5-2011

Sharp feature identification in a polygon

Joseph P. Scanlan
University of Nevada, Las Vegas

SHARP FEATURE IDENTIFICATION IN A POLYGON

by

Joseph P. Scanlan

Bachelor of Science
University of Nevada, Las Vegas
1983

A thesis submitted in partial fulfillment of
the requirement for the

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

Graduate College
University of Nevada, Las Vegas
May 2011

UNIVERSITY OF NEVADA LAS VEGAS

THE GRADUATE COLLEGE

We recommend the thesis prepared under our supervision by

Joseph P. Scanlan

entitled

Sharp Feature Identification in a Polygon

be accepted in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science
School of Computer Science

Laxmi Gewali, Committee Chair

Evangelos A. Yfantis, Committee Member

Jan B. Pedersen, Committee Member

Henry Selvaraj, Graduate Faculty Representative

Ronald Smith, Ph. D., Vice President for Research and Graduate Studies
and Dean of the Graduate College

May 2011

ABSTRACT

Sharp Feature Identification in a Polygon

by

Joseph P. Scanlan

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

This thesis presents an efficient algorithm for recognizing and extracting sharp-features from polygonal shapes. As used here, a sharp-feature is a distinct portion of a polygon that is long and skinny. The algorithm executes in $O(n^2)$ time, where n is the number of vertices in the polygon. Experimental results from a Java implementation of the algorithm are also presented.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Problems dealing with the recognition and simplification of two dimensional shapes have been investigated extensively in robotics, geographic information system, medical imaging, and computational geometry. In computational geometry [1, 8, 10], two dimensional shapes are usually modeled by polygons. An important sub-problem used in shape recognition is the formulation of a "shape-similarity" measure. One of the first geometric algorithms for measuring shape similarity is based on the concept of a "signature function" [15]. The signature function is essentially a rectilinear step function derived from the local and global properties of the polygonal shape. The signature of an edge e of a polygon is obtained by accumulating the portion of the boundary of the polygon lying to the left of the line passing through e. The signature of the whole polygon is obtained by combining the signatures of all the edges of the polygon. If two shapes are similar then the area enclosed between their signatures is very small. For identical shapes the area enclosed between their signatures is zero. The technique of signature analysis has been found to be very effective for comparing orthogonal shapes and in recognizing hand-written characters [15]. Algorithms for measuring shape similarity based on signature functions are not easy to implement. Another approach for measuring shape similarity is based on the notion of a "turning function". The turning function of a polygonal shape is also a rectilinear step function. Shape-similarity measures based on the turning function have been used successfully for developing efficient recognition algorithms and these algorithms are not difficult for practical implementation. One drawback of the turning function method is that it does not produce acceptable results when the boundary of the input polygon contains noisy edges.

Chapter 2 presents a review of the existing boundary simplification algorithms

for two dimensional shapes. Chapter 3 proposes a new technique for shape recognition. This technique is based on a partitioning procedure that separates narrow regions from the core regions. The notion of "sharp-features" for 2-d shapes is introduced for performing such partitioning. Efficient algorithms for identifying such features are presented. The sharp feature recognition algorithm runs in $O(n^2)$ time, where n is the number of vertices in the polygon. Chapter 4 describes an implementation used to demonstrate the algorithm. Chapter 5 concludes with a discussion of possible extensions of the proposed technique.

The main contributions of this thesis are *i*) formulation of the concept of sharp features of two dimensional shapes, *ii*) development of an $O(n^2)$ algorithm for extracting sharp components from a polygon, and *iii*) experimental study of the proposed algorithm by actual implementation in Java.

CHAPTER 2

BOUNDARY SIMPLIFICATION

In this chapter we present an overview of the algorithmic techniques, particularly from computational geometry, for approximating the boundary of a complex polygonal shape with a simpler one. While a complex polygonal boundary contains large numbers of vertices, its simplification is expected to contain a smaller number of vertices. In performing boundary simplification, it is necessary to preserve the polygon's global structure and properties. Boundary simplifications have application in medical image processing [9, 13], data compression [6], cartography [5], GIS [14], and remote sensing [19]. For example, the images obtained by remote sensing techniques contain complex boundary detail. Analyzing such complex boundaries require huge amount of memory and processing time. In such cases it is desirable to replace a boundary with very large numbers of nodes with one that has fewer nodes.

2.1 Preliminaries

Interest in automated boundary simplification spans more than fifty years [18]. The input boundary is available either as a closed loop (representing the boundary of a simple polygon) or an open chain that represents a portion of linear features and contours. The strategies applied for boundary simplification depend on whether the boundary is closed or not. While representing a complex boundary with fewer number of vertices, it is necessary to preserve two types of properties *i*) linear structural properties and *ii*) shape properties. Examples of linear structural properties include winding number, number of maximal convex chains, number of spiral chains, etc. Similarly, shape properties include enclosed area, convex-hull, internal / external visibility properties, centroid, etc. In most cases, linear structural properties are applicable to both closed and open boundaries. On the other hand, shape properties are mostly applicable only to closed boundaries.

Two types of simplification approaches have been considered for boundary simplification problems [12]. In the *vertex-elimination* approach, the objective is to describe the polygonal shape with as few vertices as possible without exceeding a given error tolerance ϵ. In the *solution size* approach, the objective is to minimize the error for obtaining the given number of vertices in the approximated solution.

2.2 Boundary Vertex Retention

One approach to polygonal boundary simplification is to remove unneeded vertices from the boundary. Which vertices are not needed will be determined by the intended use of the final approximated polygon but there are some intuitive criteria that should always apply. The polygon after simplification should *i*) look similar to the original polygon to a human observer, *ii*) the resulting enclosed area should be close to the area of the original polygon, and *iii*) the centroid of the original and simplified polygon should not be far apart. Note that the perimeter of the simplified polygon could be shorter than the perimeter of the original polygon.

When David Douglas and Thomas Peucker published their algorithm [4] for polygonal chains, they cited storage capacity and bandwidth as motivators for simplification. While the ability to store data has greatly increased, so has the ability to *generate* data. Polygonal chain simplification remains just as relevant today as it was in a world of punch cards and paper tape.

Algorithm 2.1 is the iterative "method 1" from the 1973 paper. A recursive variation of Algorithm 2.1 is presented as Algorithm 2.2. In this variation, a subset of vertices of the original polygonal chain are eliminated by using the predetermined tolerance error ϵ. The algorithm is developed for open polygonal chains. The algorithm works in a sequence of recursive calls. If the vertices along the boundary chain are $p_0, p_1, \ldots, p_{n-1}$ then the algorithm finds errors ϵ_is for vertices $p_1, p_2, \ldots, p_{n-2}$. The error corresponding to vertex p_i is denoted by ϵ_i. The error for p_i is measured in

Algorithm 2.1: DouglasPeucker-Iterative

```
1: Chain Ch_s DouglasPeucker(Chain Ch, Error ε);
2: // Chain Ch has vertices p_0, p_1, ..., p_{n-1}
3: // Chain Ch_s has vertices s_0, s_1, ..., s_{m-1} where m <= n
4: a = p_0;
5: f = p_{n-1};
6: Ch_s = < a >;
7: while a ≠ p_{n-1} do
8:     for each vertex between a and f do
9:         ε_j = perpendicular distance from p_k to the line passing through a
           and f;
10:        ε_j = largest amongst ε_k s;
11:        p_j = the vertex corresponding to ε_j;
12:        if ε_j < ε then
13:            a = f;
14:            Ch_s += < a >;
15:        else
16:            f = p_j;
17:        end if
18:    end for
19: end while
```

Algorithm 2.2: DouglasPeucker-Recursive

```
1: Chain Ch_s DouglasPeucker(Chain Ch, Error ε);
2: // Chain Ch has vertices p_0, p_1, ..., p_{n-1}
3: // Chain Ch_s has vertices s_0, s_1, ..., s_{m-1} where m <= n
4: for k = 1 to n − 2 do
5:     ε_k = perpendicular distance from p_k to the line passing through p_0
       and p_{n-1};
6: end for
7: ε_m = largest amongst ε_1, ε_2, ..., ε_{n-2};
8: p_m = the vertex corresponding to ε_m;
9: if ε_m < ε then
10:    return p_{n-1}
11: else
12:    return concat(DouglasPeucker(p_0, p_k), DouglasPeucker(p_k, p_{n-1}))
13: end if
```

terms of the perpendicular distance from p_i to the line through vertices p_0 and p_{n-1}. If all errors $\epsilon_1, \epsilon_2, \ldots, \epsilon_{n-2}$ are less than the predetermined *threshold* value ϵ then the segment $\overline{p_0, p_{n-1}}$ is taken as the approximation for chain $< p_0, p_1, \ldots, p_{n-1} >$. Otherwise, the vertex p_m corresponding to the maximum error ϵ_m is determined. Then the algorithm recurses on the chains $< p_0, p_1, \ldots, p_k >$ and $< p_k, p_{k+1}, \ldots, p_{n-1} >$. Note that if more than one of the vertices are maximum then ties are broken arbitrarily. The time complexity of the algorithm is $O(n^2)$.

Note that Algorithm 2.1 travels along the chain in a linear fashion while Algorithm 2.2 uses a divide and conquer approach.

With the polygonal chain shown in Figure 2.1, Figure 2.2 illustrates the

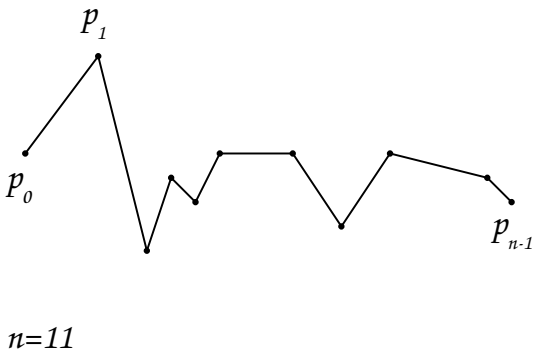


Figure 2.1: Polygonal chain with 11 vertices.

execution trace of Algorithm 2.2 for the indicated error threshold value. Figure 2.3 shows the approximated chain which contains five vertices in comparison to eleven vertices in the original chain. If the error threshold ϵ is small, then no vertices will be eliminated.

2.3 Boundary Vertex Elimination

An alternative to removing unimportant boundary vertices is to identify important boundary vertices and remove the rest.

