8-1-2013

Stability Aware Delaunay Refinement

Bishal Acharya
University of Nevada, Las Vegas, bishalacharya@gmail.com

STABILITY AWARE DELAUNAY REFINEMENT

By

Bishal Acharya

Bachelor of Computer Engineering, Kathmandu Engineering College

Tribhuvan University, Nepal

2006

A thesis submitted in partial fulfillment

of the requirements for the

Master of Science in Computer Science

Department of Computer Science

Howard R. Hughes College of Engineering

The Graduate College

University of Nevada, Las Vegas

August 2013

THE GRADUATE COLLEGE

We recommend the thesis prepared under our supervision by

Bishal Acharya

entitled

Stability Aware Delaunay Refinement

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

Master of Science in Computer Science
Department of Computer Science

Laxmi P. Gewali, Committee Chair

John T. Minor, Committee Member

Evangelos Yfantis, Committee Member

Rama Venkat, Graduate College Representative

Thomas Piechota, Ph. D., Interim Vice President for Research and Graduate Studies and Dean of the Graduate College

August 2013

ABSTRACT

Stability Aware Delaunay Refinement

by

Bishal Acharya

Dr. Laxmi Gewali, Examination Committee Chair

Professor of Computer Science

University of Nevada, Las Vegas

Good quality meshes are extensively used for finding approximate solutions for partial differential equations for fluid flow in two dimensional surfaces. We present an overview of existing algorithms for refinement and generation of triangular meshes. We introduce the concept of *node stability* in the refinement of Delaunay triangulation. We present two algorithms for generating stable refinement of Delaunay triangulation. We also present an experimental investigation of a triangulation refinement algorithm based on the location of the center of gravity and the location of the center of circumcircle. The results show that the center of gravity based refinement is more effective in refining interior nodes for a given distribution of nodes in two dimensions.

ACKNOWLEDGEMENTS

I would like to express my gratitude to my supervisor, *Dr. Laxmi Gewali* for helping me complete this thesis. I would also like to thank *Dr. Ajoy Datta*; without him I would not have been at UNLV. I would also like to thank *Dr. John Minor*, *Dr. Rama Venkat*, and *Dr. Evangelos Yfantis* for providing me with the honor being on my thesis committee and reviewing my thesis. Furthermore, my courteous appreciation goes to *Umang Amatya* for his help in the preliminary stage of my thesis and *Ravi Sharma* for reviewing my thesis. Most important of all, I would like to thank *Rachana Acharya* for her motivations which helped me find a purpose to finish my thesis. Finally, I would like to thank my parents for being by my side at every stage of my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

Partitioning a two dimensional domain into a collection of simpler shapes is called meshing. Each of the simple shapes in the mesh are called mesh elements. Widely used examples of mesh elements are triangles, rectangles, and hexagons. The two dimensional domain which is to be converted into a mesh is usually modelled by a polygon with holes. Meshing is also done in three dimensional domain where mesh elements are tetrahedrons, cubes, or a parallelepiped. Mesh generation has important applications in finite element analysis [14,17], where it is used to obtain an approximate solution to a partial differential equation for fluid flow in 2D surfaces. The quality of the generated solution increases if the aspect ratio of the mesh elements is close to 1. It is noted here that the aspect ratio of a mesh element is the ratio of the length of the sides of the smallest rectangle enclosing the element. In calculating the aspect ratio *(w/b)*, width w is the shorter side of the rectangle.

Algorithmic tools from computational geometry have been used extensively for generating quality meshes [8,13]. The most widely used geometric algorithm for generating a triangular mesh is the Delaunay triangulation algorithm obtained by using Fortune's algorithm [3].

In many applications, it is desirable to refine the Delaunay mesh by introducing new vertices. One major algorithmic work for refining Delaunay triangulation is Ruppert's algorithm [10].

In this thesis, we consider the problem of mesh refinement by introducing the concept of *node stability*. A node in a Delaunay triangulation is termed *stable* if a slight change

in its position does not result in a change in the connectivity of the nodes. We use the concept of *free region* introduced in [9,6] to develop a Delaunay refinement algorithm that tends to increase the stability of nodes.

The thesis is organized as follows : In Chapter 2, we present a brief overview of triangulation refinement algorithms reported in the algorithmic literature. In Chapter 3, we present the main contributions of the thesis. In particular, we consider two approaches for refining Delaunay triangulation. One is based on using the *largest empty circle* [4] and the other is based on using the *center of gravity* of polygonal shapes. Both algorithms produce Delaunay meshes in which the stability of the candidate node increases. In Chapter 4, we present an implementation of some of the Delaunay refinement algorithms. The implementation is done in the Java programming language with a user-friendly graphical user interface. The implemented algorithm is used to produce meshes in two dimensions. The quality of the generated mesh in the experimental investigation is measured in terms of the number of triangles with large aspect ratio. Finally, in Chapter 5, we present the conclusion observed from the analysis of the presented algorithms and propose interesting problems for further investigation.

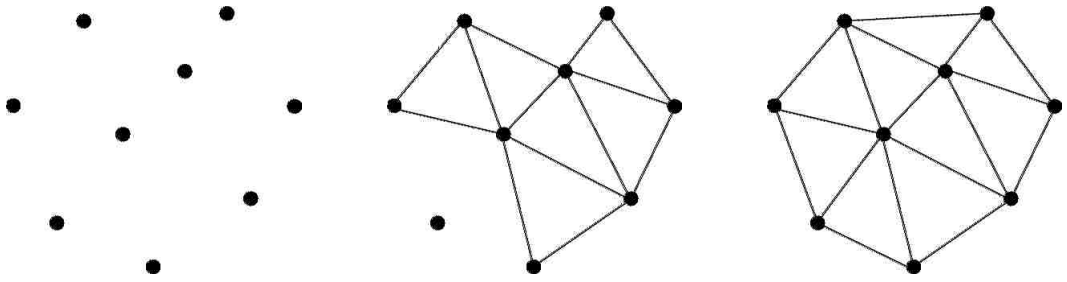
Chapter 2

REVIEW OF TRIANGULATION REFINEMENT ALGORITHMS

In this chapter, we present a critical review of the algorithms for refining triangulated mesh. In this review, the initial triangulation for refinement is assumed to be a Delaunay triangulation. Both polygon triangulation and point set triangulation are considered in the review.

