8-1-2012

Degree Constrained Triangulation

Roshan Gyawali
University of Nevada, Las Vegas, roshangyawali@gmail.com

DEGREE CONSTRAINED TRIANGULATION

by

Roshan Gyawali

Bachelor in Computer Engineering
Institue of Engineering, Pulchowk Campus, Tribhuvan University, Kathmandu
2008

A thesis submitted in partial fulfillment
of the requirements for the

Master of Science in Computer Science

School of Computer Science
Howard R. Hughes College of Engineering
The Graduate College

University of Nevada, Las Vegas
August 2012

THE GRADUATE COLLEGE

We recommend the thesis prepared under our supervision by

Roshan Gyawali

entitled

Degree Constrained Triangulation

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

Master of Science in Computer Science
School of Computer Science

August 2012

ABSTRACT

Degree Constrained Triangulation

by

Roshan Gyawali

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

Triangulation of simple polygons or sets of points in two dimensions is a widely investigated problem in computational geometry. Some researchers have considered variations of triangulation problems that include minimum weight triangulation, delaunay triangulation and triangulation refinement. In this thesis we consider a constrained version of the triangulation problem that asks for triangulating a given domain (polygon or point sites) so that the resulting triangulation has an increased number of even degree vertices. This problem is called Degree Constrained Triangulation (*DCT*). We propose four algorithms to solve *DCT* problems. We also present experimental results based on the implementation of the proposed algorithms. The implementation is done in Java programming language with user friendly graphical interface.

ACKNOWLEDGMENTS

First and foremost I offer my utmost gratitude to my supervisor, Dr Laxmi Gewali, who has supported me throughout my thesis with his dedication and knowledge while allowing me the room to work in my own way. His encouragement to implement the proposed solutions has not only made my thesis more resounding but also honed my programming skills. I would like to thank my advisor committee members Dr. Ajoy K Datta, Dr. John Minor and Dr. Rama Venkat for their continued cooperation and encouragement. Finally, I thank my wife Srijana Poudel and my family for their support, patience and sacrifices during my graduate studies.

TABLE OF CONTENTS

ABSTRACT iii

ACKNOWLEDGMENTS iv

LIST OF TABLES vi

LIST OF FIGURES vii

LIST OF ALGORITHMS viii

CHAPTER 1 INTRODUCTION 1

CHAPTER 2 LITERATURE REVIEWS 3
Preliminaries 3
Review of Triangulation Algorithms 5
Triangulation by Ear Clipping 6
Triangulation using Monotone Partitioning 7
Partitioning into Montone Pieces 7
Trapezoidalization by Plane Sweep 9
Triangulation of a Monotone Polygon 10
Delaunay Triangulation 13
Flipping 14
Triangulation Graph 15

CHAPTER 3 DEGREE CONSTRAINED TRIANGULATION 17
Preliminaries and Problem Formulation 17
Degree-Constrained Triangulation of Simple Polygons 22
Development of Scan-Flip Algorithm 22
Development of Partitioning-*AQT* Algorithm 25
Degree-Constrained Triangulation for Points in Two Dimension 28

CHAPTER 4 IMPLEMENTATION AND OBSERVATIONS 32
Interface Overview 32
Structure of the Interface 33
Generating Polygons and Point Sites 38
Observations 43

CHAPTER 5 CONCLUSION AND FUTURE WORK 45

BIBLIOGRAPHY 46

VITA 48

LIST OF TABLES

4.1 Menu bar description 33
4.2 East Panel Checkboxes description 34
4.3 East Panel Button description 36
4.4 Observation of flipping affect on different sets of point sites 44

LIST OF FIGURES

2.1 Triangulation of a polygon of $n = 14$ vertices. Dashed lines are internal diagonals. Dotted lines are some of the external diagonals 4
2.2 Comparing Monotone and Non-Monotone Polygon 5
2.3 Illustrating ear and non-ear 6
2.4 Trapizoidalization. Dotted lines show trapezoid partition chords; dashed lines are diagonals that resolve interior cusps. 8
2.5 Sweep Line Events 9
2.6 Distinguishing three cases 12
2.7 Delaunay triangulation and Incircle test. 13
2.8 Relationship between delaunay triangulation and voronoi diagram 14
2.9 Illustrating Flipping Operation 15
2.10 Triangulation Graph 16

3.1 Even Degree Triangulation 17
3.2 No Even Degree Triangulation 18
3.3 Illustrating the proof of Lemma 3.2 19
3.4 Partial Even Degree Triangulation 20
3.5 Ear Diagonal 20
3.6 Not a true Odd Degree Triangulation 21
3.7 Odd Degree Triangulation 22
3.8 Flipping in Monotone Triangulation 23
3.9 Even Degree Constrained Triangulation using Convex Components . . . 26
3.10 Scan-Flip Algorithm applied on triangulated polygon generated by Partitioning-*AQT* Algorithm 27
3.11 Ineffective Even Degree Constraint Triangulation using Convex Components 28
3.12 Scan-Flip Algorithm applied to triangulated point sites generated by Delaunay Triangulation Algorithm 29

4.1 GUI Layout 34
4.2 Starting Interface 34
4.3 Layout for Point Sites Based Triangulation Applications 35
4.4 Layout for Polygon Based Triangulation Applications 35
4.5 File Open Dialog Window 36
4.6 Polygon Frame 37
4.7 Snap-shot of polygon triangulation 37
4.8 Snap-shot of polygon triangulation after flipping operation 38
4.9 Snap-shot of initial Delaunay triangulation of 1000 point sites 39
4.10 Snap-shot of triangulated point sites after flipping operation 40
4.11 Polygon based Application Class Diagram 41
4.12 Point Sites Application Class Diagram 42

LIST OF ALGORITHMS

2.1 Triangulation by Ear Clipping 6
2.2 Plane Sweep Algorithm for Monotone Partitioning 11
2.3 Triangulation of a Monotone Polygon 12

3.1 Alternate Quadrangulation-Triangulation (*AQT*) Algorithm 19
3.2 Scan-Flip Algorithm 25
3.3 Partitioning-AQT Algorithm 27
3.4 Increased Even-Degree Triangulation of Point Sites 31

CHAPTER 1

INTRODUCTION

Triangulation is one of the fundamental topics in computational geometry and is used in many areas, such as terrain modeling (GIS), scientific data visualization and interpolation, robotics, pattern recognition, meshing for finite element methods (FEM), natural sciences, computer graphics and multimedia. Triangulation of polygons or point sites is a well studied problem [15]. It also serves as a basis for many other geometrical problems. However, triangulation of polygons or point sites may not be unique. In fact, a polygon or a set of points can be triangulated in exponentially many ways [15]. Due to the rich and fertile structure of triangulation problems, many interesting variations have been considered [15]. Generating minimum weight triangulation, triangulation that contains large proportion of fat triangles, and triangulation that maximizes the smallest angle are some of the generalizations proposed for triangulation problems [15]. One approach for expressing generalization is to impose some useful constraints on triangulation. In this thesis, we consider the problem of triangulating simple polygons and point sites subject to *vertex degree constraints*. In particular, we present efficient algorithms for triangulating simple polygons and point sites so that the number of vertices with even degree is substantially increased. This problem has applications in understanding the illumination property of polygons [17].

The thesis is organized as follows. In Chapter 2, we review important properties and algorithmic results dealing with general triangulation and triangulation satisfying certain constraints. In Chapter 3, we formulate a problem of triangulating simple polygons so that the number of vertices with even degree is significantly increased. We name the problem as *Degree Constrained Triangulation* (*DCT*). We present efficient algorithms for solving *DCT* for simple polygons. For convex polygons, the algorithm runs in linear time and for simple polygons the time complexity is $O(t(n))$, where $t(n)$ is the time complexity needed to obtain the initial triangulation. Another algorithm based on convex partitioning called $P - AQT$ *Algorithm* is presented which produces high quality results for polygons which can be decomposed into a fewer number of convex pieces. Chapter 3 also contains the presentation of an efficient algorithm called *scan-flip* that solves DCT problems for sets of point sites in two dimensions. The scan-flip algorithm for point sites runs in $O(nlogn)$ time, where n is the number of input point sites. Furthermore, we present some interesting geometric properties related to the degree constrained triangulations. One interesting property worth noting is that no simple polygon admits complete odd-degree triangulation.

In Chapter 4, we present implementation of selected algorithms described in Chapter 2 and Chapter 3. The implementations are done in the Java programming language and support a friendly user interface. An experimental investigation of the performance of the scan-flip algorithm for solving DCT problems for point sites in two dimensions is described at the end of Chapter 4. Finally, Chapter 5 is a brief discussion about the proposed algorithms and their extensions.

CHAPTER 2

TRIANGULATION AND CONSTRAINED TRIANGULATION

In this chapter we present a critical review of algorithms for triangulating a simple polygon. We first start with the definitions and terms often used for describing triangulation algorithms. We also consider the notion of constrained polygon triangulation and present a review of efficient algorithms for generating such triangulation.

2.1 Preliminaries

A *polygon* is a connected region of a plane bounded by a finite collection of line segments that intersect only at their end points.

Formally, let $v_0, v_1, v_2, \ldots, v_{\text{n-1}}$ be n point vertices in the plane. Let $e_0 = (v_0, v_1), e_1 = (v_1, v_2), \ldots, e_{\text{i}} = (v_{\text{i}}, v_{\text{i+1}}), e_{\text{n-1}} = (v_{\text{n-1}}, v_0)$ be n segments connecting the points. Then these segments bound a polygon if

1. The intersection of each pair of segments adjacent in the cyclic ordering is the single point shared between them: $e_{\text{i}} \cap e_{\text{i+1}} = v_{\text{i+1}}$, for all $i = 0, \ldots, n-1$.
2. Nonadjacent segments do not intersect: $e_{\text{i}} \cap e_{\text{j}} = \phi$, for all $j \neq i+1$.

The points v_{i}'s are the *vertices* of the polygon, and the segments e_{i}'s are its *edges*. Note that a polygon with n vertices has n edges [15]. Polygons defined in this way are called simple polygons to distinguish them from polygons that encloses holes. Now onwards, unless stated otherwise, the term "polygon" is used to indicate simple polygon.

Definition 2.1. An *internal diagonal* of a polygon P is a line segment between two of its non-consecutive vertices v_{i} and v_{j} that are clearly visible to one another. In other words, v_{i} and v_{j} can be connected by a line segment that lies completely inside the polygon.

External diagonals of a polygon are defined similarly.

Definition 2.2. *Triangulation* of a polygon P is the partitioning of its interior by diagonals into a set of non-overlapping triangles (where their interiors do not intersect) without adding new vertices. Figure 2.1 illustrates a triangulated polygon.

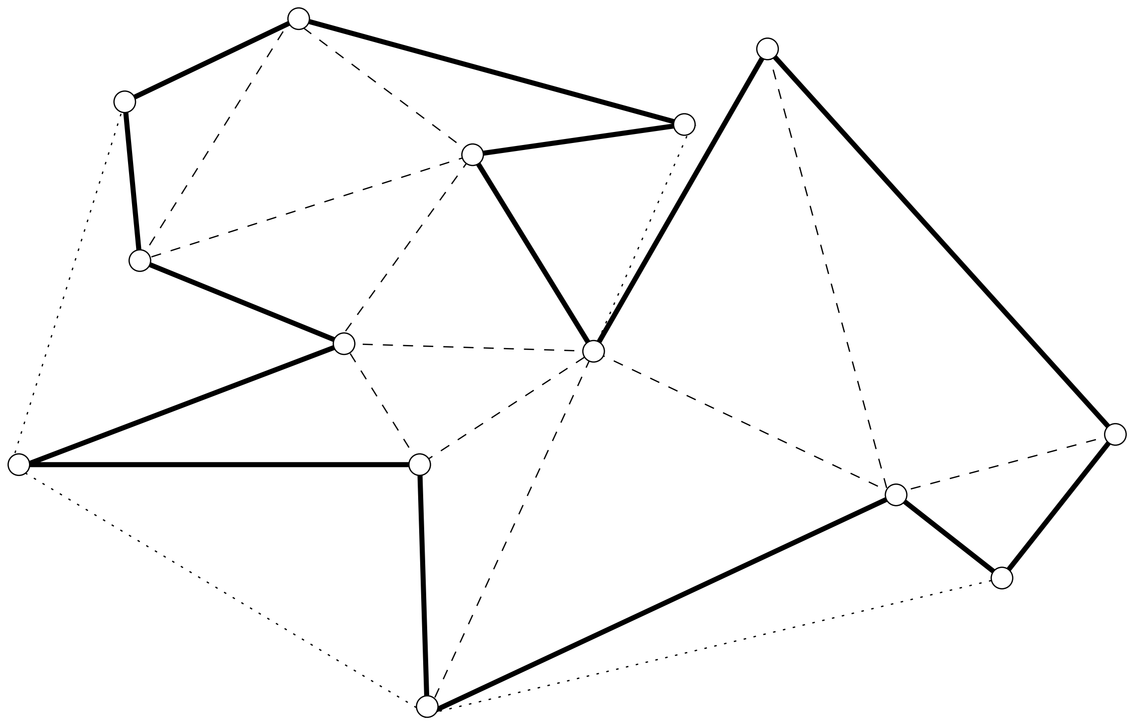


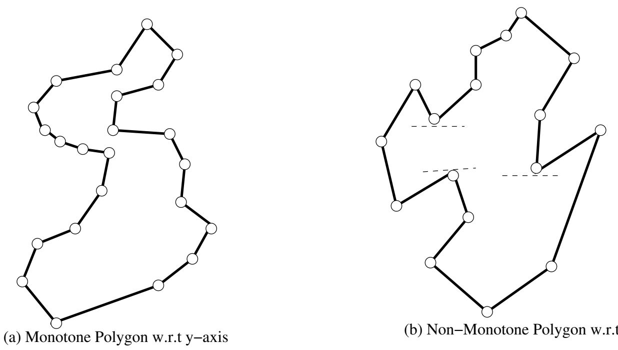
Figure 2.1: Triangulation of a polygon of $n = 14$ vertices. Dashed lines are internal diagonals. Dotted lines are some of the external diagonals

It is remarked that a polygon can be triangulated in exponentially many ways. In fact the number of ways a polygon can be triangulated is related to Catalan Number [15].

Definition 2.3. A *chain* $C = (v_{\rm i}, v_{\rm i+1}, \ldots, v_{\rm j})$ is a sequence of line segments with vertex set $\{v_{\rm i}, v_{\rm i+1}, \ldots, v_{\rm j}\}$ and edge set $\{(v_{\rm k}, v_{\rm k+1}) | k = i, \ldots, j\}$.

Definition 2.4. A chain $C = (v_{\rm i}, v_{\rm i+1}, \ldots, v_{\rm j})$ is said to be *monotone* relative to a given line l if any line orthogonal to l intersects C in exactly one point. In other words, the orthogonal projections $\{l(v_1), l(v_2), \ldots, l(v_{\rm n})\}$ of the vertices of C on l are ordered as $l(v_{\rm i}), l(v_{\rm i+1}), \ldots, l(v_{\rm j})$.

Definition 2.5. A polygon is called *monotone* if its boundary is composed of exactly two non-intersecting monotone chains relative to the same line. For example, a polygon is horizontally monotone if its boundary is composed of two horizontal monotone chains: *upper chain* and *lower chain*. In this case, each chain terminates at the polygons leftmost vertex and rightmost vertex and contains zero or more vertices in between. Figure 2.2 (a) illustrates a monotone polygon, monotone w.r.t to $y - axis$.