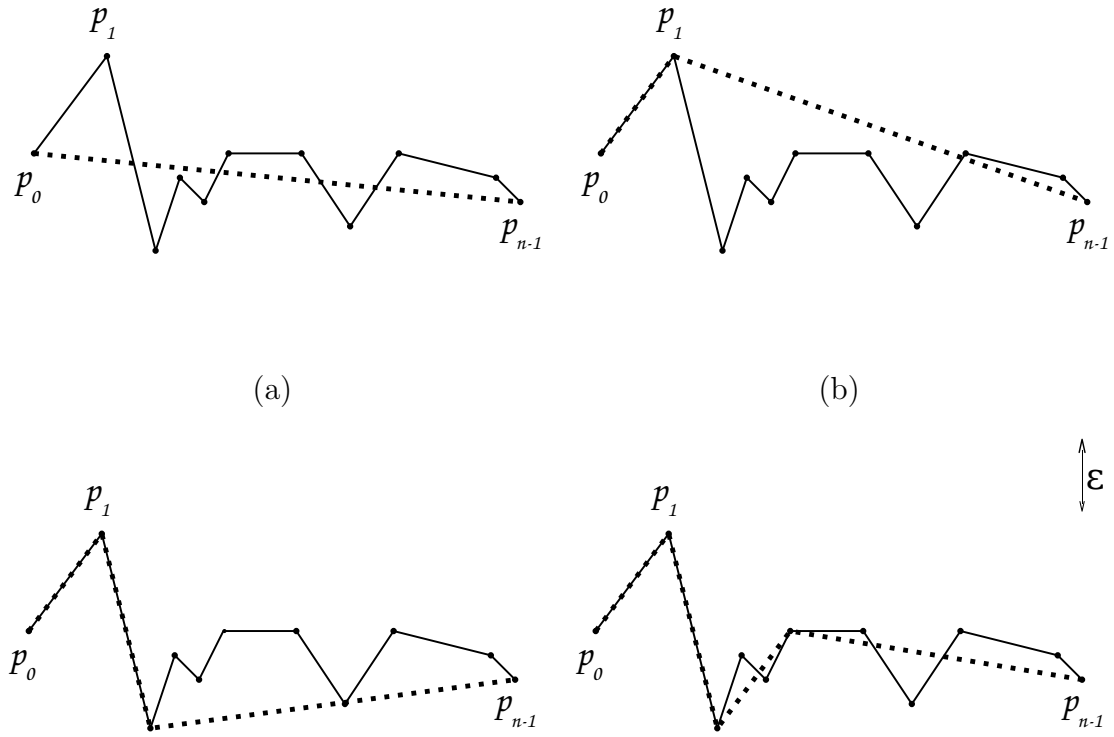


Figure 2.2: Execution of Douglas Peucker Algorithm.

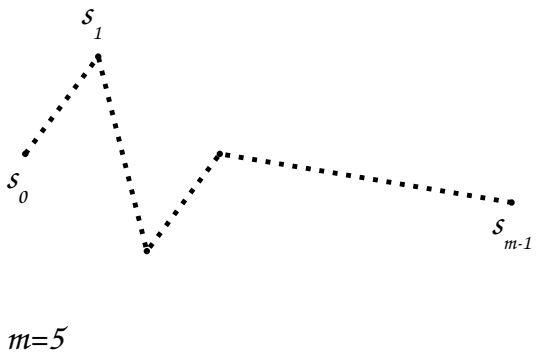


Figure 2.3: Polygonal chain with 5 vertices.

Algorithm 2.3: PikazDinstein

```
1:  Polygon P_s PikazDinstein(Polygon P, Error ε);
2:  // Polygon P has vertices p_0, p_1, ..., p_{n-1}
3:  // Polygon P_s has vertices s_0, s_1, ..., s_{m-1} where m <= n
4:  Attach to each vertex its error value, p_i.error;
5:  Build(Heap) from vertices p_0, p_1, ..., p_{n-1};
6:  repeat
7:      DeleteMin(Heap, p_i);
8:      Compute new error values for p_i's two neighbors p_{i-1} and p_{i+1};
9:      Remove p_i from polygon P;
10:     Update(Heap, p_{i-1});
11:     Update(Heap, p_{i+1});
12: until p_i.error > ε;
```

Arie Pikaz and Its'hak Dinstein [17] published an algorithm that reduces the number of vertices in a polygon by iteratively eliminating vertices from a polygon. This method is applicable to both open and closed chains.

Line 4 of Algorithm 2.3 assigns an error value to each vertex in the polygon. Two error criteria are discussed in [17] *i*) area of the triangle formed by p_{i-1}, p_i, and p_{i+1}; and *ii*) the perpendicular distance from p_i to the line passing through p_{i-1} and p_{i+1}. The example uses perpendicular distance as the error criteria. After this is done, a min-heap is built using the error value as a key.

Lines 6 through 12 repeatedly remove the vertex with the smallest error from the polygon and the heap, adjusts the error of the removed vertex's neighbors, and adjusts the heap.

Figure 2.4 is a sample polygon with 13 vertices. Figure 2.5 illustrates progress of the algorithm on the sample polygon shown in Figure 2.4. Figure 2.6 shows the approximated polygon with only 7 vertices.

Figure 2.5(c) illustrates an interesting point. Because error values are recalculated for two neighbor vertices during the execution of the algorithm, error values may not be increasing as vertices are removed. However, as pointed out in [17] the

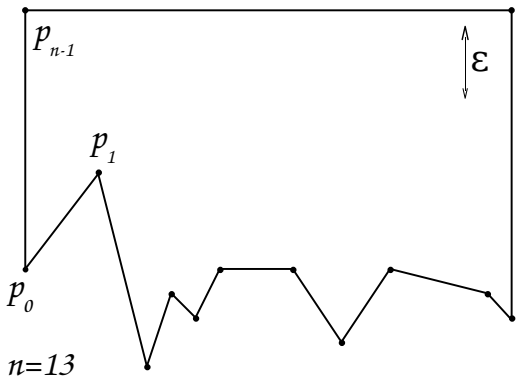


Figure 2.4: Polygon with 13 vertices.

neighbors' errors will only increase if the deleted vertex and its neighbors p_{i-1} and p_{i+1} are in the same convex chain.

The time complexity of lines 4 and 5 in Algorithm 2.3 is $O(n)$.

Errors for each vertex can be found in $O(n)$ time by scanning them from start to end in the polygon and computing the perpendicular distance. Heap operations at lines 7, 10, and 11 take $O(\log n)$ time. Computation of new error values and removing vertices from the polygon can be done in linear time, given appropriate data structures. The time complexity of the algorithm depends on the value of tolerance error ϵ. If k vertices are eliminated, the total time complexity is $O(k \log n)$. This follows from the fact that lines 7, 10, and 11 take $O(\log n)$ time each and lines 8, and 9 take $O(1)$ time.

2.4 Boundary Replacement

This section considers the problem of replacing the boundary of a polygon with a boundary of fewer vertices, where the new vertices may not have been in the original polygon at all.

Hiroshi Imai and Masao Iri [11] describe an algorithm that replaces a strictly monotone polygonal chain with a new polygonal chain containing fewer vertices. This

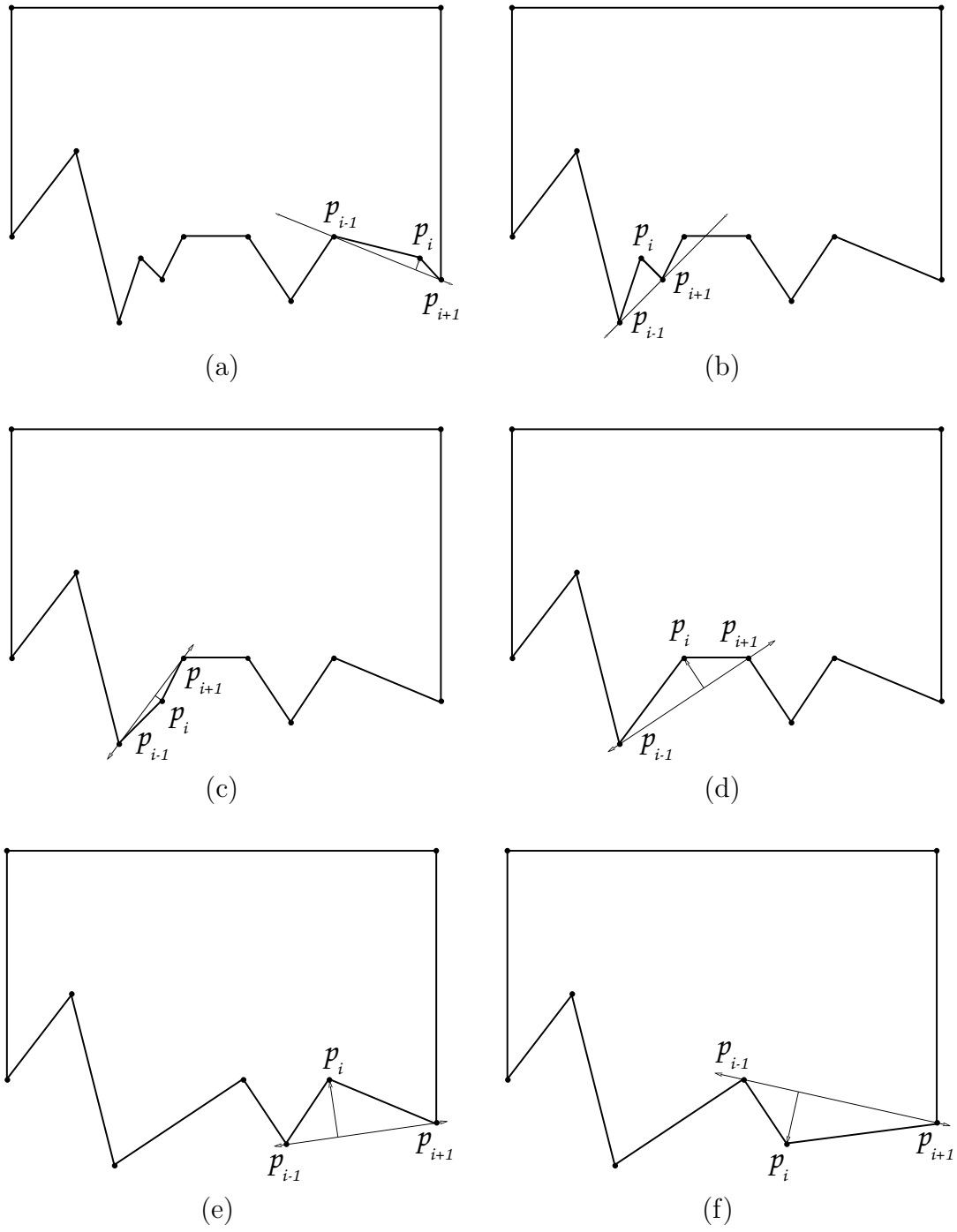


Figure 2.5: Execution of Pikaz Dinstein Algorithm.

