8-2011

Improved algorithms for ear-clipping triangulation

Bartosz Kajak
University of Nevada, Las Vegas

IMPROVED ALGORITHMS FOR EAR-CLIPPING TRIANGULATION

by

Bartosz Kajak

Bachelor of Science
Lublin University of Technology, Poland
2005

A thesis submitted in partial fulfillment
of the requirements for the

Master of Science in Electrical Engineering

Department of Electrical and Computer Engineering
Howard R. Hughes College of Engineering
The Graduate College

University of Nevada, Las Vegas
Aug 2011

THE GRADUATE COLLEGE

We recommend the thesis prepared under our supervision by

Bartosz Kajak

entitled

Improved Algorithms for Ear-Clipping Triangulation

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

Master of Science in Electrical Engineering
Department of Electrical Engineering

December 2011

ABSTRACT

Improved Algorithms for Ear Clipping Triangulation

by

Bartosz Kajak

Dr. Henry Selvaraj, Examination Committee Chair
Professor of Electrical and Computer Engineering
University of Nevada, Las Vegas

Dr. Laxmi Gewali, Associate Examination Committee Chair
Professor of Electrical and Computer Engineering
University of Nevada, Las Vegas

We consider the problem of improving *ear-slicing* algorithm for triangulating a simple polygon. We propose two variations of *ear-slicing* technique for generating "good-quality" triangulation. The first approach is based on searching for the best triangle along the boundary. The second approach considers polygon partitioning on a pre-process before applying the *ear-slicing*. Experimental investigation reveals that both approaches yield better quality triangulation than the standard *ear-slicing* method.

ACKNOWLEDGMENTS

Nearly three years ago, I submitted my admission application to Howard R. Hughes College of Engineering at University of Nevada, Las Vegas. Little did I know about significance and impact this decision would have on my life. My Master's program studies were filled with moments of happiness and grief, major challenges and triumphs. Not only did I learn the art of scientific research, but I also had some of the biggest life lessons that defined my character today. I learned how to overcome obstacles along the way and never give up. Now I know that through our biggest challenges we get greatest triumphs. That difficult and very productive time filled with pursuit for research, sow a seed of relentless hunger for knowledge and in result will undeniably bring me more fruits of success in the future.

I would like to express my utmost gratitude to dr Henry Selvaraj my mentor, advisor and master advisory committee chair. Thanks to dr Selvaraj I become a part of Electrical and Computer engineering department at UNLV. I greatly appreciate warm, friendly atmosphere at school and the fact that I always felt welcome. Dr Selvaraj helped me with my program and thesis work as well as with my private life challenges. It is impossible to describe significance and positive impact of dr Selvaraj on my life and my thankfulness to him in couple sentences. I cannot imagine better advisor. Thank you, dr Selvaraj for all your help, it will never be forgotten.

I am extremely happy to take this opportunity to acknowledge my debt and gratitude to dr Laxmi Gewali, associate chair of my graduate committee. Dr Gewali demonstrated his incredible patience and was extremely generous with his time. Words only will fail to express my gratitude for his commitment to me and my thesis. Dr Gewali taught and guided me through research, which brought the fruits of my first conference publication and this thesis. His words of wisdom are extremely significant to me and will be referenced in the future.

I could never forget to thank dr Dawid Zydek - great friend, advisor and my graduate committee member. Dawid’s thoughts, experiences and observations were many times groundbreaking for me. Dawid was always willing to share his opinion in decision making process, he helped me during my studies in professional and private life. Dawid always believed in me even in the most challenging of moments. There is no words to describe my gratitude for his help.

I am extending my sincere thanks to dr Pramen Shrestha – graduate committee member for his contribution throughout this investigation.

I would like to thank to my Las Vegas friends Juan Santana and Lina Chaparro for all of their support, for being there and for not giving up on me even in the hardest of times. Juan and Lina were always patient listeners and real friends. They radiate positive, warm energy and their help will never be forgotten.

Special thanks to my friends Mariana Rocha Rodrigues, Natarajan Pillai, Ewelina Camacho, Kasia and Rafał Kowalczuk for valuable and unforgettable conversations and support.

At last I would like to thank my mother Ela, father Andrzej, sister Kasia and her husband Leszek for persistent support and concern. They have been with me throughout my life regardless of situation. They make sure I know I am never alone.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

CHAPTER 1

INTRODUCTION

Triangulation of a simple polygon is a partitioning of its interior into triangles such that the vertices of triangles are also the vertices of the polygon. It has been established that any simple polygon can be triangulated. It can be easily verified that a triangulated polygon of *n* vertices contains exactly *n-3* diagonal and *n-2* triangles. A polygon can be triangulated in exponentially many ways. The problem of developing efficient algorithms for triangulating a simple polygon has attracted the interest of several researchers from computational geometry [1]. One of the first polygon triangulation algorithms found in standard text books is based on repeatedly slicing a triangle. This approach is often called triangulation by "ear-slicing". A straightforward implementation of ear-slicing algorithm takes $O(n^2)$ time. From the beginning of 1980, there was a flurry of research interest for developing a linear time algorithm for triangulating a simple polygon. The fastest algorithm known for the next ten years (1980-1989) had time complexity $O(n\log^* n)$. For all practical purposes this time complexity is linear. From the theoretical point of view, there was still room for improvement. Finally, Bernard Chazelle [4] reported a linear time algorithm for triangulating a simple polygon. Some investigators have commented that Chazelle's linear time algorithm is very difficult for practical implementation. Finding a simple linear time algorithm that can be implemented easily is still an open problem.

In this thesis an overview of different methods for triangulating a polygon are presented. It is shown that some algorithms can yield mesh with large number of thin-triangles which are not desired for application in finite element analysis. Quality issue for triangulation is considered. A triangulation with large proportions of "fat" triangles is said to be of high quality. Modifications of the standard ear-cutting algorithm for generating quality triangular mesh are presented. Additionally, presented method is improved by introducing polygon decomposition. Experiment results of the proposed algorithms are presented and additional approaches for further improving the quality of generated triangles are discussed.

CHAPTER 2

POLYGON TRIANGULATION

The triangulation of a simple polygon is the partitioning of its interior triangles such that the vertices of triangles are also vertices of the polygon. It has been established that any simple polygon can be triangulated [1]. It can be easily verified that a triangulated polygon of *n* vertices contains exactly *n-3* diagonal and *n-2* triangles. A polygon can be triangulated in exponentially many ways. The problem of developing efficient algorithms for triangulating a simple polygon has attracted the interest of several researchers from computational geometry [1-7]. One of the first polygon triangulation algorithms found in standard text book is based on repeatedly slicing a triangle [1,7]. This approach is often called triangulation by "ear-cutting ". A straightforward implementation of ear-cutting algorithm takes $O(n^2)$ time. From the beginning of 1980 there was a flurry of research interest for developing a linear time algorithm for triangulating a simple polygon. The fastest algorithm known for the next ten years (1980-1989) had time complexity $O(n\log^* n)$[1,7]. For all practical purposes this time complexity is linear. From theoretical point of view, there was still room for improvement. Finally Bernard Chazelle [4] reported a linear time algorithm for triangulating a simple polygon. Some investigators have commented that Chazelle's linear time algorithm is very difficult for practical implementation [1]. Finding a simple linear time algorithm that can be implemented easily is still an open problem.

2.1 Ear-Slicing algorithm

Ear-slicing is one of the well-known techniques for triangulating a simple polygon [1, 2, 3, 6]. Due to its intuitive appeal, ear-slicing triangulation is usually considered as one of the fist simplest triangulation algorithms. We first present a brief review of the standard ear-slicing algorithm. Let the vertices of the polygon that appear in the counterclockwise traversal of its boundary be denoted by v_0, v_1, v_2, ... , v_{n-1}. Three consecutive vertices v_{i-1}, v_i ,v_{i+1} form an ear of the polygon if the line segment L_i = [v_{i-1}, v_{i+1}] connecting vertices $v_{i-1}$and v_{i+1}lies completely inside the polygon. Figure 1 illustrates the definition of ear. In the figure, vertex sequence <v_4, v_5, v_6> form ear because line segment L_5 = [v_4, v_6] lies completely inside the polygon. Similarly, vertex sequence <v_8, v_9, v_{10}> form another ear. On the other hand, vertex sequence <v_1, v_2, v_3> does not form an ear because the line segment joining v_1 to v_3 does not lie completely inside the polygon.

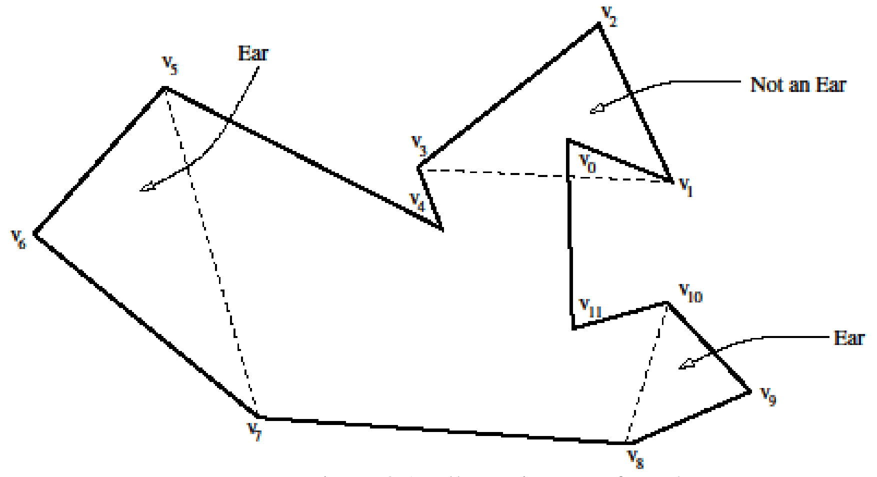


Figure 2.1 : Illustrating ear of a polygon

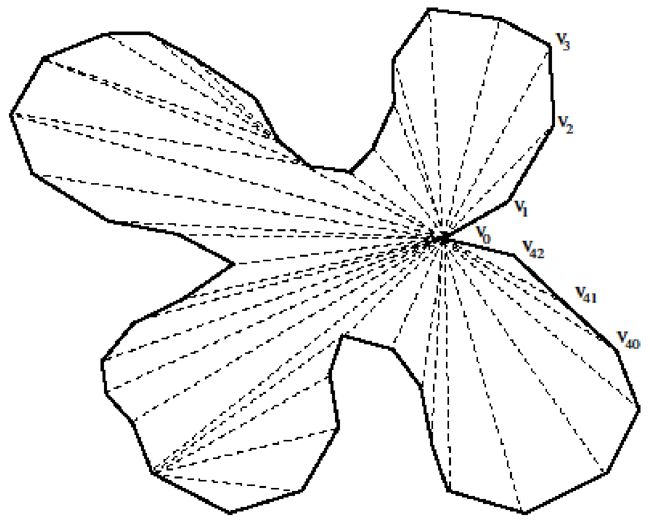


Figure 2.2 Triangulation by traditional ear-cutting method

It is known that any polygon with number of vertices greater than 3 has at least two ears [1]. If we have a simple polygon P with large number of vertices then the residual shape P' obtained by slicing off an ear from P is also a simple polygon. This observation reveals that any simple polygon with at least three vertices can be triangulated by slicing an ear repeatedly. Ear-slicing stops when the residual polygon is a triangle. This algorithm can be formally sketched as follows.

```
Algorithm 1: Triangulation by Standard Ear-Slicing
    Input: A simple polygon with $n$ vertices $v_0, v_1, v_2, \ldots, v_{n-1}$ stored in a list $L$.
    Output: A set of $n-3$ diagonals that triangulate the polygon
    Step 1: Let $D$ be the empty diagonal list.

    Step 2: while ($L$ has more than 3 vertices) do

    Step 3:        (a.) Locate an ear $v_{i-1}, v_i, v_{i+1}$.
                   (b.) Add diagonal ($v_{i-1}, v_{i+1}$) to $D$.

    Step 4:        Remove $v_i$ from $L$.
        endwhile

    Step 5: Output diagonals in $D$ as triangulating diagonals
```

To determine whether a candidate segment $d_i = (v_{i-1}, v_{i+1})$ is a diagonal or not, the algorithm checks the intersection of d_i with all the edges of the polygon. If this candidate segment does not intersect with any edge of the polygon then it is a valid diagonal and inserted into D. This straightforward check for Step 3, takes $O(n)$ time. Since this check is repeated $O(n)$ time the total time for Algorithm 1 is $O(n^2)$. Detailed analysis and implementation issue of triangulation by ear-slicing is available in reference [1]. A more careful analysis of ear-slicing algorithm has been investigated by ElGingy, Everett, and Toussaint [2]. The algorithm reported in [2] is simpler to implement but it still need $O(n)$time per ear-slice. Figure 2.2, shows a triangulation obtained by using the standard ear-slicing algorithm. An inspection of the triangles in the triangulation of Figure 2.2 reveals that there are several thin and skinny triangles. It is thus an interesting problem to modify ear-slicing techniques so that the resulting triangulation has reduced number of skinny triangles.