(a) Monotone Polygon w.r.t y–axis

(b) Non–Monotone Polygon w.r.t y–axis

Figure 2.2: Comparing Monotone and Non-Monotone Polygon

A y-monotone polygon has two distinguished vertices, the *top-most* vertex and the *bottom-most* vertex with obvious meaning. A concept useful in characterizing monotonicity is *cusp*. A *cusp* is a vertex v, other than the top-most or bottom-most vertex, such that both edges incident on v are either above or below the horizontal line through v. In Figure 2.2 (b), there are three cusps. It has been established that a polygon is y-monotone if it has no cusps [15].

2.2 Review of Triangulation Algorithms

In this sub-section, we present a brief overview of algorithms for triangulating a simple polygon. The first algorithm we review is based on clipping carefully selected triangles called *ear*. We next examine the algorithm for triangulating a special class of polygons - the y-monotone polygon. We also present an overview of triangulating a polygon by using monotone partitioning as a pre-processing step.

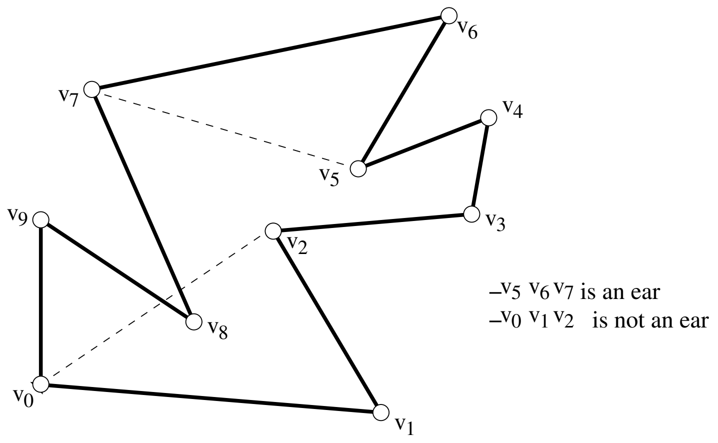


Figure 2.3: Illustrating ear and non-ear

2.2.1 Triangulation by Ear Clipping

This algorithm is based on using the concept of *ear* of a polygon. Three consecutive vertices $v_{\text{i-1}}v_{\text{i}}v_{\text{i+1}}$ are said to form an *ear* if $v_{\text{i-1}}v_{\text{i+1}}$ forms an internal diagonal. Figure 2.3 illustrates the concept of ear.

It has been established that any simple polygon with at least four vertices contain at least two ears [12]. The algorithm is based on removing ears by examining its boundary. When an ear is identified in the original polygon P_{n} (polygon with n vertices). A smaller residual polygon $P_{\text{n-1}}$ is obtained by removing an ear. The process of ear removal is continued until the residual polygon becomes a triangle.

A formal sketch of the algorithm is listed as Algorithm 2.1.

Algorithm 2.1 Triangulation by Ear Clipping

```
INPUT: A simple polygon $P_n$ with vertices $v_0, v_1, \ldots, v_{n-1}$
OUTPUT: A set of diagonals that triangulate the polygon
Step 1: $D = \Phi$; Initialize diagonals to null
        $P = P_n$;
Step 2: while($P$ is not a triangle) do
Step 3:         Let $v_{i-1}v_iv_{i+1}$ be an ear of $P$
Step 4:         $P = P - v_i$;
                $D = D \cup <v_{i-1}, v_{i+1}>$;
Step 5: end while
Step 6: Output $D$
```

The time complexity of ear-clipping triangulation depends on the implementation of identifying ears (Step 3). A brute force implementation of Step 3 takes $O(n^2)$ time which leads to $O(n^3)$ for the whole algorithm. A careful implementation can lead to $O(n^2)$ algorithm. Details are given in [15].

2.2.2 Triangulation using Monotone Partitioning

In order to improve the time complexity for triangulation, an approach based on partitioning the polygon into simpler pieces was introduced. These simpler pieces are called *monotone* pieces. There is a $O(nlogn)$ algorithm for partitioning a polygon into montone polygons and a linear time algorithm for triangulating monotone polygon, leading to an $O(nlogn)$ algorithm for triangulating the polygon. We will describe these two algorithm in detail.

2.2.3 Partitioning into Montone Pieces

It is relatively easier to triangulate a polygon with simpler shapes. Convex polygons and monotone polygons are examples of simpler shapes. To triangulate a convex polygon, it is enough to draw diagonals from a given vertex to all other vertices. Similarly, a y-monotone polygon can be triangulated by a simple top-down scan. (We will provide a short review of such an algorithm at the end of this sub-section). Based on these observations, an approach for triangulating a polygon is to partition the polygon into simpler shapes (say monotone pieces) and apply a monotone triangulation algorithm in each pieces independently. We therefore present a brief overview of partitioning a polygon into monotone pieces.

Polygons can be partitioned into monotone pieces by first breaking them into simpler quadrilaterals. This technique is called *Trapezoidalization* i.e, partitioning into trapezoids. We recall the definition of trapezoid from elementary geometry. A trapezoid is a quadrilateral with two parallel edges. This kind of partitioning was introduced by Chazelle and Incerpi(1984) as the key step for triangulation.

A *horizontal trapezoidalization* of a polygon is obtained by drawing horizontal chords through every vertex of the polygon. More precisely, we construct through each vertex v the maximal (open) horizontal segment s such that $s \subset P$ and $s \cap \partial P = v$. Here P represents the polygonal region and ∂P is the boundary of P. Thus s rep-

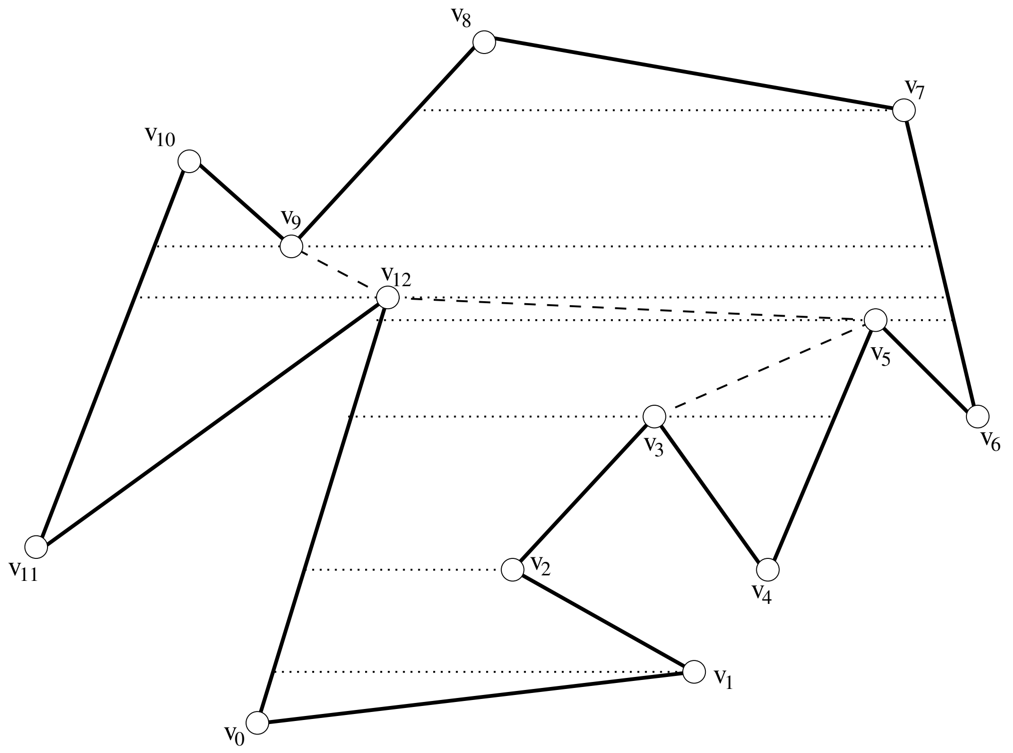


Figure 2.4: Trapizoidalization. Dotted lines show trapezoid partition chords; dashed lines are diagonals that resolve interior cusps.

resents clear lines of sight from v left and right. It may be that s is entirely to one side or the other of v; and it may be that $s = v$. An example of trapezoidalization is shown in Figure 2.4, where horizontal sides of trapezoids are drawn dotted. To simplify the exposition we will only consider polygons whose vertices have unique y coordinates, i.e, no two vertices lie on the same horizontal chord. Such a condition is known as *non-degeneracy* in computational geometry [3].

With this non-degenerate assumption, every trapezoid has exactly two *supporting vertices*, one on its upper edge and one on its lower edge. If a supporting vertex is on the interior of an upper or lower trapezoid edge, then it is an *interior cusp. Partitioning diagonals* are obtained by connecting interior cusp vertices to the opposing supporting vertex of the trapezoid. In Figure 2.4, interior cusp v_3 is connected by supporting vertex v_5 to make a partitioning diagonal $\overline{v_3, v_5}$. If this kind of operation is applied to all interior cusps, the resulting diagonals partition the polygon into monotone pieces. In Figure 2.4, four interior cusps (v_3,v_5,v_9,v_{12}) are resolved by three diagonals $\overline{v_3, v_5}$, $\overline{v_5, v_{12}}$ and $\overline{v_9, v_{12}}$. We next consider the construction of trapezoids in detail.

2.2.4 Trapezoidalization by Plane Sweep

The algorithm we use to construct a trapezoidalization depends on the well known technique called the *plane sweep* (or *sweep line*), which is useful in many geometric algorithms (Nievergelt & Preparata 1982). The main idea is to "sweep" a line over the plane, maintaining some type of data structure during the sweep. The sweep line L stops at discrete "events" where processing occurs and the data structure is updated. Sweep lines requires some preprocessing to be done with the vertices of the polygon. For performing sweep from top to bottom, the vertices are sorted by y-coordinates. Three types of events can be distinguished. When the sweep line is on the vertex v_{i}, the edges e_{i} and $e'_{\mathrm{i+1}}$ incident on it can have the following three properties:

(*i*)e_{i} and $e_{\mathrm{i+1}}$ are both above L.

(*ii*) e_{i} and $e_{\mathrm{i+1}}$ are both below L.

(*iii*) One of them is above L and the other below it.

These properties are illustrated in figure 2.5

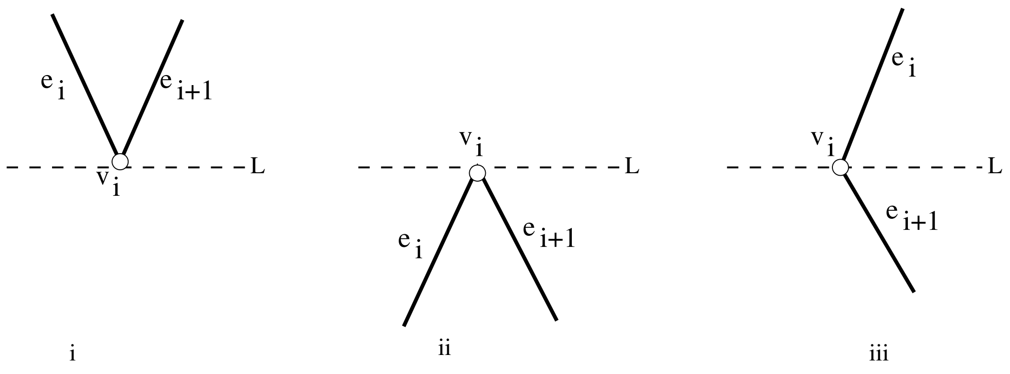


Figure 2.5: Sweep Line Events

In all of these three types, two left and right neighboring edges may exist. The processing required at each of these events is to identify the corresponding event type. The list of edges intersected by the sweep line L are maintained in a sorted list £.

This sorted list £ can be a height-balanced or 2-3 or red-black tree having $O(logn)$ height where n is the number of vertices in the polygon. The edges in a type (i) event will be deleted from the list. The edges in a type (ii) event will be inserted into the list in left to right order. The upper edge in a type (iii) event will be deleted from the list while the lower edge will be inserted into the list. In each of these events, an implied chord corresponding to vertex v is drawn, thus creating a trapezoid. After obtaining the trapezoidization, the partitioning diagonals are constructed by using the concept of interior cusp as mentioned earlier.

A formal sketch of this algorithm is presented in Algorithm 2.2.

2.2.5 Triangulation of a Monotone Polygon

There is a well known linear time algorithm to triangulate monotone polygons [15]. The basic idea for achieving linear time is to make use of the property of monotonicity. It is noticed that vertices of a montone polygon are already available in two sorted list. This means the entire sorted list of vertices (sorted by y-coordinates) can be obtained by merging the sorted sub-list in linear time. Let L_{p} denote this sorted list. The algorithm proceeds top to down in a greedy manner by processing the vertices in the sorted list, and temporarily storing the scanned vertices which cannot be used for triangulation in a stack data structure. Initially two top-most vertices p_0 and p_1 are pushed onto an empty stack S. We denote the vertices in the stack S, bottom to top as w_0, w_1,..., w_{t}. The next scanned vertex p_{i} from L_{p} is called *bottom-incident* if p_{i} is adjacent to w_0 in the polygon boundary. Similarly, p_{i} is called *top-incident* if p_{i} is adjacent to w_{t}. For constructing triangulation with p_{i} as a vertex, three cases are distinguished.

First case (Case 1) is the one in which p_{i} is top-incident and angle $<w_{\mathrm{t-1}},w_{\mathrm{t}},p_{\mathrm{i}}>$ is reflex. In the second case, p_{i} is top-incident and angle $<w_{\mathrm{t-1}},w_{\mathrm{t}},p_{\mathrm{i}}>$ is less than π. When p_{i} is bottom-incident, we have the third case. These are illustrated in figure 2.6.

For Case 1, p_{i} is pushed onto the stack. For Case 2, points are popped from the stack until $p_{\mathrm{i}}w_{\mathrm{i}}$ becomes tangent to the chain in the stack as shown in Figure 2.6.

Algorithm 2.2 Plane Sweep Algorithm for Monotone Partitioning

```
INPUT: A simple polygon $P$ with vertices $v_0, v_1, \ldots, v_{n-1}$ listed along the boundary.
OUTPUT: Monotone Components
Sort vertices of $P$ by y-coordinates. Let the sorted list be <$w_0, w_1, \ldots, w_{n-1}$>
Let £ be a height balanced search tree data structure where we can maintain polygon edges in left to right order (i.e order of x-coordinates). Initialize £ to empty.
for($i = 0; i < n - 1; i++$){

   1. Let $L_i$ be the horizontal line through $w_i$.

   2. Let $c$ and $d$ be the polygon edges incident on $w_i$.

   3. Let $a$ and $b$ (if any) be edges immediately to the left and right of $w_i$.

   4. if($c$ is above $L$ and d below){
          draw horizontal chord from $w_i$ to $a$ or $b$.
          delete $c$ from $\alpha$.
          insert $d$ to $\alpha$.
      }

   5. if(both $c$ and $d$ are above $L$){
          draw horizontal chord from $w_i$ to $a$ or $b$.
          delete $c$ and $d$ from $\alpha$.
      }

   6. if(both $c$ and $d$ are below $L$){
          draw horizontal chord from $w_i$ to $a$ or $b$.
          insert $c$ and $d$ to $\alpha$.
      }

}
for cusp vertex $w_i$ in $w$ do
   draw diagonal from $w_i$ to appropriate supporting vertex in the trapezoid.
end for
Output components by traversing the boundary of the polygon.
```

For Case 3, triangulation is performed by drawing edges from p_2 to the vertices in the reflex chain as shown in figure 2.6 case 3. It is remarked that the vertices in the stack always form a reflex chain. The algorithm is formally listed as Algorithm 2.3.