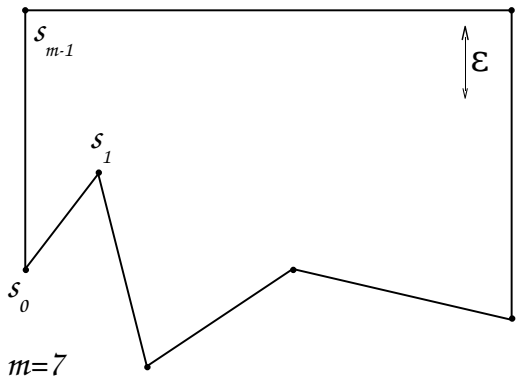


Figure 2.6: Polygon with 7 vertices.

algorithm can be used for simple polygons by breaking the polygon boundary into separate monotone chains. A brief overview of this algorithm can be described as follows.

The algorithm takes the monotone chain $Ch =< p_0, p_1, \ldots, p_{n-1} >$ and predetermined threshold error ϵ as the input. Without loss of generality, the chain Ch can be assumed to be monotone along the x-axis. The algorithm constructs a monotone polygon $P(\epsilon)$ by shifting the chain Ch above and below by ϵ, as shown in Figure 2.7. Specifically, let the vertices of the chain formed by shifting Ch above by ϵ be $Ch_q =< q_0, q_1, \ldots, q_{n-1} >$. Similarly the lower chain is $Ch_r =< r_0, r_1, \ldots, r_{n-1} >$. The polygon $P(\epsilon)$ is formed by connecting r_0 to q_0 forming the *leftmost window* e_0 and r_{n-1} to q_{n-1} forming the *rightmost window* e_{m-1}. Creation of these windows is shown in Figure 2.8.

It can be easily verified that any monotone chain connecting the leftmost chain to the rightmost chain will be within ϵ error from the original chain Ch.

The algorithm is based on finding a monotone chain with minimum number of vertices that connects e_0 with e_{m-1}. For constructing such a chain the algorithm uses the concept of visibility polygon from line segment. It finds the edge-visibility polygon from e_0. To determine the first *internal window* e_1, let $VP(P, e_0)$ denote the

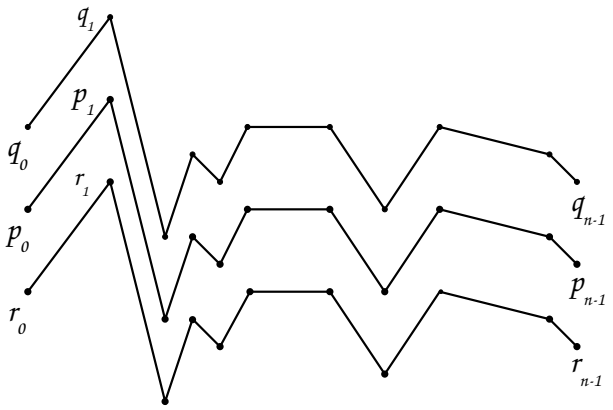


Figure 2.7: Monotone chain with corresponding upper and lower chains.

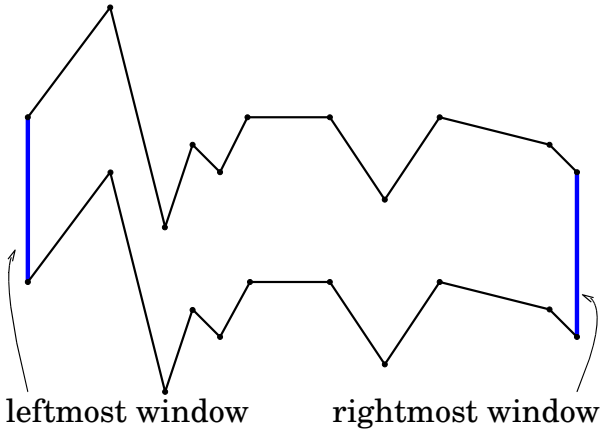


Figure 2.8: Strip polygon.

Algorithm 2.4: ImaiIri

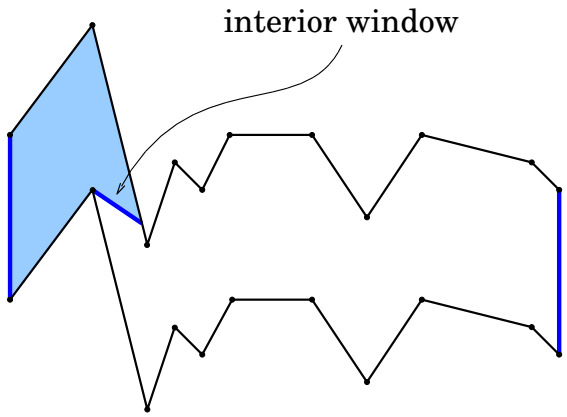
1: Chain Ch_s ImaiIri(Chain Ch, Error ϵ);
2: // Chain Ch has vertices $p_0, p_1, \ldots, p_{n-1}$
3: // Chain Ch_s has vertices $s_0, s_1, \ldots, s_{m-1}$ where $m <= n$
4: Construct strip polygon $P(\epsilon) = r_0, r_1, \ldots, r_{n-1}, q_{n-1}, q_{n-2}, \ldots, q_0$;
5: $e_0 = \overline{q_0 r_0}$;
6: $P_0 = P(\epsilon)$;
7: $i = 0$;
8: **while** $\overline{q_{n-1} r_{n-1}}$ *is not visible from* e_i *in polygon* P_i **do**
9: $\quad e_{i+1}$ = the window from e_i to $\overline{q_{n-1} r_{n-1}}$ in P_i;
10: $\quad P_{i+1}$ = the invisible polygon of e_i containing $\overline{q_n r_n}$ in polygon P_i;
11: $\quad s_i$ = point of Intersection of edge e_i and the line containing e_{i+1};
12: $\quad i = i + 1$;
13: **end while**
14: $m = i + 1$;
15: Find the point s_{m-1} on edge e_{m-1} and a point s_m on $\overline{q_{n-1} r_{n-1}}$ which are visible from each other in polygon P_{m-1};
16: **return** $< s_0, s_2, \ldots, s_{m-1} >$;

edge visibility polygon from e_0. Let P' be the invisible sub-polygon of P containing the rightmost window e_{m-1}. The common boundary between $VP(P, e_0)$ and P' gives the first internal window e_1. The algorithm determines the second internal window e_2 by computing $VP(P', e_1)$. This is continued until the rightmost window e_{m-1} is visible from the most recent internal window. Computation of internal windows and the construction of the final approximation chain for the strip polygon in Figure 2.8 are illustrated in Figure 2.9 and Figure 2.10

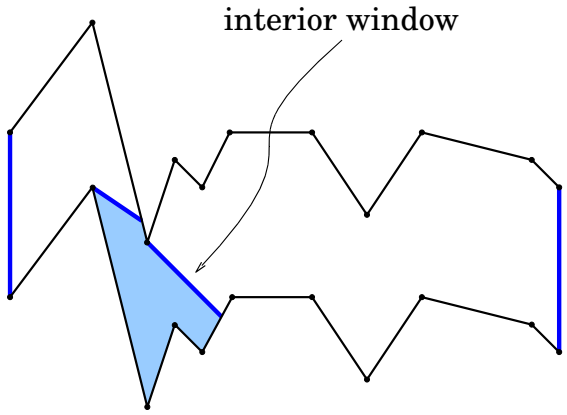
Figure 2.11 shows the original chain overlaid with it approximation. A formal sketch of the algorithm is listed as Algorithm 2.4.

Time complexity analysis can be done in a straightforward manner. Construction of the monotone polygon (line 4) from the given monotone chain can be done trivially in $O(n)$ time.

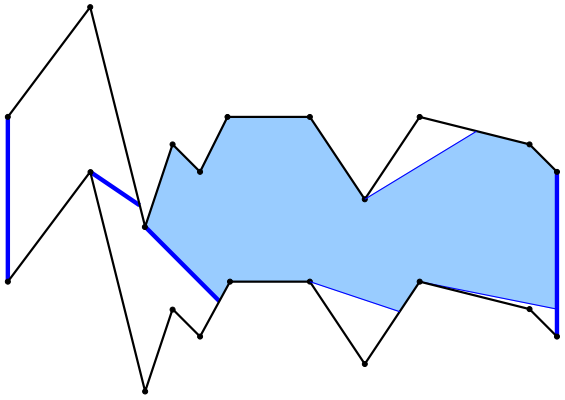
Imai and Iri [11] have shown that by careful use of the convex hull of a monotone chain, the interior windows can be determined in $O(n)$ time. Hence lines 5



(a)



(b)



(c)

Figure 2.9: Interior windows and visibility polygons.

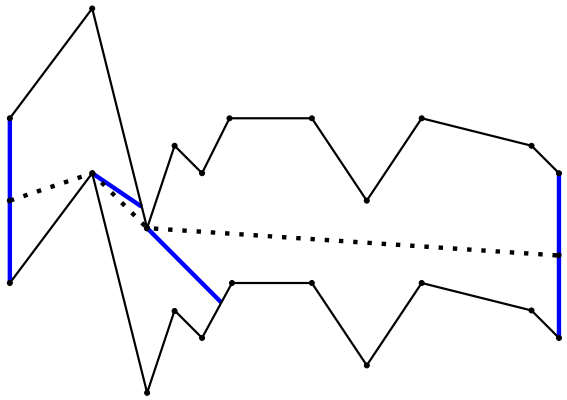


Figure 2.10: Approximation chain shown with strip polygon and interior windows.

through 12 takes $O(n)$ time.

Lines 14 through 16 take $O(n)$ time. Hence Algorithm 2.4 can be executed in $O(n)$ time.

2.5 More Boundary Vertex Retention

A class of boundary chain approximation algorithms have been reported where the approximated solution contains only the sub-set of vertices of the original chain as in the Douglas-Pucker algorithm. We briefly describe a few such algorithms here.