2.1 Point Set Triangulation

A triangulation of a set of point sites $P = \{p_0,p_1,p_2,p_3,...p_n\}$ is a partitioning in the plane of the convex hull of P into triangles, where points in P constitute the vertices of the triangles. Figure 2.1 illustrates the triangulation of nine point sites in the plane. Figure 2.1c is a triangulation of point sites of Figure 2.1a. The triangulation in Figure 2.1b is only *partial* as it does not contain the entire set of edges of the convex hull.

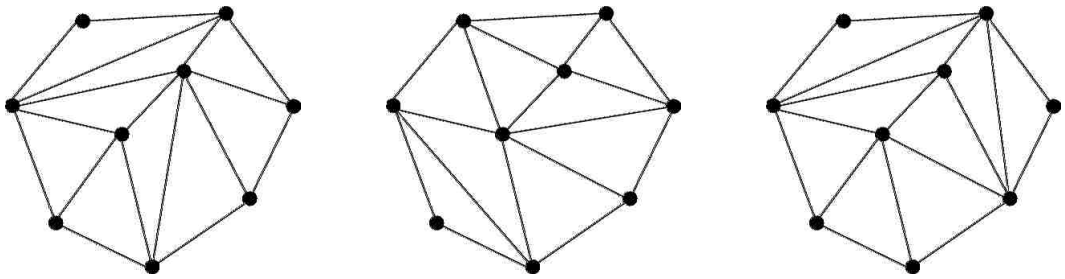


(a) Point Sites (b) Partial Triangulation (c) Maximal Triangulation

Figure 2.1: Triangulation types

When we use the term 'triangulation' of point sites, it is understood to mean a *maximal* triangulation. A given set of n point sites can be triangulated in many ways. Figure 2.2 shows three different triangulations of the point sites in Figure 2.1.

In fact, it is known that a given point site may admit an exponential number of triangulations.



(a) (b) (c)

Figure 2.2: Three different Triangulation of same set of points

2.2 Delaunay Triangulation

Delaunay triangulation [8] for a set of points P in the plane is a triangulation $DT(P)$ such that no point in P is inside the circumcircle of any triangle in $DT(P)$. One of the nice features of *Delaunay triangulation* is that it tends to maximize the minimum angle of all triangulations and thereby avoids skinny triangles which can reduce the overall quality of the triangulation. There is an interesting relationship between Delaunay triangulation and the convex hull [8]. It is known that Delaunay triangulation in 2D is related to the faces in the convex hull of 2-d points projected on a paraboloid in 3D. This can be generalized to d dimensions. Figure 2.3 shows the *Delaunay triangulation* of a set of points.

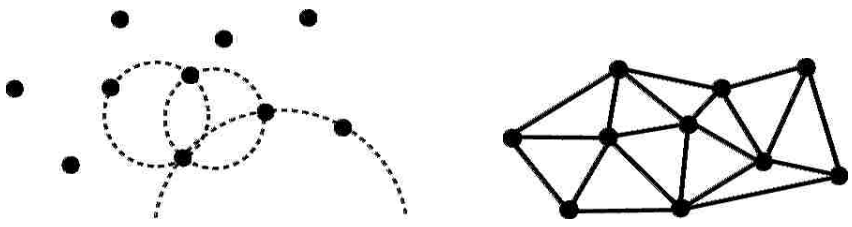


Figure 2.3: Delaunay Triangulation

2.2.1 Empty Circle Test

The empty circle test [8] for the Delaunay triangulation can be formulated as :

If we can draw an *empty circle* through two point sites a and b circumscribing the edges as shown in Figure 2.4 then we can connect them by a Delaunay edge, otherwise we cannot. Since there is an *empty circle* passing through an edge (a,b) it is a Delaunay edge. However, edge (p_1,p_2) is not a Delaunay edge since we cannot construct any *empty circle* passing through the edge (p_1,p_2).

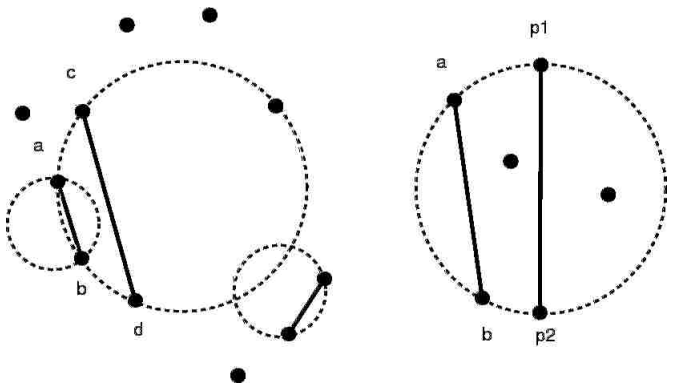


Figure 2.4: Empty Circle Test

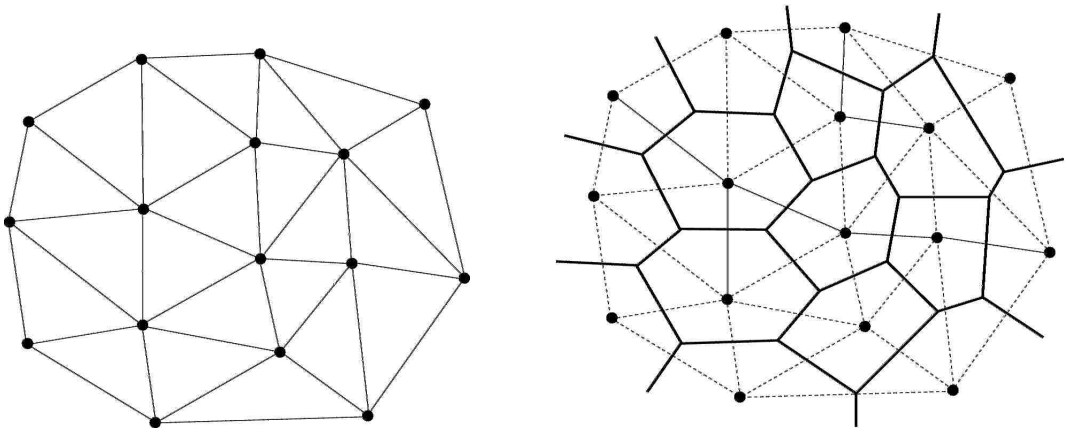
2.2.2 Dual of Delaunay Triangulation

The Delaunay triangulation of a set of two dimensional points corresponds to a dual graph known as the *Voronoi tesselation*.