2.2 Toussaint's strip triangulation

An efficient method of triangulating a simple polygon was developed by Godfried Toussaint in 1988. His adaptive algorithm runs in $O(n(1+t_0))$ where $t_0 < n$ is the resulting number of triangles that share no edges with the processed polygon. Therefore t_0 depends on shape complexity of input polygon. Due to its low complexity the algorithm has found immediate application in computer graphics. The algorithm requires no sorting or usage of complicated data structures. The approach is to partition complex polygon into set of smaller polygons called **sleeves**. Note that a polygon is called a sleeve if it can be triangulated so that the triangulation dual is a chain. The polygon in Figure 3a admits triangulation whose dual is a chain and hence it is a sleeve. On the other hand polygon in

Figure 3b is a non-sleeve because all triangulation duals are not chains. In the first step algorithm finds a diagonal and perform triangulation in both directions assuming the polygon is a sleeve. If the polygon indeed happens to be a sleeve the algorithm terminates successfully. On the other hand if the polygon is not a sleeve, the algorithm partition polygon into components by inserting appropriate diagonals and proceeds to triangulate the components separately. In the worst case, the time complexity $O(n(1+t_0))$ could be $O(n^2)$ for polygon where number of triangles that do not share polygon edges is $O(n)$. But for polygon with simpler shape complexity the value of t_0 is small and usually constant. In some case the algorithm runs in linear time.

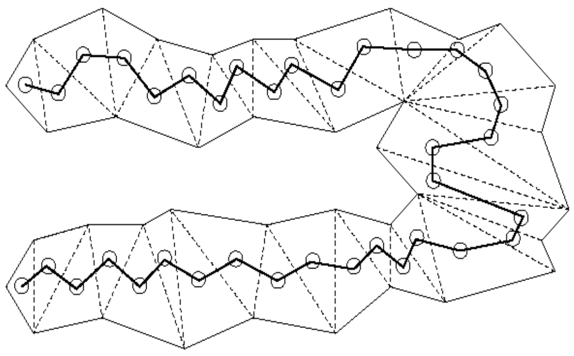


Figure 2.3a: A sleeve polygon

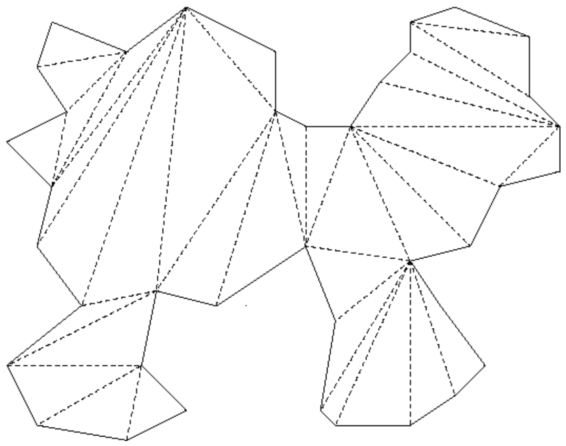


Figure 2.3b: A non-sleeve polygon

2.3 Decomposition into monotone polygons

Improved triangulation algorithm that executes in $O(nlogn)$ time was first introduced by Garey, Johnson, Preparata & Tarjan in 1978. It first partitions the polygon into simpler pieces separately in efficient manner. The simpler pieces are called monotone polygons. A polygon P is monotone in y-direction if any horizontal line intersects P in exactly one line segment or empty. The step for partitioning into monotone components is done in *O(nlogn)* time. This is achieved by constructing diagonals from "cusp" vertices as shown in Figure 4. It is remarked that a horizontal line segment s1 can be drawn in a cusp vertex so that the polygon edge at the cusp vertex are either both above or below s1. In figure 5 the polygon is partitioned into six monotone pieces. Triangulating monotone polygon can be achieved in linear time as briefly described next.

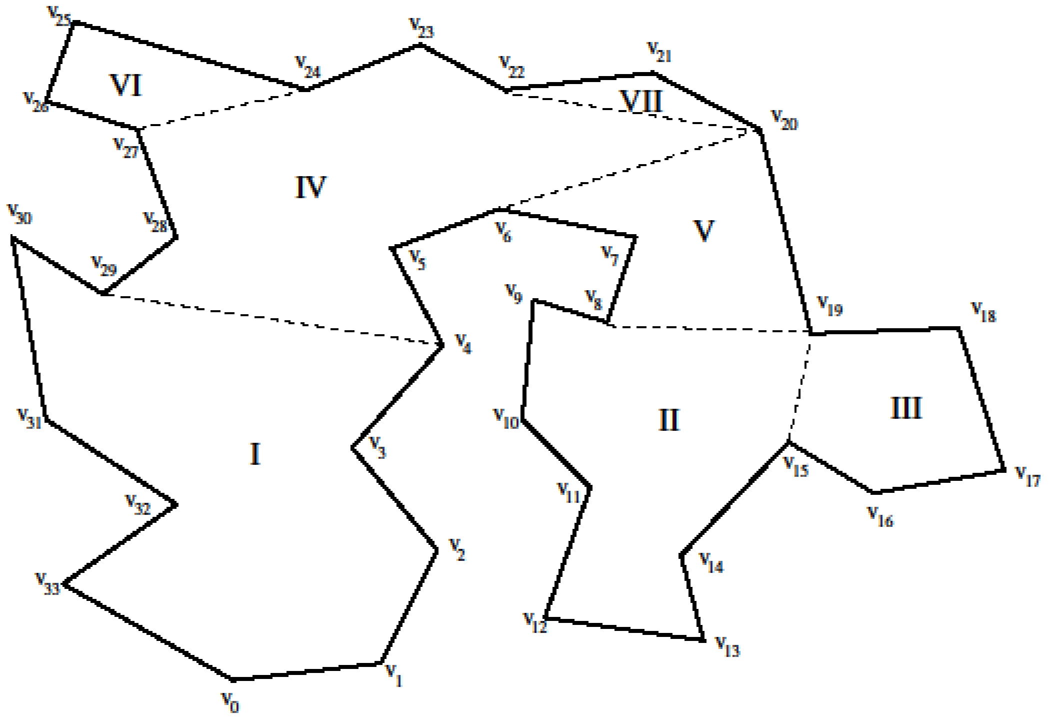


Figure 2.4 Polygon partitioned into monotone pieces – I, II, … VII.

2.4 Triangulating a monotone polygon

Given polygon is called *monotone* with respect to line L if it can be split into two polygonal chains, where each chain is monotone with respect to L. Note that a chain *Ch1* is monotone with respect to a line L if the intersection of any line parallel to L with Ch1 is either empty or one point. In Figure 6 a monotone polygon with respect to y axis is shown and the two monotone chains are $<v_0, v_1, \ldots, v_{14}>$ *and* $< v_{14}, v_{15}, \ldots, v_{26}, v_0>$. It is easily observed that the two monotone chains *left-chain* and *right-chain,* as described above are such that their vertices are already sorted by y-coordinate. This ordering property of monotone chains can be used to develop an efficient algorithm. The algorithm obtains the sorted list of the vertices of the polygon by merging the *left-chain* and *right-chain.* Since merging of two sorted list can be done in linear time the sorted list of vertices (sorted by y-coordinate) can be done within the same time. The top-most and the bottom-most vertices can also be determined in linear time by simply scanning the boundary. After having the sorted list of vertices the polygon can be triangulated in *greedy* manner by walking top to down and by using stack. The details can be found in [1, 15, 16].

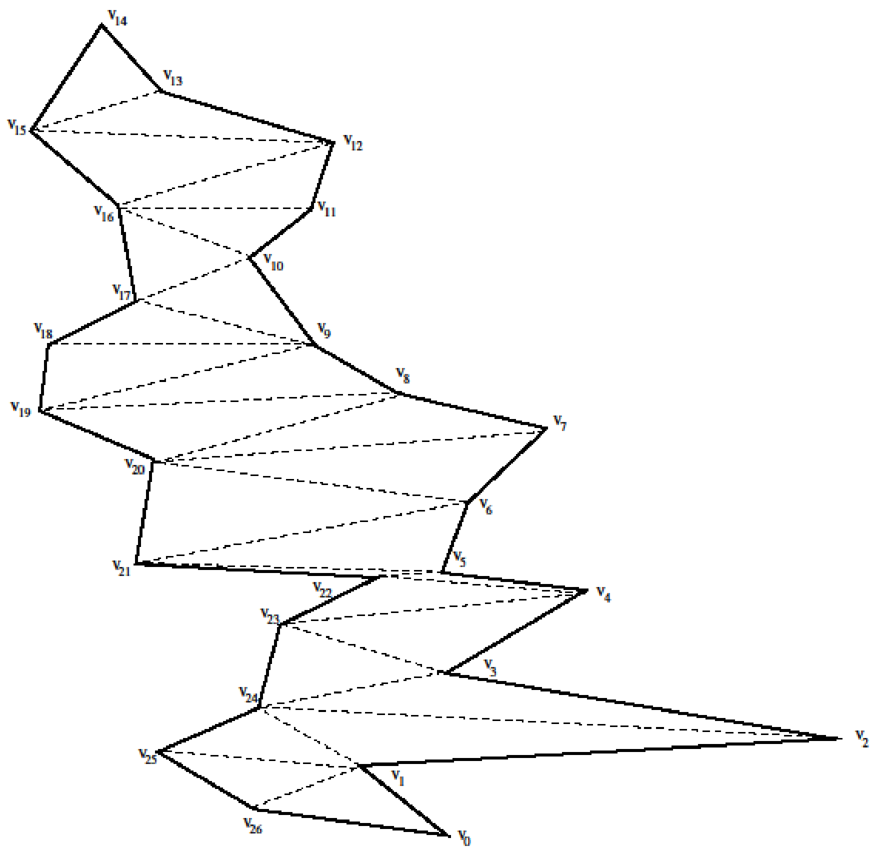


Figure 2.5 Triangulated Monotone Polygon.

