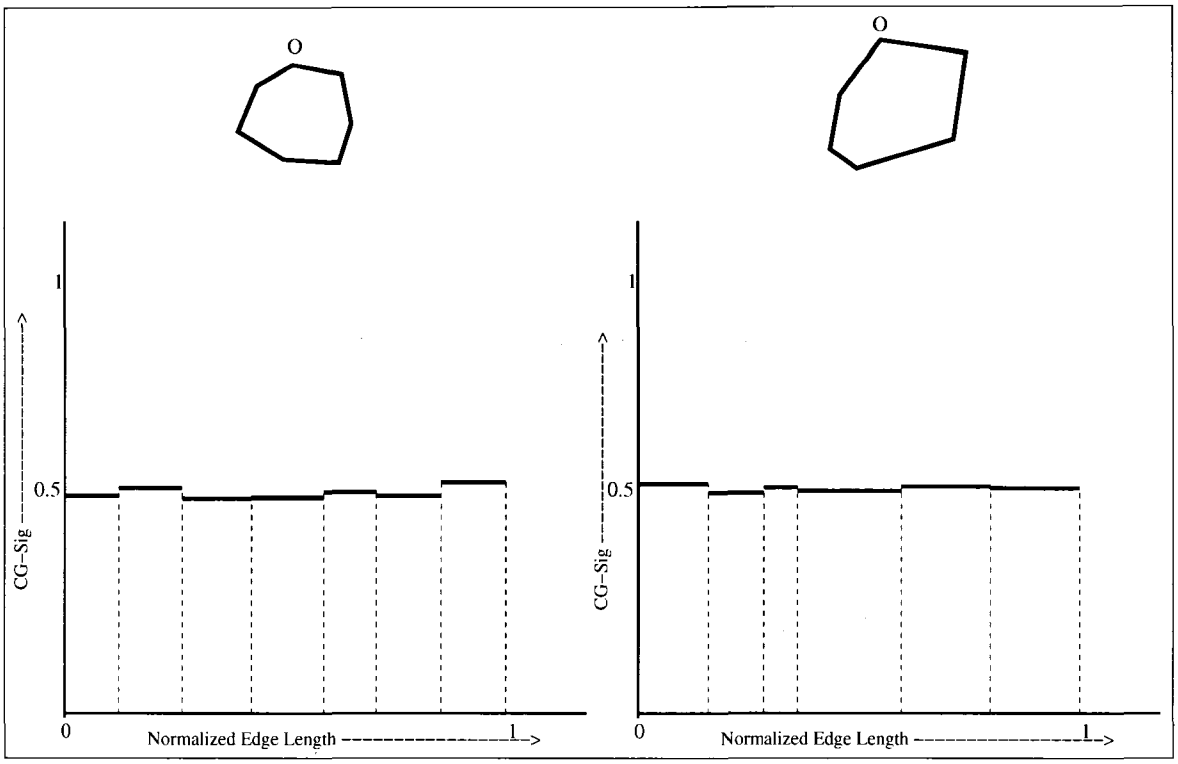


Figure 3.3: Center of gravity guided signatures of two different convex polygons

Observation 3.1: The center of gravity guided signature distinguishes convex polygons with unsymmetrical shapes.

Next, we describe the development of the algorithm for computing the center of gravity guided signature of an edge e of a simple polygon. The center of gravity of a polygon is found by using the method described in Section 3.2. The line e' parallel to e can be found by taking the slope of e and the co-ordinates of the center of gravity. The equation of e' is given by the line passing through center of gravity and having the slope of e. Once, we have the line e', we can determine the portion of the perimeter of P lying on or to the left of e'. Whether or not an edge e_x lies to the left of e' can be determined by performing a simple

left-turn check between the end points of e' and the end points of e_x. This is a standard method found in computational geometry textbook [5]. If an edge e_x does intersects with e' then it is straightforward to determine the portion of e_x lying to the left of e' by finding the point of intersection between e_x and e' and by performing a couple of left turn checks. After this step, the portion of boundary lying to the left of e' are known, the center of gravity guided signature of e is given by accumulating the identified edge lengths. This process of computing center of gravity guided signature of an edge is repeated for all edges to obtain the center of gravity guided signature of the whole polygon. A formal sketch of algorithm is listed as CG Signature algorithm below.

Algorithm: CG Guided Signature Algorithm.

Input: **a.** A simple polygon P with n vertices $v_1, v_2, ..., v_n$
b. Origin point O on the boundary

Output: CG Guided signature available as pairs $[d(x_1), sig(x_1)], [d(x_2), sig(x_2)]$, $..., [d(x_n), sig(x_n)]$.

Step 1: // Normalize the perimeter length to 1.
a. $L1$ = Perimeter of the polygon P.
b. for(int i = 1; i≤n; i++)
c. $nl(i) = len(e_i)/L1$;

Step 2: // Compute origin-distanve of vertices.
a. cumDist = 0;
b. for(int i = 1; i≤n; i++)
c. $d(i) = cumDist + nl(i)$;
d. $cumDist = cumDist + nl(i)$

Step 3: //Compute signature of edges.
a. for(int i = 1; i≤n; i++)
b. $sig(x_i)$ = FindSigCG(P, i)

Step 4: Output $d(i)'s$ and $sig(x_i)$

Algorithm: FindSigCG(P, i).

Input: Vertices of a polygon P: $v_0, v, ..., v_{n-1}$, where $v_i = (x_i, y_i)$

Output: Signature of edge (v_i, v_{i+1}).

Step 1: **a.** Point p = (x[i], y[i]), Point q = (x[i+1], y[i+1])
b. Create line segment $s_i(p,q)$

Step 2: **a.** Find Line parallel L_p to segment s_i, Cut line segment s_p out of it.
b. Set $Sig(x_i)$ to 0;

Step 3: for(int j=0; j<n; j++)
i. Point a = (x[j], y[j]), b = (x[j+1], y[j+1])
ii. Create line segment $s_j(a,b)$
iii. if(Points a and b are on the same side of s_p)
iv. if(a is on the left of s_p) $sig(x_i) = sig(x_i)$ + LengthOf(s_j)
v. else if (s_j intersects L_p) Find the point of intersection p_i.
vi. if(a is on the left of s_p) Create segment p_iSeg = seg(a, p_i)
vii. else Create segment p_iSeg = seg(p_i, b)
viii. $sig(x_i) = sig(x_i)$ + LengthOf(p_iSeg)

Step 4: Signature of the edge.

Theorem 3.1: CG Guided Signature Algorithm can be executed in $O(n^2)$ time, where n is the number of vertices of the polygon.

Proof: Step 1 and Step 2 of function FindSigCG() takes $O(1)$ time. The loop of Step 3 executes n time and each execution of the loop takes constant time. Hence function FindSigCG() takes $O(n)$ time. Step 1 and Step 2 of CG Guided Signature Algorithm take $O(n)$ time each. Step 3 mkes n calls for function FindSigCG() and hence takes $O(n^2)$ time. Step 4 takes $O(n)$ time and consequently, the total time for CG Guided Signature Algorithm becomes $O(n^2)$. □

3.4 Removing Noise Edges

When a polygon model is constructed by extracting boundary edges from an image, some noise edges may be present. Intuitively, noise edges are the edges, which do not carry any feature of the original object. Figure 3.4 shows an example of noise edges. Consider a triangular shape shown in Figure 3.4a below. Figures 3.4b-3.4d show three other triangular

shapes with embedded noise edges.

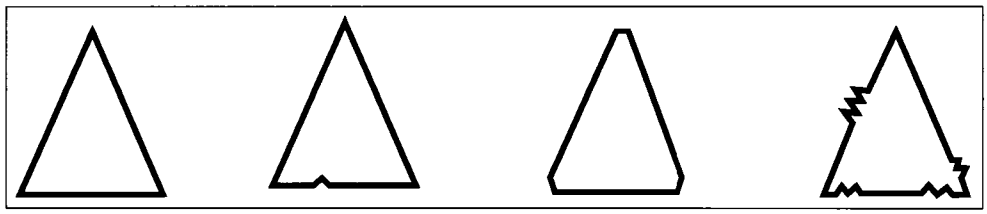


Figure 3.4a-d: Illustrating Noise edges in Polygons

If we compute the signatures of these shapes then the presence of noise edges have considerable effect on their signatures. This suggests that if we can somehow remove noise edges then the signatures can be more effective in measuring the similarities between the shapes.

Characterizing and distinguishing noise edges from regular edges are difficult tasks. Removal of noise edges could be addressed to a certain extent by using the technique of boundary smoothing. A variety of boundary smoothing techniques have been reported in the computational geometry and the pattern recognition literature [2,3]. It is yet to be investigated how effective are the boundary smoothing techniques as a pre-processing step in signature analysis.

We propose a technique for filtering noise edges from the boundary of a polygon. We use the following two characteristics of noise edges.

(i) **A short edge** is defined as an edge whose length is no more than $q\%$ (q can be taken as 10) of the average length of boundary edges. A short-edge is a possible candidate for noise edges.

(ii) Two consecutive short edges forming a very small angle between them are possible candidates as noise edges.

We can determine the length of all the edges and their average length l_{av} by one scan of the boundary of the polygon. By performing a second scan we can identify all the short edges. During the first and second scan, edges forming very small angle (say no more than 8^o), which we refer to as **acute edges**, can be marked. During the third scan two or more consecutive edges that are short as well as acute edges are taken as noise edges. If $v_i, v_{i+1}, ..., v_j$ are the vertices of a maximal sequence of noise edges then this sequence is replaced with a single edge (v_i, v_j). A formal description of the algorithm for replacing noise edges is listed below.