Defnition 2.2.2 (Voronoi Tessellation) [8] : Let $P = \{p_0,p_1,p_2,p_3,...p_n\}$ be a set of points in the two dimensional Euclidean plane. These are called the sites. Partition the plane by assigning every point in the plane to its nearest site. All those points assigned to p_i form the Voronoi region V(p_i). V(p_i) consists of all the points at least as close to p_i as to any other site :

$$\mathrm{V}(p_i) = \{\mathrm{x} : \mid p_i - x| \leq |P_j - x| \forall j \neq i \}$$

Note that we have defined this set to be closed. Some points do not have a unique nearest site, or nearest neighbor. The set of all points that have more than one nearest neighbor form the Voronoi diagram *V(P)* for the set of sites.



(a) Delaunay Triangulation (b) Voronoi diagram

Figure 2.5: Dual Of Delaunay

Many algorithms for producing Delaunay triangulation exists, and the same can be used to produce *Voronoi diagrams* as well [12]. Incremental and Divide and Conquer algorithms produce Delaunay triangulations both of time complexity O(nlogn). Figure 2.5b shows the *Voronoi Tesselation* of Figure 2.5a. Every edge of the Delaunay triangulation is bisected and projected to meet other bisectors producing the dual known as the *Voronoi Diagram* shown with dark edges in Figure 2.5b.

2.2.3 Constrained Delaunay Triangulation

Defnition 2.3 (Constrained Delaunay Triangulation [CDT]) : A triangulation T of any straight line planar graph G is called a *Constrained Delaunay Triangulation* [11] of G if each edge of G is an edge of T and for each remaining edge e of T, there

exists a circle in which the endpoints of e are on the boundary of c and if any vertex v of G is in the interior of c then it cannot be "seen" from at least one of the endpoints of e. From the diagram below, we can figure out that the *Constrained Delaunay triangulation* and *Delaunay triangulation* are more or less the same. The only difference is that for *CDT*, some portions of the circle may be ignored, but not for the *Delaunay triangulation*.

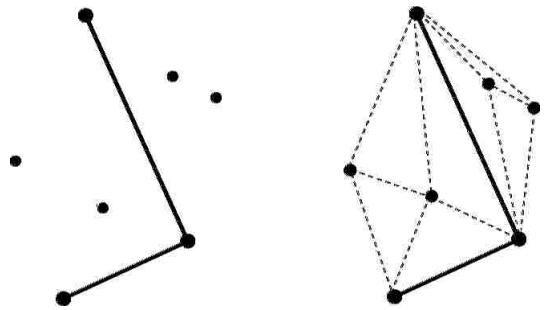


Figure 2.6: Constrained Delaunay Triangulation

2.3 Data Structure for representing 2D meshes

A widely used data structure for representing planar graphs is a doubly connected edge list. A doubly connected edge list [DCEL] [8] is a pointer based data structure which is very useful for finding neighboring vertices and faces in a polygonal mesh. A *DCEL* does not require us to search through all polygons while searching for nearby nodes and edges. A *DCEL* consists of vertices, half edges and face objects with pointers between the vertices. The advantage that *DCEL* has is that it allows direct access to the pointed objects in the mesh without the need of searching.

Each edge in *DCEL* bounds two faces and hence is also known as a *half edge*. Each *half edge* has a pointer to the next *half edge* and previous *half edge* of the same face. Each *half edge* only bounds a single face. In order to reach the other face we need to visit the twin of the half edge and traverse the other face. Table 2.1 shows the

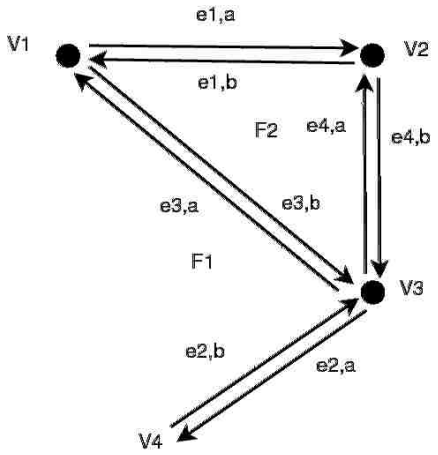


Figure 2.7: Doubly Connected Edge List

half-edge	origin	twin	incidentFace	next	previous
e1,a	v1	e1,b	F1	e4,b	e3,a
e1,b	v2	e1,a	F2	e3,b	e4,a
e2,a	v3	e2,b	F1	e2,b	e4,b
e2,b	v4	e2,a	F1	e3,a	e2,a
e3,a	v3	e3,b	F1	e1,a	e2,b
e3,b	v1	e3,a	F2	e4,a	e1,b
e4,a	v3	e4,b	F2	e1,b	e3,b
e4,b	v2	e4,a	F1	e2,a	e1,a

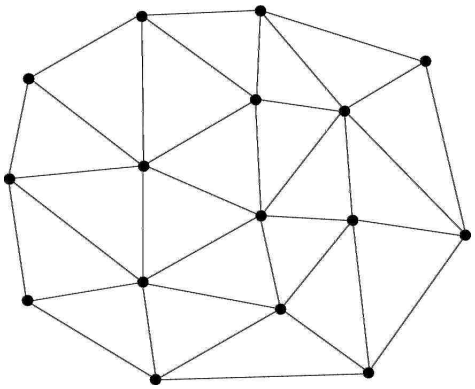
Table 2.1: Doubly Connected Edge List

records for DCEL representation for the planar graph of Figure 2.7.