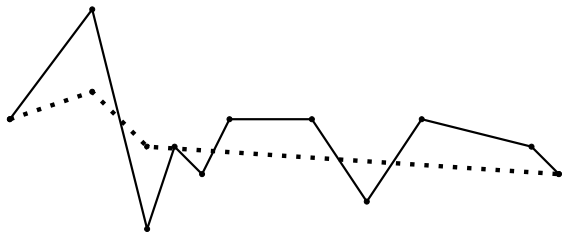
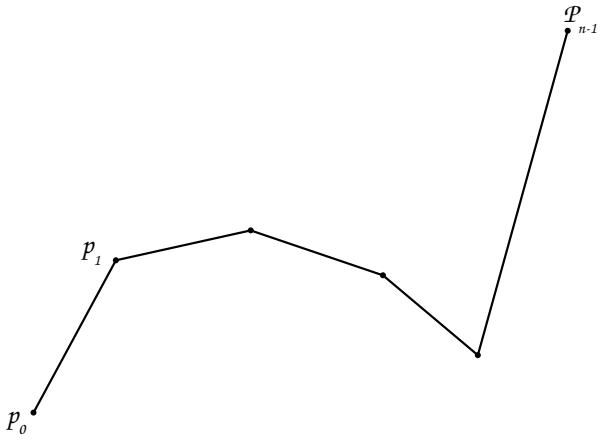
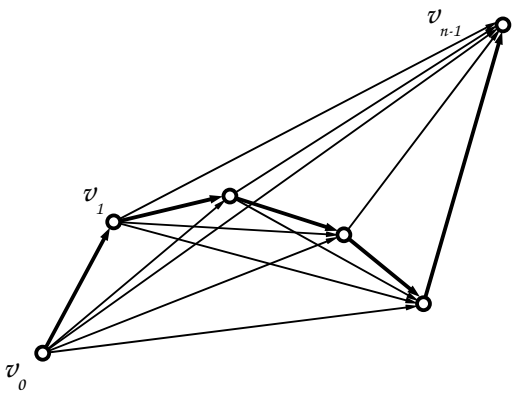


Figure 2.11: Monotone chain shown with its approximation.



(a) Given Chain.



(b) Resulting Directed Acyclic Graph.

Figure 2.12: Directed Graph.

2.5.1 Directed-Graph Approach

Imai and Iri [11] have reported a boundary approximation algorithm based on the construction of directed graphs from the given chain $Ch =< p_0, p_1, \ldots, p_{n-1} >$. The set of vertices V of the directed graph $G(V, E)$ corresponds exactly to the set of vertices of the chain Ch, that is $V = \{v_i | p_i \in Ch\}$. Two vertices v_i and v_j are connected by an edge $\overrightarrow{v_i v_j}$ if $i < j$. It is straightforward to see that the resulting graph $G(V, E)$ is a directed acyclic graph. Figure 2.12 shows an example of constructing the directed graph for a given chain.

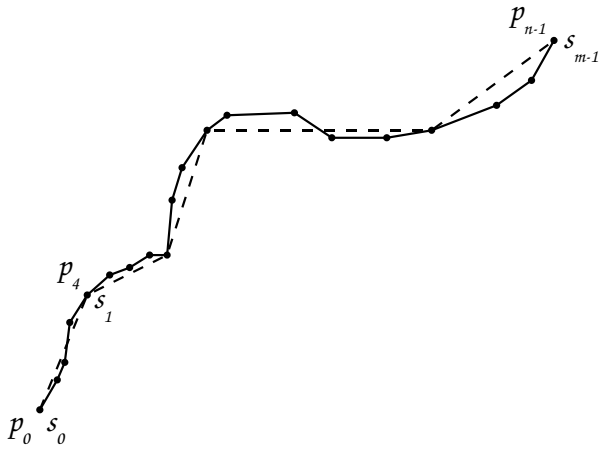


Figure 2.13: Monotone chain with its approximation.

2.5.2 Area-Preserving Approximation

When the boundary of a polygon is approximated by a simpler chain, the area enclosed by the approximated chain changes. It is thus interesting to seek for the development of boundary approximation algorithms that preserve the enclosed area. Very few research results have been reported on polygonal chain approximation that address preservation of enclosed area. One of the first interesting algorithms on *area-preserving* chain approximation was given by Bose, et al. [2]. This paper deals with the approximation of monotone chains. Consider an x-monotone chain with vertices n vertices and its approximation by a chain of m vertices as shown in Figure 2.13. In the figure the approximated chain is drawn with dashed edges. The area enclosed between the original chain and the approximated chain can be distinguished into two kinds: *i*) The enclosed area lying below the original chain can be viewed as *loss-area*. *ii*) Similarly, the enclosed area lying above Ch is *gain-area*. In Figure 2.14 the loss-area and gain-area are drawn shaded. Three criteria have been considered in [2] to obtain the approximation. In the first criteria the objective is to minimize the sum of enclosed areas without distinguishing loss or gain. In the second criteria, the objective is to minimize the maximum of total loss-area and total gain-area. The

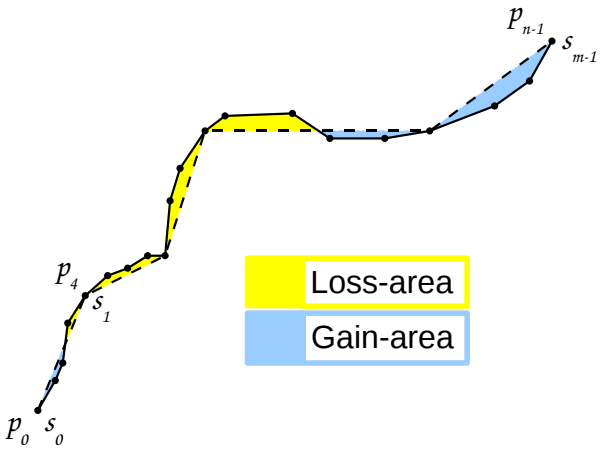


Figure 2.14: Highlighted Loss-area and Gain-area.

objective of the third criteria is to minimize the difference between total loss-area and total gain-area.

2.5.3 Boundary Simplification and Rendering

Almost all boundary simplification algorithms have been developed by modeling boundary as a sequence of line segments without addressing the issue of digitization. For practical applications in GIS and medical imaging, the approximated boundary should preserve shape properties even when it is digitized and rendered in graphic display devices. One of the first results on curve simplification that examines the issue of rendering has been reported by Lilian Buzer in [3]. Buzer's paper introduces the digital zoning criteria for estimating approximation error.

In the standard way, a line segment connecting two given pixels can be uniquely digitized by making use of Bresensham's line drawing algorithm. The digital zoning technique reported in [3] shows that a line segment connecting two pixels can be digitized in more than one way. The existence of more than one solution is useful for polygonal chain approximation. The idea is to select the solution that best fits the approximation. It is argued in [3] that the approximation based on digital zoning

criteria results in solutions that do not compromise topological consistency. The algorithm presented in [3] runs in $O(n \log n)$ time. Furthermore, the approximated solution is within half a pixel from the original chain.

CHAPTER 3

SHARP FEATURE RECOGNITION

3.1 Characterizing Sharp Features of Polygonal Shapes

Examples of polygonal shapes (the outlines of a tea pot and a spider) are shown in Figure 3.1 and Figure 3.2. Consider a polygonal shape P, such as the

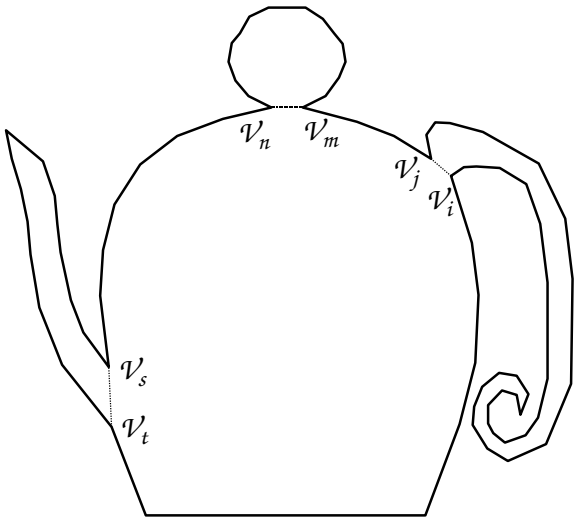


Figure 3.1: Tea pot polygon and its subpolygons.

one in Figure 3.1, whose vertices, in counterclockwise order along the boundary, are $v_0, v_1, \ldots, v_{n-1}$. The *sub-polygon* $P(m, n)$ induced by vertices v_m and v_n is the portion of the polygon lying to the right of the diagonal (v_m, v_n).

Some sub-polygons are *round* or *broad* and others are *sharp* or *skinny*. In Figure 3.1, the sub-polygon to the right of the diagonal (v_m, v_n) is broad or round and those to the right of the diagonals (v_s, v_t) and (v_i, v_j) are *skinny* or *sharp*. The notion of sharpness of a sub-polygon can be formalized in terms of its structural properties. It can be observed that for a given perimeter, the area enclosed by a round sub-polygon is more than the area enclosed by a sharp sub-polygon. This leads to the following definition.

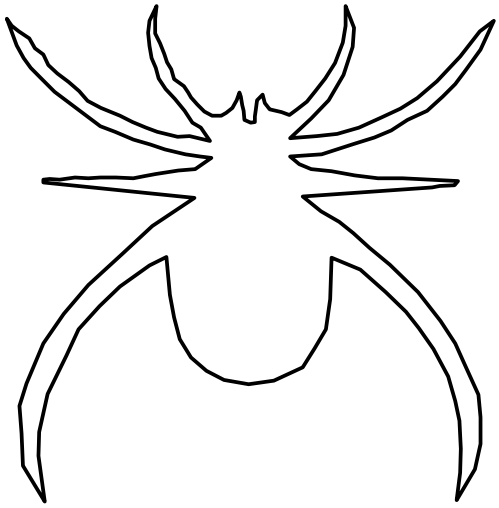


Figure 3.2: Spider polygon.

Definition 3.1. The *sharpness* of a sub-polygon $P(i,j)$ denoted $\alpha(i,j)$ is defined in terms of the ratio of perimeter squared over the enclosed area, that is $\alpha(i,j) = \frac{R(i,j)^2}{A(i,j)}$ where $R(i,j)$ and $A(i,j)$ represent the perimeter and area of the sub-polygon $P(i,j)$, respectively.

It is noted that the value of sharpness is small for circular shapes and for skinny ones it can become very large. It can be easily verified that for circular shapes the sharpness value can be a minimum of 4π and for very skinny ones it becomes very high, approaching infinity, for long, narrow, hair-like shapes. The lid of the tea pot in Figure 3.1 has a small sharpness value, 12.93. The spout and handle are much sharper with values of 48.80 and 103.16, respectively.

Sharpness values for some example shapes are given in Table 3.1.

For identifying and extracting sharp-features (i.e., sharp sub-polygons) of a polygonal shape, additional conditions of sub-polygons listed below are used. Preliminary experiments suggest that polygons with α values ≥ 20 are perceptually sharp. This leads to the following condition.

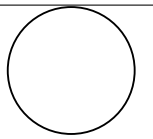
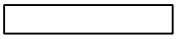
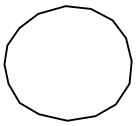
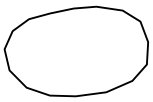
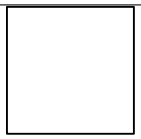
	12.57		32.66
	12.81		67.17
	13.92		81.23
	16.00		159.41
	16.76		288.57

Table 3.1: Sample sharpness values.

Condition 3.1. (Sharpness threshold)

A sub-polygon $P(i, j)$ is sharp if its sharpness value $\alpha(i, j)$ is at least 20.