Algorithm: Noise Removal Algorithm.

Input: A simple polygon P with ordered boundary vertices $v_1, v_2, ..., v_n$

Output: A filtered polygon P' obtained by removing noise edges.

Step 1: // Determine the edge lengths and l_{av}.
- **a.** totalLength = 0;
- **b.** for(int i = 1; i≤n; i++)
- **c.** $l_i = dist(v_i, v_{i+1})$;
- **d.** totalLength = totalLength + l_i;
- **e.** $l_n = dist(v_n, v_1)$;
- **f.** totalLength = totalLength + l_n;
- **g.** l_{av} = totalLength/n;

Step 2: // Mark short and acute edges.
- **a.** for(int i = 1; i≤n; i++)
- **b.** if($l_i < 0.1 * l_{av}$) $e_i.short$ = true;
- **c.** else $e_i.short$ = false;
- **d.** if(angle at $l_i < 10$deg) $e_i.acute$ = true;
- **e.** else $e_i.acute$ = false;

Step 3: //Mark noise edges.
- **a.** for(int i = 1; i≤n; i++)
- **b.** if($e.short$ && $e.acute$) $e_i.noise$ = true;
- **c.** else $e_i.noise$ = false;

Step 4: //Replace noise edges by a single edge.
- **a.** for(int i = 1; i≤n; i++)
- **b.** $v_{start} = v_i$; $j = i$;
- **c.** while($e_i.noise$) i++;
- **d.** if($i > j$)
- **e.** replace $e_i, e_{i+1}, ..., e_j$ by the edge connecting v_i and v_j

Step 5: Output the resulting polygon P'.

3.4.1 Time Complexity

Time complexity for *Noise Removal Algorithm* can be deduced as follows. Step 1 and Step 2 of the algorithm takes $O(n)$ time each as there are n vertices to be analyzed as candidate noise edges. Step 3 of the algorithm also takes $O(n)$ time for identifying candidate edges as noise edges. Step 4 of the function executes n times for replacing identified noise edges with a noise-free edge, thus taking $O(n)$ time. Hence the Noise-Removal algorithm takes total time of $O(n)$.

CHAPTER 4

IMPLEMENTATION AND EXPERIMENTAL RESULTS

In this chapter, a brief description of the implementation of the algorithm for computing the cg-guided signature of a polygon is presented. For the purpose of comparative evaluation, we also implemented the algorithm for computing the standard signature of a polygon reported in [4]. We also implemented a preprocessing step that can be used to remove "noise edges" from the boundary of the polygon. We picked the Java programming language for implementing the algorithms. We used the Eclipse Java Programming suite which is available for free down load. We tried to make a user friendly graphical user interface through which polygon data can be entered and signature computation can be performed. To enter the polygon manually, the user can specify vertices by clicking the mouse button at desired locations. Consecutive vertices entered by the user are automatically connected by an edge. At the same time, the first and the last vertices are connected by an edge to form a closed boundary. The entered polygon can be edited visually by using mouse. A user can select a vertex by mouse click and move it to a new position by dragging the mouse button. For performing simple geometric computations such as finding the intersection between line segment, detecting the turn made by three points, etc. we used the java class available for download at the web-site of O'Rourke [6]. Other classes for computing signature, file I/O, editing, etc. were developed from the scratch.

4.1 Interface Design

We use the swing classes available from javax for designing the graphical user interface. The main container of the GUI is the JFrame class imported from javax. The main frame contains three panels for holding buttons, canvas, text area, menu bar, etc. The layout of the panels on the frame is shown in Figure 4.1

a. **Panel 1:** Panel 1 is the main canvas to draw the polygon. Depending upon what avaialble attribute the user has selected, the polygon can be created, deleted or modified in this panel.

b. **Panel 2:** Panel 2 is divided into three sub-panels namely, Panel 2.1, Panel 2.2 and Panel 2.3. Panel 2.1 contains options to create and modify the polygon. Panel 2.2 contains buttons to move the polygon over the canvas and to calculate the standard signautre, CG-guided signature and center of gravity of the polygon. Panel 2.3 contains a text box in which the coordinates of the polygons are displayed.

c. **Panel 3:** Panel 3 has option button control using which the color of the edges can be changed to either red, blue or green.

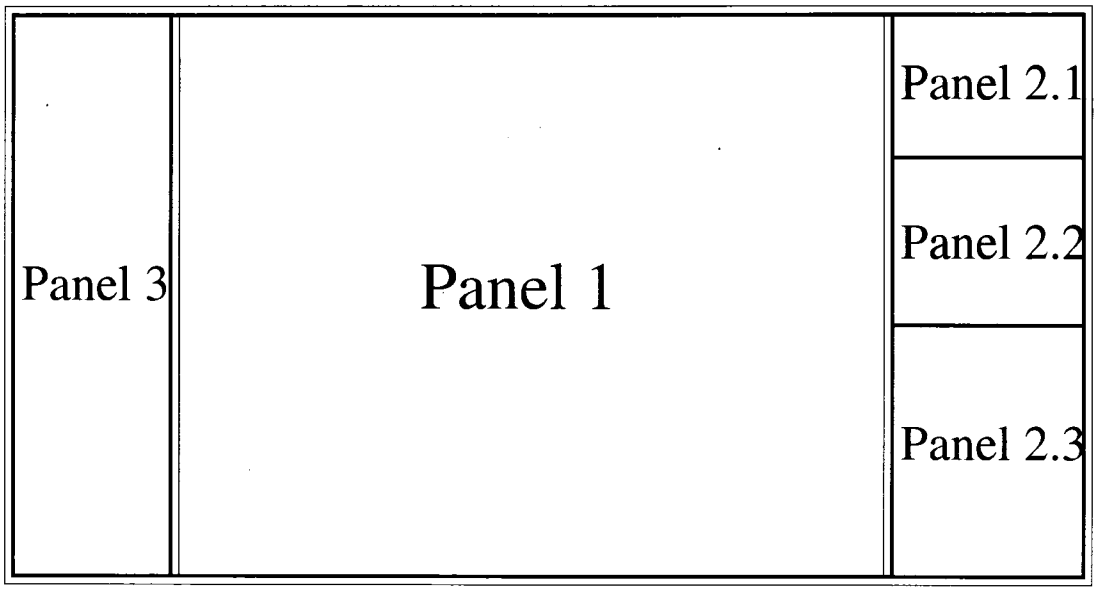


Figure 4.1: The Layout of the Interface.

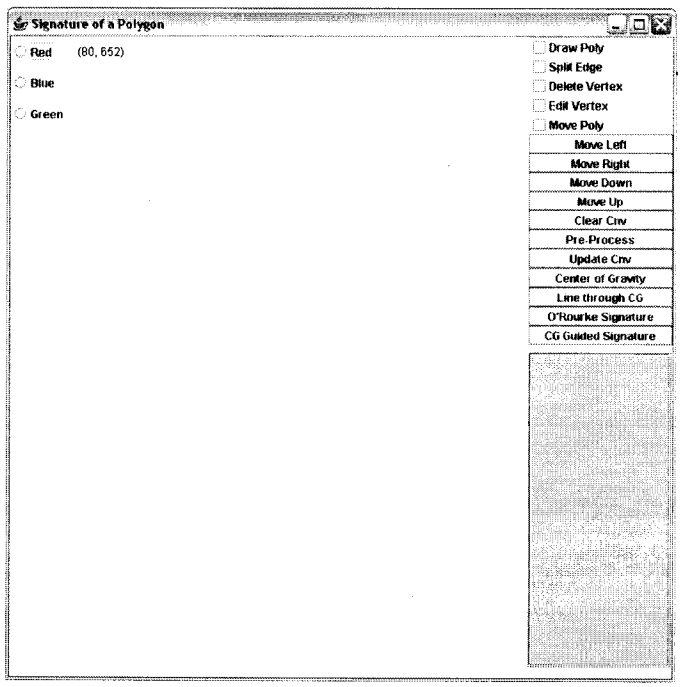


Figure 4.2: A Snapshot of GUI.

To create a polygon in the canvas, *DrawPoly* option is available in Panle 2.1. The vertices of the polygon will lie within the width and height of the canvas. If the width and height of the canvas is say w and h, respectively, the vertices of the polygon will lie within [0,w] and [0, h]. To generate the signature of the polygon, the polygon vertices need to be created in the counter-clockwise direction. While creating a polygon it may happen that the user creates two or more vertices at the same coordinate. The user can edit the position of those vertices by using "Edit Poly" functionality provided in the Panel 2.1 of the GUI. The coordinates of the polygon are stored a "my_point" object in the Vector variable. Figure 4.2 shows an illustration of creating a polygon and its coordinates being displayed in Panel 3 at the bottom right region of the GUI. The vertices are formed as a circle of a small radius just for the convenience of better visibility of the vertex.