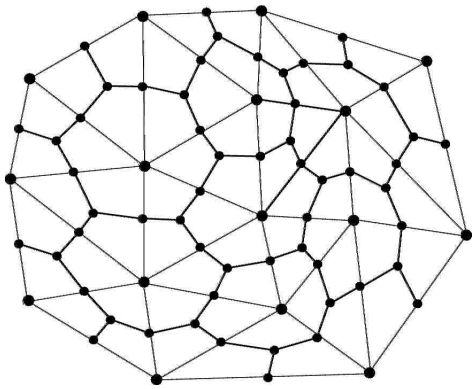
2.4 Triangulation refinement by quadrangulation

A very straightforward way of refining triangulation is to first convert triangulation to quadrangulation and convert quadrangulation to triangulation by introducing new vertices. Efficient algorithm for quadrangulating point sets in 2D is reported in [16]. It is interesting to note that while any point set convex hull can be partitioned into triangles, it is not always possible to partition into quadrilaterals. For many point sites, an attempt to quadrangulate leads to some trapped triangles.

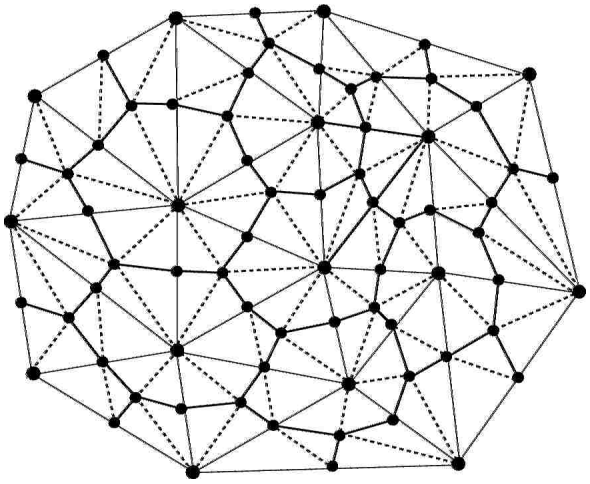
In order to produce a *quadrangulation*, we need to add *Steiner* points to the triangulation. The problem of mesh refinement by *quadrangulation* starts with a triangulation of the points set.



(a) triangulation



(b) quadrangulation



(c) Refinement by diagonal

Figure 2.8: Quadrangulation from the triangulated polygon

A very simple way of refining a triangulated mesh is to first convert each triangle element into a quadrilateral and then partition each quadrilateral into triangles by adding diagonals [16]. A popular technique to convert a triangle into a quadrilateral is to introduce a new vertex inside each triangle. The location of the new vertex can be taken at the centroid of the triangle. The centroid can be connected to the mid-points of the edges of the corresponding triangle.

Figure 2.8b shows a *quadrangular* mesh obtained in this manner. Now, each quadrilateral can be refined by adding a new diagonal. The refined triangulated mesh obtained in this manner is shown in Figure 2.8c. One of the drawbacks of this approach is that the resulting triangles can be very *skinny* [10]. This is due to the fact that each angle of the original triangle may be partitioned into two angles. The algorithm can be implemented to run in linear time in a straightforward way if the original triangulation is available in a doubly connected edge list data structure [8]. It is apparent that if the original mesh has k triangles then the refined mesh has $6k$ triangles.

2.5 Delaunay Refinement

Delaunay refinement algorithms [18,19] typically serve to fulfil certain common goals, like bounding the small angles and offering control over the size of the triangles in the mesh. The measure of quality in any triangle is typically dictated by the small or large angle constituting the triangle. *Delaunay Refinement* algorithms typically improve the *circumradius-to-shortest* edge ratio. The implication of a smaller ratio is that the smallest angle becomes larger. In order to refine any triangular mesh we need to insert new vertices. The core problem of mesh refinement is solved only if the new vertex is inserted optimally. The angle so formed by inserting new vertices should be chosen optimal so as to avoid the formation of *skinny* triangles.

Some of the *Delaunay refinement algorithms* [18,19] were first introduced by *L Paul Chew* and *Jim Ruppert*. Both *Chew's* and *Ruppert's algorithms* [10] work by inserting a new vertex at the circumcenter of a triangle of poor quality. A triangle is said to be of poor quality if its circumradius to shortest side ratio is smaller than a predefined bound B. The value of B for *Ruppert's* algorithm is $\sqrt{2}$ and for *Chew's* algorithm [11] it is 1 [18]. Figure 2.9 shows the refinement process in which a poor triangle with circumradius to shortest side ratio smaller than B is split by inserting a new vertex at the circumcircle. The split process maintains the *Delaunay* property and thereby eliminates the poor quality triangle from the mesh.

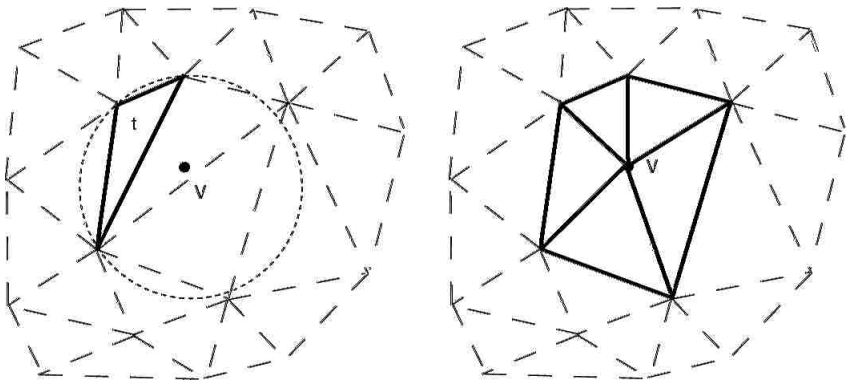
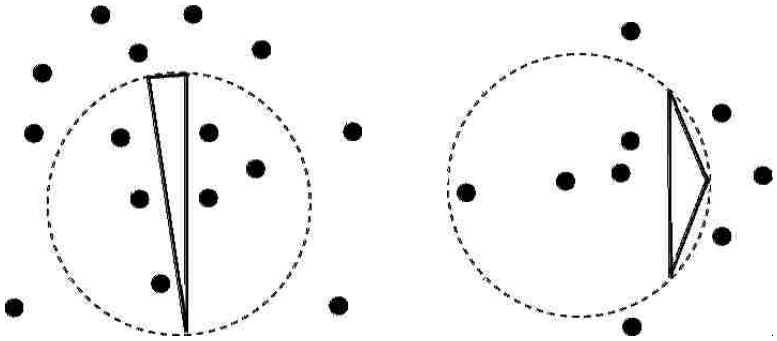


Figure 2.9: Mesh refinement

2.5.1 Skinny Triangle

Skinny triangles degrade the overall quality of the Delaunay triangulation. These triangles are eventually removed by the mesh refinement process. The circumcircle of a *skinny* triangle is larger compared to its shortest edge. As shown in Figure 2.10 *skinny* triangles can be classified as being either a *Needle* [10] or a *Cap* [10]. In a *Needle*, the longest edge is much larger as compared to its shorter edge, whereas a *Cap* has one of its angle which is close to 180^0.