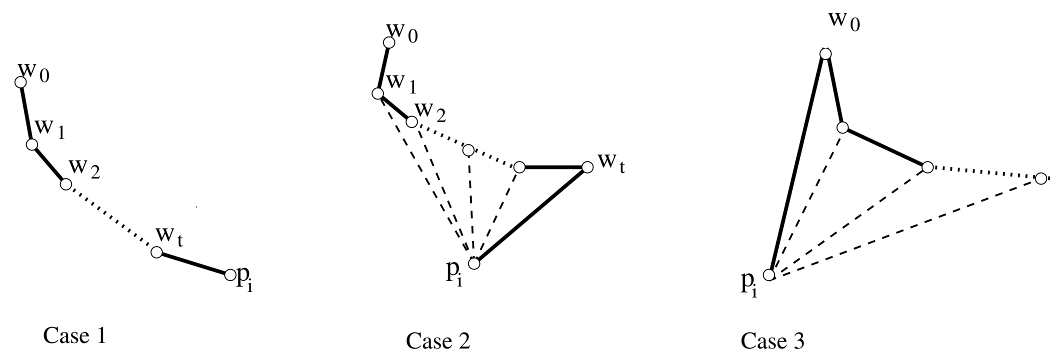


Figure 2.6: Distinguishing three cases

Algorithm 2.3 Triangulation of a Monotone Polygon

```
**INPUT**: Monotone polygon $P$ with vertices $v_0, v_1, \ldots, v_{n-1}$.
Sort vertices of $P$ by y-coordinates. Let the sorted list be <$p_0, p_1, \ldots, p_{n-1}$>
Push $p_0, p_1$ onto stack $S$.
Let <$w_0, w_1, \ldots, w_t$> denote stack content from bottom to top order.
for $i = 2$ to $n - 1$ do
  if $p_i$ is adjacent to $w_0$ then
    $x = w_t$
    while $t > 0$ do
      draw diagonal $p_i \rightarrow w_t$
      pop $S$
      $t - -$
    end while
    push $x$, push $p_i$
  else if $p_i$ is adjacent to $w_t$ then
    while $t > 0$ and $w_t$ is not reflex do
      draw diagonal $p_i \rightarrow w_{t-1}$
      pop $S$
      $t - -$
    end while
    push push $p_i$
  end if
end for
```

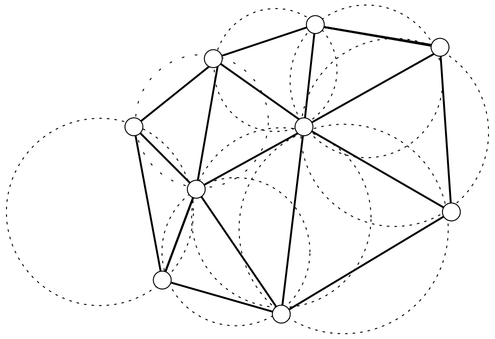


Figure 2.7: Delaunay triangulation and Incircle test.

2.2.6 Delaunay Triangulation

Delaunay triangulation $D_{\mathrm{T}}(P)$ of a finite set $P = \{p_0, p_1, \ldots, p_{\text{n-1}}\}$ in $2D$, is the triangulation that fulfills the condition that no points of point sites P is inside the circumcircle of any triangle in $D_{\mathrm{T}}(P)$. This condition is often called *Empty Circumcircle or Incircle Test.* Figure 2.7 illustrates a Delaunay triangulation of point sites and circum-circles drawn by dashed arcs. Delaunay triangulation satisfies many interesting properties that includes:

1. The interior of each face of $D_{\mathrm{T}}(P)$ contains no point sites.
2. The circumcircle of each triangle contains no point sites.
3. The convex hull of point sites encloses all triangles.
4. The dual of a Delaunay triangulation is a Voronoi diagram. In Figure 2.8, the Voronoi diagram is drawn by dashed lines.
5. Among all possible triangulations of point sites P, Delaunay triangulation has the largest minimum inner angle. It is also referred as $MaxMin$ angle criterion.
6. No four points of P are co-circular.

Many algorithms for generating Delaunay triangulation have been reported [15]. Some algorithms make use of the duality property of Voronoi diagrams. However, It is equally efficient to use direct algorithms to construct Delaunay triangulation.

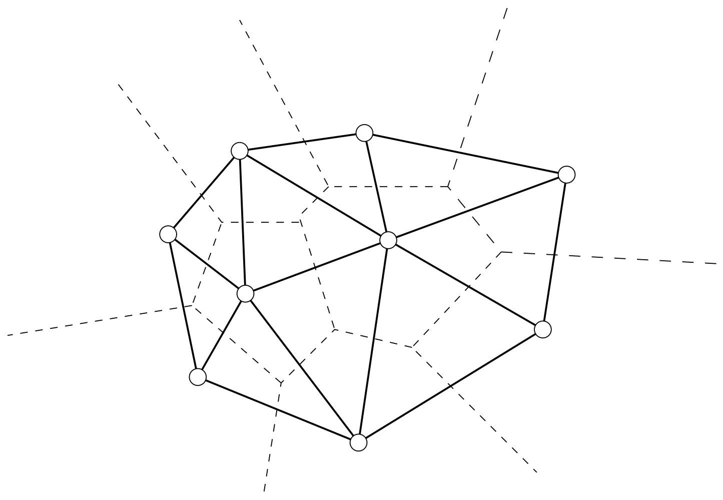


Figure 2.8: Relationship between delaunay triangulation and voronoi diagram

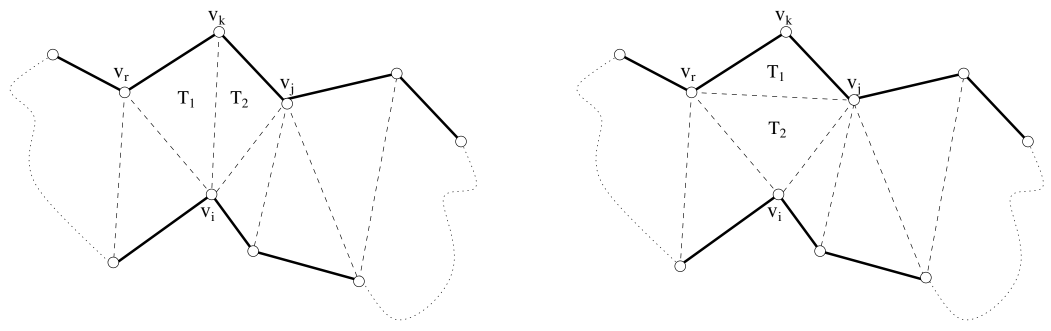
Some of the well known delaunay triangulation algorithms are local improvement [2], incremental construction [7], incremental insertion [11], higher dimension embedding [15] and divide & conquer [6].

2.2.7 Flipping

As mentioned in chapter 1, a polygon or point set can be triangulated in exponentially many ways. It is thus itself an interesting problem to obtain some other triangulation from a given triangulation. The *flipping operation* in triangulation has been considered to obtain one kind of triangulation from an other. Informally, the flipping operation is obtained by replacing diagonals of the quadrilateral of two adjacent triangles in the triangulation. It has been established that any triangulation of a point set (or polygon) can be obtained from an other by a sequence of flipping operations [16]. A formal definition of flipping operation is given in the next definition.

Definition 2.6. Consider a triangulated polygon $T(P)$ obtained by triangulating a simple polygon P. Let $<v_{\mathrm{i}}, v_{\mathrm{j}}, v_{\mathrm{k}}, v_{\mathrm{r}}>$ be the convex quadrilateral formed by adjacent triangles T_1 and T_2 sharing edge $<v_{\mathrm{i}}, v_{\mathrm{k}}>$ as shown in Figure 2.9a.

The modification of $T(P)$ obtained by replacing edge $<v_{\mathrm{i}}, v_{\mathrm{k}}>$ with crossing-diagonal $<v_{\mathrm{j}}, v_{\mathrm{r}}>$ is called *flipping operation.* Modified $T(P)$ is shown in Figure 2.9b.



(a) Before Flipping (b) After Flipping

Figure 2.9: Illustrating Flipping Operation

It is noted that an edge is *flippable* if it is a common edge to two adjacent triangles that forms a convex quadrilateral. In Figure 2.9a, edge <v_i, v_k> is flippable because it is the common edge of adjacent triangles T_1 and T_2 that forms a convex quadrilateral <$v_i, v_j, v_k v_r$>. Not every two adjacent triangles of a arbitrary triangulation forms a convex quadrilateral. This leads to a quest to know how many edges of arbitrary triangulation are flippable or how many flips are needed to transform from one triangulation to another. Studies have shown that, for n point set, there are $\frac{n-4}{2}$ edges that can be flipped [9]. It is also known that, it takes at most $O(n + k^2)$ flips for a simple triangulated polygon P with n vertices and k reflex vertices to transform from one triangulation $T_1(P)$ to another triangulation $T_2(P)$. There is a close relationship between visibility graph, flipping operation, and triangulated graph which we briefly review next.

2.2.8 Triangulation Graph

Given a set P of points in general position in the plane, the *graph of triangulations* $T_G(P)$ has a vertex for every triangulation of P, and two of them are adjacent if they differ by a single edge flip. It is well-known that $T(P)$ is a connected graph [9]. There are a number of interesting properties discovered about Triangulation Graph. The diameter of $T_G(P)$ is at most $O(n^2)$ where n is the size of its point set [9]. Note that *diameter* of triangulation graph $T_G(P)$ can be defined as the maximum distance between two nodes of the entire $T_G(P)$ where distance is measured by the number

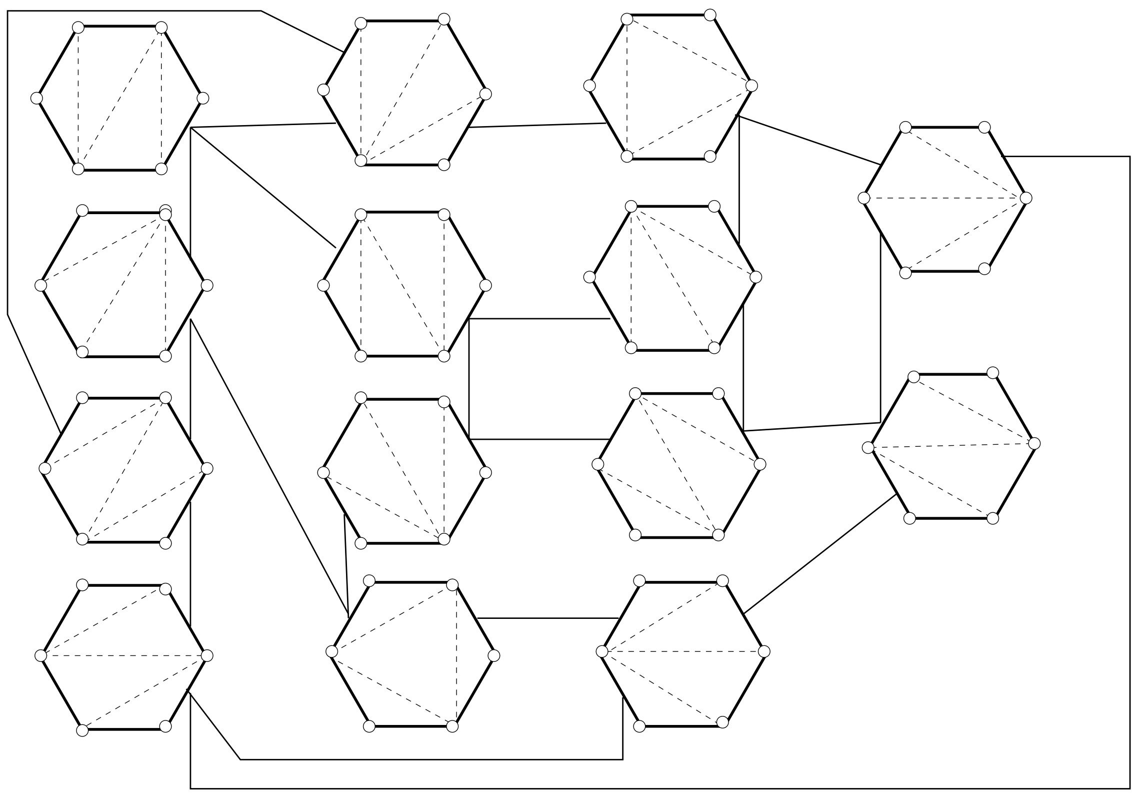


Figure 2.10: Triangulation Graph

of edges in the path. Diameter is also related to its visibility graph. It is at most equal to the number of edges of the visibility graph of a simple polygon. Note that the *Visibility graph* V_G of a simple polygon P is defined by associating a vertex v_i with each point p_i of P such that (v_i, v_j) is an undirected edge of V_G if p_i and p_j are mutually visible [14]. Some further results on the graph of triangulations of convex polygons have been reported in [10]. Figure 2.10 depicts the triangulation graph of a regular hexagon. Each node of the graph represents a unique triangulation of the hexagon. Also each connected node differs only in one flipped diagonal. Since triangulation graph $T_G(P)$ is connected, it can be easily observed that each node can be transformed to another node in $T_G(P)$ with a series of flips.

CHAPTER 3

DEGREE CONSTRAINED TRIANGULATION

3.1 Preliminaries and Problem Formulation

In this chapter, we present the main contribution of the thesis. We first formulate the Degree Constrained Triangulation (*DCT*) problem that asks for a triangulation of a simple polygon or set of nodes with higher number of even-degree nodes. We then describe the development of efficient algorithms for solving *DCT* problem. The first algorithm we develop called *AQT Algorithm*, is designed to work only for "*near convex*" polygons. For non-convex polygons we propose two algorithms - one based on scanning and flipping (*Scan-Flip Algorithm*) and the other based on convex partitioning (*Partitioning AQT Algorithm*). Finally, we present a *Scan-Flip Algorithm* for solving *DCT* problem for node distribution in two dimensions. This algorithm solves *DCT* by flipping adjacent triangles on Delaunay triangulation.

Definition 3.1. A simple polygon P is said to admit *even-degree triangulation* if every vertex in the triangulated graph of P is of even degree. Figure 3.1 shows an example of even degree triangulation.

The notion of odd-edge triangulation can be defined similarly.

Lemma 3.1. Not every polygon admits even degree triangulation. Figure 3.2 illustrates this lemma. Vertices v_i and v_j will never be of even degree in the triangulation.

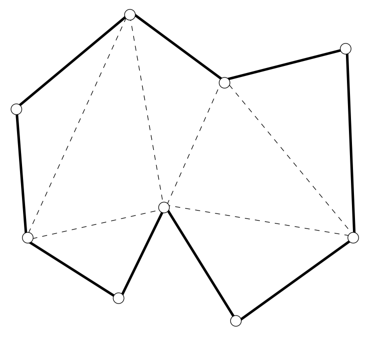


Figure 3.1: Even Degree Triangulation

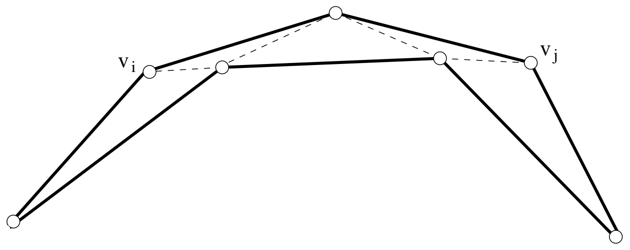


Figure 3.2: No Even Degree Triangulation

This polygon can be triangulated only in one way.

Degree Constrained Triangulation (DCT) Problem: Given a polygon P, the *DCT* problem asks to triangulate P with increased number of even-degree vertices. *DCT* problem for a set of nodes in two dimensions can be defined similarly.