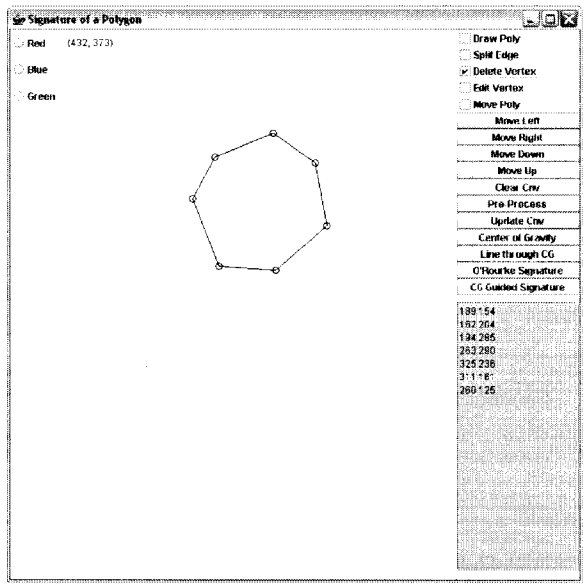


Figure 4.3: Illustrating drawing polygon in GUI.

The standard signature and center of gravity guided signatures are drawn in sepearate frames. After creating the polygon, the user can click *O'Rourke's Signature* and *CG Guided Signature* options and both the signatures will be displayed in seperate frames. The user can select *Center of Gravity* option from the Panel 2.2. The center of gravity of the polygon is then displayed at the bottom left corner of the canvas. If the polygon is modified, the center of grvity gets recalculated smultaniously and is displayed at the lower left corner of the canvas. The center of gravity point of the polygon is formed as a circle of small radius displayed in color red, which usually is in the interior of the polygon.

The GUI provides a hidden functionality to save the polygon, its O'Rourke Signature and its cg-Guided Signature in a separate file with a "*.fig" format. As soon as the user clicks the O'Rourke and cg-guided signature buttons, the polygon and graphs gets saved at a pre-mentioned location on the hard drive. The user can open those files in xfig and analyze them for later use.

Implementing O'Rourke's Signature

The functionality of the standard signature explained in Chapter 2, has been implemented in the button captioned as "O'Rourke Signature" in Panel 2.2 of the GUI. After creating the polygon, if the user clicks this button, its signature gets displayed in a separate frame, which has been implemented using JFrame and JPanel class of javax. JFrame acts as the parent frame to hold the panel, which can also be termed as a window. A JPanel is a light weight container, which like an object to be held by the parent frame. The signature values and the distance values of the polygon is stored in a Vector, which then is taken out in the panel and plotted with red (x-value) and blue (signature not defined at vertices) lines. Figure 4.3 below shows an illustration of O'Rourke's Signature produced by using the GUI.

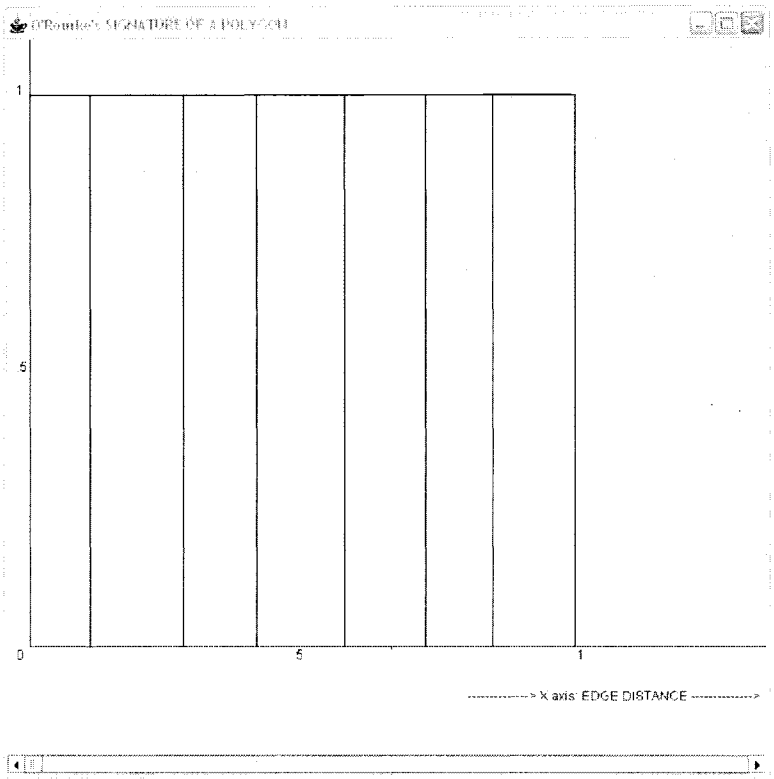


Figure 4.4: Illustrating O'Rourke's Signature.

Implementing CG-Guided Signature

The proposed algorithm of cg-guided signature explained in Chapter 3 has been can be accessed using the "CG Guided Signature" button provided in Panel 2.2 of the GUI. Similar to the standard signature, the cg-guided signature of the polygon gets displayed in a separate frame, which has also been implemented using JFrame and JPanel class of javax. The cg-guided signature is displayed in a separate frame and the signatures and distance values of the polygon are stored in a Vector variable, which then in displayed in the frame. Figure 4.4 below shows an illustration of cg-guided signature produced by using the GUI.

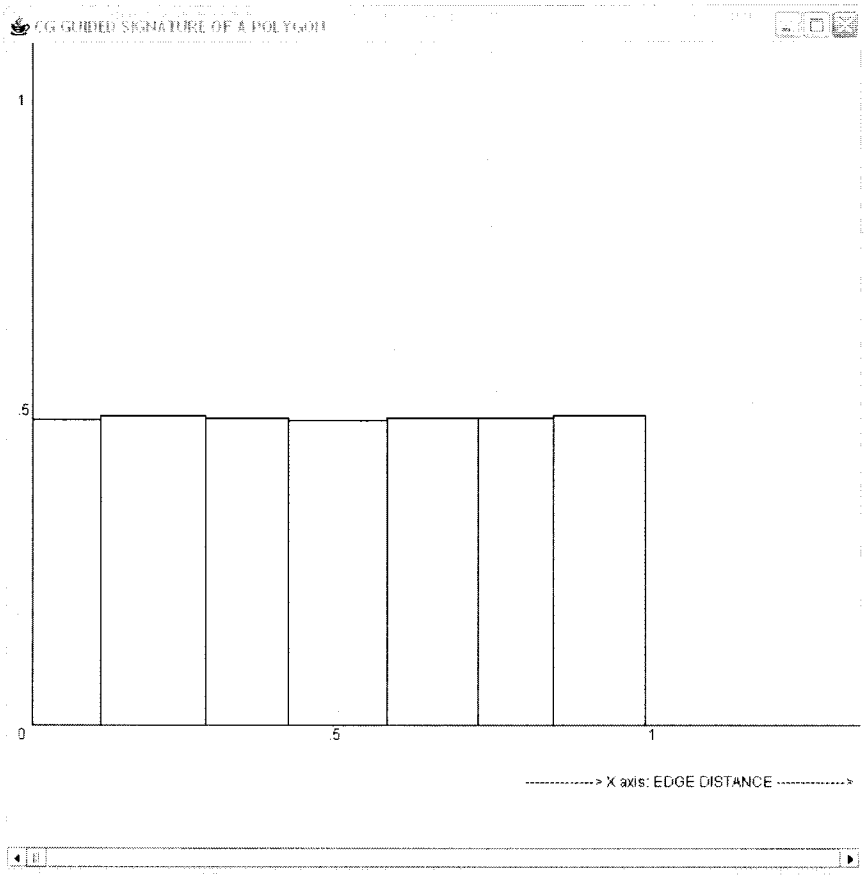


Figure 4.5: Illustrating CG-Guided Signature.

4.2 Functionalities of GUI Components

Table 4.1(a-c) contains a brief description of the GUI components. The first column in the table contains the names of the components and the second column contains a brief description of the functionality corresponding to those components.

Table 4.1a: Functionality of GUI

Name	Description
Draw Poly	This option enables the user to draw polygon vertex when a mouse is clicked. The coordinates of the vertex of a polygon are shown on the top left corner of the frame in the GUI.
Split Vertex	This feature enables the user to split an edge of a polygon. The edge nearest to the mouse click will be splited by a new vertex. The coordinates of the new vertex are displayed at the top left corner of the canvas.
Delete Vertex	This attribute enables the user delete one vertex at one mouse click. The vertex closest to the mouse pointer will be deleted. The deleted vertex also gets removed from text panel, which displays the coordinates of the vertices of the polygon.
Edit Vertex	This element lets the user edit a vertex of a polygon by moving and repositioning the polygon on the screen. The coordinates of the edited vertex are displayed on the left top corner of the canvas.
Move Poly	This option enables the user to move the polygon and reposition it on the canvas. The new coordinates of the polygon are displayed on the coordinate panel on the bottom right corner of the GUI.
Move Left	This button enables user to move the polygon to the left by 10 pixel units (in x-axis). The new coordinates of the polygon are displayed in the coordinate panel to the bottom right of the GUI.
Move Right	This button enables user to move the polygon to the right by 10 pixel units (in x-axis). The new coordinates of the polygon are displayed in the coordinate panel to the bottom right of the GUI.

Table 4.1b Part A: Functionality of GUI

Name	Description
Move Down	This feature enables the user to move the polygon down by 10 pixel units in the y-direction. The new coordinates of the polygon are displayed in the coordinate panel to the bottom right of the GUI.
Move Up	This feature enables the user to move the polygon up by 10 pixel units in the y-direction. The new coordinates of the polygon are displayed in the coordinate panel to the bottom right of the GUI.
Clear Cnv	This button enables the user to clear the canvas. The polygon pre-drawn is deleted.
Pre-Process	This functionality enables the user to remove the noisy points from the polygon. The updated polygon vertices are displayed in the coordinate panel at the bottom right of the GUI.
Update Cnv	This feature enables the user to update the polygon when a vertex of the polygon is updated in the coordinate panel at the bottom right part of the canvas
Center of Gravity	This function calculates the center of gravty point of the polygon. The center of gravity point is displayed at the bottom left corner of the canvas.
Line Through CG	This function shows the user all the lines, which are parallel to the respective edges of the polygon and also pass through the center of gravity of the polygon.
O'Rourke Signature	This button calculates the O'Rourke's Signature of the polygon. The signature of the polygon is shown in a seperate frame titled *O'Rourke's Signature of a Polygon.*
CG Guided Signature	This function calculates the center of gravity guided signature of the polygon. The CG-Guided signature is shown in a seperate frame. titled *CG Guided Signature of a Polygon*
Red	This option changes the color of the edges of the polygon to red.