(a) Needle (b) Cap

Figure 2.10: Skinny Triangles

2.5.2 Ruppert's Delaunay Refinement Algorithm

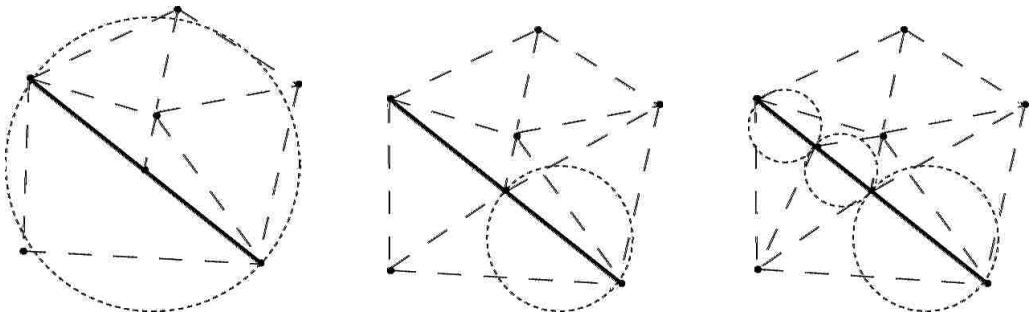
Ruppert's algorithm [10] for *Delaunay refinement* is used for producing a mesh with quality triangles. The input to the algorithm is a planar straight line graph [PSLG] and follows an iterative approach to produce quality triangles. Upon the termination of this program, all the triangles will have aspect ratios at most $|2/sin\alpha|$ since all angles smaller than the threshold angle α are removed [10]. The two basic operations in *Ruppert's* algorithm are to split an encroached segment by adding a new vertex at its midpoint and similarly splitting a skinny triangle by putting a vertex at its circumcircle. The formal sketch of *Ruppert's* algorithm is given below :

The first stage of *Ruppert's algorithm* involves finding the Delaunay triangulation for the set of given points. After computing the Delaunay triangulation, the algorithm follows two iterative steps to refine the mesh. First, it divides the encroached segments shown with the dark solid lines in half by inserting a new vertex at the midpoint of the encroached segment as shown in Figure 2.11a. The encroached segments are split recursively until no segments are encroached as shown in Figure 2.11b and 2.11c.

After splitting the encroached segment into half, the algorithm checks for any skinny triangles formed due to split and re-triangulation operations. The algorithm inserts a

Algorithm 1 Ruppert's Algorithm

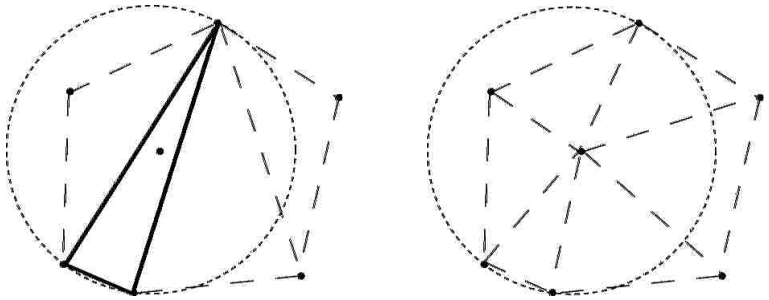
```
public T Ruppert(points,segments,threshold){
        T = Delaunay(points);
        Q = encroached-segments-poor-triangles();
        while (! Q.isEmpty())
                if (Q.contains(segment))
                        insertMidpoint(s, T) ;
                else Q contains poor quality triangle t:
                        if the circumcenter of t encroaches a segments s:
                                add(s ,Q);
                        else:
                                insertCircumcenter(t, T);
                        end if;
                end if;
                update(Q);
        end while;
        return T;
}
```



(a) (b) (c)

Figure 2.11: Ruppert's Encroached segment splitting

circumcenter in any skinny triangle in the mesh as shown in Figure 2.12a and initiates an iterative split process until no other skinny triangles exist. The overall procedure maintains the *Delaunay* property and finally the mesh is refined. A formal sketch of Ruppert's algorithm is listed as Algorithm 1.



(a) (b)

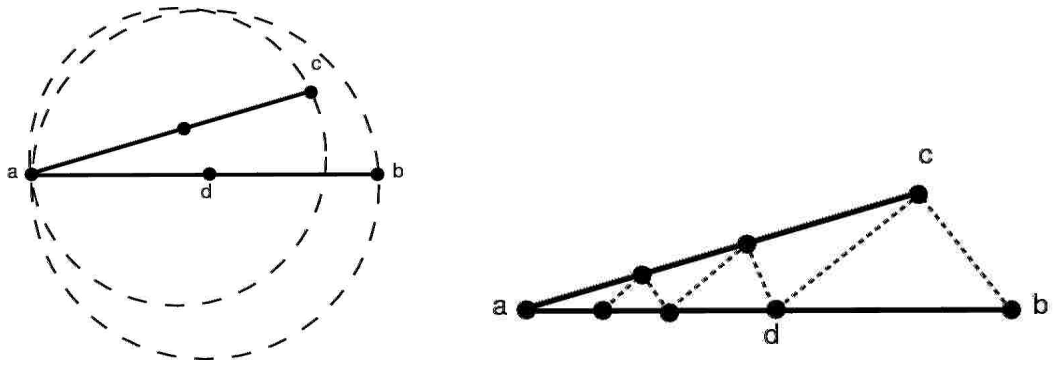
Figure 2.12: Ruppert's bad Triangle splitting

2.5.3 Nontermination Caused by Skinny Triangles

Ruppert's segment split operation can lead to a non-terminating state if two segments intersect at a very skinny angle or two segments have a large angle, as shown in Figure 2.13. As shown in Figure 2.13, segment ab encroaches upon point c so we split the segment ab by inserting a new vertex at mid point d. Now, this midpoint d is again encroached upon by segment ac and we split segment ac into half as well. Therefore, this process of split and encroachment can go on indefinitely and can cause possible non-termination of Ruppert's algorithm.