2.5 Converting triangulations to quadrangulations

For certain problems in finite element analysis and scattered data interpolation decomposing a polygon into quadrangle (quadrilateral) elements is more beneficial than a triangular decomposition. Unfortunately algorithms for high quality quadrangle meshes are not as well developed as algorithms for triangular meshes. It is known that a polygon may not admit a qaudrangulation if we restrict diagonals to be inserted only between existing vertices (where Steiner points are not permitted). Additionally it was proven that qaudrangulation without adding Steiner points can be done only if number of vertices of a figure is even. Unlike quadrangular, computing of triangular meshes is well known and developed for years, due to that fact scientist took an insight into converting triangulations to quadrangulations. A triangular mesh generated on a simple polygon can be converted into quadrangular in O(n) time. The restriction is that obtained quadrangles have to be *strict* quadrangles – no three vertices can be collinear (that would make them triangles). O(n) time can be achieved by inserting Steiner points on all of the edges and diagonals of a triangulated polygon. Then extra Steiner points are inserted in the interior of each of the triangles and connected to the other 3 points on the diagonal and the edges (Figure 2.6a and 2.6b). Connected Steiner points yield quadrangular mesh. Such algorithm is very simple to implement and run in linear time. Disadvantage of that solution is the fact that large number of Steiner points is generated, while the goal is to keep that amount low. For simple *n-gon* such approach always uses *3n-5* Steiner points.

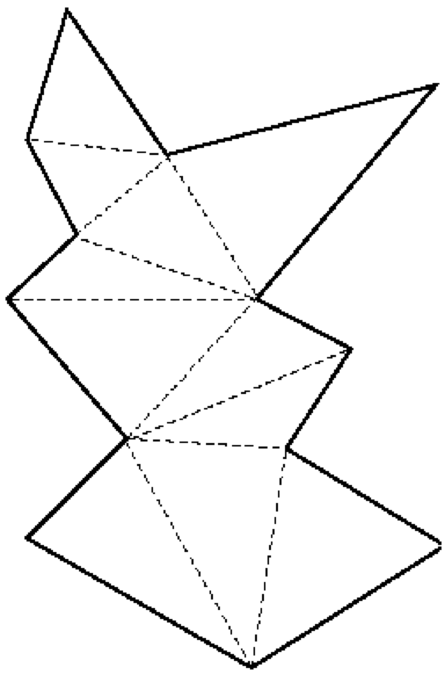


Figure 2.6a: Triangulated simple polygon

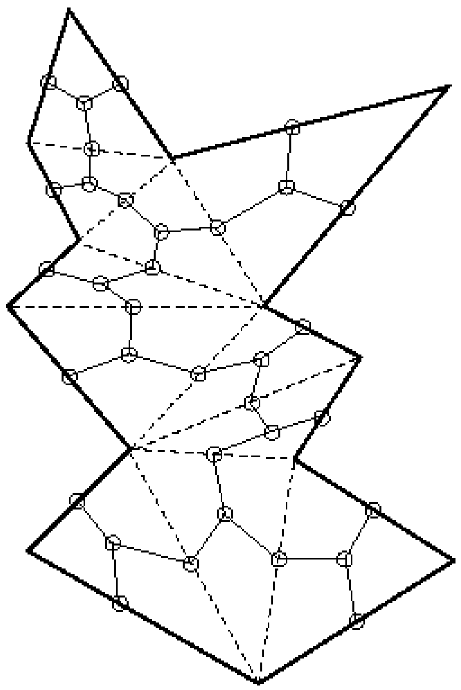


Figure 2.6b: Quadrangulation obtained by inserting 3*n-5* Steiner points

A slightly more complicated algorithm was developed that decreases number of Steiner points. Algorithm’s first step is to obtain a Hamiltonian-cycle triangulation by Arkin’s Algorithm [17]. A planar dual tree is inserted into triangulated polygon. Once the tree is constructed each triangle’s interior node of a dual tree is connected with three vertices of that triangle. Now diagonals of an original polygon triangulation are erased and Hamiltonian triangulation is obtained. Next, we need to visit polygon’s triangles in Hamiltonian order. We can do that by performing a tree traversal of the geometrical dual tree – Hamiltonian cycle. By visiting each triangle and erasing every other diagonal polygon quadrangulation is achieved (Figure 2.8a, 2.8b, 2.8c, 2.8d). One outside Steiner point may be required to quadrangulate last remaining triangle. Algorithm performs in O(n) time and always generates n-2 Steiner points. Further improvements of converting triangular meshes to quadrangulations were proposed. In [18] presented method require at most *n/3* outer Steiner points or at most *n/4* inner Steiner points and at most one outside the polygon.

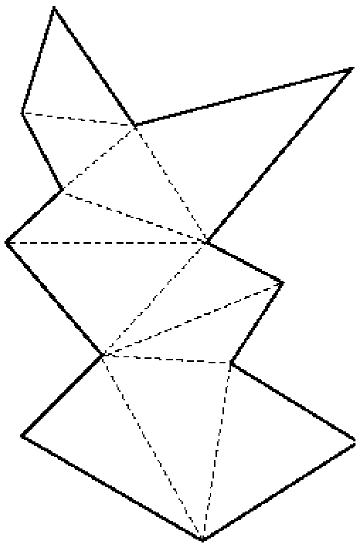


Figure 2.7a: Triangulated simple polygon

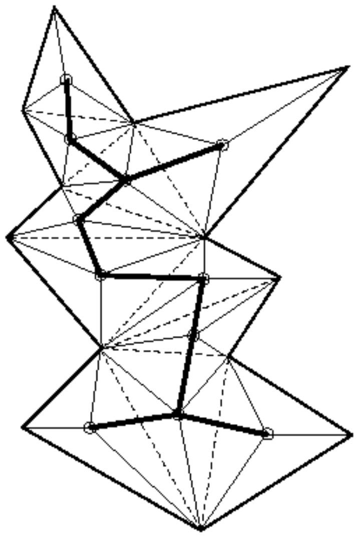


Figure 2.7b: Dual tree inserted into triangulated polygon with each triangle's interior node connected to vertices of corresponding triangle

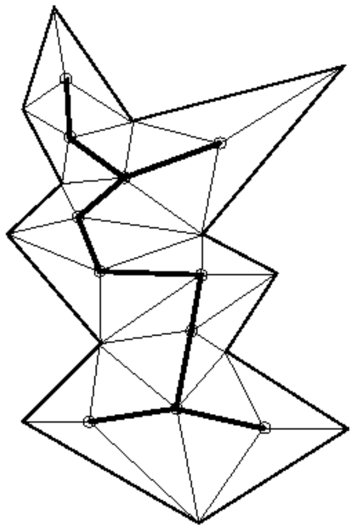


Figure 2.7c: Polygon with triangulating diagonals removed

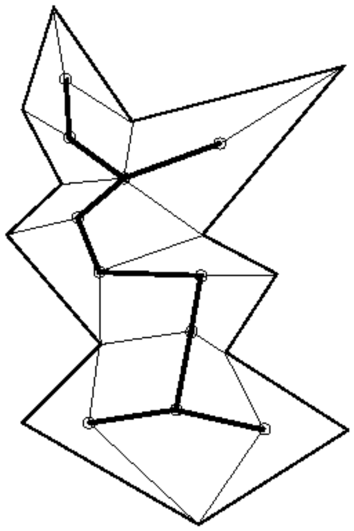


Figure 2.7d: Quadrangulated polygon with *n-2* Steiner points inserted (one single triangle remaining)

CHAPTER 3

IMPROVED TRIANGULATION

This chapter presents the main contributions of the thesis. We propose two approaches for improving performance of *ear-slicing* techniques for generating quality triangulation. The first approach is based on searching for best diagonal to slice an ear. The second approach relies on partitioning polygon after examining the output generated from regular triangulation.

3.1 Notion of quality triangulation

We first consider the quality measure of a triangle. One of the applications of triangular mesh is in finite element analysis [1,7], where a complicated domain need to be partitioned into union of simple shapes such triangles, quadrilaterals, hexagons, etc. For computing fluid flow and heat transfer in a given domain, it is necessary to solve partial differential equations on triangles and quadrilaterals rather than the whole domain. The quality of solution obtained by using finite elements (triangles, quadrilaterals) method depends on the shape of the elements. The finite elements that are not skinny yield better approximation for the generated solution. For such applications, it is beneficial to generate triangular mesh with larger proportion of quality triangles.

One way to measure the quality of a triangle is by finding its smallest enclosing bounding rectangle. An easy way of finding such a rectangle is to construct the smallest iso-thetic (axis parallel) rectangle. It is very easy to construct smallest enclosing iso-thetic rectangle. We just have to select appropriate x- and y-coordinates from the coordinates of the vertices of the triangle. An example of smallest iso-thetic rectangle is shown in Figure3a. It turns out that the smallest enclosing rectangle in not necessarily iso-thetic as shown in Figure 3b. Let l and w ($l >= w$) be the height and width, respectively, of the smallest enclosing rectangle. Then the **aspect ratio**$ar(T)$ of triangle T is defined as the ratio w/l. It is obvious that aspect ratio of any triangle T satisfies the condition $0 < ar(T) <= 1$. A high quality triangle should have large (> 0.5) aspect ratio.

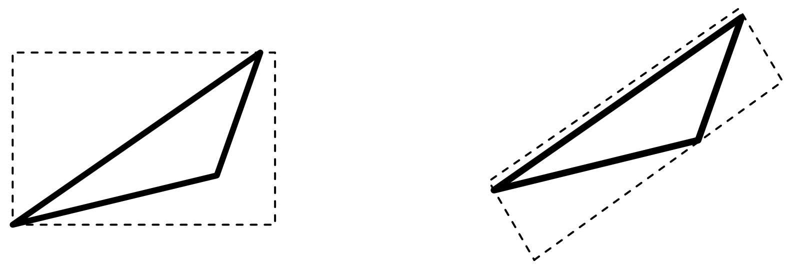
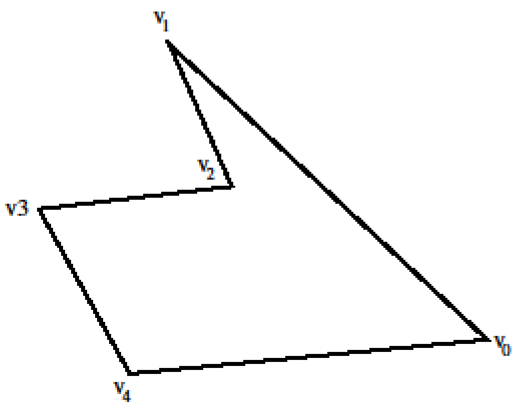

(a): Iso-thetic bounding box (b): General bounding box

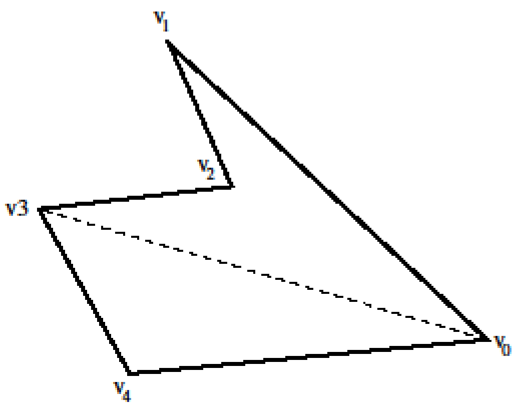
Figure 3.1: Two ways of measuring aspect ratio

3.2 Greedy searching.

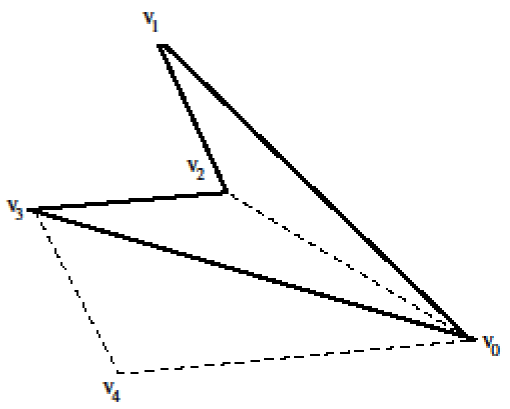
To modify the performance of ear-clipping triangulation it is necessary to examine the aspect ratios of all possible ears by scanning the whole boundary. The algorithm examines each three consecutive vertices $<v_{i-1}, v_i, v_{i+1}>$ one by one along the boundary starting from vertex v_0. It checks if v_{i-1}, v_{i+1} is an internal diagonal or not. If v_{i-1}, v_{i+1} is indeed an diagonal then it computes aspect ratio of the triangle ("ear") $<a_{i-1}, a_i, a_{i+1}>$. The algorithm keeps track of the triangle that maximizes the aspect ratio by constantly updating the desired "search-ear" as the scan proceeds along the boundary. When the first quality triangle identified in greedy-manner is completed, the initial polygon is made smaller by a vertex by *chopping-off* the ear. The process of *ear-slicing* is continued until the residual polygon is a triangle. The running snap-shot of the algorithm is shown in Figure 3.2.