Table 4.1b Part B: Functionality of GUI

Name	Description
Blue	This option changes the color of the edges of the polygon to blue.
Green	This option changes the color of the edges of the polygon to green.

Several java classes were written to implement the GUI and the proposed algorithms. The UML diagrams of the classes are as displayed below.

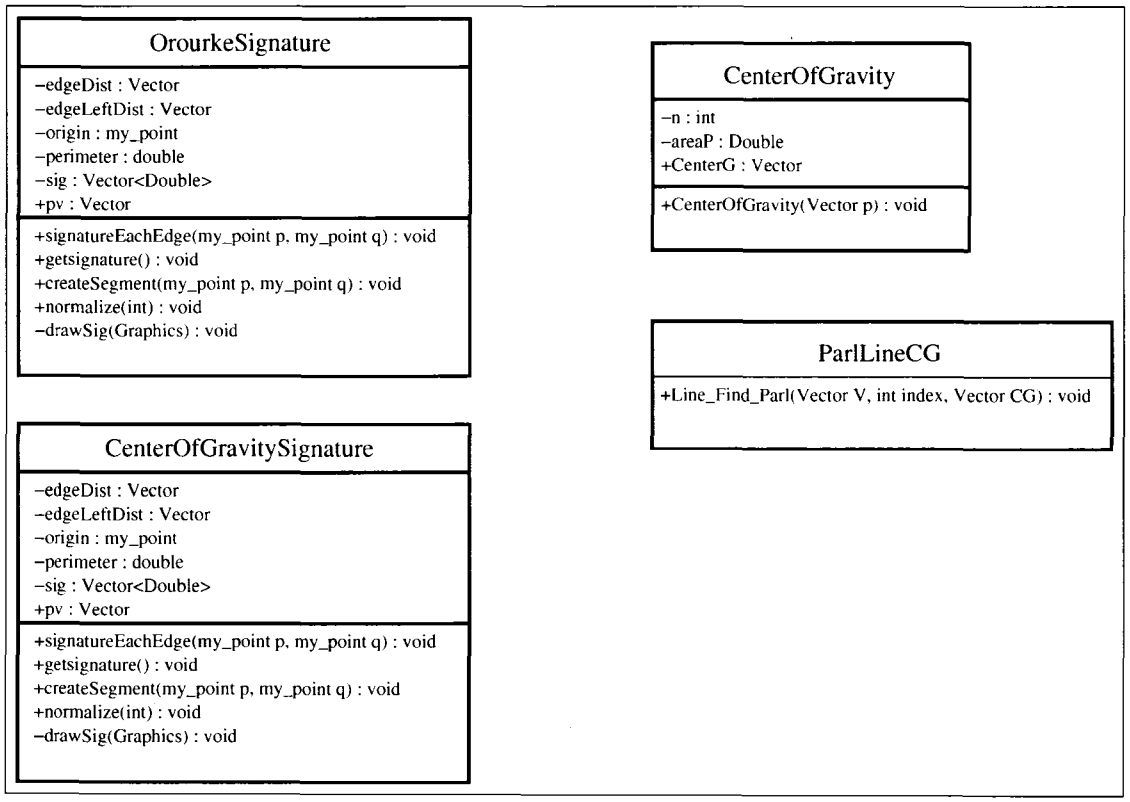


Figure 4.6: UML Model - 1

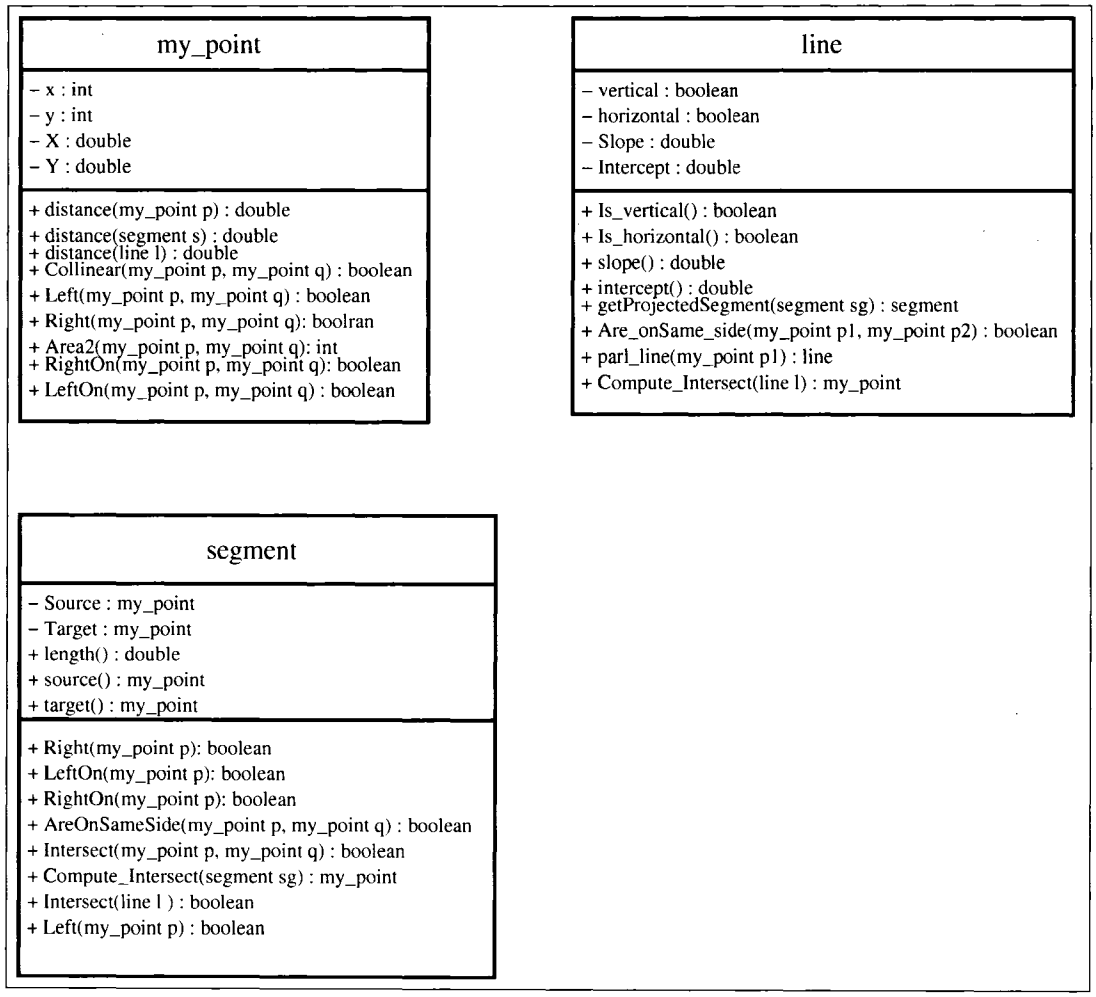


Figure 4.7: UML Model - 2

The experiment was carried out on several convex and non convex polygons and the plot of the computed signature is shown in figures Figure 4.8 - Figure 4.10. Figures 4.11-4.15 shows the comparison between signatures of non-convex polygons. The results of the comparison of the signatures of similar looking shapes are shown in Figures 4.16-1.10. Figures on Left side are the standard signautres of the respective polygons and the figures on the right are the cg-guided signautres of the respective polygons.