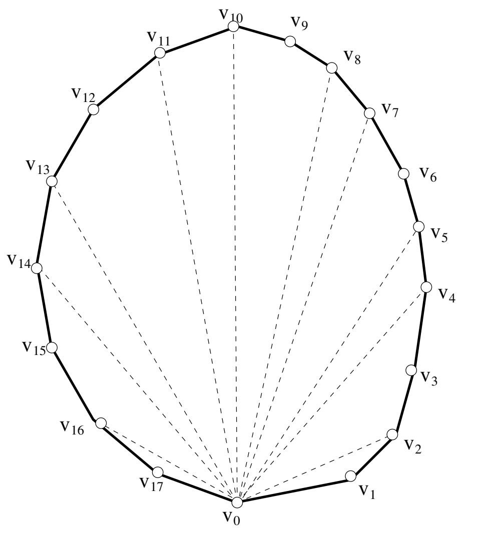
Lemma 3.2. Any convex polygon with $n = 3k$ (k>1) vertices admits even degree triangulation.

Proof We sketch a constructive proof. Note that the vertices of polygon P are $v_0, v_1, v_2, \ldots, v_{\text{n-1}}$ We first partition the convex polygon of $3k$ vertices into alternate triangles and quadrilaterals by following Rule 1. **Rule 1** requires to draw diagonals from vertex v_0 to all other vertices v_{i} such that index i is not a multiple of 3. The partitioning of the convex polygon by executing Rule 1 is shown in Figure 3.3a, where the polygon is partitioned into alternate triangles and quadrilaterals.

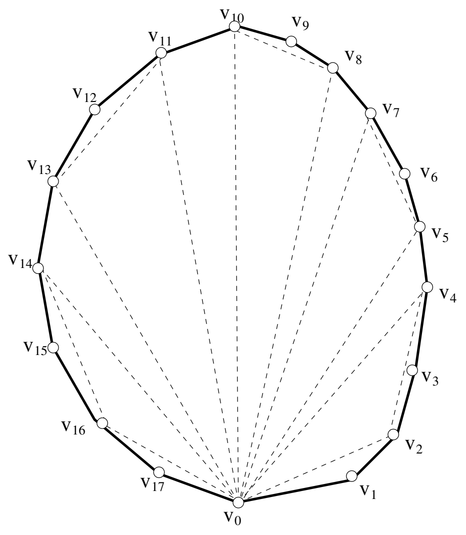
Partition each quadrilateral by executing Rule 2. **Rule 2**: Each quadrilateral $v_0 v_{\text{i}} v_{\text{i+1}} v_{\text{i+2}}$ is partitioned into triangles by drawing diagonal $\overline{v_{\text{i}} v_{\text{i+2}}}$. The resulting triangulation is shown in 3.3b. It is easily observed that the degree of vertices in the triangulation is either 2 or 4. □

A formal sketch of the algorithm for solving *DCT* problem for convex polygon as outlined in the proof of Lemma 3.2 is written in Algorithm 3.1.

Lemma 3.3. Any convex polygon with $n = 3k + 1$ or $3k + 2$ (k>0) vertices can be triangulated to have *at least* $n - 2$ vertices of even degree.



(a) Partitioning to alternate triangles and quadrilaterals with diagonals obtained by following Rule 1



(b) Triangulation obtained by following Rule 2

Figure 3.3: Illustrating the proof of Lemma 3.2

Algorithm 3.1 Alternate Quadrangulation-Triangulation (AQT) Algorithm

```
**INPUT**: A convex polygon $P = <v_0, v_1, \ldots, v_{n-1}>$
**OUTPUT**: Triangulation $T(P)$ of $P$ with higher number of even-degree vertices
Step 1: $T(P) = P$
Step 2: for $i = 2$ to $n - 2$ do
Step 3:         if ($i$ is not a multiple of 3)
Step 4:             $T(P) = T(P) \cup <v_0, v_i>$
Step 5:         end if
Step 6: end for
        // Now $T(P)$ consists of alternate Triangles and Quadrilaterals
Step 7: $i = 2$; //Skip the first triangle
Step 8: while ($i < n - 3$) do
Step 9:         $T(P) = T(P) \cup <v_i, v_{i+2}>$ //partition quadrilateral $<v_0, v_i, v_{i+1}, v_{i+2}>$
Step 10:         $i = i + 3$; //Skip the triangle
Step 11: end while
Step 12: Output $T(P)$
```

Proof First consider the case when the number of vertices is $n = 3k + 1$. We chop a triangle $T' = v_i v_{i+1} v_{i+2}$ from polygon P to obtain a polygon P' with $3k$ vertices. Polygon P' can be triangulated to have all its vertices of even degree by Lemma 3.2. When we put back T' to triangulated P' all vertices except v_i and v_{i+2} are of even degree. The case for $n = 3k + 2$ (k>0) follows similarly. Figure 3.4 illustrates this lemma. Vertices v_{i+} and v_{i+2} are the only odd degree vertices. $\square$

Theorem 3.1. *DCT* problem for convex polygon can be solved in $O(n)$ time

Proof The for-loop of *AQT* algorithm executes $O(n)$ time and each execution takes $O(1)$ time. Also, the while-loop of *AQT* algorithm executes $O(n)$ time and one execution of the body of while-loop takes $O(1)$ time. Hence the entire algorithm takes $O(n)$ time. $\square$

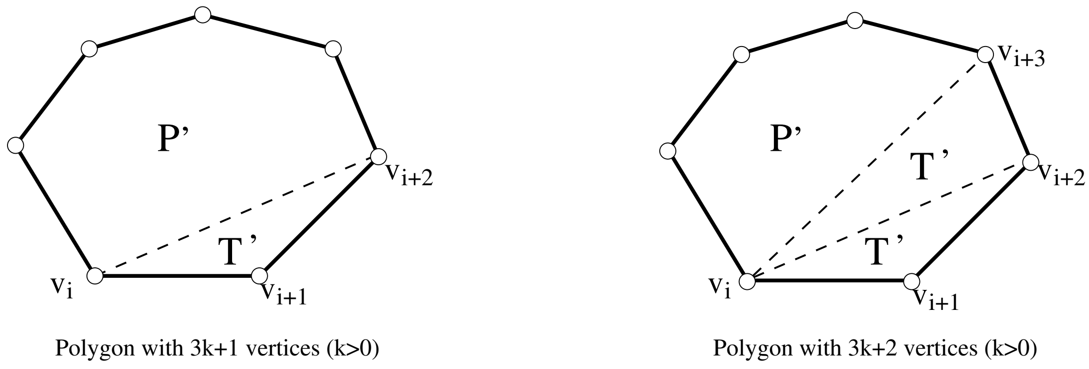


Figure 3.4: Partial Even Degree Triangulation

Definition 3.2. If (v_i, v_{i+2}) is a diagonal of a triangulated polygon $T_1(p)$ then it is called *ear-diagonal*. In the Figure 3.5 there are three ear diagonals.

The following lemma directly follows from Meister's two-ears theorem [12].

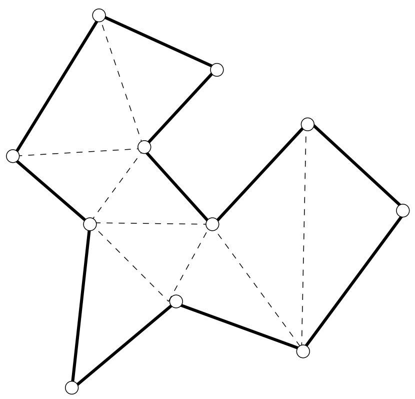


Figure 3.5: Ear Diagonal

Lemma 3.4 (Ear-Diagonal Lemma). Every triangulated polygon with at least four vertices contains two ear-diagonals.

It is interesting to examine the possibilites of triangulating a simple polygon so that all its vertices have odd degree. It turns out that no polygon of $n > 3$ vertices admit odd-degree triangulation. This is established by the following Lemma.

Lemma 3.5. No simple polygon admits odd-degree triangulation.

Proof Assume to the contrary that some simple polygon of $n > 3$ vertices admits odd-degree triangulation $T(Q)$. Then there will be at least one diagonal emanating from every vertex of Q in $T(Q)$ as shown in Figure 3.6.

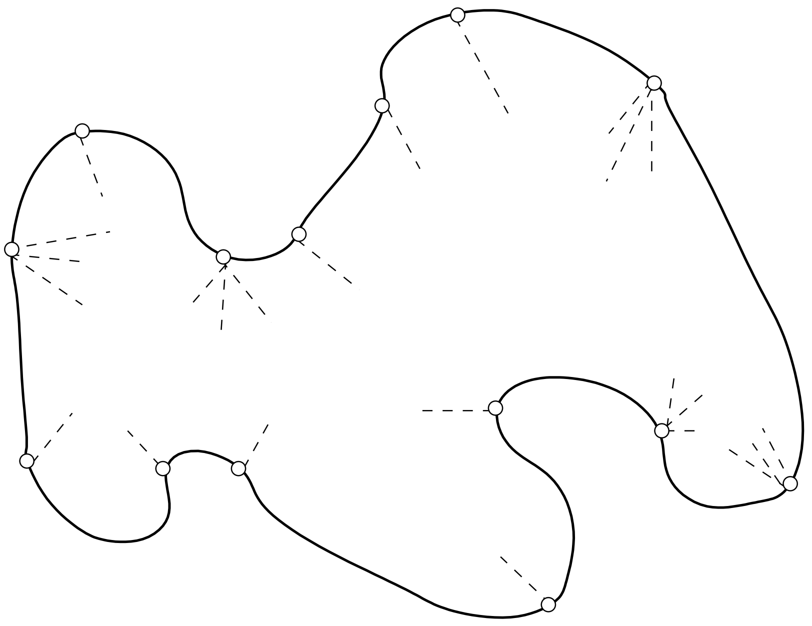


Figure 3.6: Not a true Odd Degree Triangulation

The presence of diagonal incident at all vertices imply that $T(Q)$ has no ear-diagonal. This is contradictory to ear-diagonal lemma (lemma 3.4). □

Observation 3.1. It turns out that a polygon with holes could admit odd-degree triangulation. This is illustrated in Figure 3.7

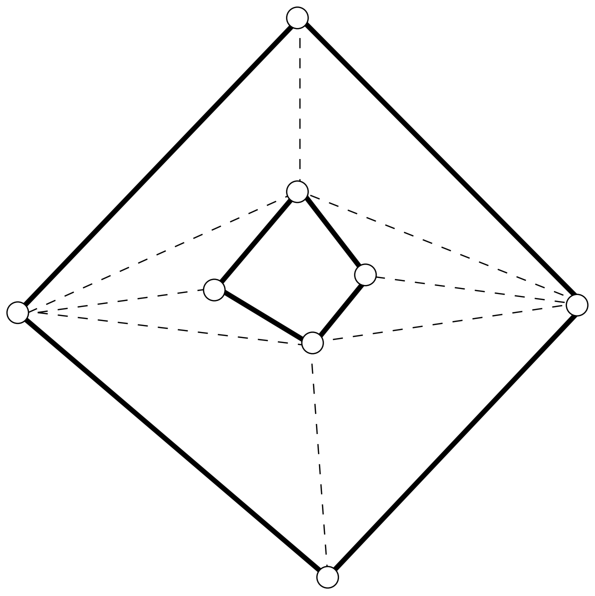


Figure 3.7: Odd Degree Triangulation

3.2 Degree-Constrained Triangulation of Simple Polygons

We now develop algorithms to solve DCT problems for simple polygons, not necessarily convex. The first algorithm we present works by applying flipping operations on a triangulated partitioning of the polygon. The second algorithm applies flipping operations by first partitioning the input polygon into convex components.

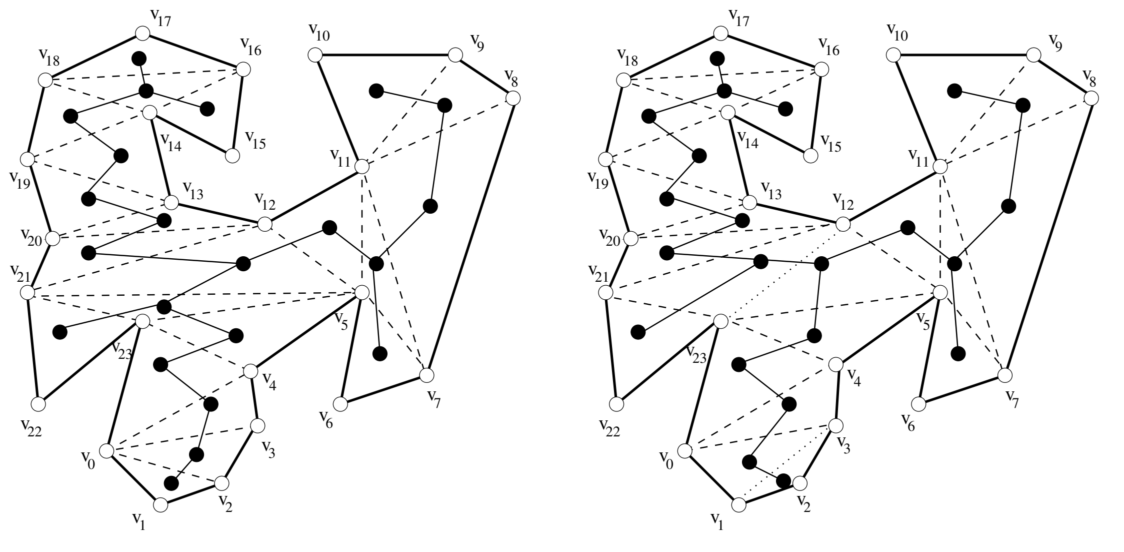
3.2.1 Development of Scan-Flip Algorithm

The input is a simple polygon P. The polygon P is first triangulated by using any suitable triangulation algorithm available in the literature [15]. We use doubly connected edge list data structures ($DCEL$) [13] to store the resulting triangulated polygon $T(P)$. The algorithm proceeds by processing diagonals incident on a vertex. The vertices are visited by traversing them along the boundary of the polygon. It is noted that each diagonal d of triangulated polygon $T(P)$ corresponds to a unique quadrilateral $Q(d)$ formed by combining two triangles incident on it. The algorithm processes $Q(d)$ to check if its diagonal can be flipped to increase the number of even-degree vertices. The two rules for checking valid flipping conditions for quadrilateral $Q(d)$ can be listed as follows.

Flippability Rules

Rule 3: $Q(d)$ must be convex.

Rule 4: $Q(d)$ must have more than two odd-degree vertices.



(a) Triangulated simple polygon T(p) with its dual

(b) T(p) after flipping operation
Dotted line represents flipped diagonal

Figure 3.8: Flipping in Monotone Triangulation

We can illustrate the progress of the algorithm with a running example. Figure 3.8(a) represents a triangulated simple polygon $T(P)$ with vertices $v_0, v_1 \ldots, v_{23}$. The *Triangulation dual* of $T(P)$ is drawn as thick segments connecting black dots. Note that the dual of a triangulated polygon $T(P)$ is a graph whose nodes are the triangles and edges are formed by connecting adjacent triangles. The dual of a triangulated simple polygon is a tree.

In order to traverse diagonals of $T(P)$, we perform a vertex scan starting with any vertex say, v_0. For each vertex v_{i}, we find the diagonals incident on it. In Figure 3.8(a), the incident diagonals for vertex v_0 are $\overline{v_0, v_2}$, $\overline{v_0, v_3}$, $\overline{v_0, v_4}$. These diagonals are checked for possible flipping. In order to apply the flipping opertation, the diagonal must satisfy the two rules stated above. Furthermore, there are two additional conditions that the candidate diagonal d must satisfy.