2.5.4 Small angle Nontermination solution

A possible solution to this problem as suggested by Ruppert is to perform splitting using concentric shells [10] as shown in Figure 2.14. In this approach we first split the segment at the midpoint. However, the subsequent split is done at the intersection of the concentric shells with the nearest midpoint.



(a) (b)

Figure 2.13: NonTermination caused by small angle

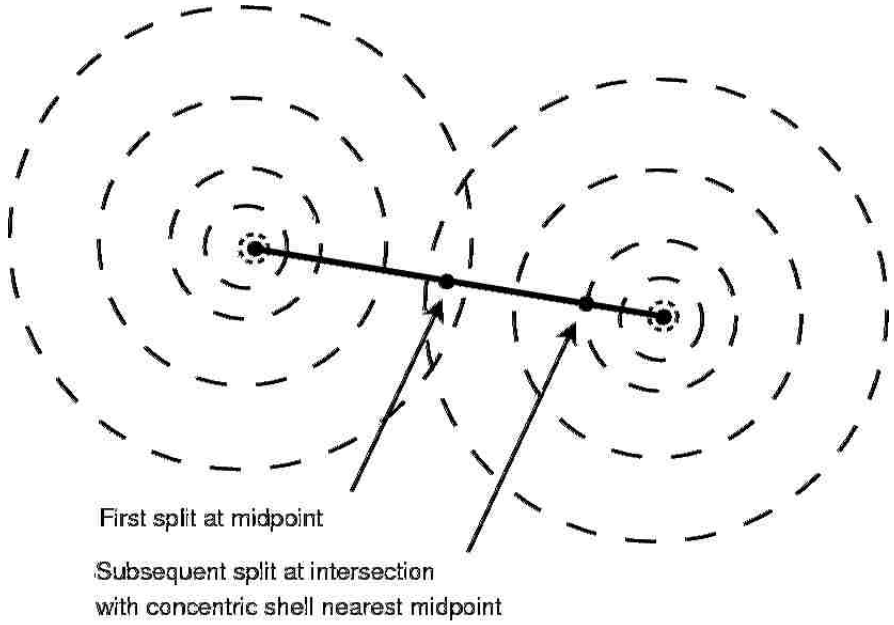


Figure 2.14: Concentric Shell splitting

2.5.5 Chew's Delaunay Refinement Algorithm

Chew's Delaunay Refinement algorithm [11] is an algorithm that produces a quality *Constrained Delaunay Triangulation*. The mesh it produces guarantees that all angles are between 30^0 and 120^0. It starts with a *Constrained Delaunay triangulation* of a *PSLG*. The algorithm removes *skinny* triangles by *Delaunay Refinement*. *Chew's* triangulation, unlike Ruppert's, is not *Delaunay* but rather a *Constrained Delaunay*. After getting the *Constrained Delaunay triangulation* for the set of input vertices, *Chew's* algorithm inserts the circumcenter of a poor quality triangle into the triangulation as shown in Figure 2.15 with the exception that if the circumcenter lies on the opposite side of the input segment as the poor quality triangle then the midpoint is inserted instead. Further, any previously inserted circumcenters inside the diametrical ball of the original segment are removed from the *triangulation*. This is repeated until no poor quality triangles remain in the mesh. It has been shown that Chew's algorithm terminates for an angle bound of up to 30^0 but may not guarantee size optimality and good grading [11]. Under cases when the segments present in the mesh are shorter, only a few of them are ever encroached. This makes Chew's and Ruppert's algorithms produce similar meshes.

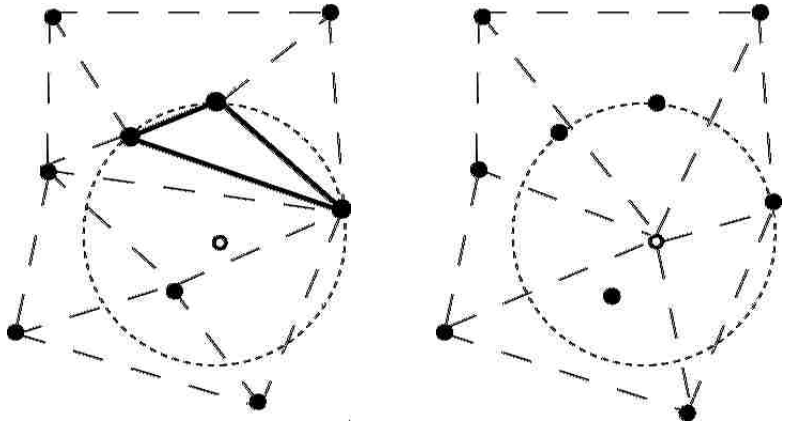


Figure 2.15: Chew's bad Triangle splitting

Chapter 3

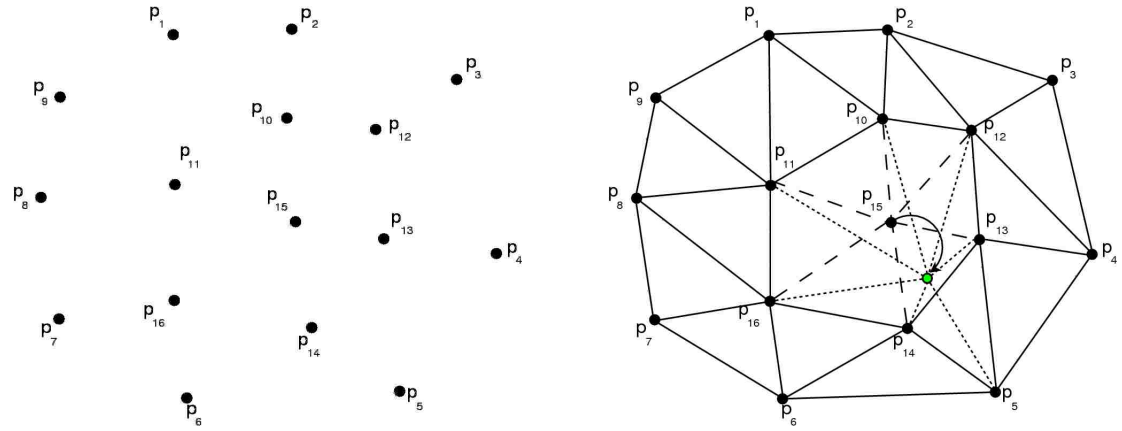
STABILITY AWARE DELAUNAY REFINEMENT

In this chapter, we present the main contribution to the thesis. We introduce the concept of *stable refinement* of Delaunay triangulation. We propose two criteria for placing a new node in the context of Delaunay triangulation refinement. Furthermore, we present two algorithms that use the proposed criteria.