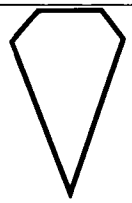
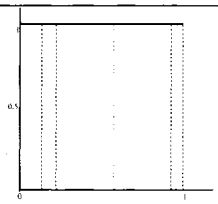
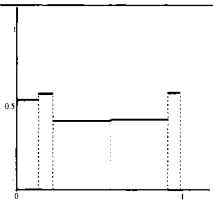
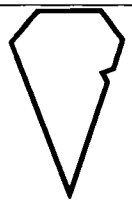
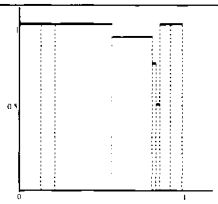
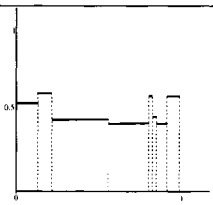
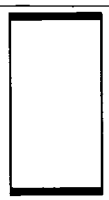
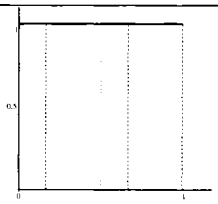
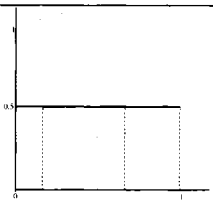
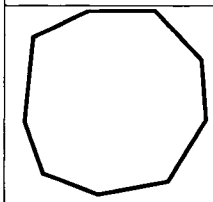
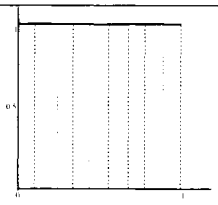
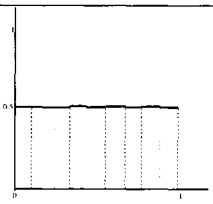
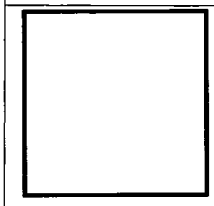
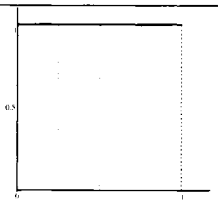
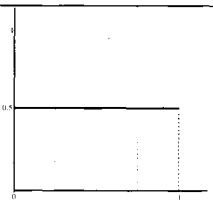
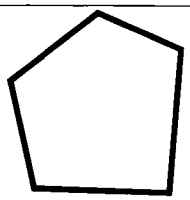
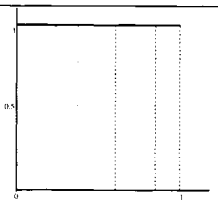
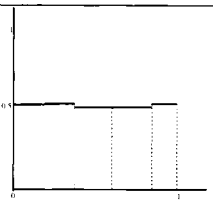
Figure	Area: O'Rourke Signautre	Area: CG Guided Signature	O'Rourke Signature	CG Guided Signature
	1.0	0.4605		
	0.9573	0.4639		
	1.0	0.5		
	1.0	0.5017		
	1.0	0.5		
	1.0	0.5062		

Figure 4.8: Signatures of several Convex (and Convex with Noise Edges) Polygons

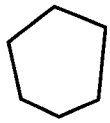
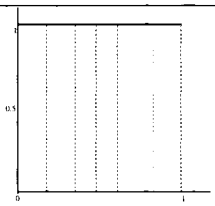
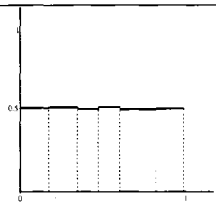
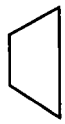
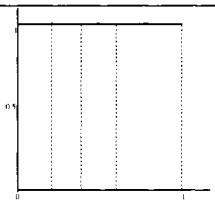
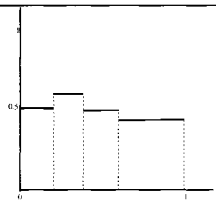
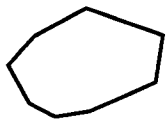
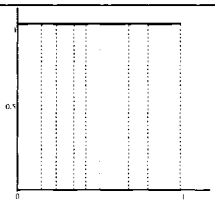
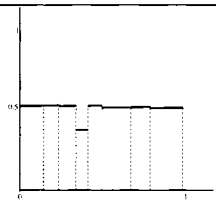
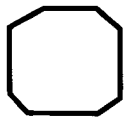
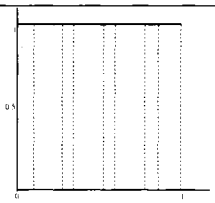
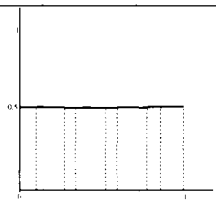
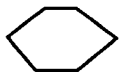
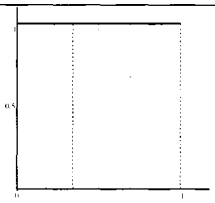
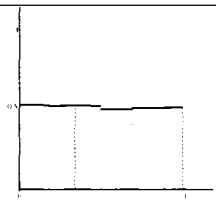
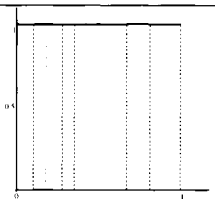
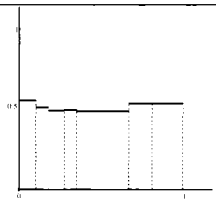
Figure	Area: O'Rourke Signautre	Area: CG Guided Signature	O'Rourke Signature	CG Guided Signature
	1.0	0.5015		
	1.0	0.4764		
	1.0	0.4928		
	1.0	0.4995		
	1.0	0.4998		
	1.0	0.4975		

Figure 4.9: Signatures of several Convex Polygons (First Example)

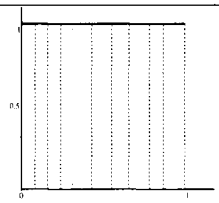
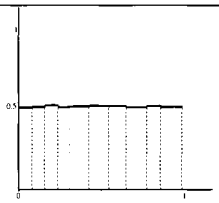
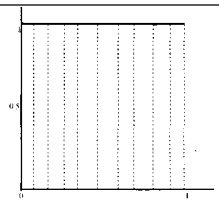
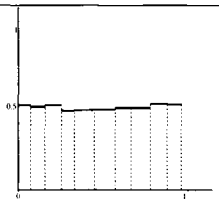
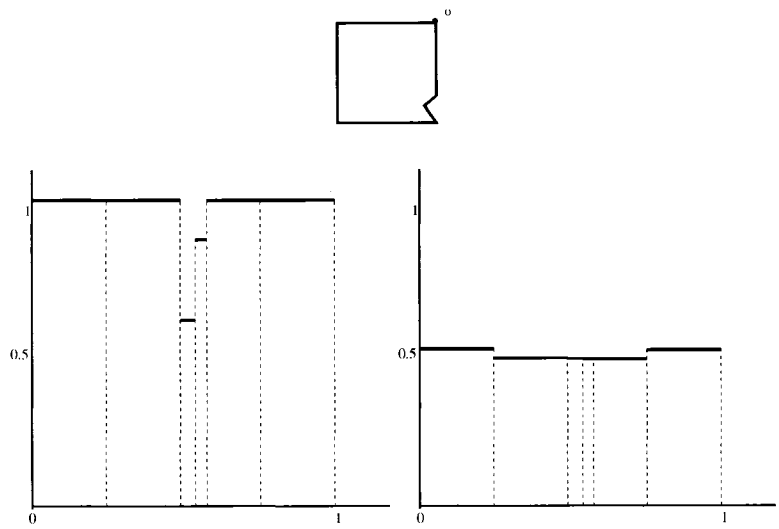
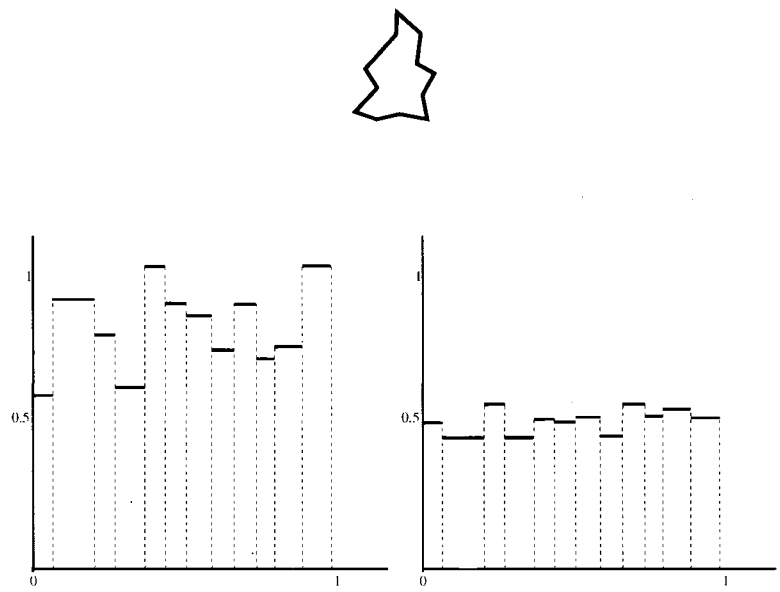
Figure	Area: O'Rourke Signautre	Area: CG Guided Signature	O'Rourke Signature	CG Guided Signature
	1.0	0.4987		
	1.0	0.4975		

Figure 4.10: Signatures of several Convex Polygons (Second Example)