Condition 1: Diagonal d was not processed before.
Condition 2: Diagonal d cannot be a newly created diagonal, obtained by applying flipping operation.

In Figure 3.8(a), $<v_0, v_1, v_2, v_3>$ is the quadrilateral represented by diagonal $\overline{v_0, v_2}$. Quadrilateral $<v_0, v_1, v_2, v_3>$ satisfies both rules and both conditions, and hence diagonal $\overline{v_0, v_2}$ is flipped by replacing it with new diagonal $\overline{v_1, v_3}$. The diagonals that bound the flipped quadrilateral are marked processed. In our example, diagonal $\overline{v_0, v_3}$ is marked processed. The other diagonal $\overline{v_0, v_4}$ incident on v_0 is inspected. It satisfies Rule 3 but fails on Rule 4. Hence it is not flipped. Next, we move to vertex v_1. Newly created diagonal $\overline{v_1, v_3}$ is skipped (Condition 2 fails). Now, we move to vertex v_2, which has no diagonal incident on it as it was already flipped, see Figure 3.8(b). Next, vertex v_3 is skipped because diagonal $\overline{v_0, v_2}$ incident on it was already processed (Condition 1). For vertex v_4, diagonal $\overline{v_0, v_2}$ is skipped (Rule 3 fails). Diagonal $\overline{v_4, v_{23}}$ fails on Condition 2. Next, we move to vertex v_5. There are five diagonals incident on v_5, $\overline{v_5, v_{23}}$, $\overline{v_5, v_{21}}$, $\overline{v_5, v_{12}}$, $\overline{v_5, v_{11}}$ and $\overline{v_5, v_7}$. We process $\overline{v_5, v_{23}}$ and find out that it does not satisfy Condition 1. Diagonal $\overline{v_5, v_{21}}$ satisfies both rules and both conditions, therefore it is flipped with $\overline{v_{23}, v_{12}}$. In this way, diagonal $\overline{v_5, v_{12}}$ fails on Rule 4; diagonals $\overline{v_5, v_{11}}$ and $\overline{v_5, v_7}$ fail for Rule 3 and Rule 4 respectively. This kind of processing is done until all the vertices of polygon are scanned and thus all the diagonals are recorded and processed. Figure 3.8(b) represents the polygon after applying this algorithm to $T(P)$. Diagonals $\overline{v_1, v_3}$ and $\overline{v_{23}, v_{12}}$ are the newly created diagonals after the execution of Scan-Flip algorithm. It can be observed that triangulation dual can change due to flipping operation. A formal sketch of this algorithm is listed in Algorithm 3.2.

Theorem 3.2. *DCT problem for simple polygons can be solved in $O(n)$ time.*

Algorithm 3.2 Scan-Flip Algorithm

```
INPUT: A simple polygon $P$ with vertices $v_0, v_1, \ldots, v_{n-1}$
OUTPUT: Triangulated polygon $T(P)$ with increased number of even-degree
        vertices
Step 1: Obtain a triangulated polygon $T(P)$ of $P$ by applying any standard polygon
        triangulation algorithm
Step 2: $i = 0$;
Step 3: repeat
Step 4:     for each diagonal d incident on $v_i$ do
Step 5:         if ($Q(d)$ satisfy flippability rules and condition 1 and 2)
Step 6:             $T(P) = \text{flip}(T(P), d)$
Step 7:         end if
Step 8:     end for
Step 9:     $i = (i + 1) \bmod n$
Step 10: until ($i \neq 0$)
Step 11: Output $T(P)$
```

Proof Step1 can be done in $O(n)$ time by appealing to Chazzele's linear time polygon triangulation algorithm [4]. Each diagonal is examined at most two times for flippability test. By representing $T(P)$ obtained in Step1 in $DCEL$, the for-loop (Step4 - Step8) can be done in $O(deg_i)$ time, where deg_i is the degree of vertex v_i. This implies that the repeated-loop (Step 3 - Step 10) takes $O(n)$ time. Hence the entire algorithm takes $O(n)$ time. □

3.2.2 Development of Partitioning-AQT Algorithm

The quality of the solution obtained by applying Scan-Flip Algorithm (3.2) may not always yield a good solution. However, the quality of the solution obtained by applying AQT to convex polygons is such that almost all vertices have even degree. This motivates us to take a slightly different approach for developing the algorithm. A promising approach based on this observation is to first break the given simple polygon into convex pieces and apply AQT algorithm to each piece separately. Breaking a simple polygon into convex pieces is itself a very difficult problem. Some of the well-know algorithms for breaking a polygon into convex pieces are reported in [15]. For our investigation, we pick Hertel-Melhorn's algorithm [8] (HM-algorithm, for short) for convex decomposition. HM algorithm is simple to understand and implement.

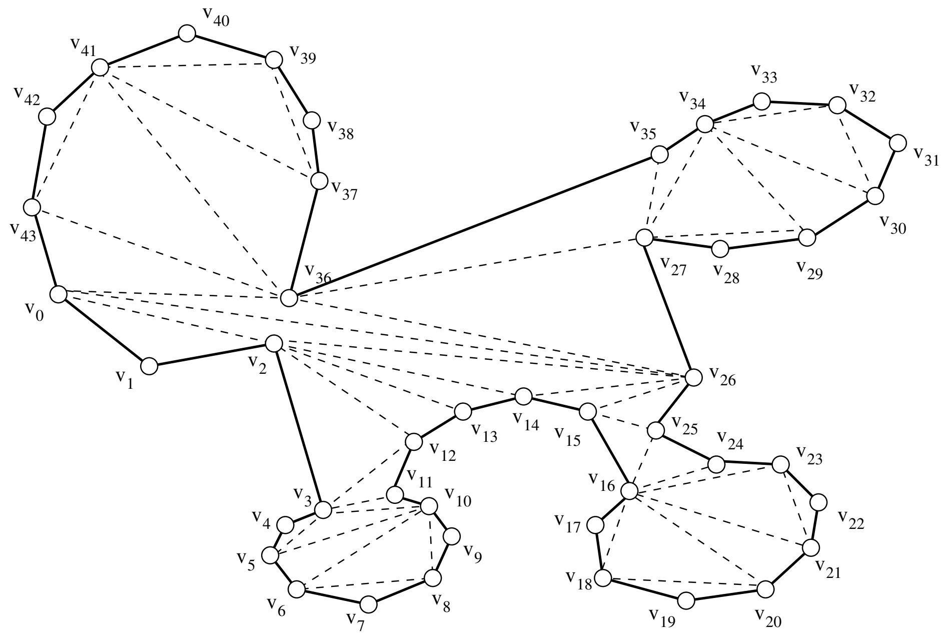


Figure 3.9: Even Degree Constrained Triangulation using Convex Components

An Additional merit of HM algorithm is the fact that it has a guaranteed bound for the quality of the resulting solution. In fact, HM algorithm obtains the solution which is no more than four times the optimal solution [15]. HM algorithm works by first triangulating the given polygon. In the triangulated polygon, the diagonals are carefully removed (removal of "non-essential" diagonals) to obtain the convex components.

The algorithm we propose that makes use of convex partitioning is called the *partitioning-AQT algorithm*. Let $c_0, c_1, \ldots, c_k$ be the k convex components when the input polygon P is partitioned by applying HM algorithm. The alternate AQT algorithm of Section 3.1 is applied on each component. The application of this algorithm is illustrated in Figure 3.9. This example shows that 36 out of 44 vertices are of even degree. In this figure there are four major convex components. We can see that if a larger number of convex pieces are generated in the convex decomposition then the quality of the solution obtained by applying partitioning-AQT algorithm improves significantly. When we apply Scan-Flip algorithm to the triangulated polygon obtained by applying partitioning-AQT algorithm, it is likely that only few number of

Algorithm 3.3 Partitioning-AQT Algorithm

```
INPUT: A simple polygon $P$ with vertices $v_0, v_1, \ldots, v_{\text{n-1}}$
OUTPUT: Triangulated polygon $T(P)$ of $P$ with increased number of even-degree vertices
Step 1: Obtain convex components of $P$ by using HM-Algorithm. Let $c_0, c_1, \ldots, c_{\text{k}}$
        be the resulting convex components
Step 2: for each component $c_{\text{i}}$ of $P$ do
Step 3:         $c_{\text{i}} = AQT(c_{\text{i}})$
Step 4: end for
Step 5: $T(P) = \phi$
Step 6: for $i = 1$ to $k$ do
Step 7:         $T(P) = T(P) \cup c_{\text{i}}$;
Step 8: end for
Step 9: Output $T(P)$
```

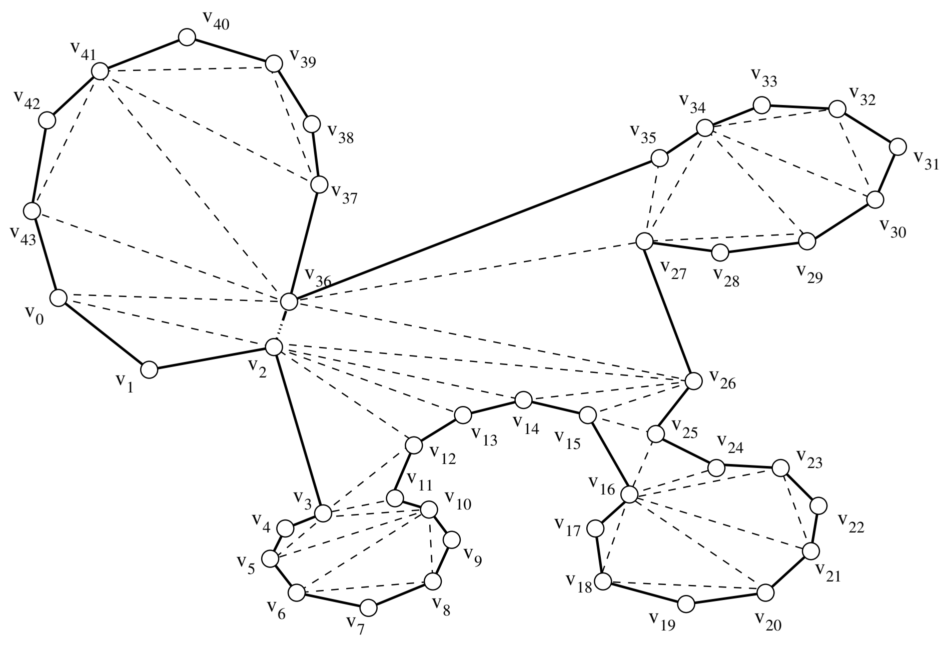


Figure 3.10: Scan-Flip Algorithm applied on triangulated polygon generated by Partitioning-*AQT* Algorithm

flips are possible. This is illustrated by figure 3.10 which has just one flip and whose resulting triangulated polygon has 40 out of 44 even-degree vertices. A formal sketch of partitioning-AQT algorithm is listed as Algorithm 3.3.

This algorithm does not always provide an effective even degree triangulation. It is highly dependent on the quality of convex decomposition provided by HM Algorithm.

Some shapes having a series of reflex chains can only be decomposed into thin convex polygons. Our algorithm is not effective for those shapes. Figure 3.11 illustrates a polygon for which this algorithm is not effective.

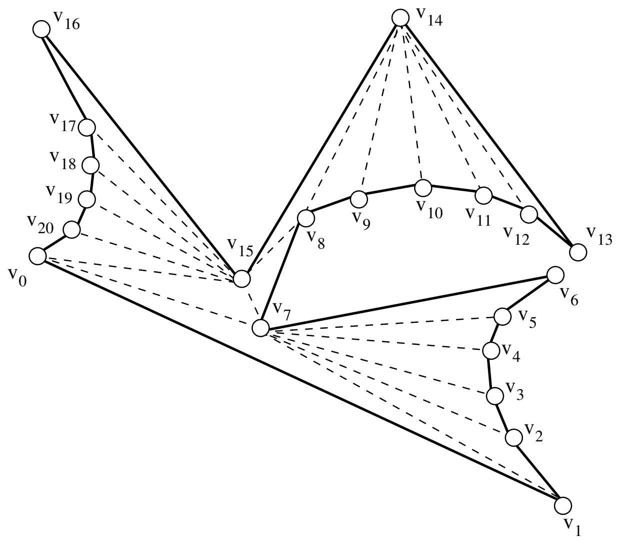
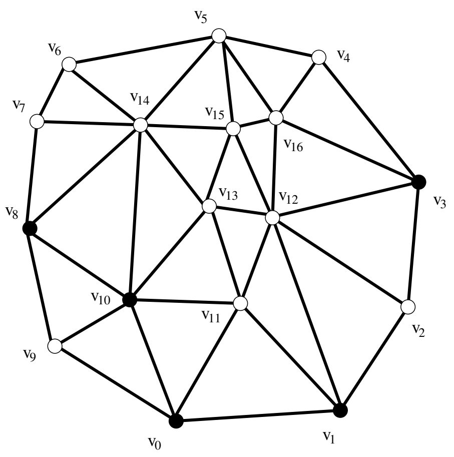


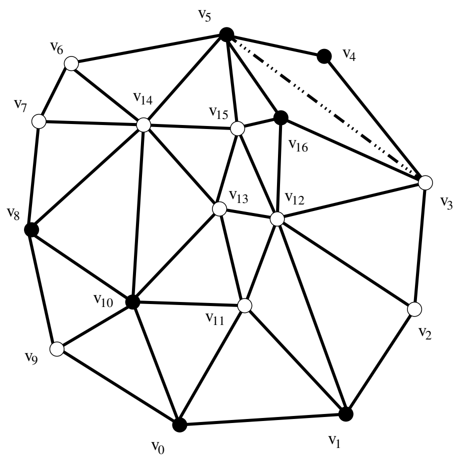
Figure 3.11: Ineffective Even Degree Constraint Triangulation using Convex Components

3.3 Degree-Constrained Triangulation for Points in Two Dimension

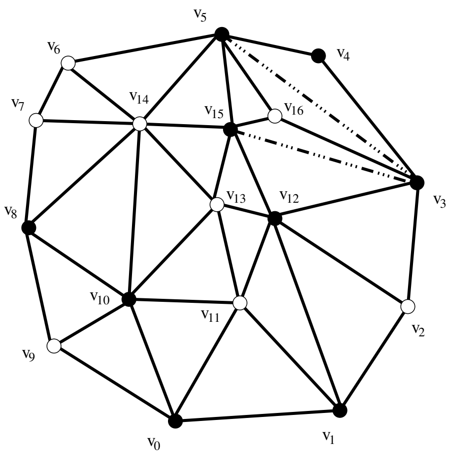
We now consider the development of efficient algorithm for solving DCT problem for a set of points in two dimensions. Our approach is to start an initial triangulation of the given input point site $S = \{p_0, p_1, \ldots, p_{\text{n-1}}\}$ and apply a sequence of feasible flipping to obtain the desired solution. Since Delaunay triangulation is one of the most widely used triangulations algorithm, we pick it as the initial triangulation. Let $DT(S)$ be the Delaunay triangulation of S obtained by using Fortune's plane sweep algorithm [5]. The edges of Delaunay triangulation can be distinguished into two kinds: (i) *external edges* are those that lie on the convex-hull boundary, and (ii) *internal edges* are the edges inside the convex-hull. Figure 3.12(a) is the Delaunay triangulation of 17 point sites. Among the 38 edges of $DT(S)$, 10 are external and the remaining 28 are internal edges. The algorithm we propose processes each internal edge, one at a time, to apply a flipping operation. If a selected internal edge e_{i} satisfies flipping conditions then the triangles of quadrilateral $Q_{\mathrm{e_i}}$ are modified by applying a flipping operation. It is remarked here that the order in which the internal edges are processed exactly depends on the order in which the Delaunay triangulation outputs the edges of the triangulation.