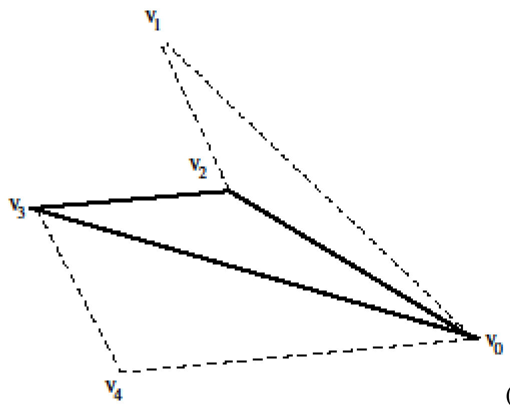
(a): Polygon to be triangulated



(b): First quality triangle found

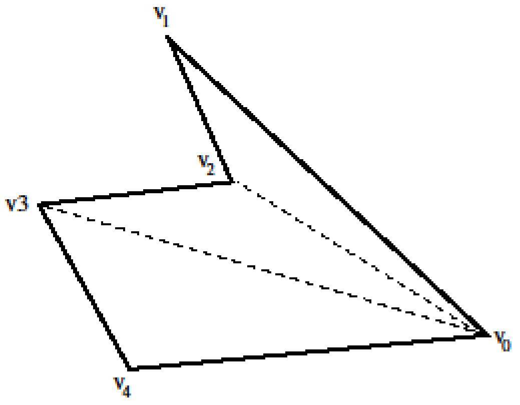


(c): First “ear” chopped and second quality triangle found



(a):

(d): Residual polygon is a triangle



(e): Triangulated polygon

Figure 3.2: Improved *ear-slicing* triangulation

The diagonal corresponding to the ear that maximizes the aspect ratio is taken as the desired diagonal for triangulation. An algorithm based on this search scheme by comparing aspect ratio is sketched as Algorithm 2. A triangulation obtained by applying Algorithm 2 is shown in Figure 3.2, which clearly has larger number of quality triangles than a triangulation obtained by traditional method showed in Figure 2.2.

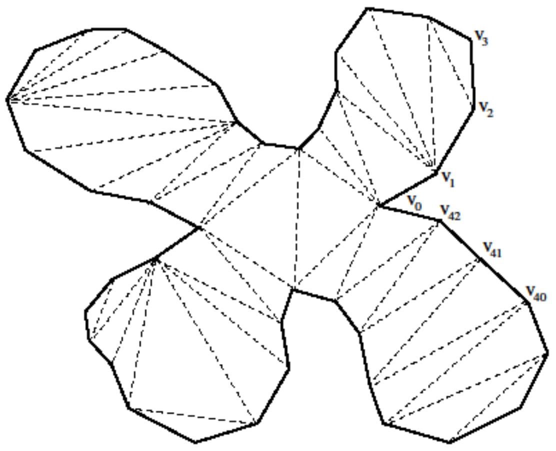


Figure 3.3 Triangulation by improved ear-cutting method

Algorithm 2: **Triangulation by Modified Ear-Slicing**

```
Input:   A simple polygon with $n$ vertices $v_0, v_1, v_2, \ldots, v_{n-1}$ stored in a list $L$.
Output:  A set of $n-3$ diagonals that triangulate the polygon
Step 1:  Let $D$ be the empty diagonal list.

Step 2:  while ($L$ has more than 3 vertices) do
             a.Let$T_i$ be the next ear
             b.$T_{max} = Ti; max_{val} = ar(T_i);$
             c. $T_i = getNextEar();$
             d. while ($T_i$ is not$null$) do
                    if ($ar(T_i)>max_{val}$) then
                           $T_{max} = Ti;$
                           $max_{val} = ar(T_i);$
                    endif
                    $T_i = getNextEar();$
             endwhile
             e. Add the diagonal corresponding to $T_{max}$ to $D$.

Step 3:      Remove middle vertex of$T_{max}$ from$L$.
         endwhile

Step 4:  Output diagonals in $D$ as triangulating diagonals
```

Time complexity analysis of Algorithm 2 is straightforward. The inner while loop need to examine all triangles to determine the best one. Hence one execution of the inner while-loop takes $O(n^2)$ time. The outer while-loop executes $n-3$ times and hence the total time complexity of Algorithm 2 is $O(n^3)$.

Theorem 1: Modified ear-slicing algorithm can be executed in $O(n^3)$ time

3.3 Polygon Partitioning and Ear Slicing.

On closer examination of the triangulation obtained by using the standard *ear-slicing* algorithm we find that there could be regions where many skinny triangles are crowded. This is illustrated in Figure 3.4.

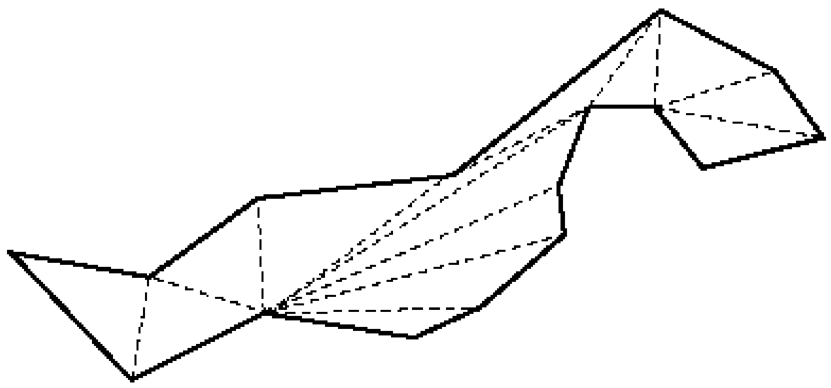


Figure 3.4 Ilustating the crowding of skinny triangles

Definition 3-1: Given a triangulated polygon the internal diagonal that intersects with most number of skinny triangles is called the stabbing diagonal (Figure 3.5).

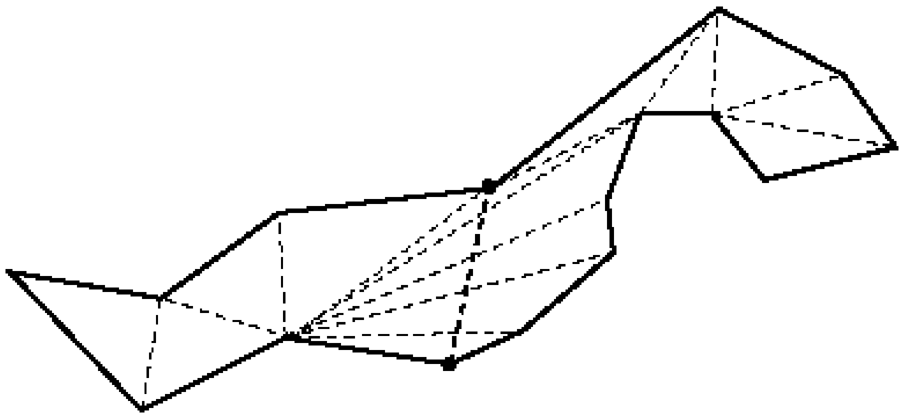


Figure 3.5 Ilustating stabbing diagonal

In order to improve the number of quality triangles our approach is to first partition the polygon into components by using the stabbing diagonals. The critical issue here is come up with technique for identifying appropriate stabbing diagonal efficiently. An obvious way is to try all possible diagonals as possible stabbing candidates. For this purpose we need to use the concept of visibility graph of a polygon investigated in computational geometry [1] which can be described as follows:

Visibility Graph: Given a simple polygon *P* the visibility graph of *P*, denoted as *VG(P)* consist of set of vertices *V* which are exactly the set of vertices of the polygon and the set of edges are the set of internal diagonals of the polygon.

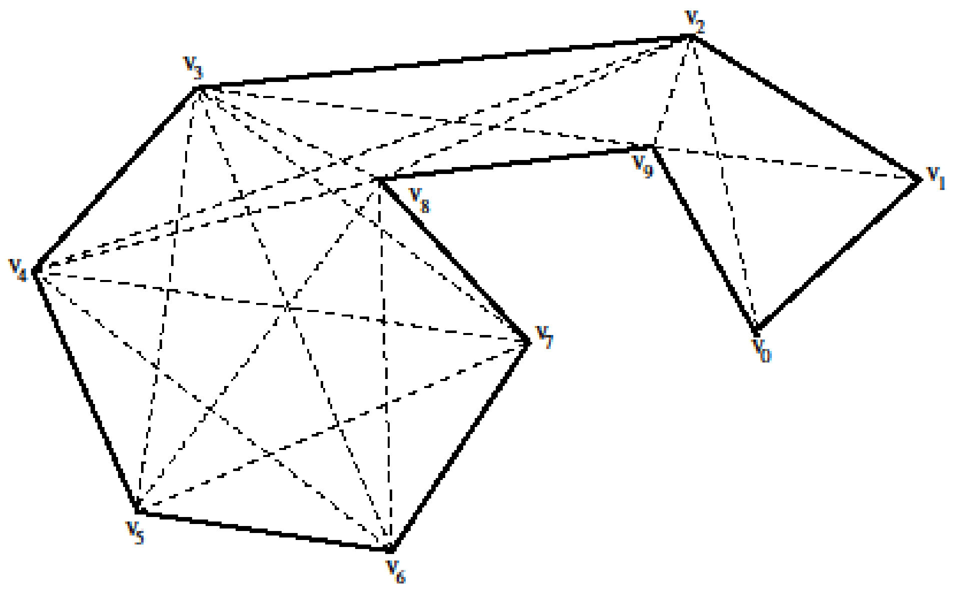


Figure 3.6 Illustrating visibility graph.

To determine the stabbing diagonal we can first compute the visibility graph to get all possible candidate internal diagonals. Each diagonal from the visibility graph is checked for the intersection with the triangles of the triangulation. The number of triangles intersected by a diagonal can be referred to as its stabbing number. The diagonal that maximizes the stabbing number is taken as the stabbing diagonal.