Remark. The threshold value of 20 was chosen rather subjectively. Another value could have been chosen to formalize the concept of sharpness.

Closer examination of the boundary of a complex polygon shows sharp features generally start from a reflex vertex. This leads to the next condition.

Condition 3.2. (Originating from reflex vertex)

At least one end-point of the diagonal separating a sharp feature is incident on a reflex vertex.

The area of a sharp feature must not be a large fraction of the area of the whole polygon. An analogous condition applies for the perimeter.

Condition 3.3. (Threshold area and threshold perimeter)
The perimeter $R(i,j)$ (or area $A(i,j)$) of the sub-polygon $P(i,j)$ is no more than δ_1 (δ_2) times the perimeter (area) of the whole polygon. A typical value of δ_1 could be 0.35.

For most maximal sharp features the diagonal that separates it from the whole polygon is of relatively shorter length.

Condition 3.4. (Short Diagonal)
The diagonal $d = (v_i, v_j)$ on which the sub-polygon is subtended should be of short length. 0.24 of the total perimeter works in tests.

3.2 Algorithm Development

An algorithm for identifying sharp features of a polygon uses the visibility graph of the polygon to identify candidate diagonals on which sharp features can be subtended. Note that the visibility graph of a polygon is the graph $VG(V,E)$, where V is the set of vertices of the polygon and E is the set of its internal diagonals. This is illustrated in Figure 3.3. The algorithm examines each diagonal from the visibility graph to determine the sharpness of the subtended sub-polygon $P(i,j)$. Only those sub-polygons are considered for sharpness computation that satisfy conditions 3.1 through 3.4. The area of the polygon or sub-polygon is computed by using the following expression. $A(P) = \frac{1}{2}\sum_{i=1}^{n-1}(x_i + x_{i+1})(x_{i+1} - y_i)$ [16]

The perimeter of a sub-polygon is determined in a straightforward manner by adding the lengths of the edges bounding the sub-polygon. The values of the area and perimeter are used to determine the sharpness value α.

3.3 Dominated and Prime Candidate Diagonals

Definition 3.2. A diagonal whose sub-polygon satisfies sharpness conditions is called a *feasible diagonal*.

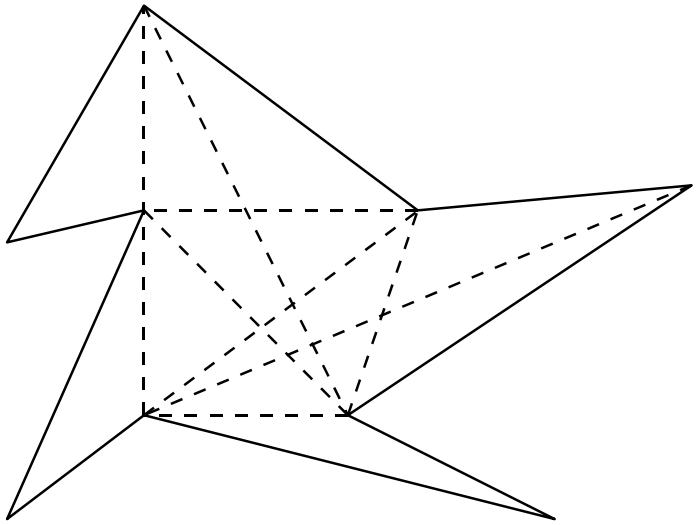


Figure 3.3: Visibility graph in a polygon.

Definition 3.3. A feasible diagonal inside a sharp feature is called a *dominated diagonal*.

Definition 3.4. A *prime diagonal* is a feasible diagonal that is not inside any sharp feature.

The notion of feasible and prime diagonals are illustrated in Figure 3.4 and Figure 3.5, respectively. Two prime diagonals are retained in Figure 3.5 when all non-prime diagonals are removed from Figure 3.4.

To recognize and extract all sharp features of a polygon, areas $A(i, j)$'s and perimeters $R(i, j)$'s must be calculated for all sub-polygons. A brute force approach would be to compute these quantities separately for all diagonals. Time needed to compute area $A(i, j)$ for one pair is $O(n)$. There can be $O(n^2)$ diagonals and consequently the total time for computing all sub-areas in this way is $O(n^3)$.

A faster algorithm makes use of one sub-area to compute another related sub-area. This approach is based on angularly sweeping the sub-areas corresponding to diagonals originating from a vertex. Let the list of diagonals in the counterclockwise

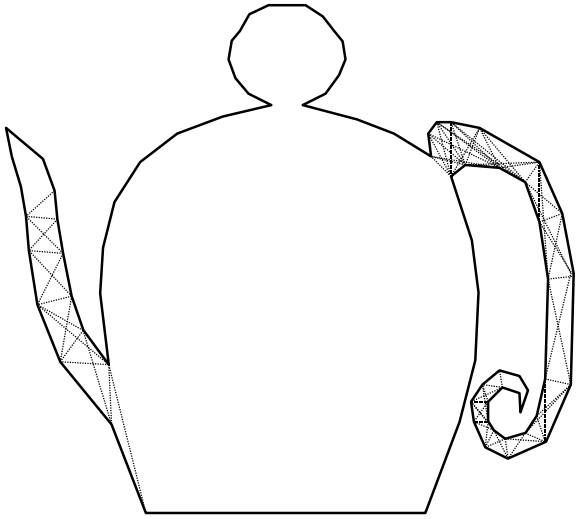


Figure 3.4: Tea pot polygon with feasible diagonals.

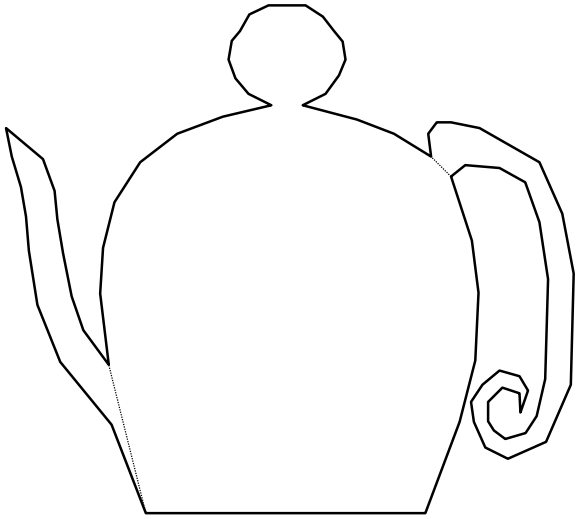


Figure 3.5: Prime diagonals after dominated diagonals are removed.

angular order originating from vertex v_i be $d_{i_1}, d_{i_2}, \ldots, d_{i_m}$. Suppose $A(i, i_{k-1})$ has been computed. Then $A(i, i_k)$ can be expressed as

$$A(i, i_k) = A(i, i_{k-1}) + A(v_i, v_{i_k}, v_{i_k+1}, \ldots, v_{i_{k+1}}) \tag{3.1}$$

This is illustrated in Figure 3.6.

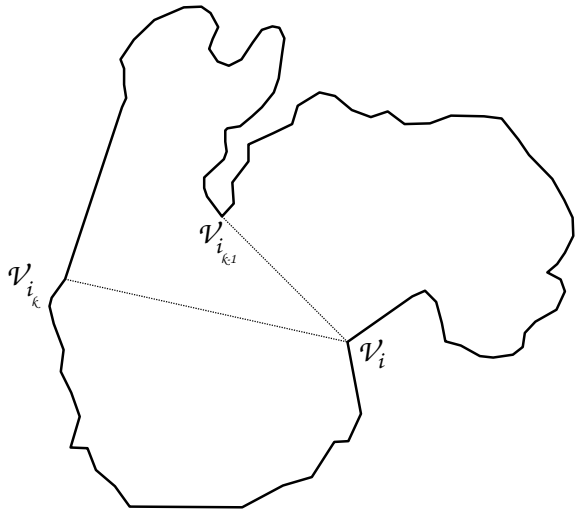


Figure 3.6: Relating $A(i, i_{k-1})$ and $A(i, i_k)$.

It is thus not necessary to recompute the area $A(i, i_k)$ when computing $A(i, i_{k+1})$. Sub-areas corresponding to the diagonals can be processed in the angular order d_i, $d_{i_2}, \ldots, d_{i_k}$, to obtain corresponding sub-areas and also sub-perimeters in $O(n)$ time. Consequently, all sub-areas and all sub-perimeters can be computed in $O(n^2)$ time. A formal listing of the algorithm is given as Algorithm 3.1 (AngularSweep).

Lemma 3.1. *Algorithm 3.1 (AngularSweep) can be executed in $O(n^2)$ time.*

Proof. Computing the visibility graph of a polygon (line 1) can be done in $O(n^2)$ time [16]. It is noted that the angularly ordered diagonals originating from a vertex can be extracted from the visibility graph in $O(n)$ time and hence the time needed for

Algorithm 3.1: AngularSweep

```
1: Compute visibility graph VG of polygon P;
2: foreach vertex v_i of P do
3:     Let d_{i_1}, d_{i_2}, ..., d_{i_k} be the counterclockwise ordered list of diagonals
       emanating from vertex v_i;
4:     A(i, j_0) = 0;
5:     R(i, j_0) = 0;
6:     for j = 1 to k do
7:         A(i, i_j) = A(i, i_{j-1}) + Ar(v_i, v_{i_k-1}, v_{i_k-1+1}, ..., v_{i_k})
           R(i, i_j) = R(i, i_{j-1}) + Pr(v_i, v_{i_k-1}, v_{i_k-1+1}, ..., v_{i_k})
8:     end for
9: end foreach
```

Algorithm 3.2: SharpFeatureRecognition

Input: A simple polygon P with vertices $v_0, v_1, \ldots, v_{n-1}$
Output: Set of prime diagonals

```
1: Compute area A(P) of polygon P;
2: Perform AngularSweep;
3: // Determine feasible diagonals
4: Mark all diagonals unfeasible;
5: foreach diagonal d_ij of P do
6:     if d_ij satisfies all conditions then mark d_ij feasible
7: end foreach
8: // Determine prime diagonals
9: Let d_yk be the starting prime diagonal;
10: while all vertices are not processed do
11:     foreach feasible diagonal d within indexRange(y, k) do
12:         mark d dominated;
13:     end foreach
14:     d_yk = next prime diagonal;
15: end while
```

one execution of line 3 is $O(n)$. The for-loop in line 6 executes in $O(n)$ time. Hence the total time for all steps adds-up to $O(n^2)$. □

The set of feasible diagonals that separate a sharp feature are illustrated in Figure 3.4. By examining the indices of the vertices of a feasible diagonal d_k it can determine whether or not d_k is dominated. Let I_{k_1} and I_{k_2} be the indices of the end vertices of a feasible diagonal d_k. Let J_{r_1} and J_{r_2} be the indices of the end vertices of another feasible diagonal d_r. Then d_k is dominated by d_r if I_{k_1} and I_{k_2} are within the range of J_{r_1} and J_{r_2}. When checking range it is necessary to take index addition modulo n. In this way all dominated diagonals are marked. The set of unmarked feasible diagonals are the prime diagonals. The sub-polygons subtended by prime diagonals give the sharp features. A formal sketch of the algorithm is listed as Algorithm 3.2: SharpFeatureRecognition.