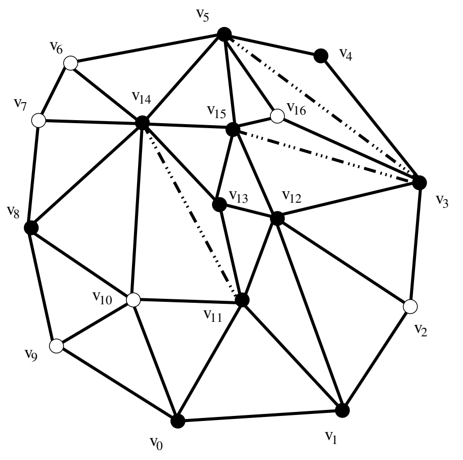
(a) Original Triangulation



(b) Triangulation after first flip



(c) Triangulation after second flip



(d) Triangulation after third flip

Figure 3.12: Scan-Flip Algorithm applied to triangulated point sites generated by Delaunay Triangulation Algorithm

We illustrate the application of flipping operations on a running example as shown in Figure 3.12. The algorithm picks $e_{\mathrm{i}} = <v_4, v_{16}>$ as the first internal edge to process. The quadrilateral $Q_{\mathrm{e_i}}$ corresponding to edge e_{i} is convex and it has three odd-degree vertices. (In the figure, vertices with odd-degree are drawn white and those with even-degree are drawn black. In the original triangulation there are 5 even-degree vertices and the remaining 12 are odd-degree vertices.)

The algorithm thus finds $Q_{\mathrm{e_i}}$ fit for flips. Figure 3.12(b) is the triangulation after one flip. In this triangulation the number of even-degree vertices increases by 2 to a total of 7. Next, the algorithm checks internal edge $e_2 = <v_3, v_{16}>$ and finds that the corresponding quadrilateral $Q_{\mathrm{e_2}}$ is not convex and it is rejected for flipping. The third internal edge $e_3 = <v_{12}, v_{16}>$ is appropriate for flipping and results in the triangulation shown in Figure 3.12(c), where the total number of even-degree vertices increases to 9. This is continued and the final triangulation is shown in Figure 3.12(d) where the total number of even-degree vertices is 11.

A formal sketch of the algorithm is listed as Algorithm 3.4.

Theorem 3.3. Algorithm 3.4 can be executed in $O(nlogn)$ time.

Proof Step 1 can be done in $O(nlogn)$ time by using Fortune's plane-sweep algorithm. The triangulation given by Fortune algorithm can be implemented using a doubly connected edge list structure [13] so that faces, edges and vertices can be accessed quickly. Interior edges from $T(P)$ can be accessed in $O(n)$ time and hence Step 2 takes $O(n)$ time. The while loop executes at most n time. Validity of flippability rules, and Condition 1 and Condition 2 can be checked in $O(1)$ time by using the dcel data structure. Hence each execution of the body of while loop takes $O(1)$ time implying that Step 3 - Step 8 takes $O(n)$ time. Thus the time for the entire algorithm is $O(nlogn)$. □

Algorithm 3.4 Increased Even-Degree Triangulation of Point Sites

```
INPUT: Set of point sites $S = \{p_0, p_1, \ldots, p_{\text{n-1}}\}$
OUTPUT: Triangulated point sites with increased number of even-degree point sites
Step 1: Obtain triangulation of $S$ ($T(S)$) using delaunay triangulation algorithm
Step 2: $\alpha \leftarrow$ Interior edges of $T(S)$
Step 3: while($\alpha$ is not empty) do
Step 4:         Find corresponding quadrilateral $Q(e)$ w.r.t interior edge $e$ from $\alpha$
Step 5:         if ($Q(e)$ satisfy flippability rules and condition 1 and condition 2)
Step 6:             $T(S) = \text{flip}(T(S), d)$
Step 7:         end if
Step 8: end while
Step 9: Output $T(S)$
```

CHAPTER 4

IMPLEMENTATION AND OBSERVATIONS

This chapter describes the implementation and experimental study of the proposed algorithms for solving *DCT* problems. We have used Java programming language for implementing the algorithms. The implemented algorithms include

1. Triangulation of monotone polygons
2. Plane-Sweep algorithm for monotone partitioning
3. Hertel-Melhorn's convex decomposition algorithm
4. Delaunay triangualation of point sites
5. Even-degree triangulation of convex polygons.
6. Scan-Flip algorithm for solving *DCT* problems.

4.1 Interface Overview

A graphical user interface (GUI) is designed so that the users can execute selected algorithms easily and intuitively. The GUI contains serveral components that include buttons, textArea, dropdown menus, canvas, menubar, check boxes and radio buttons. The input data in the form of co-ordinates of nodes can be read from files or manually by mouse click. The implementation allows the user to generate upto 1000 nodes. The location of generated nodes can be visually altered by mouse drag. Furthermore, the implementation allows the random generation of nodes. Random generation of nodes is done by randomly generating x and y co-ordinates in a range that corresponds to a pixel size of the canvas. The maximum pixel size of the display canvas in the implementation is $1000X700$ and can be altered to a smaller size.

Table 4.1: Menu bar description

	Item Name	Functional Description
1	Open	Brings up a file selection panel, user can choose a pregenerated graph file
2	Save	Brings up a file save panel, user can save a new generated file or replace an existing file, Naming convention is maintained to distinguish files for two application which are <*filename*>_polygonbased and <*filename*>_pointbased
3	Exit	Brings up a confirmation dialog box, which if selected, closes the application

4.2 Structure of the Interface

The initial interface frame on which components are placed is imported from Java.swing API JFrame. Four components are placed on the initial frame which are (*i*) Menu bar, (*ii*) Central panel and (*iii*) East panel. The layout of these components on the frame is as shown in Figure 4.1. Menu Bar has three basic items: open, save and exit selections. Table 4.1 lists the functionalities of these items. Center panel contains main display area that allows user to manually draw, edit, split, delete nodes or display nodes that are read from a file. Mouse coordinates are shown in the upper left corner to help navigate or draw objects within the center area. The east panel is divided into three parts. The Top part of the east panel is the checkbox grid, which contains checkboxes labeled draw, edit, split and delete. These check box functions are used for setting a mode for drawing in the canvas. The Middle Part of the east panel is the button grid, which has buttons labeled "clear canvas", "perform flip" and "random points" in Point sites based applications and "clear canvas", "convex polygon even triangulation", "convex polygon odd triangulation", "monotone polygon triangulation" and "perform flip" in Polygon based applications. Clicking of these buttons executes respective algorithms that are labeled with them. Tables 4.2 and Table 4.3 describe the functionalities of checkbox grid and button grid, respectively. The Bottom part of east panel is the text area, where co-ordinates of the displayed nodes are listed in order. Figure 4.2, Figure 4.3 and Figure 4.4 illustrate the starting interface, layout of point sites, and layout of polygon based applications respectively

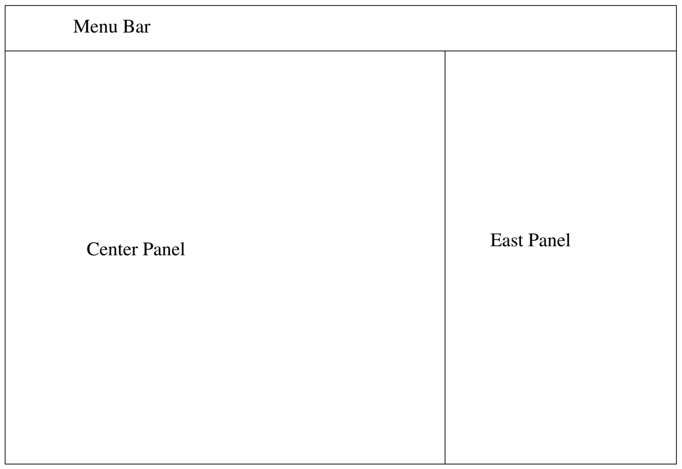


Figure 4.1: GUI Layout

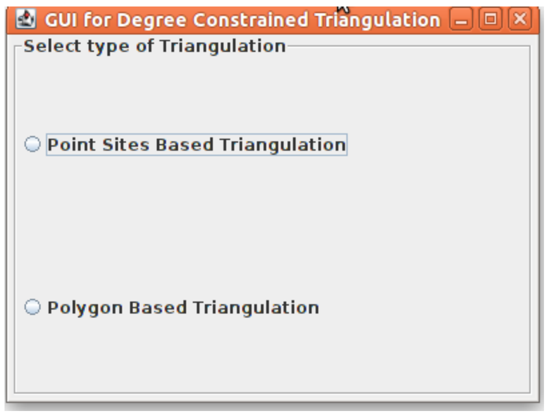

GUI for Degree Constrained Triangulation
Select type of Triangulation
Point Sites Based Triangulation
Polygon Based Triangulation

Figure 4.2: Starting Interface

Table 4.2: East Panel Checkboxes description

	Item Name	Functional Description
1	Draw Vertex	Adds a vertex v_{n} to a polygon or point sites $v_0, \dots, v_{\text{n-1}}$ In point sites application, delaunay triangulation is calculated with each drawn vertex
2	Delete Vertex	Deletes clicked vertex of a polygon by updating the values to the connecting vertices
3	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
4	Split Vertex	Splits the closest edge into two parts by generating new vertex to the closest edge

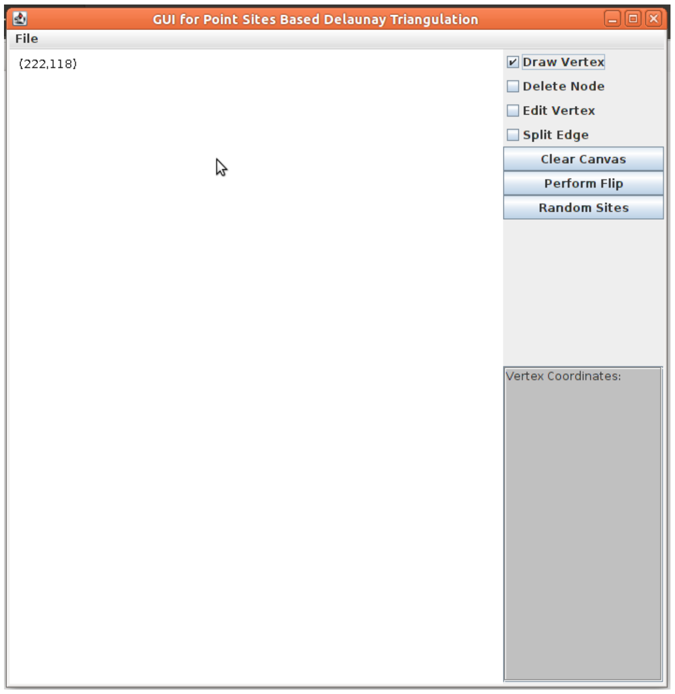


Figure 4.3: Layout for Point Sites Based Triangulation Applications

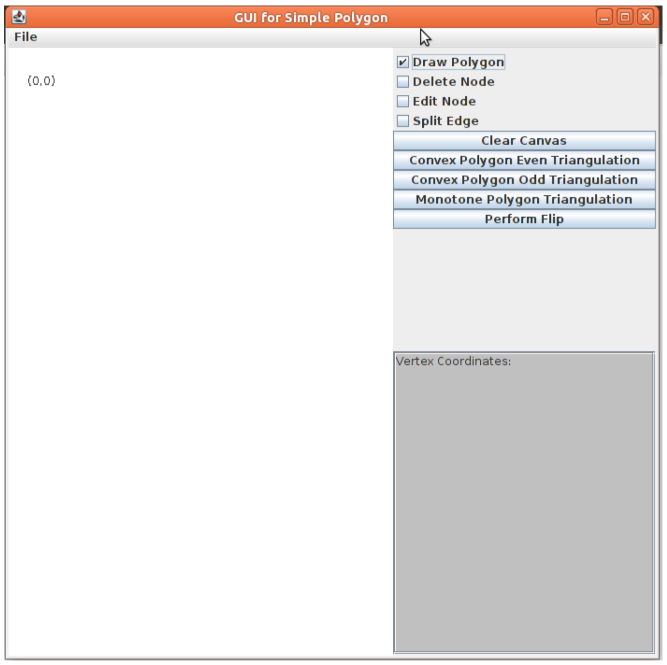
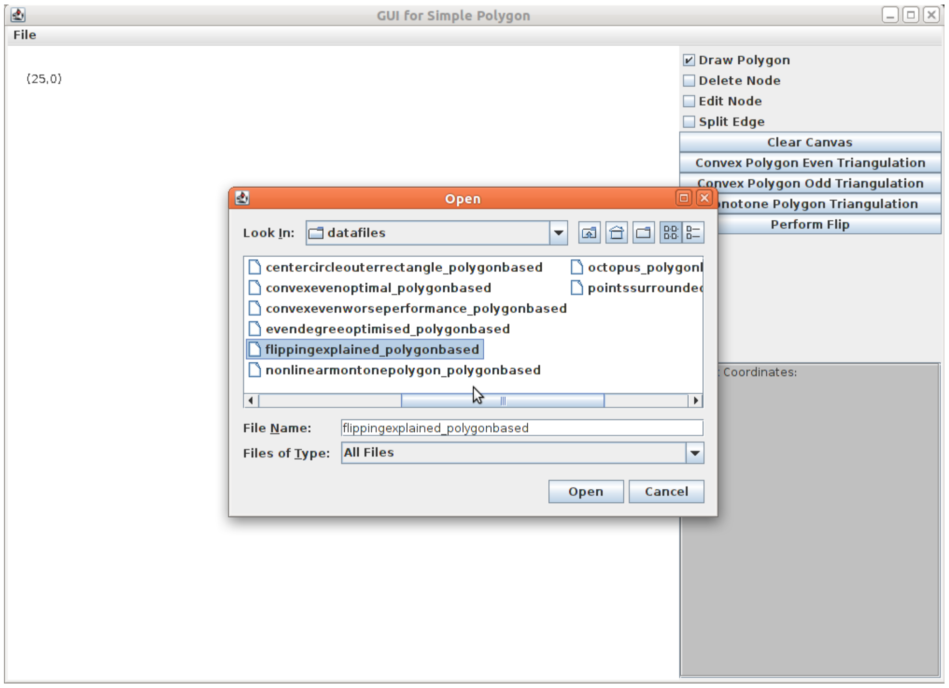


Figure 4.4: Layout for Polygon Based Triangulation Applications

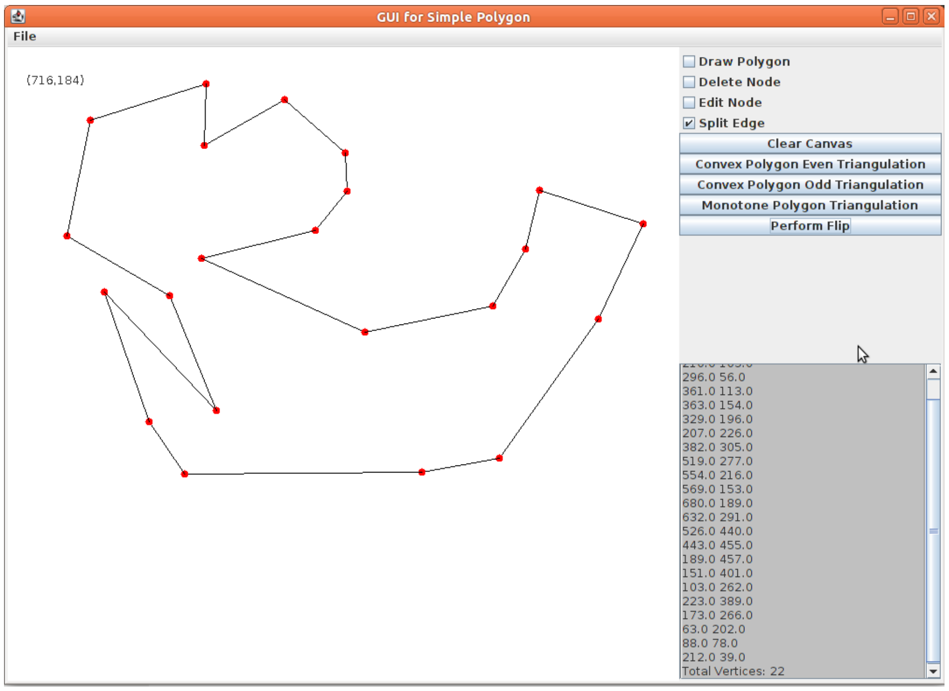
Table 4.3: East Panel Button description