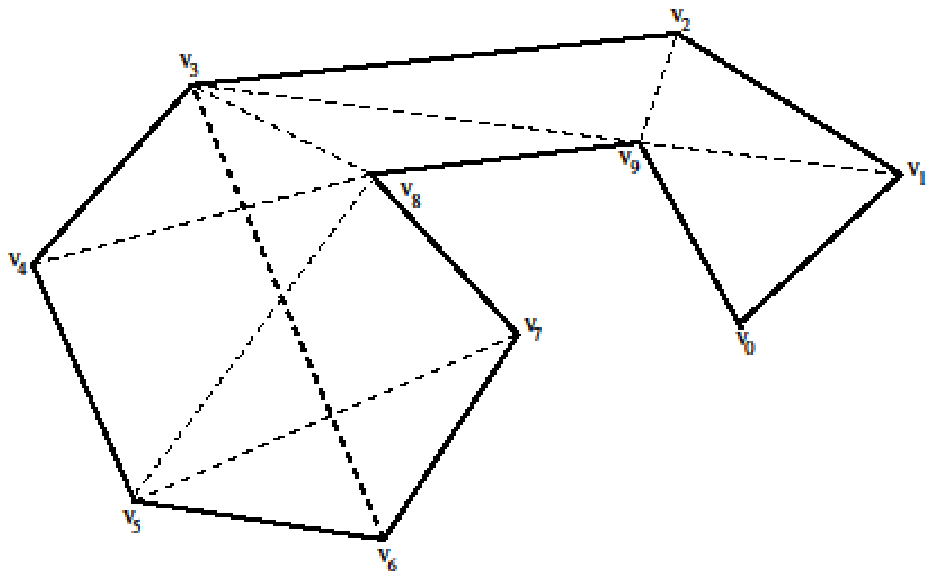


Figure 3.6 Illustrating visibility edge with triangulating edges.

A formal sketch of the algorithm is as follows:

```
Algorithm 2: Partitioning and Ear-Slicing
    Input:   (i) A simple polygon P with $n$ vertices $v_0, v_1, v_2, \dots, v_{n-1}$ stored in a list $L$.
             (ii) Integer m
    Output:  A set of $n-3$ diagonals that triangulate the polygon.
    Step 1:  Compute the visibility graph VG(P) of the given simple polygon.

    Step 2:  Determine the triangulation T(P) by applying improved ear-slicing
             triangulation algorithm

    Step 3:  // Determine stabbing number for diagonals of Visibility Graph.
               For each diagonal edge $e_i$ in VG(P) do
               $sn(e_i)$=Number of diagonals of T(P) intersected by $e_i$.

    Step 4:  Let E' be the list of diagonals of VG(P) sorted by stabbing number (in
             non-incrementing order).

    Step 5:  Let $P_1, P_2 \dots P_m$ be the sub-polygons of P implied by the first m
             diagonal in E'.

    Step 6:  Triangulate $P_1, P_2 \dots P_m$ by applying the improved ear-slicing
             algorithm.

    Step 7:  Output the diagonal of triangulated polygon of $P_1, P_2 \dots P_m$.
```

The time complexity of Algorithm 2 can be analyzed in straightforward manner. Visibility graph can be computed in $O(n^2)$ time [1] and hence step 1 takes $O(n^2)$ Improved ear-slicing (step 2) can be done in $O(n^3)$ time. Stabbing numbers $sn(e_i)$ can be computed by checking each edge of visibility graph again triangulation in $O(n^3)$ time. Sorted list of diagonals (step 4) can be done in $O(n^2 \log n)$ time by sorting all $O(n^2)$ diagonals. Once we have E', step 5 can be obtained in $O(n^2)$ time. Each of the m polygon's component in step 6 can have O(n) sizes and hence step 6 is takes $O(n^3m)$ time. Step 7 takes O(n) time. Since step 6 is the dominating step, the total time of the algorithm is $O(mn^3)$.

CHAPTER 4

IMPLEMENTATION

This chapter describes implementation and study of the *ear-cutting* and improved quality triangulation algorithms. Program was implemented in Java Version 1.6.

Application consists of three algorithms implemented to triangulate the polygons. First algorithm performs standard *ear-cutting* triangulation, second one performs improved quality triangulation and last one allows for manual decomposition of polygons by diagonal *stabbing* and individual triangulation of decomposed pieces.

4.1 Application interface

Implementation is done by permitting user to generate a figure or read any predesigned polygon from a file consisting of *n* vertices. Polygon size and shape can be adjusted by adding and deleting vertices or splitting edges. Once figure is finalized user has a choice to triangulate it using original *ear-cutting* method or execute an improved quality triangulation algorithm. As a result program outputs triangulating diagonals. Slight code modification allows user to manually decompose polygon by inserting stabbing diagonals in the places where large number of triangulating diagonals exist. Decomposed polygon is then triangulated using improved *ear-cutting* method and yields ameliorated results.

4.1.1 Interface description

Figure 4.2 shows an implementation of the main Graphic User Interface. GUI was implemented by extending the JFrame class component in java.swing which consists of four panels. Application layout is presented in Figure 4.1. File menu is contained within

JFrame’s top menu bar and contains two basic items: read and save. All other panels contained within JFrame object are constructed by using JPanel class. Main panel area is divided into four sub-panels: left, right, center and bottom. Center panel contains main display area that allows user to manually draw, edit or display polygons read from a file. Mouse coordinates are provided in the upper left corner to help navigate or draw objects within center area. The right panel is divided into two windows. First one is used to display *x* and *y* vertex coordinates of the polygon. Appropriate coordinates are displayed each time user clicks inside the center panel to draw or modify a polygon by adding a vertex. Second window displays triangles’ quality statistics. Information is classified into 5 groups with respect to triangles’ aspect ratio. Large number of triangles in the first two groups indicates a lot of skinny triangles and low quality of triangulation, accordingly large number of triangles in groups four and five indicate good triangulation quality. First group contains triangle with aspect ratio in range 0:0.2, second group in range 0.2:0.4, third in 0.4:0.6, last two groups contain good quality triangles with aspect ratio in range 0.6:0.8 and 0.8-1 accordingly. Total number of triangles in each of the groups is displayed as a result of successful triangulation.

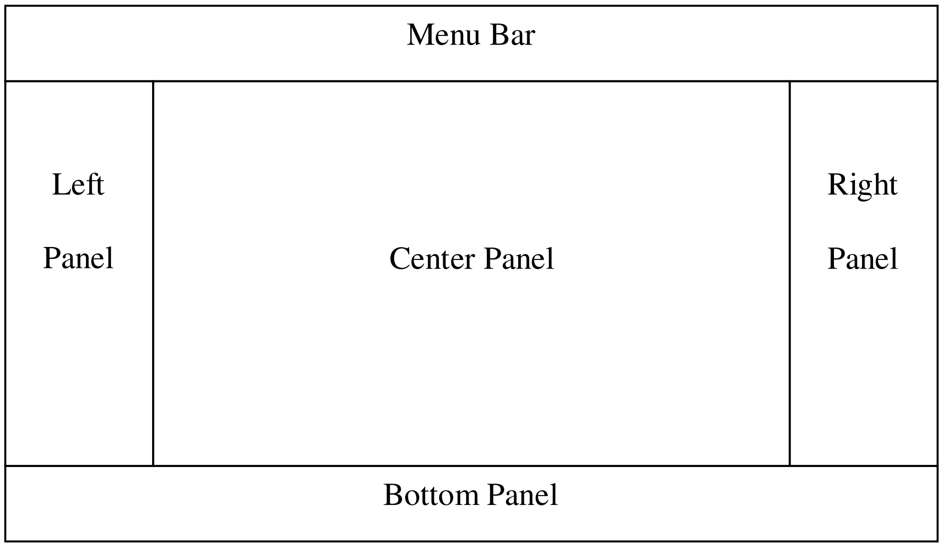
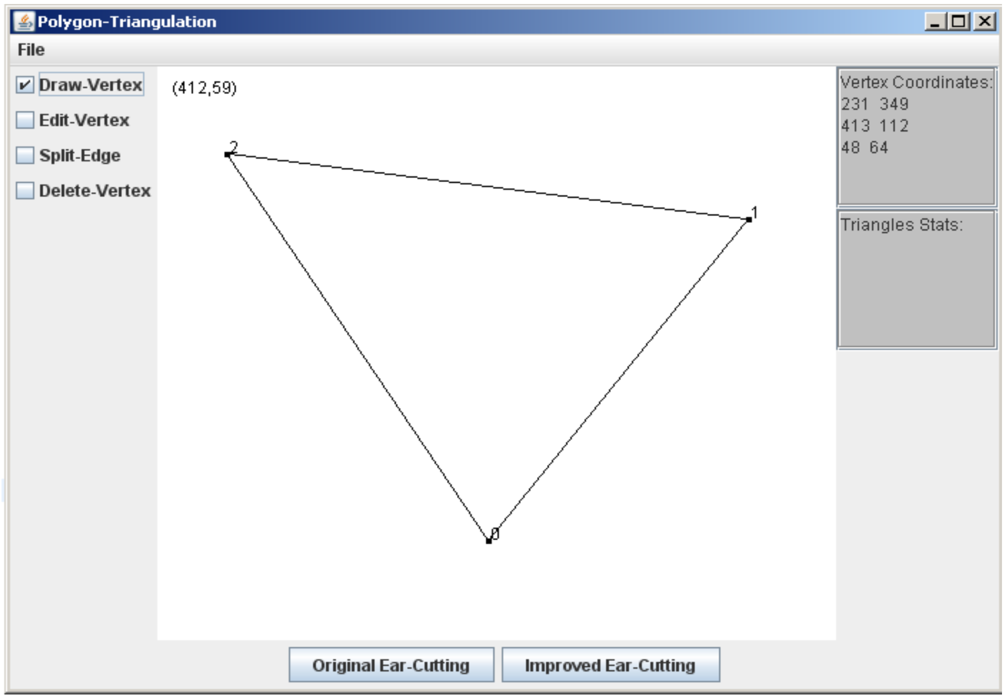


Figure 4.1: GUI Layout



Vertex Coordinates:
231 349
413 112
48 64

Figure 4.2: The Initial Display of GUI for polygon triangulation

The right panel contains four checkboxes used to add and manipulate the edges and vertices of a polygon. Application starts with no vertices or edges displayed, user can initiate drawing figure by selecting *Draw-Vertex* checkbox. Figure 4.2 presents simple three vertices object drawn by user in the center panel. Such triangle can be grown to a bigger polygon by adding consecutive vertices and splitting the edges. User adds vertex simply by clicking inside the main panel area. *Draw-Vertex*, *Edit-Vertex*, *Delete-Vertex* and S*pli-edge* can be done one at a time. Functionality of right panel checkboxes is described in Table 4.1. Finally, the bottom panel contains two buttons used to execute polygon triangulation. First button to the left triangulate polygon using original *ear-cutting* method, second button executes improved triangulation. Polygon can be triangulated only once for each start of the application. Multiple instances of the same application can be used for comparison of results. Additionally source code can be edited to manually decompose polygon by stabbing diagonals between the vertices where large number of diagonals exist. Such decomposition further improves the quality. Once decomposed polygon can be executed by clicking *Improved Ear-cutting* method button. Saved polygon can be used multiple times to run different algorithms to compare the results. However, store option will not save triangulating diagonals, therefore algorithm has to be executed again in order to restore previous triangulation results.