It is interesting to note that the set of prime diagonals found by Algorithm 3.2 do not necessarily partition the polygon. That is, sharp features can overlap. This is illustrated in Figure 3.7.

Theorem 3.2. *All sharp features of a polygon can be recognized in $O(n^2)$ time.*

Proof. We proceed to perform an analysis of Algorithm 3.2 line by line. The area of a polygon can be computed in $O(n)$ time by using the well known formula and hence line 1 takes $O(n)$ time. By Lemma 3.1, line 2 can be done in $O(n^2)$ time. There can be $O(n^2)$ diagonals in the worst case and hence the marking task in line 4 can take $O(n^2)$ time. Since all sub-areas and sub-polygons have been pre-computed, each of the four conditions for sharpness satisfaction can be verified in $O(n)$ time. Thus the first for-loop can be done in $O(n^2)$ time. The starting prime diagonal can be obtained in $O(n^2)$ time by scanning the diagonals along the boundary and by using the pre-computed sub-areas and sub-perimeters. Whether or not an index i lies between the

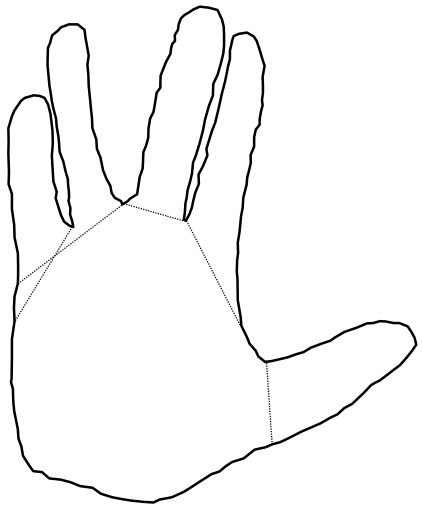


Figure 3.7: Sharp features overlap when prime diagonals intersect.

other two indices y and k can be done in constant time, hence the while loop can be done in $O(n^2)$ time. The total time complexity adds up to $O(n^2)$. □

CHAPTER 4

IMPLEMENTATION

The algorithms proposed in this thesis were implemented to observe the quality of generated results. The actual programs were implemented in the Java programming language. The user can enter the polygon by clicking the points on a graphic canvas. The points are used to draw the polygon. The polygon can then be edited by adding, deleting, and moving vertices. Files with polygon coordinates can be included on the command line when invoking the program.

4.1 User Interface

The main window consists of a drawing canvas to displayed the polygons, a console tree listing the polygons on the canvas, and buttons to open auxiliary windows. The layout is shown in Figure 4.1 with an example shown in Figure 4.2.

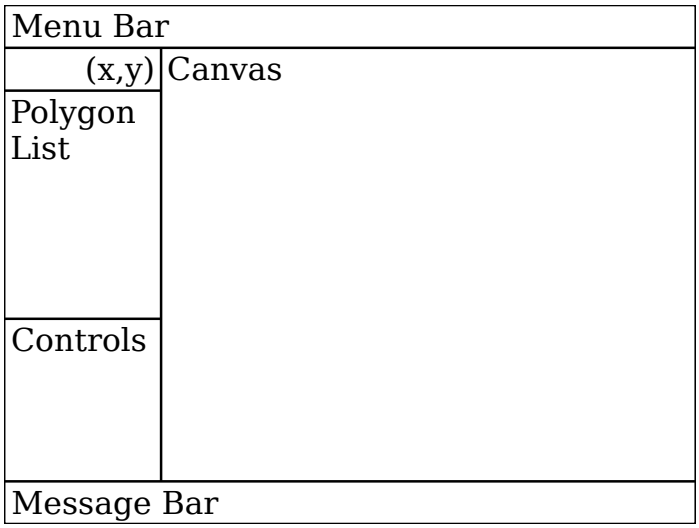


Figure 4.1: Main Window layout.

Only one polygon is "active" at any time. The active polygon is selected by clicking on its name in the console tree. The active polygon is drawn on the canvas in black, all other polygons are drawn in red.

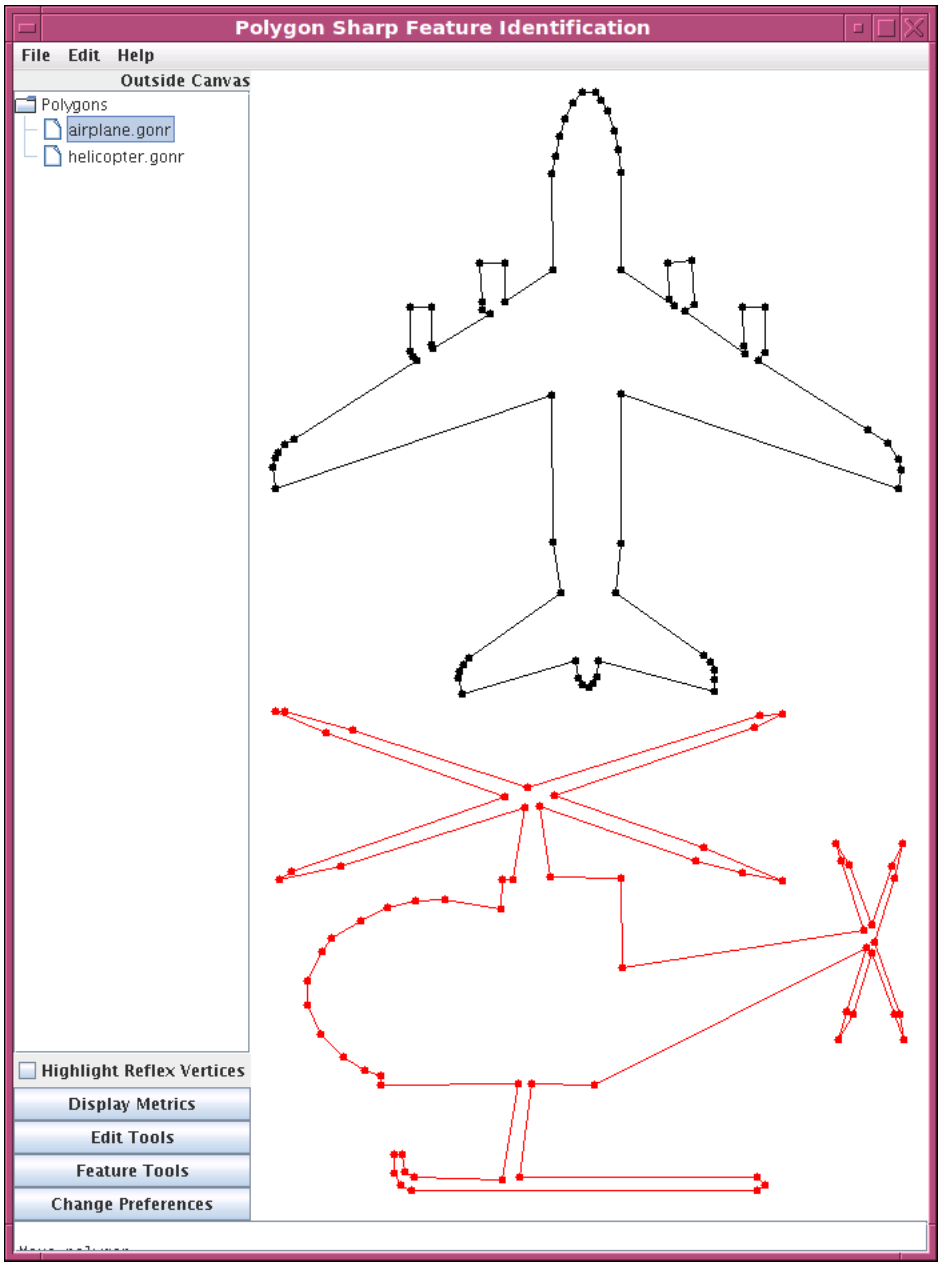


Figure 4.2: Example Main Window with two polygons.

Four auxiliary windows can be displayed for each polygon.

The *Metrics Window*, in addition to the polygon name, displays the number of vertices, area, and total perimeter length of the polygon. An example is given in Figure 4.3(a).

The user interface includes the ability to edit a polygon. These features are controlled in the *Edit Window*, shown in Figure 4.3(b). The Edit Window features a list of the vertices on the perimeter of the polygon and a set of buttons to control the edit mode. These buttons are latched, that is, the mode remains in effect until a different button is pressed.

With the *Draw Polygon* button pressed, each click on the canvas adds that point to the perimeter of the polygon. Vertices are added to the end of the list.

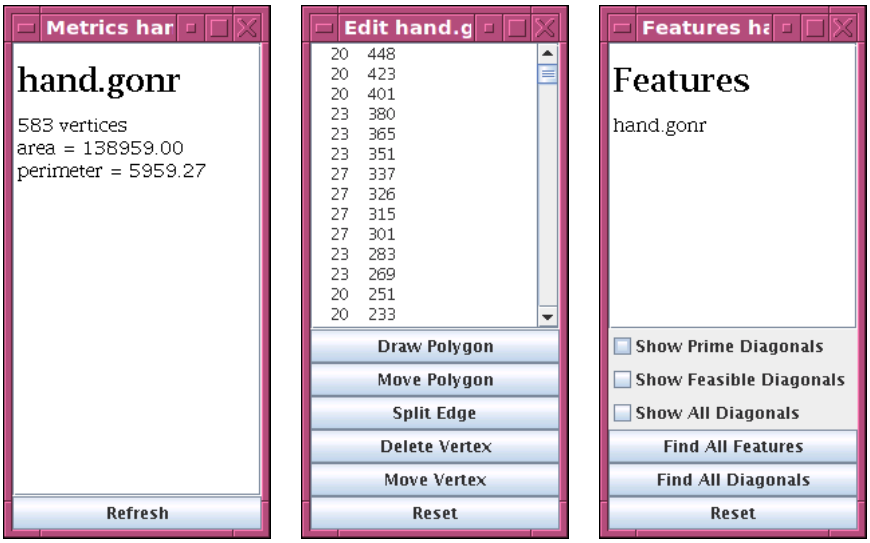
Polygons are dragged across the canvas when the *Move Polygon* button is pressed. The mouse can be dragged along any part of the canvas. Only the bounding box of the polygon is displayed while it is being dragged.