3.1 Characterizing stable node position

Consider the Delaunay triangulation of randomly generated nodes as shown in Figure 3.1a. Suppose we move an interior node P_{15} slightly in its neighbourhood. If the movement is very small, the resulting triangulation does not change in the sense of connectivity i.e., the connectivity between the nodes before and after the movement remains the same. However, if we move the node further sufficiently, the connectivity relation could change. Existing edges in the triangulation can disappear and new edges may appear. This effect of the change in the triangulation due to node movement is illustrated in Figure 3.1b.

Delaunay edges of the candidate node P_{15} are drawn dashed. When the candidate node P_{15} is moved to a new position in its neighbourhood, the edges incident on node P_{15} are drawn dotted. By comparing dashed and dotted edges we find that a new edge connecting P_5 and P_{15} appears and the edge that was connecting P_{13} and P_{14} before disappears. The region in the neighbourhood of a node where the connectivity does not change has been investigated [9]. This can be stated in the following definition.



(a) Point Sites (b) Node Movement

Figure 3.1: Moving Nodes in Delaunay Triangulation

Defnition 3.1 (Roaming region)[9,7] : The *Roaming region* for a Delaunay node P_i is defined as the maximal region R enclosing node P_i where the relocation of the node P_i does not change connectivity.

In Figure 3.2, a small region $R^{'}$ around P_{15} is shown which is a subset of its roaming region. The reader can verify that as long as P_{15} remains within $R^{'}$ the connectivity does not change.

To define *roaming region* formally we start with a few definitions. Consider a node P_i in the interior of a Delaunay triangulation as shown in Figure 3.3. A Delaunay node is called *internal* if (i) it is not on the convex hull and (ii) it is not incident to a node on the convex hull.

Two types of triangles with respect to a candidate internal node P_i are used to formalize the notion of *roaming region*. The first type of triangles are the *radial triangles* and the others are the *lateral triangles*.

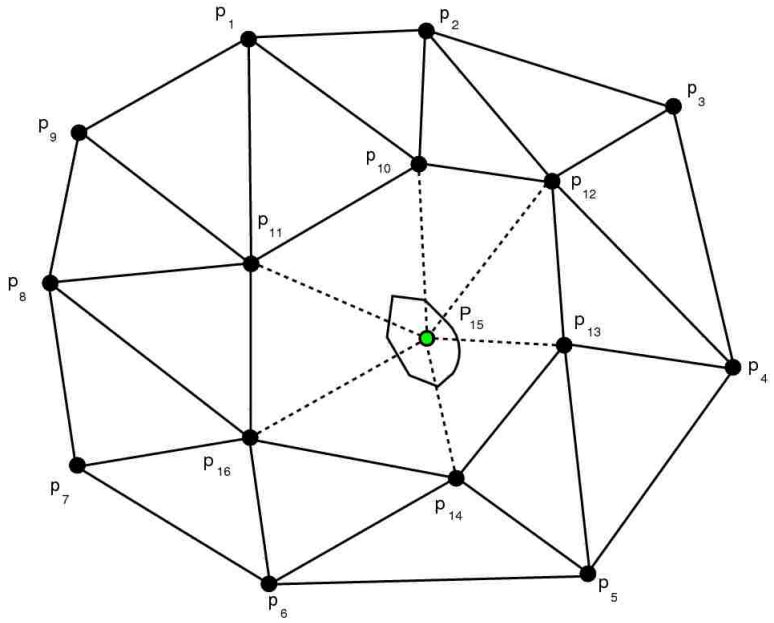


Figure 3.2: Illustration of Roaming Region

3.2 Radial Triangles

Defnition 3.2 [9] : The edges of the triangles incident on candidate node P_i that do not have endpoints at P_i are the *limit edges*. The triangles incident on a *limit edge* away from P_i are called the *radial triangles*. In Figure 3.3, there are six *radial triangles* for internal node P_i. The *radial triangles* are drawn with thick edges.

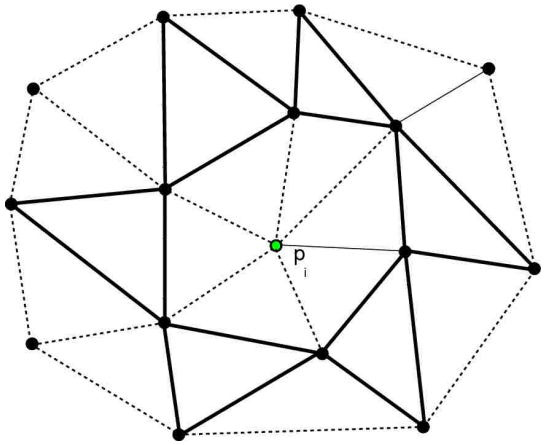


Figure 3.3: Illustration of Limit-Edge and Radial Triangle

3.3 Lateral Triangles

Defnition 3.3 [9] : Lateral triangles are not the triangles of Delaunay triangulation. Consider two adjacent triangles incident on a candidate node P_i. These two triangles form a quadrilateral. If we flip the diagonal of the quadrilateral, two new triangles are formed called *flipped triangles*. The *flipped triangle* away from P_i is a *lateral triangle*. The six lateral triangles for P_i are drawn with thick edges in Figure 3.4.