Table 4.2 Right Panel checkboxes description.

1	Draw-Vertex	Adds a vertex v_n to edge v_0, v_{n-1}.
2	Edit-Vertex	Changes x and y coordinates of a vertex, update is done by clicking the vertex and dragging it into desired place within a main panel area.
3	Delete-Vertex	Deletes clicked vertex of a polygon by updating the values to the connecting vertices.
4	Split-Edge	Splits the closest edge into two parts by generating new vertex to the closest edge.

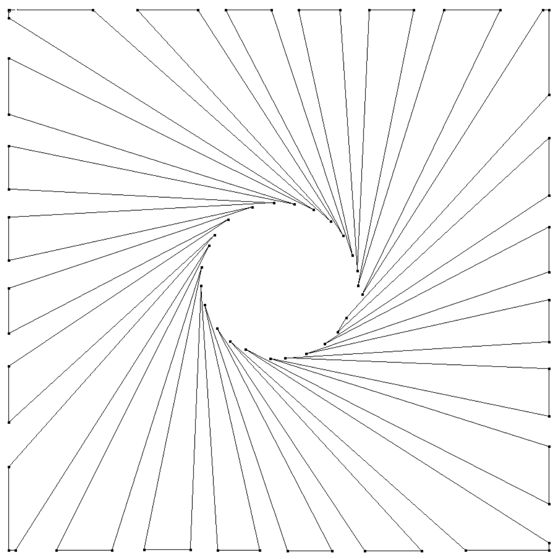


Figure 4.3a: AutoCAD figure used for triangulation

```
221
515  844
656  892
574  824
646  854
705  845
650  822
… …
```

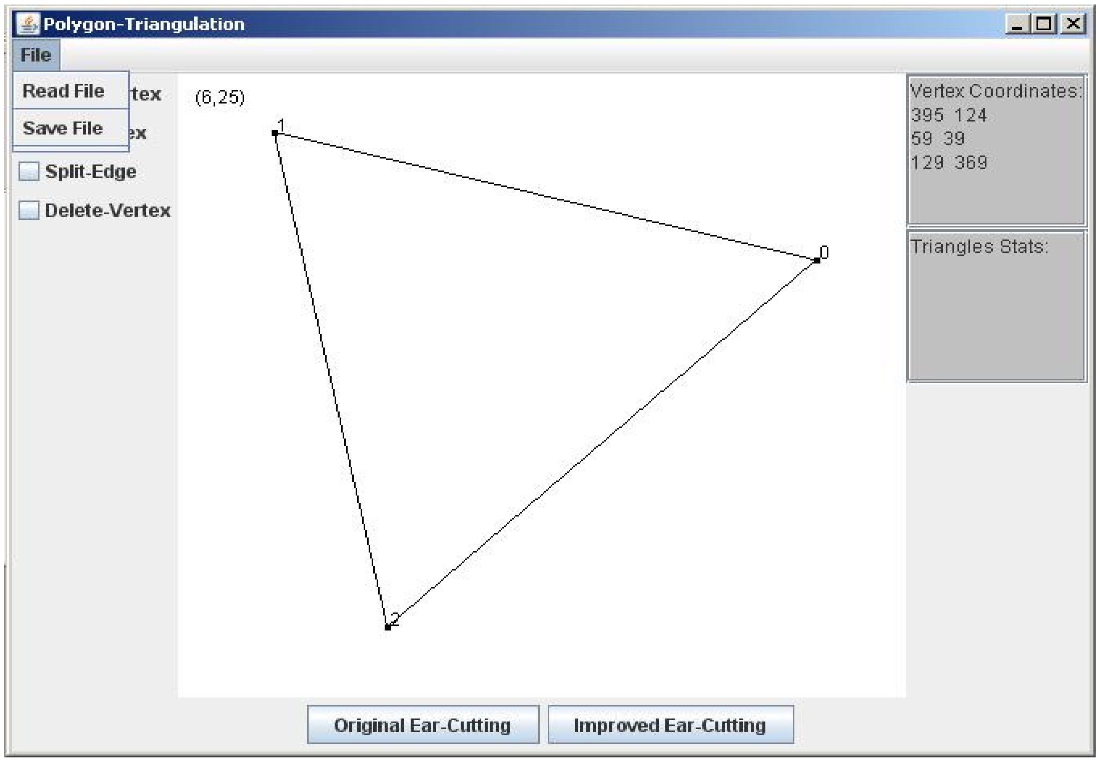
Figure 4.3b: Part of input file generated by AutoLISP script

4.1.2 Program menu items.

File menu is located in top menu bar and contains two basic items: read and save. Generated or modified objects can be saved to repeat research and execution on multiple algorithms. Quality improvement can be measured using *Triangles Stats* data if the same polygon is used to execute various algorithms. Second option allows reading stored objects from a file. Figure contained in the file can be created by user in main panel or generated and extracted from external application, i.e. AutoCAD. File has to be in the format where the first line contains number of vertices, followed by lines containing *x* and *y* coordinates of each vertex. AutoCAD was used to generate desired, complex shapes and AutoLISP script was written to export them to appropriate, readable file format. Figure 4.3a presents figure draw in AutoCAD and Figure 4.3b presents part of input file generated by AutoLISP script. Figure 4.4 and Figure 4.5 shows the GUI representation of the File menu and selection panel to choose or save the polygon respectively.

Table 4.2 File Menu Items description.

1	Read File	Brings up a file selection panel, user can choose a pre generated graph file.
2	Save File	Brings up a file save panel, user can save a new generated file or replace an existing file



```
Vertex Coordinates:
395 124
59 39
129 369
```

```
Triangles Stats:
```

Figure 4.4: File-menu pull down.

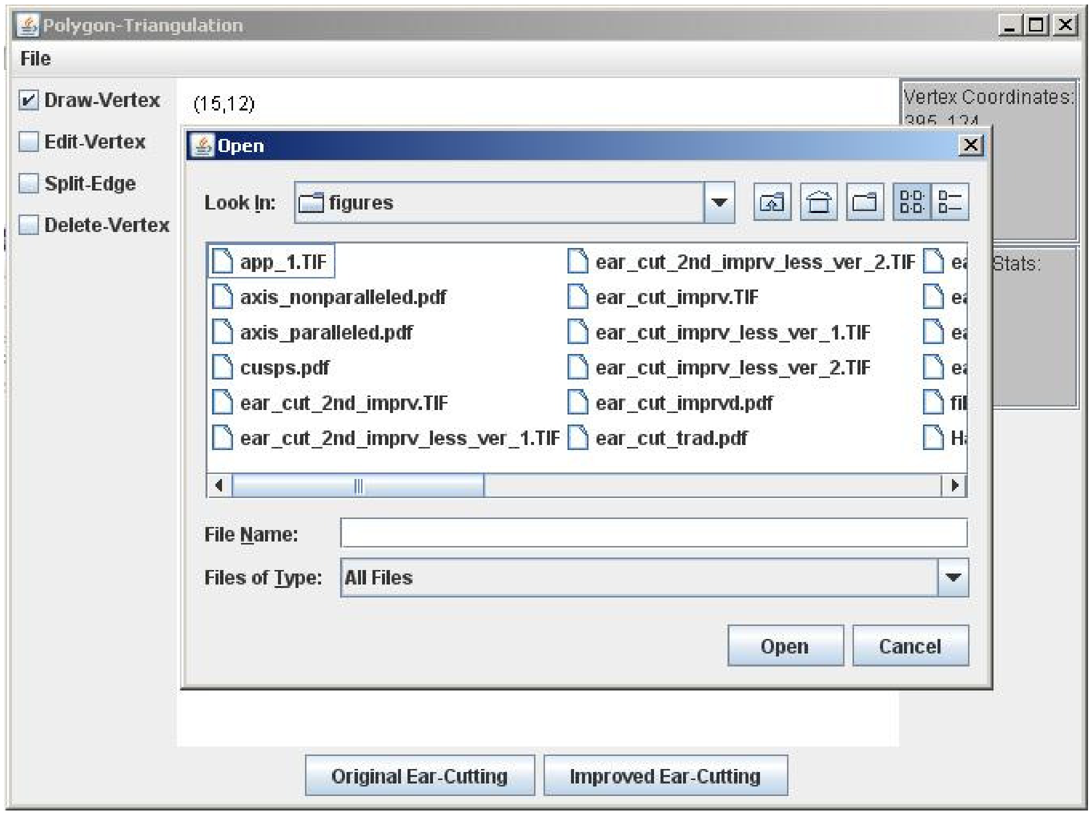
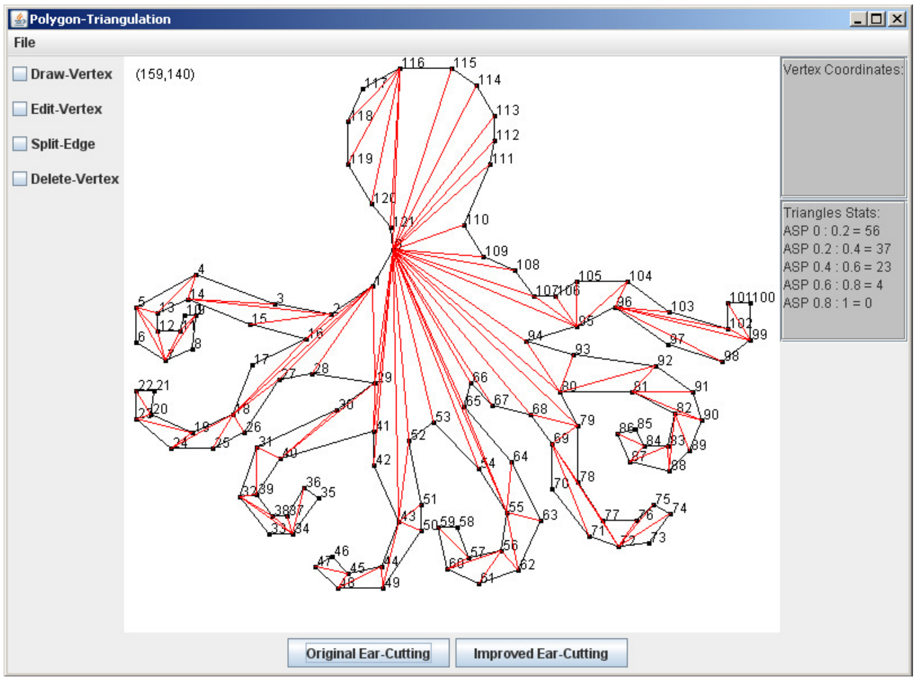


Figure 4.5: Prompting user for File selection.

4.2 Description of methods and classes.

Program Cross Triangulation uses three distinct ways to decompose a polygon drawn by user or read from a file into triangles. Execution is triggered by clicking either *Original Ear-cutting* or *Improved Ear-Cutting* button. Standard *ear-cutting* decomposition is performed using algorithm described in Chapter 2, improved method is based on extended *ear-slicing* algorithm as discusses in Chapter 3. User can also modify code to perform manual decomposition into sub-polygons and then triangulate decomposed pieces using improved method. Resulted triangulating diagonals are painted in red and displayed inside black boundary of a polygon. Main driver of a program is class **public class Cross_Triangulation**. It contains definitions of all methods used to set up GUI components, including panels, buttons and checkboxes. It defines event driven program behavior. It also contains all methods responsible for successful execution of a polygon triangulation. Two main methods that execute triangulation are: **public void triangulate1(Vector)** – implements standard *ear-slicing* algorithm, **public void triangulate2(Vector) –** implements improved *ear-slicing.* Methods **public void triangulateBetween(Vector, int, int, int, int), public void triangulateLeftof(Vector, int, int)** and **public double triangulateRightof(Vector, int, int)** perform manual polygon decomposition and extended triangulation.