Polygon Based Triangulation		
1	Clear Canvas	Clears the canvas, flushing out all the objects created
2	Convex Polygon Even Triangulation	Breaks down a simple polygon to a number of convex pieces and applies even triangulation algorithm to them
3	Convex Polygon Odd Triangulation	Breaks down a simple polygon to a number of convex pieces and applies odd triangulation algorithm to them
4	Monotone Polygon Triangulation	Breaks down a simple polygon to a number of monotone pieces and applies monotone triangulation algorithm to them
5	Perform Flip	Apply flipping algorithm to the triangulated simple polygon
Point Sites Based Triangulation		
1	Clear Canvas	Clears the canvas, flushing out all the objects created
2	Perform Flip	Apply flipping algorithm to the triangulated simple polygon
3	Random Sites	Pops up a dialog input box to set the number of points say n generates n number of random point sites


GUI for Simple Polygon
File
(25,0)
Draw Polygon
Delete Node
Edit Node
Split Edge
Clear Canvas
Convex Polygon Even Triangulation
Convex Polygon Odd Triangulation
notone Polygon Triangulation
Perform Flip
Coordinates:
Open
Look In:
datafiles
centercircleouterrectangle_polygonbased
convexevenoptimal_polygonbased
convexevenworseperformance_polygonbased
evendegreeoptimised_polygonbased
flippingexplained_polygonbased
nonlinearmontonepolygon_polygonbased
octopus_polygon
pointssurrounde
Open
Cancel

File Name: flippingexplained_polygonbased
Files of Type: All Files

Figure 4.5: File Open Dialog Window



```
296.0 56.0
361.0 113.0
363.0 154.0
329.0 196.0
207.0 226.0
382.0 305.0
519.0 277.0
554.0 216.0
569.0 153.0
680.0 189.0
632.0 291.0
526.0 440.0
443.0 455.0
189.0 457.0
151.0 401.0
103.0 262.0
223.0 389.0
173.0 266.0
63.0 202.0
88.0 78.0
212.0 39.0
Total Vertices: 22
```

Figure 4.6: Polygon Frame

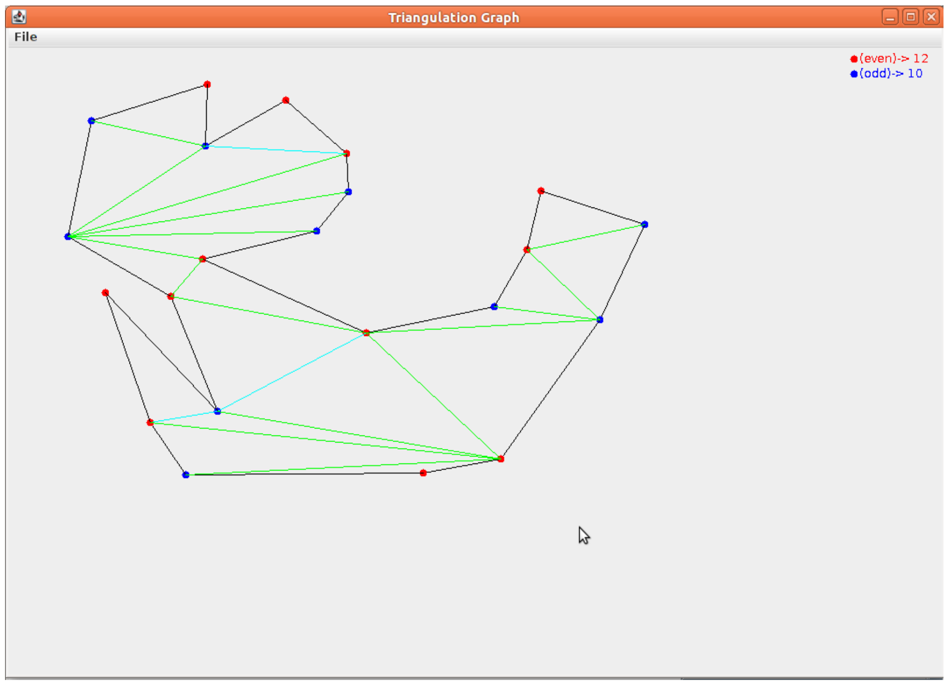


Figure 4.7: Snap-shot of polygon triangulation

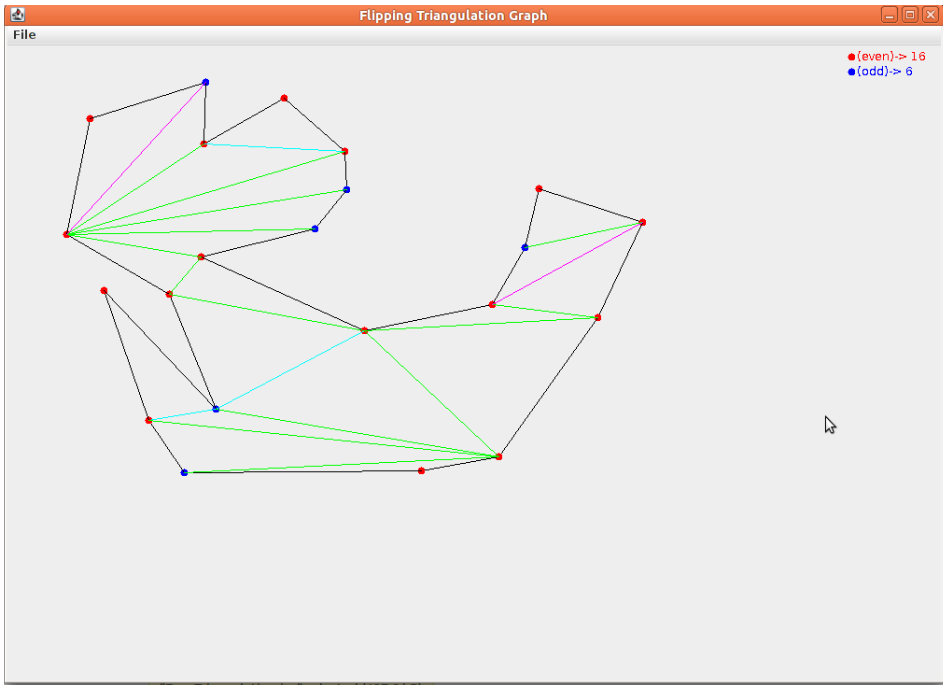
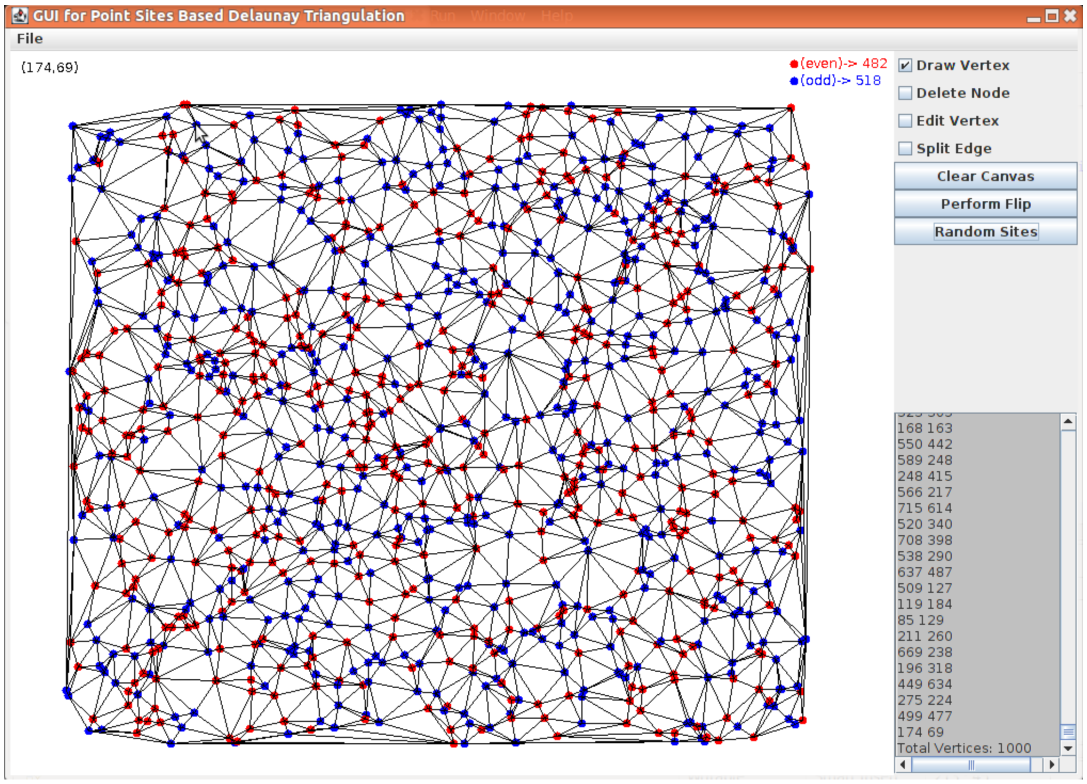


Figure 4.8: Snap-shot of polygon triangulation after flipping operation

4.3 Generating Polygons and Point Sites

There are two approaches to generating polygons or point sites in the application. The first one is to load a file (Figure 4.5), which has informations regarding the polygon or point sites for the application. The other way is to make sure that the application is in draw mode and create vertices by clicking in the desired place within the central panel. Once the object is drawn, it can be saved to a file using the save option in the file menu. However, these two applications behave differently while drawing. The point sites application calculates Delaunay triangulation with each added point (after initial three points) as shown in Figure 4.9. On the other hand, the polygon application simply draws the polygon connecting those drawn points as shown in Figure 4.6. When the user clicks any of the triangulation buttons in the polygon application, a new frame is displayed showing the resulting triangulation. There are certain graphical features that have been used to represent triangulated point sites or polygons. While even degree vertices are drawn as "red" dots, odd degree vertices are displayed with "blue" color. On the top right corner of the frame,



```
168 163
550 442
589 248
248 415
566 217
715 614
520 340
708 398
538 290
637 487
509 127
119 184
85 129
211 260
669 238
196 318
449 634
275 224
499 477
174 69
Total Vertices: 1000
```

Figure 4.9: Snap-shot of initial Delaunay triangulation of 1000 point sites

the count of odd/even degree vertices is shown. Furthermore, all the interior edges of point sites and diagonals of polygon are colored green. The diagonal determined by the decomposition algorithm for breaking into component polygons are displayed in "cyan" color. Snap-shots of a triangulated polygon and an initial Delaunay triangulation of point sites are shown in Figure 4.7 and Figure 4.9 respectively. Once we have the triangulation of polygon or point sites, the flipping operation can be started by clicking corresponding buttons. The flipped diagonals are colored "magenta" to make them distinctly visible. Figure 4.8 shows the result after applying flipping operations. Similarly, Figure 4.10 is the snap-shot of triangulation of point sites after applying flipping.

The class interface diagrams in UML style used for implementing the algorithms are shown in Figure 4.11 and Figure 4.12. It is remarked that this is only a partial list.