The *Split Edge* button also adds vertices to the polygon's perimeter with each mouse click on the canvas. In this case, each vertex is added after the nearest vertex to the mouse pointer.

With the *Delete Vertex* button pressed, a vertex is removed from the perimeter each time the mouse is clicked on the canvas. The vertex nearest to the mouse pointer is removed. Note that the mouse click can occur anywhere on the canvas for a point to be deleted. The nearest vertex need not be "close" to the mouse pointer.

Dragging the mouse along the canvas with the *Move Vertex* button pressed, moves the closest vertex to the mouse pointer. The mouse pointer need not be precisely on the vertex when the mouse button is pressed but the new value of the vertex is the point where the mouse button was released.

The behavior of the *Reset* button is a little different from the rest. All vertices are deleted from the polygon and mouse clicks on the canvas will add new vertices to



(a) (b) (c)

Figure 4.3: Auxiliary windows for each polygon.

the polygon. The *Draw Polygon* button is shown pressed in the edit window.

Testing of Algorithm 3.2 is done in the *Features Window* show in Figure 4.3(c). Check boxes are provided to control the display of prime diagonals, feasible diagonals, and all diagonals. The algorithm is executed using the *Find All Features* button. Buttons for *Find All diagonals* and *Reset* are also provided.

In addition to the auxiliary windows provided for each polygon, the *Parameters Windows* controls some parameters used by the algorithm. *Diag Max Fraction* limits the length of any diagonal used to separate a feature from the rest of the polygon. The diagonal length is expressed as a fraction of the polygon's perimeter. *Peri Max Fraction* sets the largest fraction of the polygon's perimeter. *Diag Max Growth* sets a factor that can not be exceeded when searching "outward" for the next diagonal.

Changing any of these parameters and pressing the *Set* button causes the algorithm to use the new values. It also deletes all diagonals, including prime and feasible diagonals, from all polygons. The *clear* button deletes any changes and restores the current values. *Get me out of here!* will close the window without

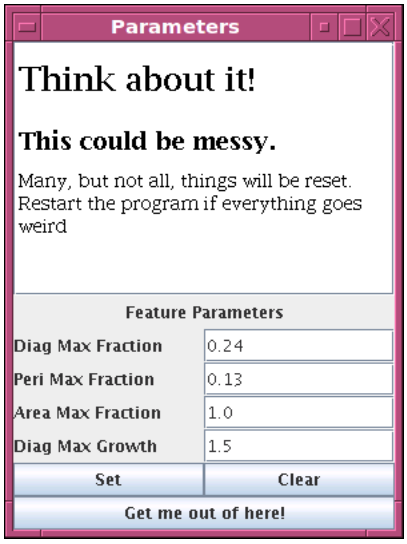


Figure 4.4: Parameters window.

changing any values.

The window also provides a message that encourages the user to exercise caution when changing values. An example of the Parameters window is shown in Figure 4.4.

4.2 Experimental Results

A variety of polygons were constructed to test the performance of the proposed algorithm. Since the central problem in the investigation is the identification of sharp features, the polygonal shapes contain long and narrow structures.

The polygon shown in Figure 4.5 resembles the outline of an ant. When the algorithm limits the perimeter of a sharp feature to 10% of the total perimeter, it is able to clearly identify the ant's antennae and front legs. Part of the rear legs, however, are not identified. The figure shows this clearly and displays both prime and feasible diagonals.

Figure 4.6 shows the prime diagonals when the algorithm only limits feature perimeters to 15% of the total perimeter. In this case, the rear legs are defined well

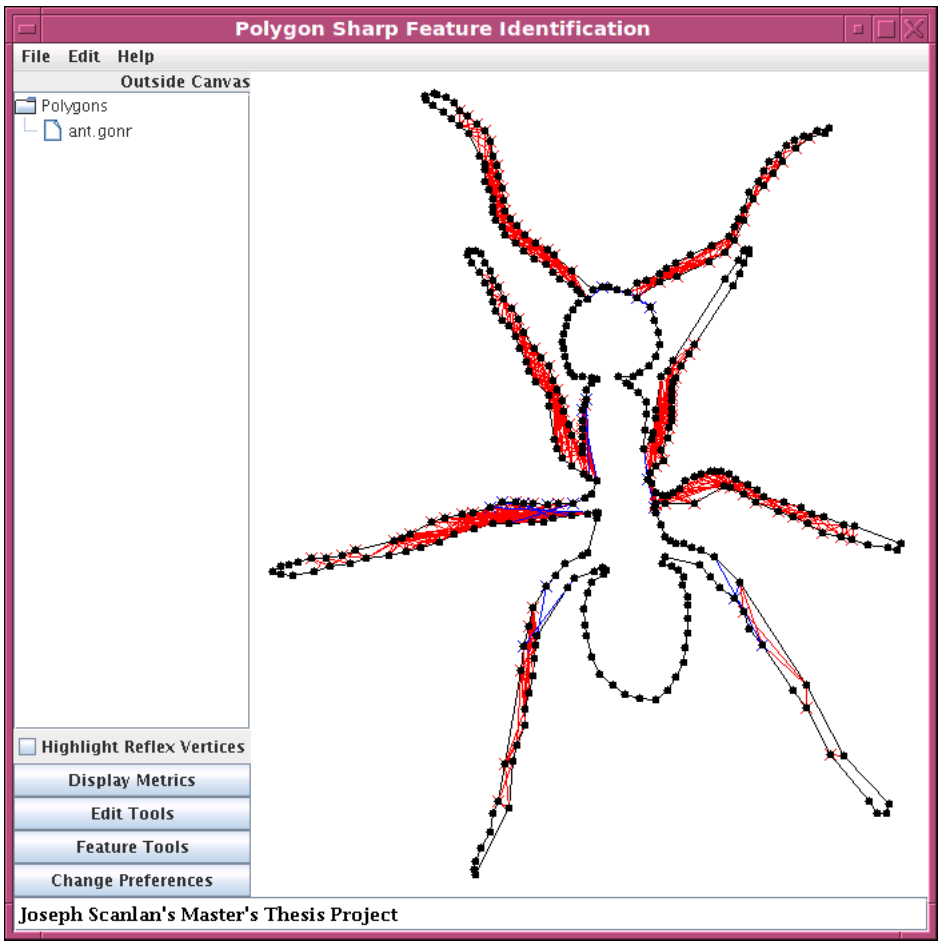


Figure 4.5: Ant Polygon with 10% limit.

enough but part of the head was identified as parts of each antenna.

The algorithm can clearly identify the tail when the polygon resembles a turtle, such as the one shown in Figure 4.7, even when a feature's perimeter can be as large as 15% of the total perimeter. Notice the head and legs of the turtle are not sharp enough to be identified.

The 15% limit also works well with the Kiwi Bird polygon show in Figure 4.8. One of the kiwi bird's legs clearly shows that prime diagonals do not partition the polygon. The foot can be identified by any one of three prime diagonals.

Some polygons are very sensitive to the perimeter limit imposed on the algorithm. The spiky polygon shown in Figure 4.9 looks like it has sharp features when

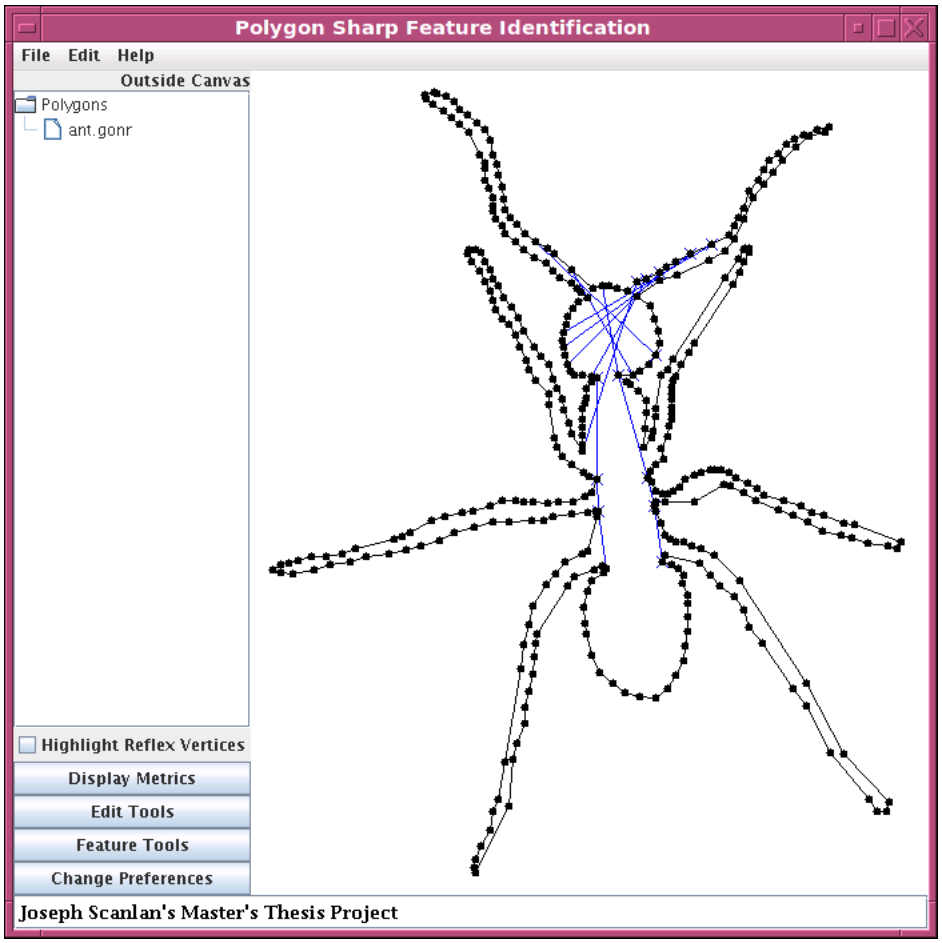


Figure 4.6: Ant Polygon with 15% limit.

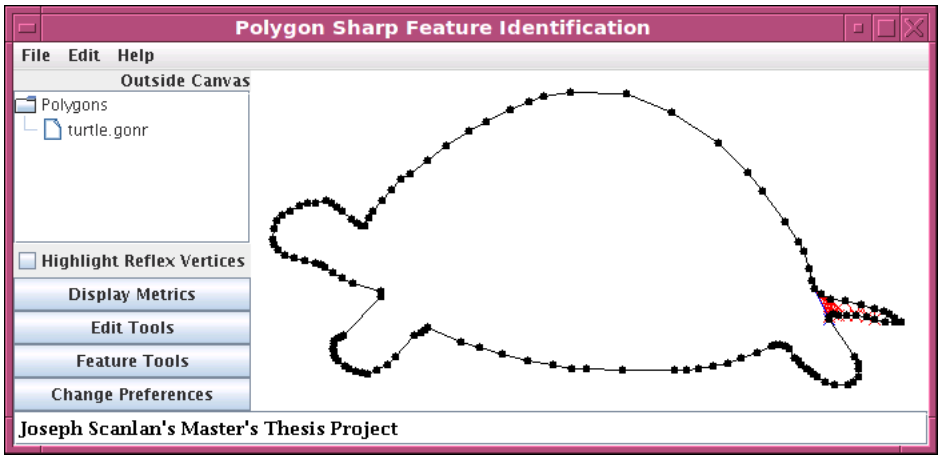


Figure 4.7: Turtle Polygon with 15% limit.