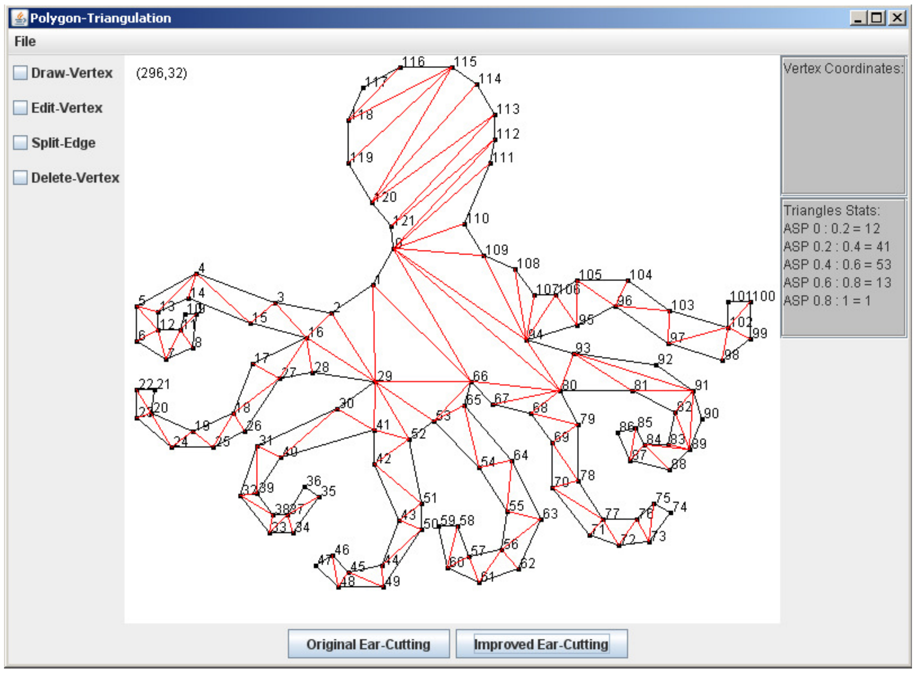
Implemented method **public void triangulate1(Vector)** takes a Vector containing polygon vertices' coordinates as a parameter. It clones the polygon vector and uses new copy to process triangulation algorithm. Method attempts to find a diagonal for each vertex of a polygon starting from vertex *0* until *n-1.* Algorithm checks if diagonal exist for every other vertex i.e. vertex *i* with vertex *i+2*. Standard algorithm blindly searches for diagonals and if a diagonal is found program immediately stores it in a Vector called *diagonal*s and removes vertex *i+1* from cloned polygon (*slicing an ear*). Each algorithm method uses **public boolean isDiagonal_ie(Vector, int, int). isDiagonal_ie** takes polygon (Vector) and two vertices' indices (int) as the parameters. It returns Boolean value *true* if there exists a diagonal between provided vertices. Program repeats finding diagonals and removing *ears* until number of vertices is greater than 3 (residual part is a triangle). Program uses **protected void paintComponent(Grapics g)** method to draw polygon and output triangulating diagonals from Vector(Point) *diagonals* into center panel of GUI. Figure 4.6 presents figure triangulated using original *ear-slicing* method.



```
Triangles Stats:
ASP 0 : 0.2 = 56
ASP 0.2 : 0.4 = 37
ASP 0.4 : 0.6 = 23
ASP 0.6 : 0.8 = 4
ASP 0.8 : 1 = 0
```

Figure 4.6: Figure triangulated using traditional ear-cutting method

Method **public void triangulate2(Vector)** extends **triangulate1** by including method **public int getFatEar(Vector)** . **getFatEar** takes a polygon as a parameter and returns *ear tip* index of a polygon *ear* with the largest aspect ratio. Method searches polygon boundary in *greedy* manner, verifies if triangle is an *ear* (public Boolean isEar(Vector, int)) computes aspect ratio of each of the *ears* and returns index of the one with the largest value. Program continues by *slicing* polygon *ear* with returned index of an *ear tip* until residual part is a triangle. Method **public double getAspectRatio(Point, Point, Point)** takes coordinates of three consecutive vertices of a polygon as a parameters, computes and returns decimal value of an aspect ratio of triangle created by given points. Method **getAspectRatio** uses three additional classes and their methods to obtain lengths and heights of a triangle required to compute Aspect Ratio. Classes **public class my_point**, **public class segment** and **public class line** were used. Once three edges and heights are found method **getAspectRatio** computes three distinct aspect ratios and returns the smallest of them. Such method uses general bounding box as described in Chapter 3 and presented in Figure 3.1b. Figure 4.7 presents the polygon triangulated using improved method which exhibits essential quality improvement over triangulation showed in Figure 4.6.



```
Triangles Stats:
ASP 0 : 0.2 = 12
ASP 0.2 : 0.4 = 41
ASP 0.4 : 0.6 = 53
ASP 0.6 : 0.8 = 13
ASP 0.8 : 1 = 1
```

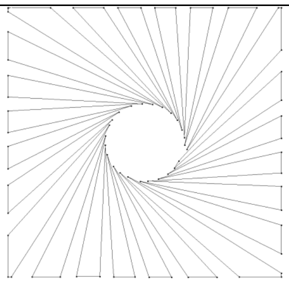
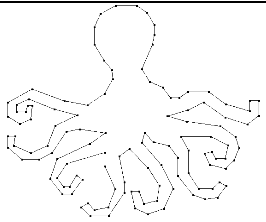
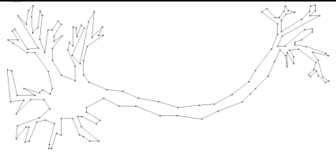
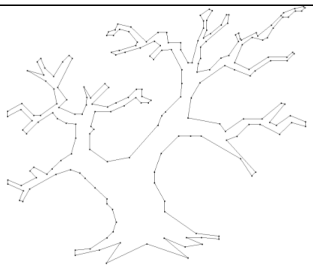
Figure 4.7: Figure triangulated using improved ear-cutting method

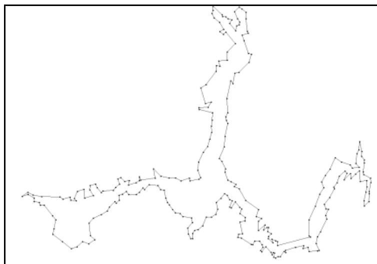
Methods **public void triangulateBetween(Vector, int, int, int, int), public void triangulateLeftof(Vector, int, int)** and **public double triangulateRightof(Vector, int, int)** were implemented to ameliorate obtained results even further by decomposing polygon into sub-polygons in the areas where large number of skinny triangles exist. Method **public void triangulateBetween(Vector, int, int, int, int)** takes as an input Vector containing vertices of a polygon, and four integers with vertices indices. Each pair of vertices indicates start and end of decomposing diagonal. User can provide indices of two diagonals and triangulate area in between them. Method will copy area restricted by two diagonals into new polygon Vector and triangulate it using improved *ear-slicing* technique. Such method can be combined with **public void triangulateLeftof(Vector, int, int)** and **public double triangulateRightof(Vector, int, int). triangulateLeftof** and **triangulateRightof** takes Vector containing vertices of a polygon and two integers with vertices indices as a parameter. Two integers indicate starting and ending index of a decomposing diagonal. First method copies and triangulates area enclosed within polygon boundary to the right of provided diagonal and second method copies and triangulates area enclosed within polygon boundary to the left of provided diagonal. Experiments with manual decomposition yielded surprisingly good results, that encouraged author to develop an automatic method. Polygon partitioning and ear-slicing algorithm was proposed and described in chapter 3 (Algorithm 3).

4.3 Experimental Results

Numbers of experiments with complicated polygonal shapes to test the performance of algorithms were conducted. Table 4.3 presents samples of experimental results. It contains five columns with five different aspect ratios. Each column is divided into two sub-columns that contain number of triangles with respect to aspect ratio for original *ear-slicing* algorithm (Org) and improved triangulation (Imp). We showed results for figures of different shapes and sizes. Number of vertices range between 70 and 300. Regardless of size and shape of the polygon improved *ear-slicing* algorithm yields better quality.

Table 4.3 Triangulation Quality comparison.

Sample Polygon	**Aspect Ratios**									
	Asp= 0:0.2		Asp= 0.2:0.4		Asp= 0.4:0.6		Asp= 0.6: 0.8		Asp= 0.8: 1	
	Org	**Imp**	Org	**Imp**	Org	Imp	Org	Imp	Org	Imp
 n=77	75	**73**	0	**2**	0	**1**	0	**0**	0	**0**
 n=122	56	**12**	37	**41**	23	**53**	4	**13**	0	**1**
 n=170	118	**71**	35	**64**	12	**27**	3	**6**	0	**0**
 n=221	126	**74**	71	**97**	19	**38**	3	**10**	0	**0**

 n=295	203	**126**	60	**90**	21	**50**	9	**24**	0	**3**

Experiments with polygon partitioning were done for complex polygon. Figure 4.8 represents triangulated Lake Mead, Nevada shape with partitioning diagonals showed as bold lines. Simulation is created by reading over three hundreds of vertices that form a complicated polygonal shape. When the component polygons are separately triangulated by using the modified ear-slicing algorithm the result is shown in Figure 4.9, which has large proportions of "quality" triangles. Our experiments based on Lake Mead shape prove amelioration of triangulation quality for improved ear-cutting algorithm.

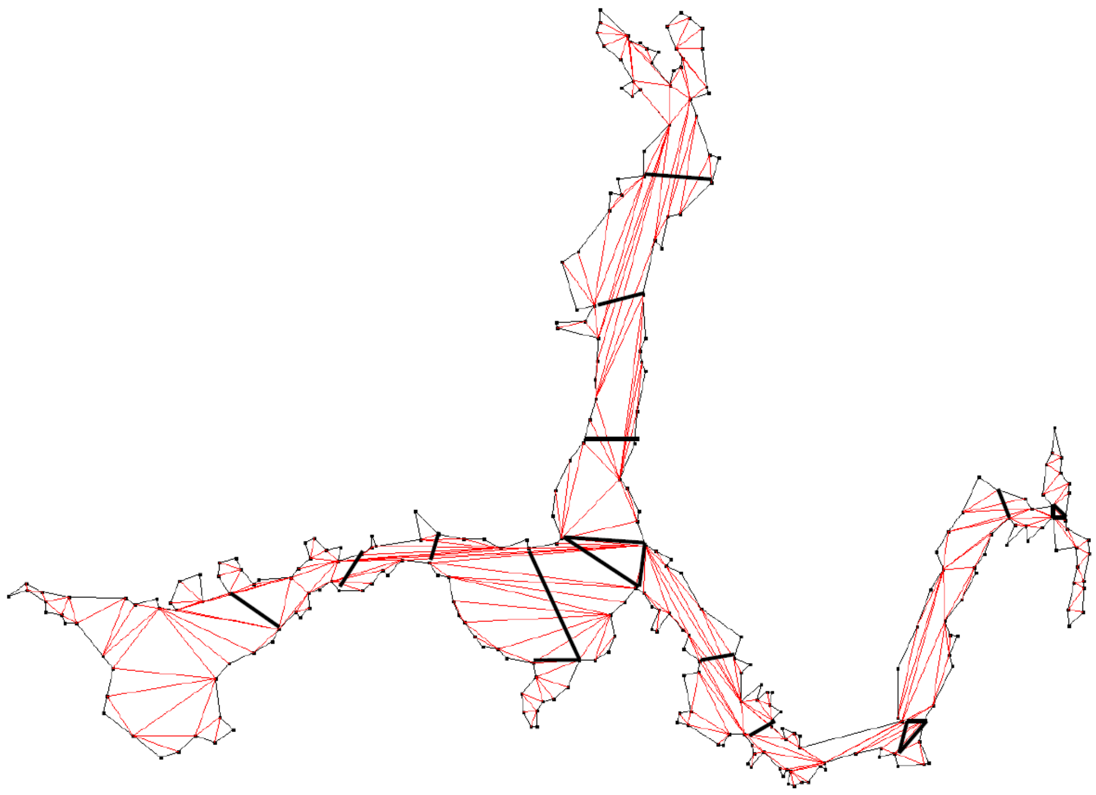


Figure 4.8: Stabbing diagonals (drawn with thick edges) of a triangulated polygon