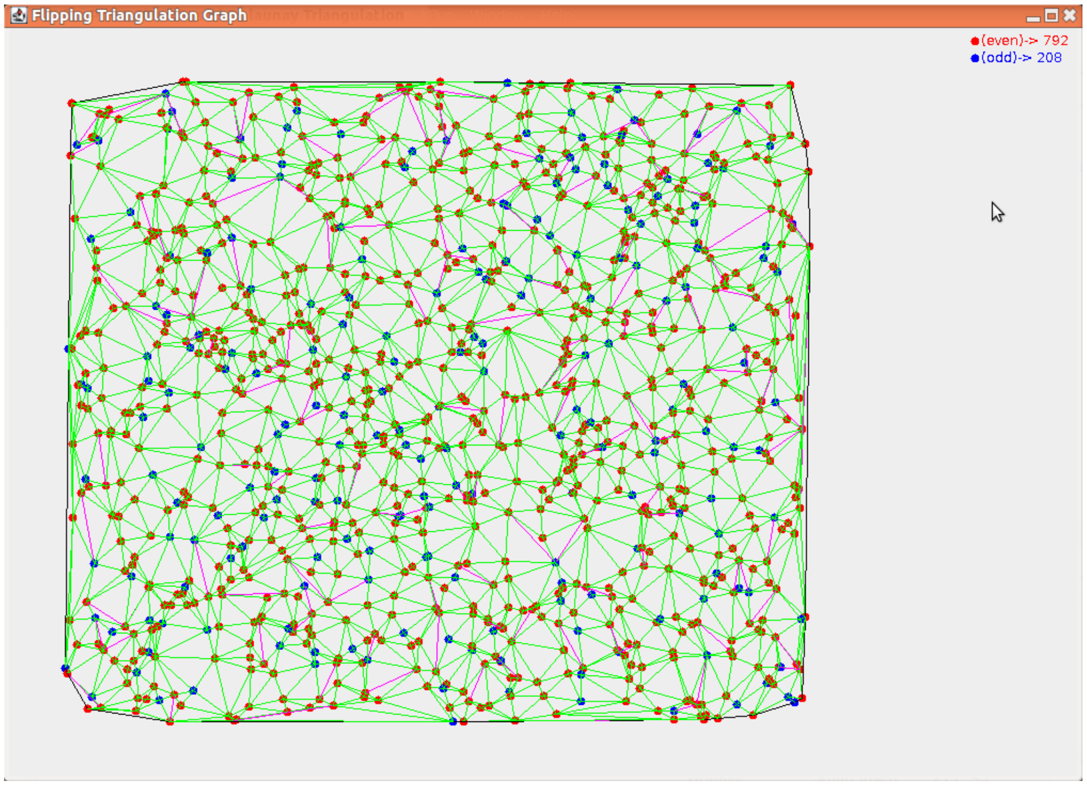


Figure 4.10: Snap-shot of triangulated point sites after flipping operation

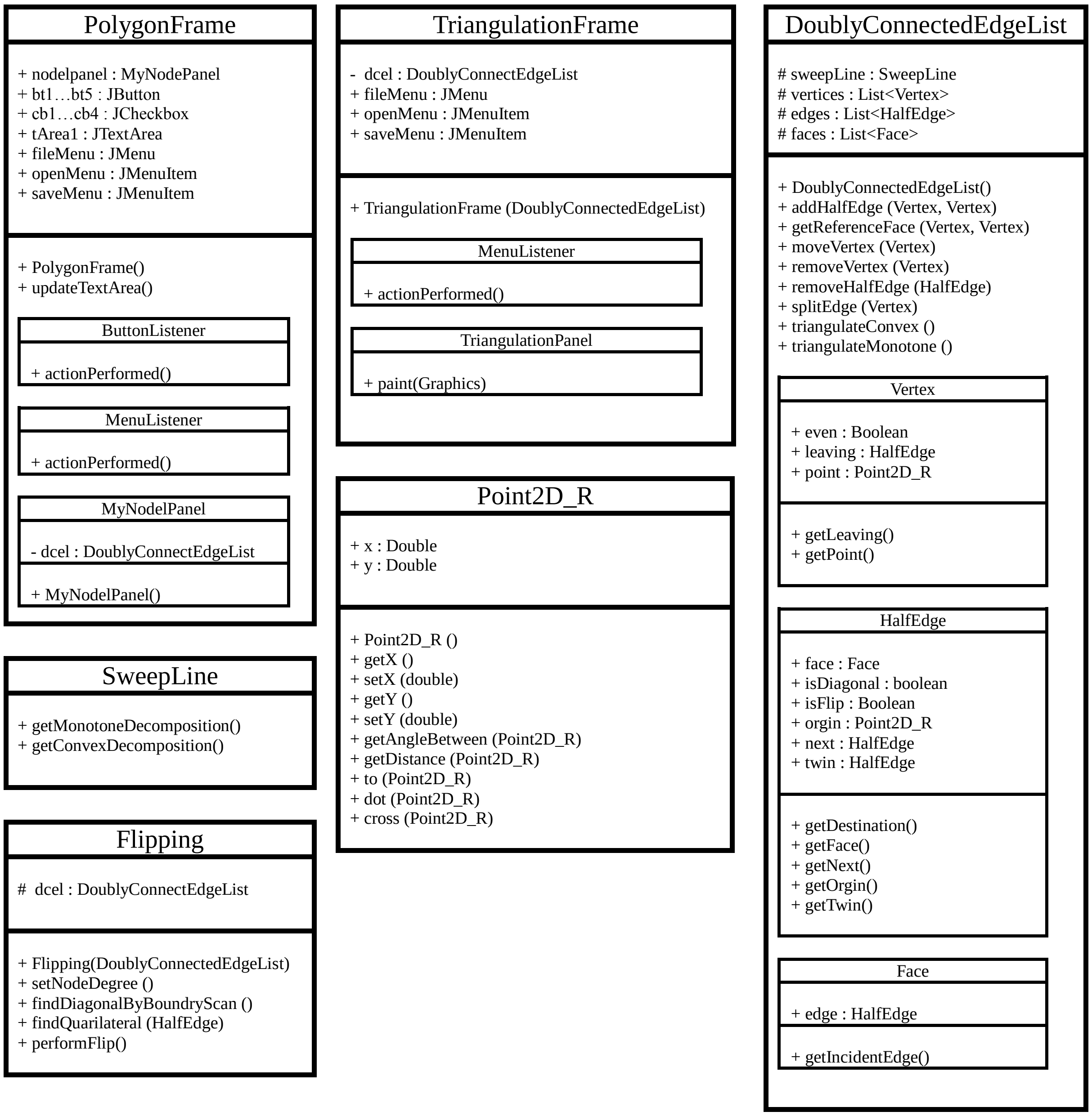


Figure 4.11: Polygon based Application Class Diagram

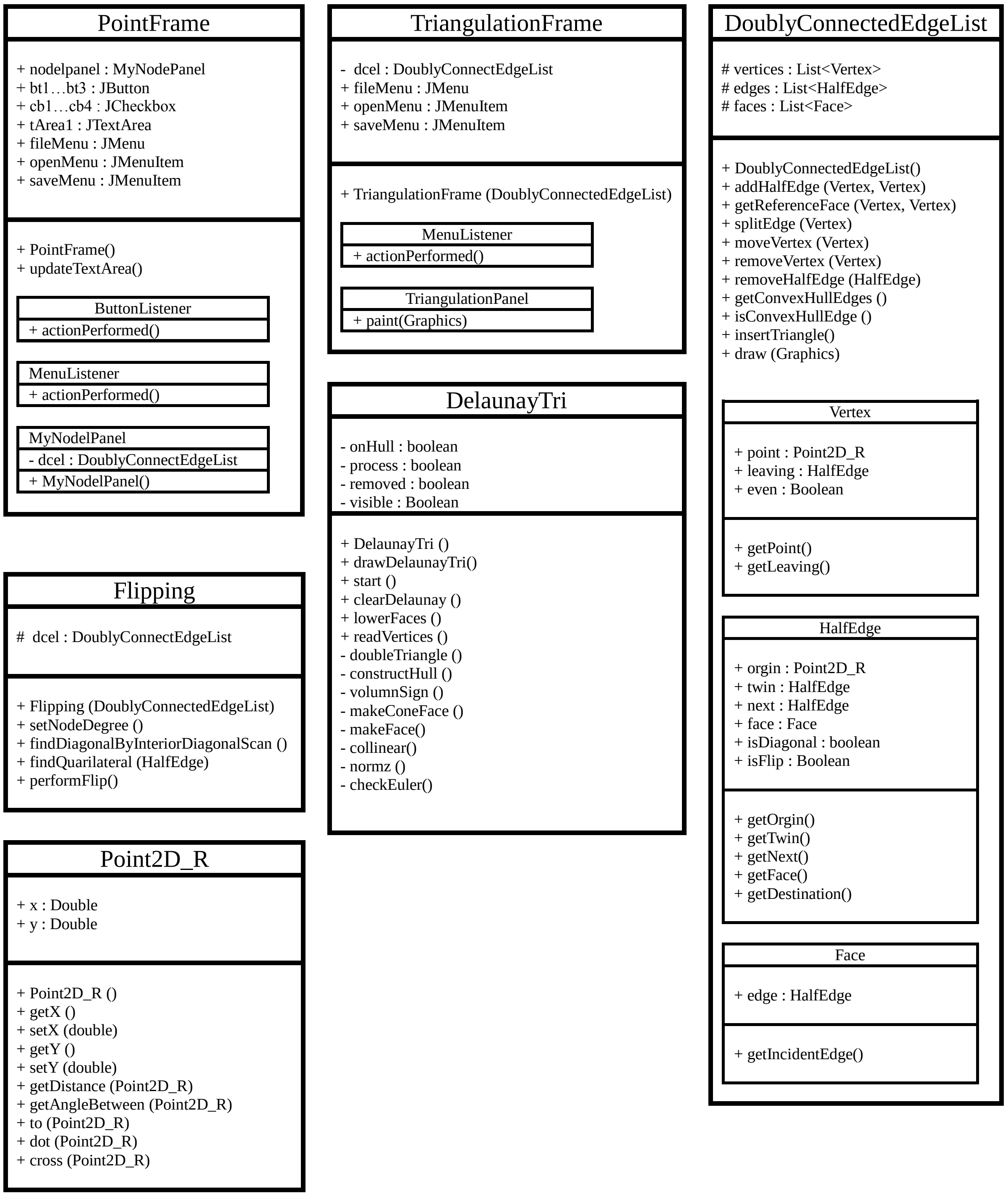


Figure 4.12: Point Sites Application Class Diagram

4.4 Observations

We conducted several experiments on the performance of the scan-flip algorithm for solving DCT problem on point sites in two dimensions. The initial triangulation of the input point sites was the Delaunay triangulation. Our implementation of Delaunay triangulation was done by using some Java classes available in [15]. Our program converts the Delaunay triangulation into Doubly Connected Edge List ($DCEL$) data structures so that the faces, vertices, and edges can be traversed quickly. Initial Delaunay triangulation (DT) was done on randomly generated point sites. The scan-flip algorithm was then applied to DT to increase the number of even-degree vertices. The algorithm was executed on 5 sets of randomly generated 10 subsets of point sites. Each of the 10 subsets contain a number of points starting from 50 up to 500 in increments of 50. The number of vertices with even degree was recorded for the initial triangulation and the triangulation after applying scan-flip algorithm. These results are tabulated in Table 4.4. An inspection of the results in the table reveals that the percentage increase in the number of even-degree vertices ranges from as little as 31% to as much as 111%. Furthermore, it seems that the percentage increase does not depend on the number of the number of input points.

It is not easy to conduct such an experiment for triangulation of polygons. This is due to the fact that there is no accepted algorithm for generating random polygons [1]. The quality of the result produced by the scan-flip algorithm for polygons depends on the initial triangulation of the polygon. As expected, the quality of the generated solution is substantially increased if the polygon has a fewer number of convex components as in Figure 3.9.

Table 4.4: Observation of flipping affect on different sets of point sites

Number of Points	Number of Odd Degree Vertices (n_o)		Number of Even Degree Vertices (n_e)		% increase of (n_e)
	Before	After	Before	After	
Set 1					
50	24	12	26	38	46.15%
100	48	28	52	72	38.46%
150	80	30	70	120	71.43%
200	98	46	102	154	50.98%
250	130	54	120	196	63.33%
300	162	70	138	230	66.67%
350	174	78	176	272	54.55%
400	200	82	200	318	59.00%
450	238	100	212	350	65.09%
500	260	102	240	398	65.83%
Set 2					
50	20	10	30	40	33.33%
100	50	18	50	82	64.00%
150	82	32	68	118	73.53%
200	104	52	96	148	54.17%
250	128	56	122	194	59.02%
300	152	56	148	244	64.86%
350	174	74	176	276	56.82%
400	190	78	210	322	53.33%
450	222	95	228	355	55.70%
500	268	114	232	386	66.38%
Set 3					
50	18	8	32	42	31.25%
100	64	24	36	76	111.11%
150	80	32	70	118	68.57%
200	112	40	88	160	81.82%
250	116	56	134	194	44.78%
300	158	72	142	228	60.56%
350	174	68	176	282	60.23%
400	220	94	180	306	70.00%
450	226	94	224	356	58.93%
500	254	100	246	400	62.60%
Set 4					
50	26	14	24	36	50.00%
100	46	23	54	77	42.59%
150	68	28	82	122	48.78%
200	98	42	102	158	54.90%
250	124	52	126	198	57.14%
300	156	58	144	242	68.06%
350	180	78	170	272	60.00%
400	198	94	202	306	51.49%
450	222	98	228	352	54.39%
500	260	108	240	392	63.33%
Set 5					
50	24	12	26	38	46.15%
100	50	20	50	80	60.00%
150	74	28	76	122	60.53%
200	104	46	96	154	60.42%
250	144	58	106	192	81.13%
300	168	72	132	228	72.73%
350	170	68	180	282	56.67%
400	198	82	202	318	57.43%
450	230	96	220	354	60.91%
500	250	106	250	394	57.60%

CHAPTER 5

CONCLUSION AND FUTURE WORK

We presented a critical review of the existing algorithms for triangulating simple polygons and point sites. We formulated a variation of triangulation problems called Degree Constrained Triangulation (DCT). To solve DCT problems for convex polygons we presented an algorithm called AQT algorithm. The quality of the solution generated by applying AQT algorithm to convex polygons is near-perfect in the sense that almost all vertices in the triangulation are of even degree. We presented a formal proof that no simple polygon admits odd-degree triangulation.

We proposed two algorithms for solving DCT for simple polygons. The first algorithm, called scan-flip polygon triangulation algorithm, solves DCT problems by applying flip operations on a carefully selected initial triangulated polygon. The time complexity of scan-flip algorithm is $O(n)$. The second algorithm called Partitioning $AQT(P-AQT)$ algorithm, solves the DCT problem by using a convex-decomposition tool from computational geometry. The quality of the solution generated by $P-AQT$ algorithm depends on the quality of the convex-decomposition algorithm.

For solving DCT problems for set of points in $2D$, we use the well known Delaunay triangulation as the initial triangulation to apply scan-flip operations. The time complexity for the scan-flip algorithms for point sets in $2D$ is $O(nlogn)$. Scan-flip algorithm for point sites and simple polygons were implemented for testing the performance of proposed algorithms. Experimental results show that the algorithm for points sites is fairly effective in generating good quality solutions for DCT problems.

We could not perform an extensive experimental investigation of the scan-flip algorithm for polygons. If we could develop a good algorithm for generating random polygons then it would be feasible to perform a serious experimental investigation. Our future work is planned in this direction.

BIBLIOGRAPHY

[1] Thomas Auer and Martin Held. Heuristics for the generation of random polygons. In Frank Fiala, Evangelos Kranakis, and Jrg-Rdiger Sack, editors, *Proceedings of the 8th Canadian Conference on Computational Geometry, Carleton University, Ottawa, Canada, August 12-15, 1996*, pages 38–43. Carleton University Press, 1996.

[2] Marshall W. Bern, Herbert Edelsbrunner, David Eppstein, Sandra L. Mitchell, and Tiow Seng Tan. Edge insertion for optimal triangulations. *Discrete & Computational Geometry*, 10:47–65, 1993.

[3] Christoph Burnikel, Kurt Mehlhorn, and Stefan Schirra. On degeneracy in geometric computations. In *Proceedings of the fifth annual ACM-SIAM symposium on Discrete algorithms*, SODA '94, pages 16–23, Philadelphia, PA, USA, 1994. Society for Industrial and Applied Mathematics.

[4] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete Comput. Geom. 6*, pages 485–524, 1991.

[5] Steven Fortune. A sweepline algorithm for voronoi diagrams. *Algorithmica*, 2:153–174, 1987.

[6] Leonidas Guibas and Jorge Stolfi. Primitives for the manipulation of general subdivisions and the computation of voronoi. *ACM Trans. Graph.*, 4(2):74–123, April 1985.

[7] Leonidas J. Guibas, Donald E. Knuth, and Micha Sharir. Randomized incremental construction of delaunay and voronoi diagrams. *Algorithmica*, 7(4):381–413, 1992.

[8] Stefan Hertel and Kurt Mehlhorn. Fast triangulation of simple polygons. In *Proceedings of the 1983 International FCT-Conference on Fundamentals of Computation Theory*, pages 207–218, London, UK, UK, 1983. Springer-Verlag.

[9] F. Hurtado, M. Noy, and J. Urrutia. Flipping edges in triangulations. In *Proceedings of the twelfth annual symposium on Computational geometry*, SCG '96, pages 214–223, New York, NY, USA, 1996. ACM.

[10] Ferran Hurtado and Marc Noy. The graph of triangulations of a convex polygon. In *Proceedings of the twelfth annual symposium on Computational geometry*, SCG '96, pages 407–408, New York, NY, USA, 1996. ACM.

[11] C. L. Lawson. Software for c1 surface interpolation. *Mathematical Software III*, pages 161–194, 1977.

[12] G. H. Meisters. Polygons have ears. *American Mathematical Monthly*, 82:648651, 1975.

[13] David E Muller and Franco P Preparata. Finding the intersection of two convex polyhedra. *Theoretical Computer Science*, 7(2):217–236, 1978.

[14] Joseph O'Rourke. *Art gallery theorems and algorithms.* Oxford University Press, Inc., New York, NY, USA, 1987.

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

[16] E. Osherovich and A. M. Bruckstein. All triangulations are reachable via sequences of edge-flips: an elementary proof. *Comput. Aided Geom. Des.*, 25(3):157–161, March 2008.

[17] Jorge Urrutia, Canek Pelez, and Adriana Ramrez-Viguer. Triangulations with many points of even degree. In *Proceedings of the 22nd Annual Canadian Conference on Computational Geometry, Winnipeg, Manitoba, Canada, August 9-11, 2010*, pages 103–106, 2010.

VITA

Graduate College
University of Nevada, Las Vegas

Roshan Gyawali

Degrees:
Bachelor of Computer Enginnering 2008
Institute of Engineering, Pulchowk Campus, Tribhuvan University

Thesis Title: Degree Constrained Triangulation

Thesis Examination Committee:
Chairperson, Dr. Laxmi Gewali, Ph.D.
Committee Member, Dr. Ajoy K. Datta, Ph.D.
Committee Member, Dr. John Minor, Ph.D.
Graduate Faculty Representative, Dr. Rama Venkat, Ph.D.