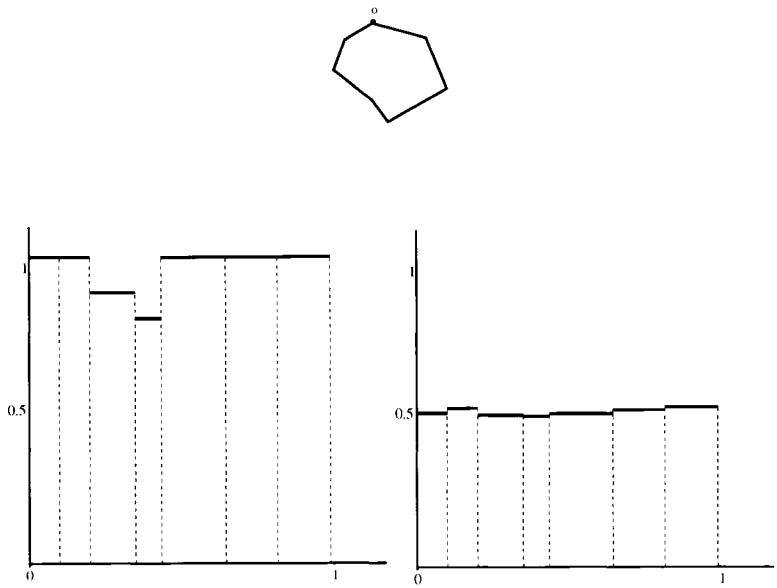
Standard Signature CG-Guided Signature

Figure 4.11: Results for Non-Convex Polygon - Example 1



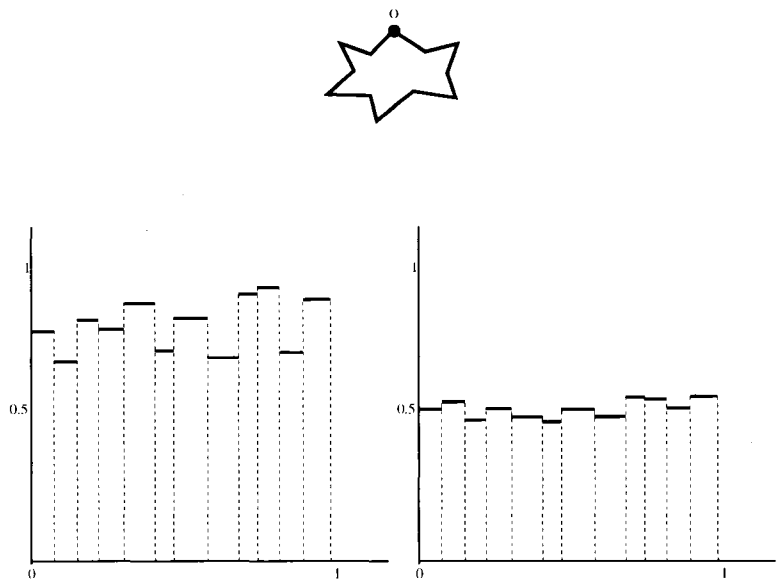
Standard Signature CG-Guided Signature

Figure 4.12: Results for Non-Convex Polygon - Example 2



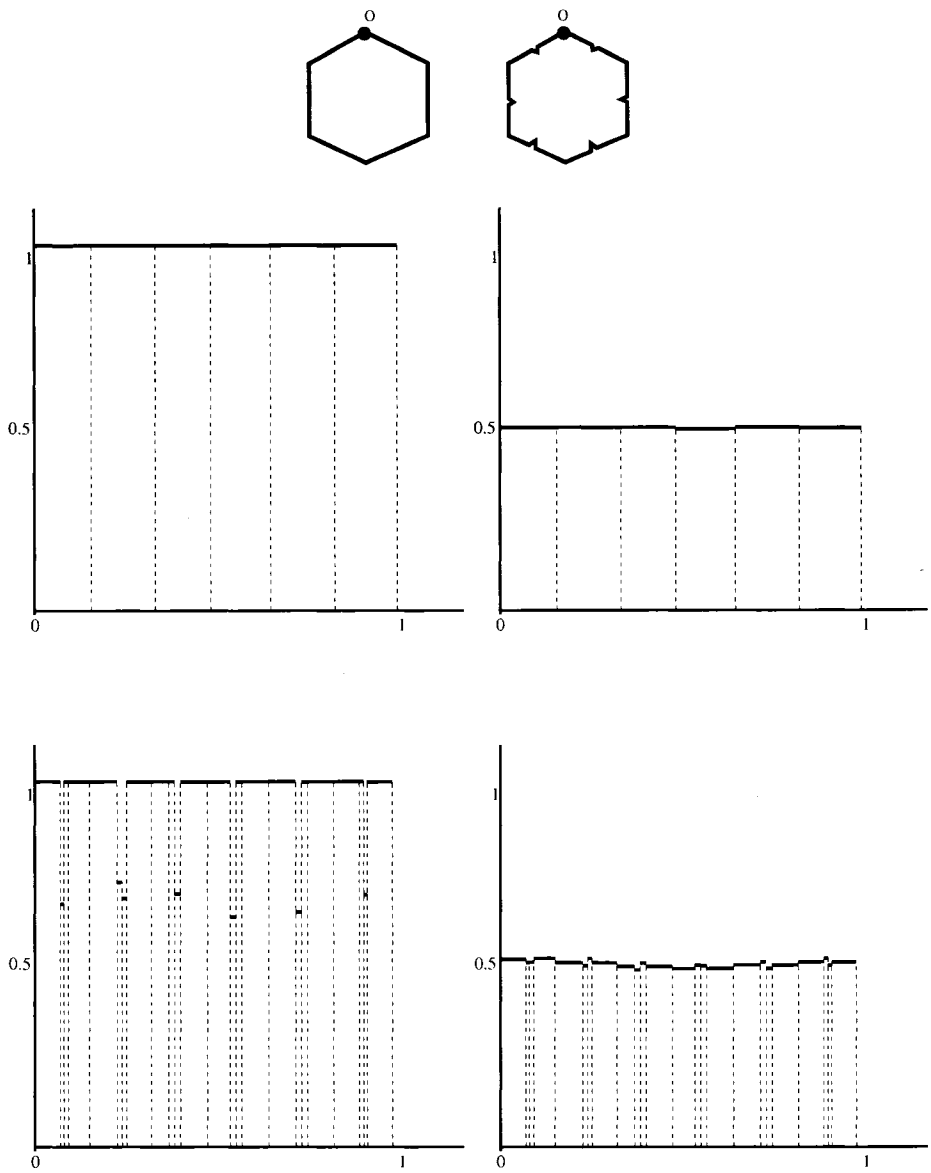
Standard Signature CG-Guided Signature

Figure 4.13: Results for Non-Convex Polygon - Example 3



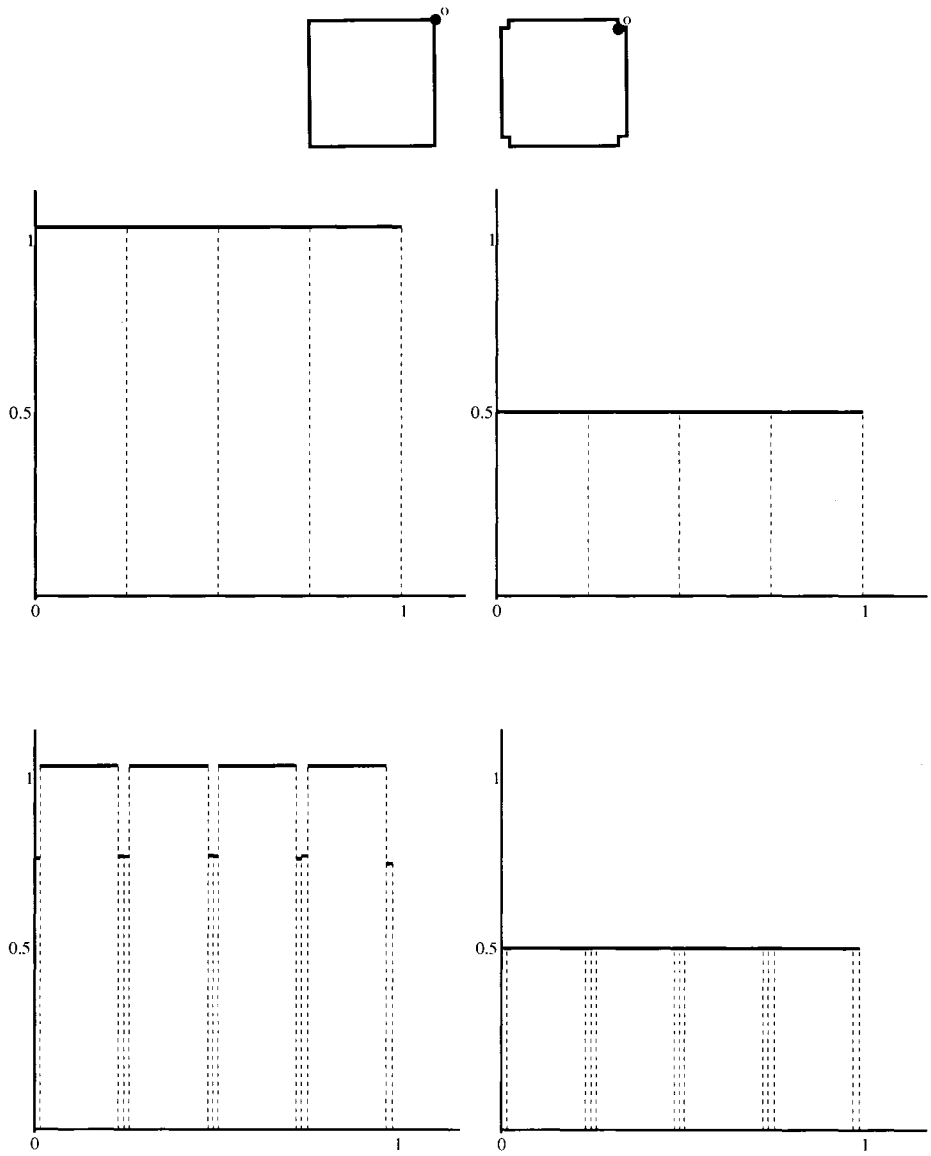
Standard Signature CG-Guided Signature

Figure 4.14: Results for Non-Convex Polygon - Example 4



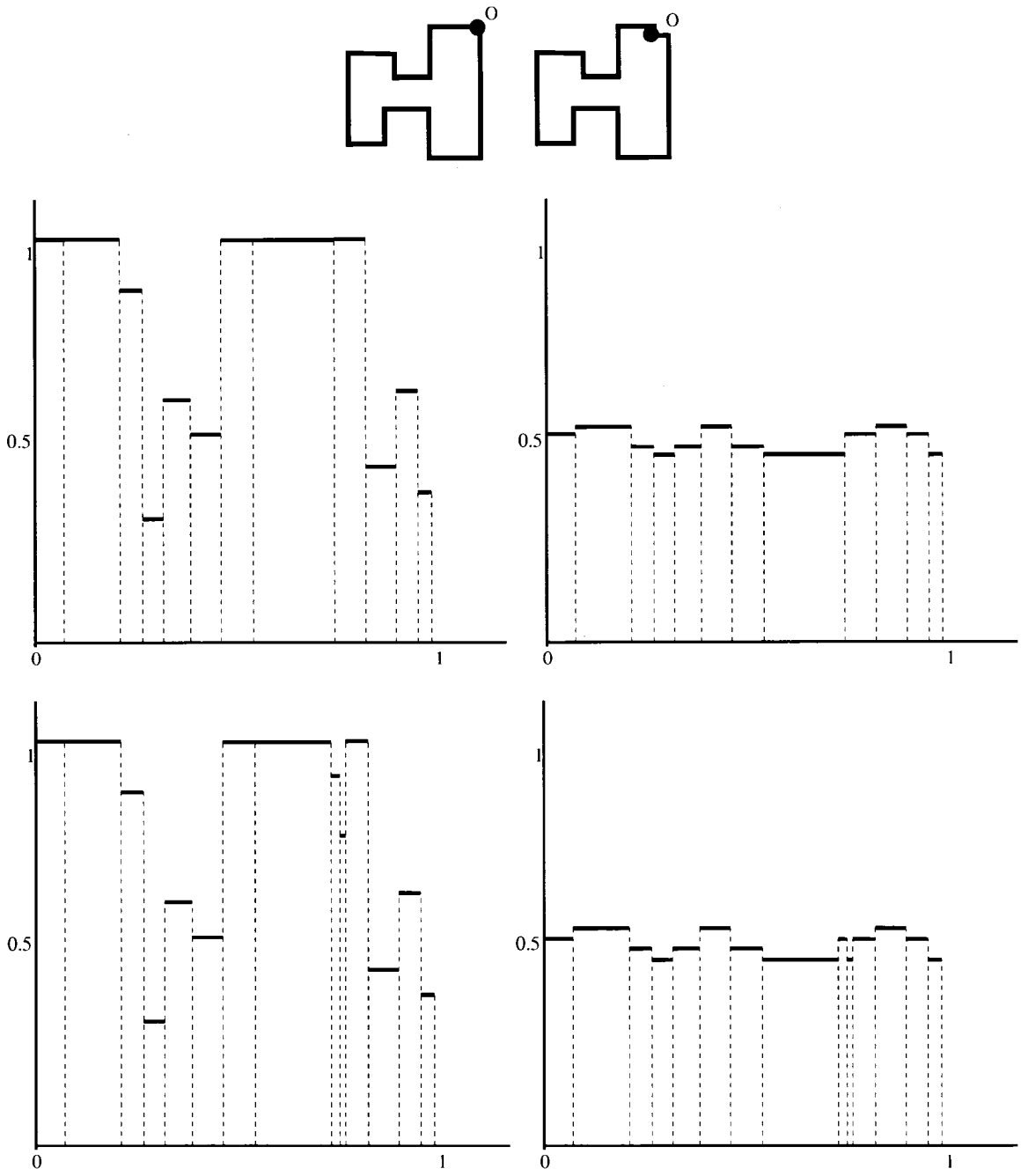
Standard Signature CG-Guided Signature

Figure 4.15: Results for Similar Polygons - Example 1



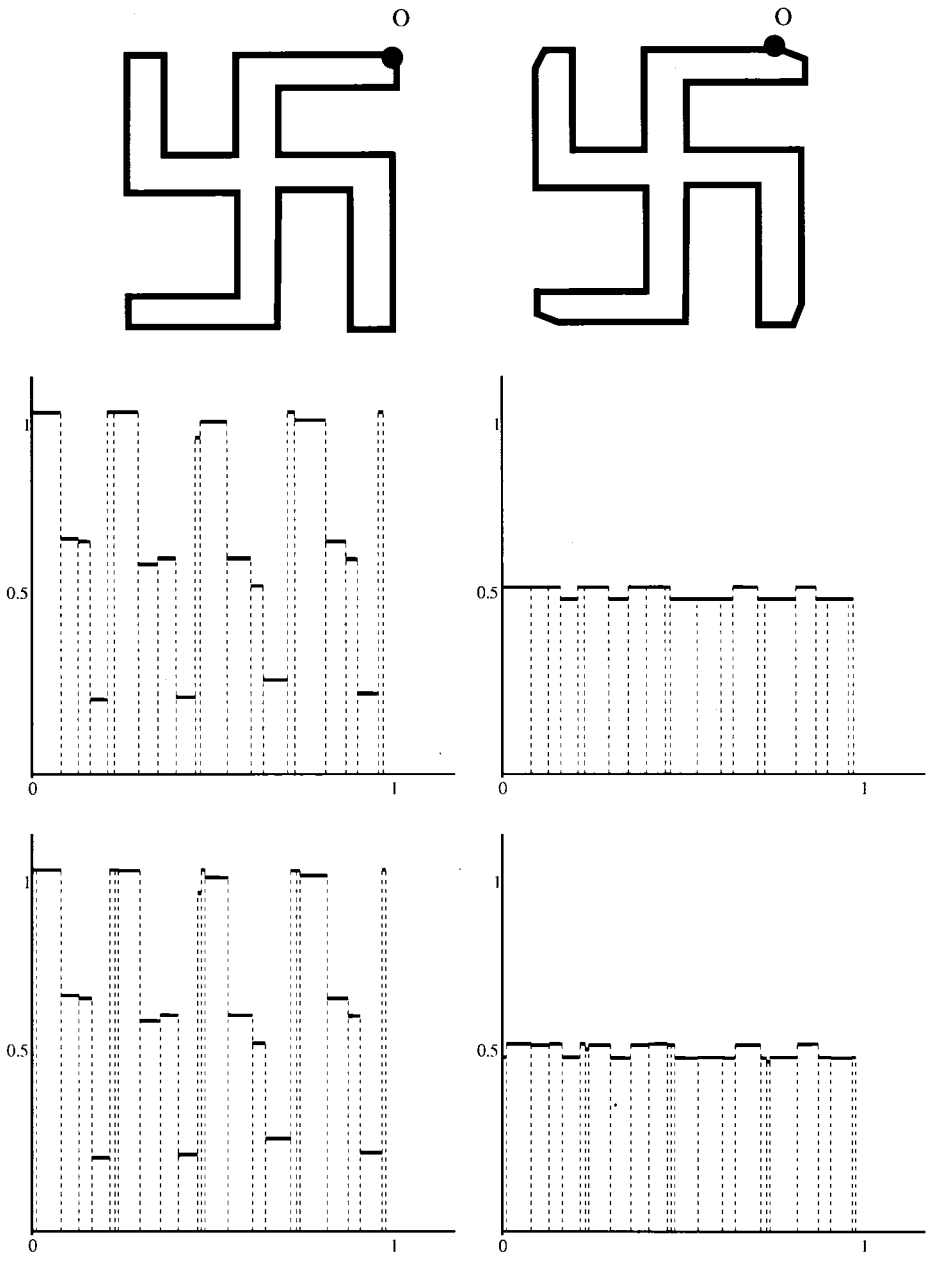
Standard Signature CG-Guided Signature

Figure 4.16: Results for Similar Polygons - Example 2



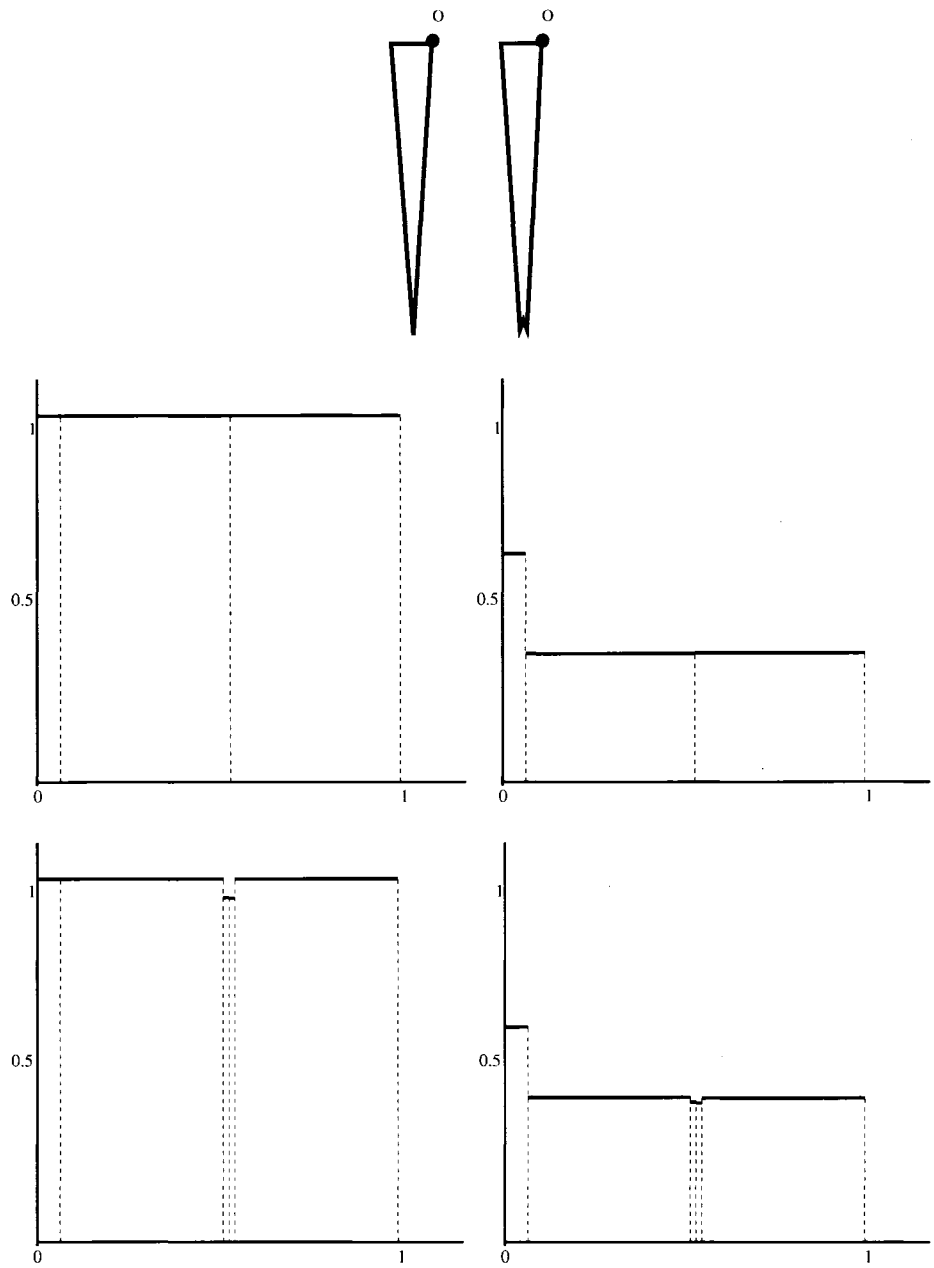
Standard Signature CG-Guided Signature

Figure 4.17: Results for Similar Polygons - Example 3



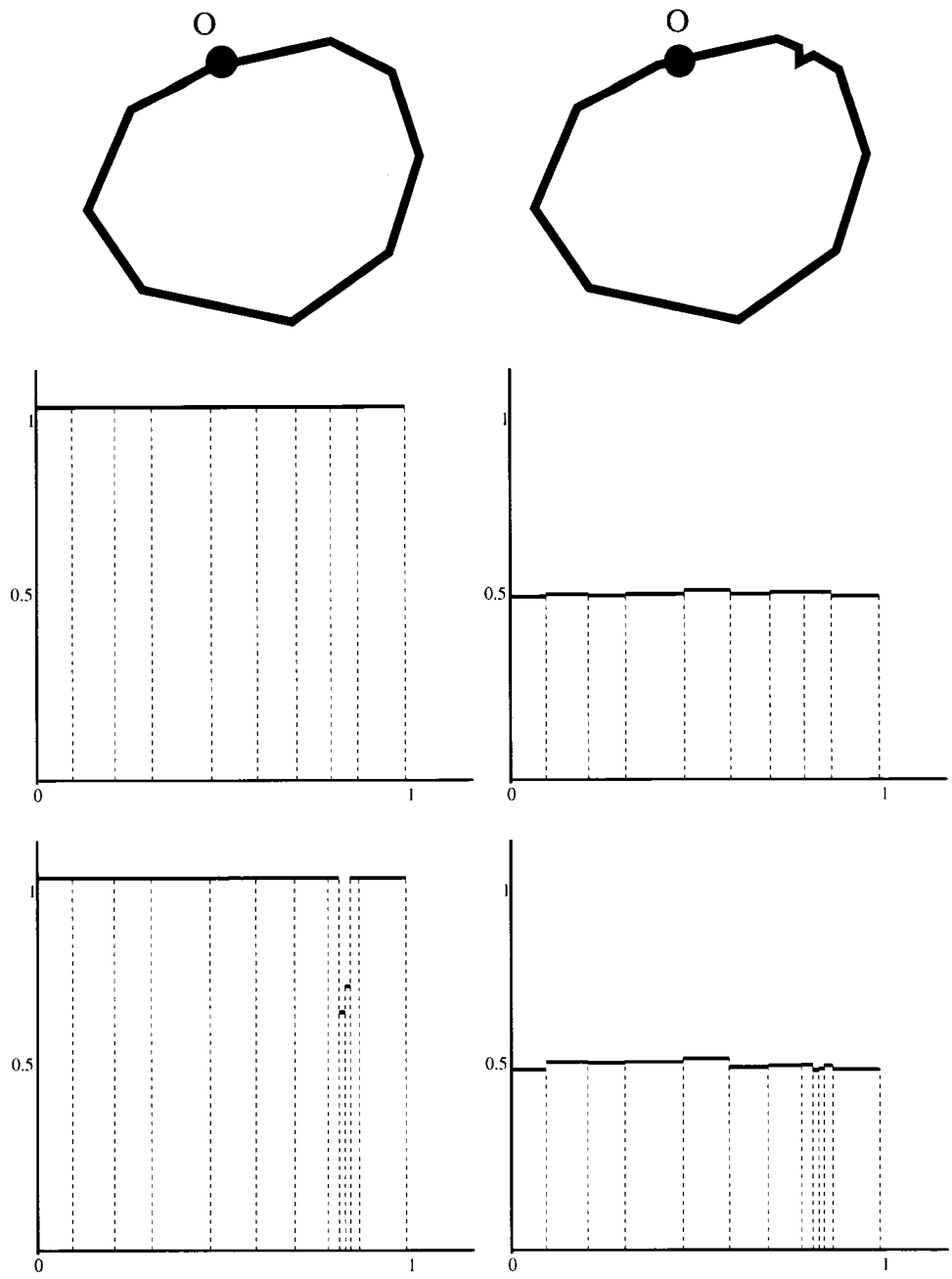
Standard Signature CG-Guided Signature

Figure 4.18: Results for Similar Polygons - Example 4



Standard Signature CG-Guided Signature

Figure 4.19: Results for Similar Polygons - Example 5



Standard Signature CG-Guided Signature

Figure 4.20: Results for Similar Polygons - Example 6

CHAPTER 5

CONCLUSION

We presented a brief review of existing geometric algorithms for measuring similarity between polygonal shapes. We introduced a variation of the standard signature called cg-guided signature which is capable of distinguishing convex polygons with different shapes. We