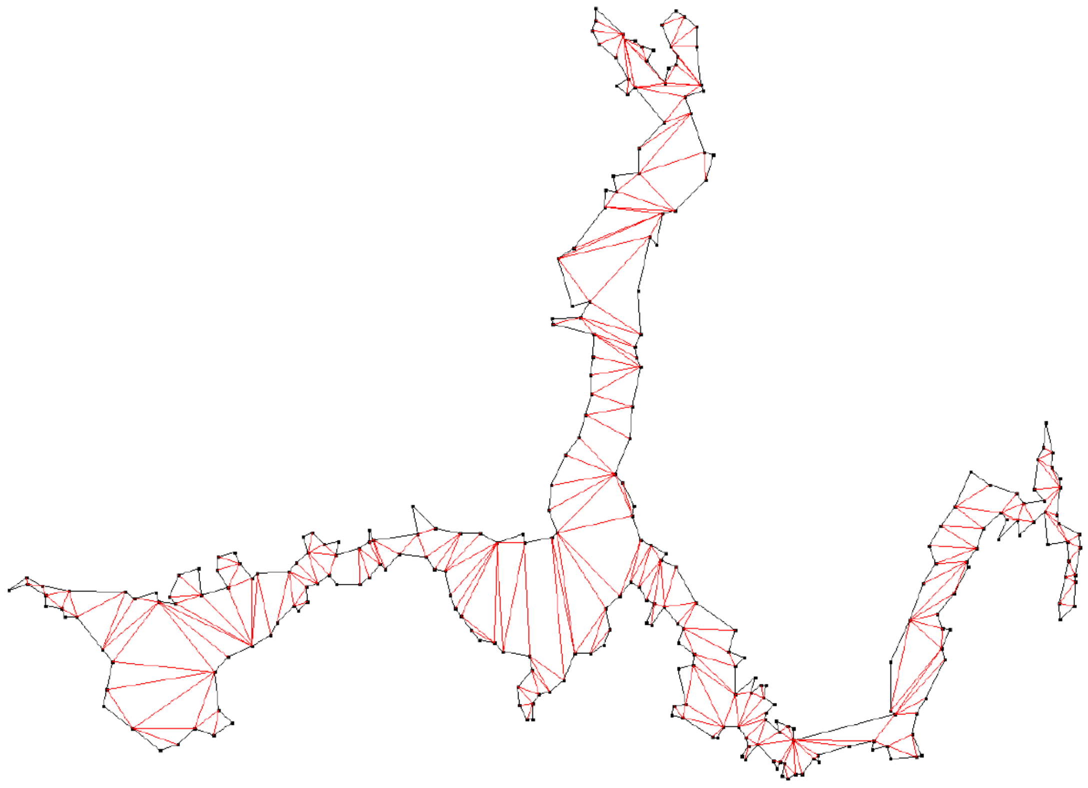


Figure 4.9: Triangulated polygon by partitioning and ear slicing

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

We presented a critical review of polygon triangulation algorithms and underlined the need for improving the ear-slicing polygon triangulation algorithm. We presented two approaches from improving the performance of standard ear-slicing algorithm. The proposed approaches are sketched in a formal algorithm. The time complexity of the first algorithm is $O(n^3)$ where n is the number of vertices in the polygon. The time complexity of the second algorithm is $O(mn^3)$, where *m* is the number of components obtained by partitioning.

We presented an extensive experimental investigation of the first algorithm in Java programming language. The front-end of the implementation has user-friendly interface for entering and displaying polygon and triangulation. The performance of the first algorithm is experimentally investigated on several types of polygons. The compiled results show that the triangulation obtained by the first indeed contains high proportion of quality triangles.

Due to time constraints we were not able to perform extensive experimental investigation of the second algorithm. However a few test cases show that the second algorithm has a potential to generate good quality triangulation.

Several future research activities can be carried out by extending the presented algorithm. One problem would be to extend proposed algorithm to polygon with holes. It would also be fruitful to perform extensive experimental investigation of the second algorithm presented in the thesis.

BIBLIOGRAPHY

[1] O'Rourke, Joseph, *Computational Geometry in C*, Second Edition, Cambridge University Press, 1998.

[2] ElGindy, H., Everett, H., and Toussaint, G. T., (1993) "Slicing an Ear Using Prune-and-Search," *Pattern Recognition Letters*, **14**, (9):719–722.

[3] Meisters, G. H., "Polygons have ears." *American Mathematical Monthly* 82 (1975). 648–651

[4] Chazelle, Bernard, "Triangulating a Simple Polygon in Linear Time", *Discrete and Computational Geometry***6**: 485–524, 1991.

[5] Stefan HertelKurt Mehlhorn, "Fast Triangulation of Simple Polygons", *Proceedings of the 1983 International FCT-Conference on Fundamentals of Computation Theory*.

[6] B. Chazelle, "A Theorem on Polygon Cutting with Applications**"**, *Proc. 23rd FOCS* (1982), 339-349.

[7] Mark de Berg, Marc van Kreveld, Mark Overmars, and OtfriedSchwarzkoft, *Computational Geometry* (2nd revised ed.), Springer Verlag, 2000.

[8] Godfried T. Toussaint, "Efficient triangulation of simple polygons," *The Visual Computer,* vol. 7, 1991, pp. 280-295.

[9] Seidel, Raimund (1991), "A Simple and Fast Incremental Randomized Algorithm for Computing Trapezoidal Decompositions and for Triangulating Polygons", *Computational Geometry: Theory and Applications***1**: 51–64.

[10] B. Preas and M. Lorenzetti, Eds., "Physical Design and Automation of VLSI Systems", Benjamin/Cummings, Menlo Park, 1988.

[11] D. Zydek, "Processor Allocator for Chip Multiprocessors," *Ph.D. Thesis, University of Nevada, Las Vegas, USA*, 2010.

[12] D. Zydek, H. Selvaraj, "Hardware Implementation of Processor Allocation Schemes for Mesh-Based Chip Multiprocessors," *Journal of Microprocessors and Microsystems*, vol. 34, no. 1, 2010, pp. 39-48.

[13] D. Zydek, H. Selvaraj, L. Koszalka, I. Pozniak-Koszalka, "Evaluation Scheme for NoC-based CMP with Integrated Processor Management System," *International Journal of Electronics and Telecommunications*, vol. 56, no. 2, 2010, pp. 157-168.

[14] D. Zydek, H. Selvaraj, "Fast and Efficient Processor Allocation Algorithm for Torus-Based Chip Multiprocessors," *Journal of Computers & Electrical Engineering*, vol. 37, no. 1, 2011, pp. 91-105.

[15] F. Preparata and K. Supowit. Testing a simple polygon for monotonicity *Information Proc. Letters*, 12(4):161–164, 1981.

[16] A. Fournier , D. Y. Montuno, Triangulating Simple Polygons and Equivalent Problems, *ACM Transactions on Graphics (TOG)*, v.3 n.2, p.153-174, April 1984.

[17] E. Arkin, M. Held, J. Mitchell, S. Skiena, Hamiltonian Triangulations for Fast Rendering, in: *J. Van Leeuven, ed., Algorithms –ESA '94, Lecture Notes in Computer Science 855,* Utrecht, Netherlands, September 1994, pp. 36-47.

[18] Suneeta Ramaswami, Pedro Ramos, Godfried Toussaint, Converting triangulations to quadrangulations, in: *Computational Geometry: Theory and Applications,* Computational Geometry 9 (1998) pp. 257-276.

[19] D. Zydek, H. Selvaraj, G. Borowik, T. Luba, "Energy Characteristic of Processor Allocator and Network-on-Chip," *International Journal of Applied Mathematics and Computer Science*, vol. 21, no. 2, 2011, pp. 385-399, DOI: 10.2478/v10006-011-0029-7.

[20] D. Zydek, H. Selvaraj, "Processor Allocation Problem for NoC-based Chip Multiprocessors," Proceedings of 6th International Conference on Information Technology: New Generations (ITNG 2009), IEEE Computer Society Press, 2009, pp. 96-101, DOI: 10.1109/ITNG.2009.182.

[21] D. Zydek, I. Pozniak-Koszalka, L. Koszalka, K. J. Burnham, "Algorithms to Managing Unicast, Multicast and Broadcast Transmission for Optical Switches," Lecture Notes in Computer Science, vol. 5297, Springer Verlag, 2008, pp. 21-30, DOI: 10.1007/978-3-540-88623-5_3.

[22] D. Zydek, H. Selvaraj, L. Gewali, "Synthesis of Processor Allocator for Torus-Based Chip MultiProcessors," Proceedings of 7th International Conference on Information Technology: New Generations (ITNG 2010), IEEE Computer Society Press, 2010, pp. 13-18, DOI: 10.1109/ITNG.2010.145.

[23] D. Zydek, N. Shlayan, E. Regentova, H. Selvaraj, "Review of Packet Switching Technologies for Future NoC," Proceedings of Nineteenth International Conference on Systems Engineering (ICSEng 2008), IEEE Computer Society Press, 2008, pp. 306-311, DOI: 10.1109/ICSEng.2008.47.

VITA

Graduate College
University of Nevada, Las Vegas

Bartosz Kajak

Degrees:

Bachelor of Science in Electrical and Computer Engineering
Lublin University of Technology, Poland 2005

Special Honors and Awards:

GPSA Service Award for displaying outstanding performance through dedication and service to the UNLV and Las Vegas community, Las Vegas 2011

Member of Tau Beta Pi - The USA Engineering Honor Society, University of Nevada, Las Vegas, 2011

Graduate and Professional Student Association Department’s Representative, University of Nevada, Las Vegas, 2010

IEEE Member, Institute of Electrical and Computer Engineers, 2010

Articles in Refereed Conference Proceedings:

B. Kajak, L. Gewali, H. Selvaraj, “Ear Slicing and Quality Triangulation” *Proceedings of Twenty First International Conference on Systems Engineering (ICSEng 2011)*, IEEE Computer Society Press, 2011.

Thesis Title:

Improved Algorithms for Ear Clipping Triangulation.

Thesis Examination Committee:

Chairperson, Henry Selvaraj, Ph.D.
Associate Committee Chair, Laxmi Gewali, Ph.D.
Committee Member, Dawid Zydek, Ph.D.
Graduate Faculty Representative, Pramen Shrestha, Ph.D.