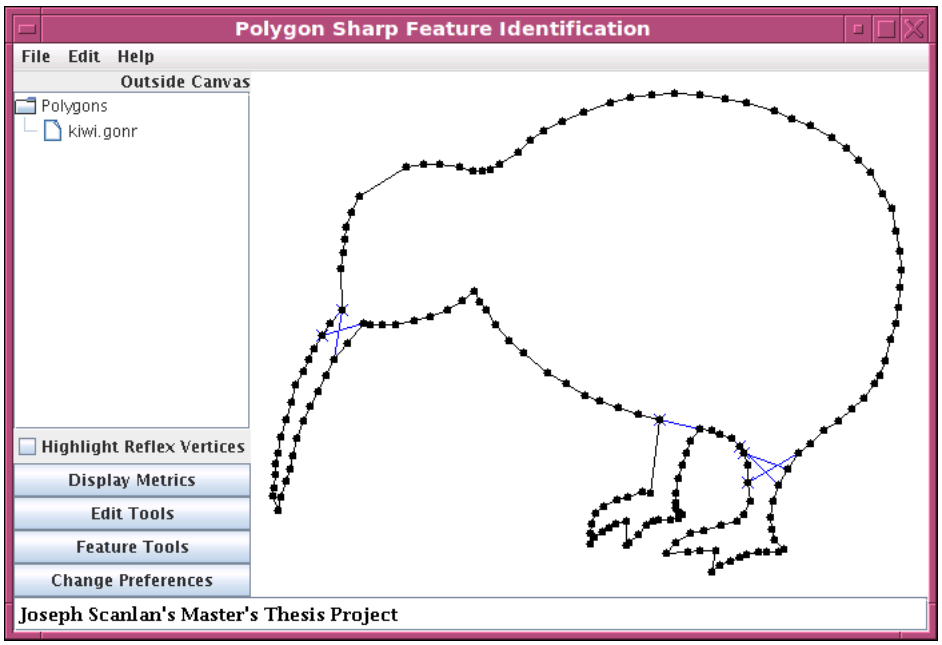


Figure 4.8: Kiwi Bird Polygon with 15% limit.

a 13% limit was used, but the results were unacceptable when 12% and 14% limits were attempted.

4.3 Application Programming Interface

The complete API document is supplied with the source code and is available at `http://www.n7xsd.us/scanlan2011sharp/`.

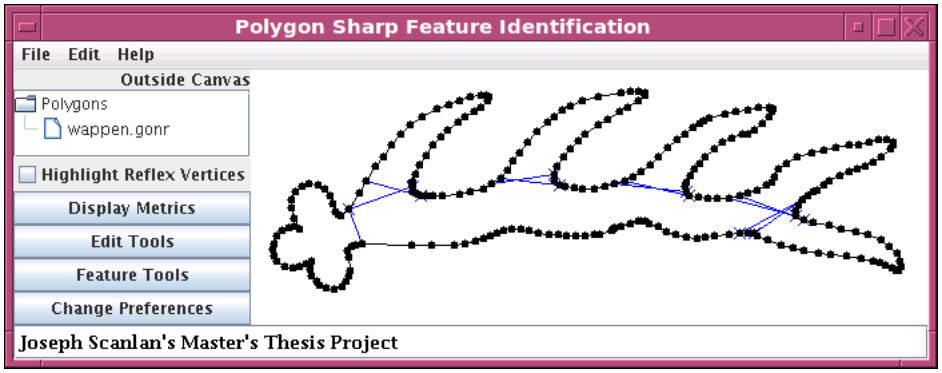


Figure 4.9: Spiky polygon with 13% limit.

CHAPTER 5

CONCLUSION

This thesis presented a survey of polygon boundary simplification algorithms. Three approaches were examined in detail: boundary vertex retention, boundary vertex elimination, and boundary replacement. David H. Douglas and Thomas K. Peucker presented their boundary vertex retention algorithm in 1973 [4]. Their original iterative algorithm and a recursive variant were presented here. Other vertex retention approaches were also touched upon. Arie Pikaz and Its'hak Dinstein's approach to boundary vertex elimination [17] was demonstrated. This approach uses a min-heap and has a time complexity of $O(k \log n)$. Hiroshi Imai and Masao Iri's algorithm replaces a monotone chain with a new chain [11] without selecting vertices from the original chain. This thesis also touched upon work by Lilian Buzer from her paper "Optimal simplification of polygonal chains for subpixel-accurate rendering" [3].

The notion of polygonal sharp features was introduced. A working definition of a sharp feature was given and the characteristics of sharp features examined. A formula was derived to quantify the sharpness of a polygon and various polygons compared to find a working threshold value to determine when a polygon can be considered sharp. An algorithm, utilizing the visibility graph of a polygon, was proposed to recognize sharp features along with the basis of its development. The time complexity of this algorithm is $O(n^2)$. The concepts of candidate diagonal, feasible diagonal, and prime diagonal were used to find the diagonals that separate sharp features from the rest of the polygon. It was also noted that prime diagonals do not partition a polygon.

A program framework used to test the algorithm was demonstrated. This framework allowed the input of polygons from simple text files as well as by drawing polygons on a canvas. The canvas supports display of multiple polygons. Polygons can also be edited after they are entered.

The framework was then used to test several polygons. Adjustments were made to the sharpness threshold and fraction of the polygon perimeter that could be incorporated into the sharp feature. No change to the sharpness threshold was needed for the tested polygons. It was found that adjustment to the fraction of perimeter the sharp feature could use was needed to produce results that were pleasing to the eye.

An interesting idea for the future would be to change the definition of sharp feature. This thesis' definition only allows for one diagonal to separate the sharp feature from the remainder of the polygon. Allowing two or more diagonals to separate the feature would show narrow structures in a polygon that connect broad shaped structures. Sharp feature recognition also has the potential for use in polygon similarity comparison, perhaps as a preprocessing step.

BIBLIOGRAPHY

[1] Esther M. Arkin, L. Paul Chew, Daniel P. Huttenlocher, Klara Kedem, and Joseph S. B. Mitchell. An efficiently computable metric for comparing polygonal shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(3):209–216, 1991.

[2] Prosenjit Bose, Sergio Cabello, Otfried Cheong, Joachim Gudmundsson, Marc van Kreveld, and Bettina Speckmann. Area-preserving approximations of polygonal paths. *Journal of Discrete Algorithms*, 4(4):554–566, 2006.

[3] Lilian Buzer. Optimal simplification of polygonal chains for subpixel-accurate rendering. *Computational Geometry*, 42(1):45–59, 2009.

[4] David H. Douglas and Thomas K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*, 10(2):112–122, December 1973.

[5] A. Garrido, N. P. Blanca, and M. Garcia-Silvente. Boundary simplification using a multiscale dominant-point detection algorithm. *Pattern Recognition*, 31(6):791–804, 1998.

[6] T. Gerstner. Multiresolution visualization and compression of global topographic data. *GeoInformatica*, 7(1):7–32, 2003.

[7] Laxmi Gewali and Joseph P. Scanlan. Recognizing Sharp Features of 2-D Shapes. *International Journal of Electronics and Telecommunications*, 56(2):153–156, 2010.

[8] Eric Guilbert and Hui Lin. B-spline curve smoothing under position constraints for line generalisation. In *GIS '06: Proceedings of the 14th Annual ACM International Symposium on Advances in Geographic Information Systems*, pages 3–10. ACM, 2006.

[9] Denise Guliato, Juliano D. de Carvalho, Rangaraj M. Rangayyan, and Sérgio A. Santiago. Feature extraction from a signature based on the turning angle function for the classification of breast tumors. *Journal of Digital Imaging*, 21(2):129–144, 2008.

[10] Martin Held and Johannes Eibl. Biarc approximation of polygons within asymmetric tolerance bands. *Computer-Aided Design*, 37(4):357–371, 2005.

[11] Hiroshi Imai and Masao Iri. Polygonal approximations of a curve – formulations and algorithms. In Godfried T. Toussaint, editor, *Computational Morphology*, pages 87–95. Elsevier Science Publishers B. V. (North-Holland), Amsterdam, Netherlands, 1988.

[12] Y. Kurozumi and W. A. Davis. Polygonal approximation by the minimax method. *Computer Vision, Graphics, and Image Processing*, 19(3):248–264, July 1982.

[13] D. J. Lee, Sameer Antani, and L. Rodney Long. Similarity measurement using polygon curve representation and fourier descriptors for shape-based vertebral image retrieval. In M. Sonka & J. M. Fitzpatrick, editor, *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 5032 of *Presented at the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference*, pages 1283–1291, May 2003.

[14] Adam Lewis, Suzanne Slegers, Dave Lowe, Leath Muller, leanne Fernandes, and Jon Day. Use of spatial analysis and gis techniques to re-zone the great barrier reef marine park. *Coastal GIS*, pages 431–451, 2003.

[15] Joseph O'Rourke. The signature of a plane curve. *SIAM Journal on Computing*, 15(1):34–51, 1986.

[16] Joseph O'Rourke. *Computational Geometry in C*. Cambridge University Press, Cambridge, UK, second edition, 1998.

[17] Arie Pikaz and Its'hak Dinstein. An algorithm for polygonal approximation based on iterative point elimination. *Pattern Recognition Letters*, 16(6):557–563, June 1995.

[18] Waldo R. Tobler. Automation and cartography. *Geographical Review*, 49(4):526–534, 1959.

[19] Shen Wei. Building boundary extraction based on lidar point clouds data. *Proceedings of the International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 37:157–161, 2008.

VITA

Graduate College
University of Nevada, Las Vegas

Joseph P. Scanlan

Degrees:
Bachelors of Science, 1983
University of Nevada, Las Vegas

Thesis Title: Sharp Feature Identification in a Polygon

Thesis Examination Committee:
Chairperson, Laxmi P. Gewali, Ph. D.
Committee Member, Evangelos A. Yfantis, Ph. D.
Committee Member, Jan B. Pedersen, Ph. D.
Graduate Faculty Representative, Henry Selvaraj, Ph. D.