presented an an algorithm for computing the cg-guided signature of polygons. The algorithm runs in $O(n^2)$ time. We also present an approach for removing noise edges from the input polygon. Removal of noise edges can be considered as a pre-processing step for computing signature. We also presented (i) an implementation of the proposed cg-guided algorithm and (ii) experimental investigation of signatures of similar looking shapes by using both the standard and the cg-guided signature. We executed the proposed algorithms on various shapes that contain noise edges. Furthermore, the results show that the cg-guided algorithm is very effective in capturing shape similarity for shapes having common features.

Several extensions of the proposed problem and algorithms can be planned for future work. One direction of investigation would be to perform extensive experimentation with many more shapes where the noise edges are injected randomly. Note that in our investigation we injected noise edges manually by visual inspection. One interesting issue would be to characterize class of polygons where the cg-guided algorithm would perform better

in general. In case of convex shapes, it would be interesting to characterize those shapes that would produce identical cg-giuded signatures.

REFERENCES

1. E. M. Arkin, L. Paul Chew, Daniel P. Huttenlocher, Klara Kedem, and J. S. B. Mitchell, "An Efficiently Computable Metric for Comparing Polygonal Shapes," IEEE Transactions on Pattern Analysis and Machine Intelligence, 13(1), 1991, pp. 209-215.
2. Eric Gulbert and Hui Lin, "B-Spline Curve Smoothing Under Position Constraints for Line Generalisation," Proceedings of ACM'GIS, 2006, pp. 3-10.
3. Martin Held and Johannes Eibl, "Biarc Approximations of Polygons Within Asymmetric Tolerance Bands," Computer-Aided Design, Vol. 37, 2005, pp. 357-371.
4. J. O'Rourke, "The Signature of a Plane Curve," Siam Journal on Computing, 15(1), 1986, pp.34-51.
5. J. ORourke, Computational Geometry in C, (Second Edition), Cambridge University Press, 1998.
6. J. ORourke, http://cs.smith.edu/ orourke ¡http://cs.smith.edu/ orourke¿
7. D. T. Lee, "SimilarityMeasurement Using Polygon Curve Representation and Fourier Descriptors for Shape-based Vertribral Retrieval," Proceedings of SPIE, Vol. 5032, 2003, pp. 1283-1291.
8. R. M. Rangayyan, D. Guliato, J. Carvalho, and S. Santiago, "Feature Extractiom from the Turning Angle Function for the Classification of Contours of Breast Tumors," Journal of Digital Imaging, 2007 (in press).

VITA

Graduate College
University of Nevada, Las Vegas

Hina Jain

Local Address:
2120 Ramrod Ave., Apt. 921
Henderson, Nevada 89014

Home Address:
84, Vivekanand Puri
Sarai Rohilla, Delhi -110007
India

Degrees:
Bachelor of Engineering, Information Technology, 2005
B.M. Institute of Engineering and Technology, India

Master of Science, Computer Science, 2007
University of Nevada, Las Vegas

Thesis Title: Center of Gravity Guided Signature of Planar Shapes

Thesis Examination Committee:
Chairperson, Dr. Laxmi P. Gewali, Ph. D.
Committee Member, Dr. David Pinelle, Ph. D.
Committee Member, Dr. Yoohwan Kim, Ph. D.
Graduate Faculty Representative, Dr. Rama Venkat, Ph. D.