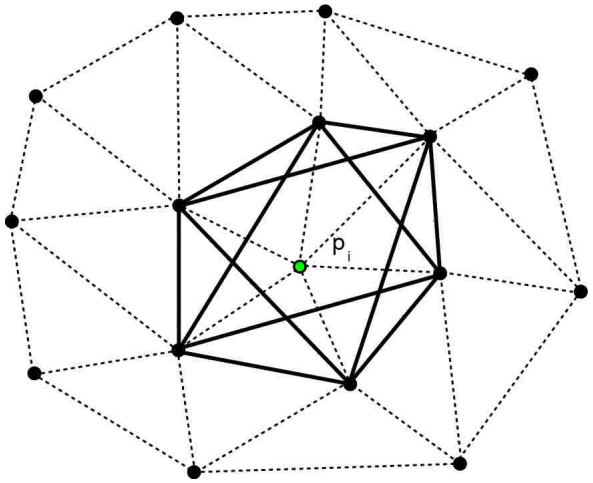
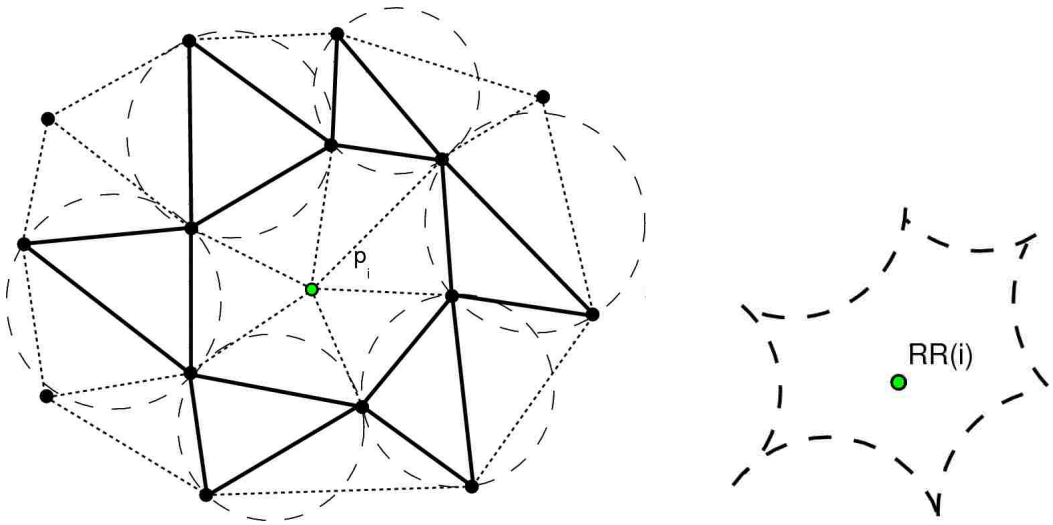


Figure 3.4: Illustration of Lateral Triangle of P_i

3.4 Formation of Radial Roaming Region RR(i)

Consider circumcircles of radial triangles of candidate node P_i as shown in Figure 3.5. The region bounded by the circumcircles of all radial triangles is the *radial roaming region* RR(i) for P_i. It is observed that as long as node P_i remains inside RR(i), no new nodes are connected to P_i due to the in-circle property of Delaunay triangulation. As soon as P_i goes outside RR(i), a new connection will be established. It is noted that when P_i moves inside RR(i), existing edges from P_i to other nodes may disappear.



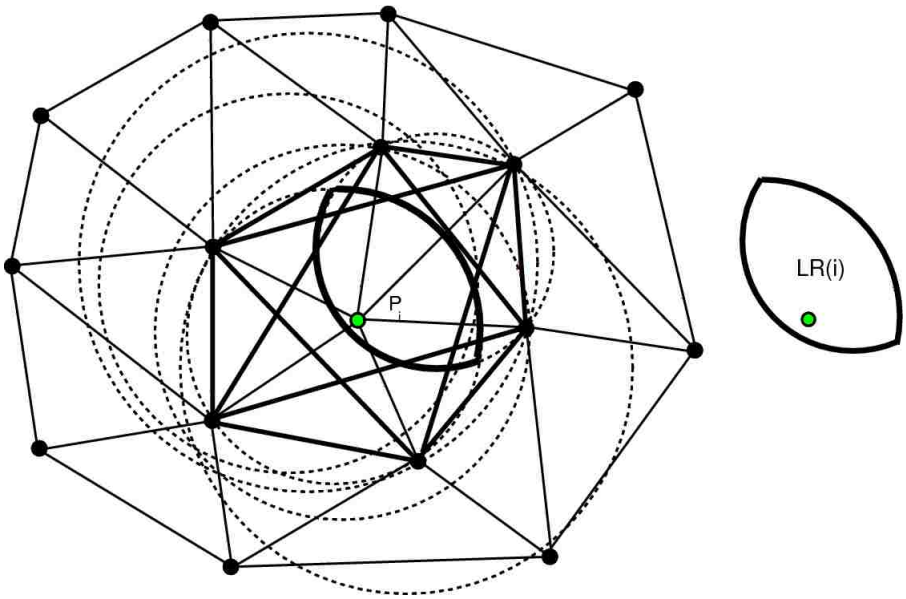
(a) Radial discs (b) Radial Roaming Region

Figure 3.5: Illustration of Radial discs and Radial roaming region of P_i

3.5 Formation of Lateral Roaming Region LR(i)

Consider the circumcircles of lateral triangles of candidate node P_i as shown in Figure 3.6. Let LR(i) denote the intersection of disks corresponding to the circumcircles of lateral triangles. The region LR(i) is called the *lateral roaming region*. It can be observed that as long as node P_i remains within LR(i), all existing edges incident on P_i will be preserved.

Lateral Roaming Region LR(i) = $\cap(L_1, L_2, L_3, ...L_n)$ where $L_1, L_2, L_3.....L_n$ are the lateral discs.



(a) Lateral discs

Figure 3.6: Illustrating the formation of Lateral Roaming Region

3.6 Formation of Free Roaming Region R(i)

If we overlay the lateral roaming region and the radial roaming region then their intersection R(i) = LR(i) ∩ RR(i) is such that as long as node P_i remains within R(i) the connectivity between the nodes does not change. This overlay region R(i) is called the *free region* of node P_i as shown in Figure 3.7.

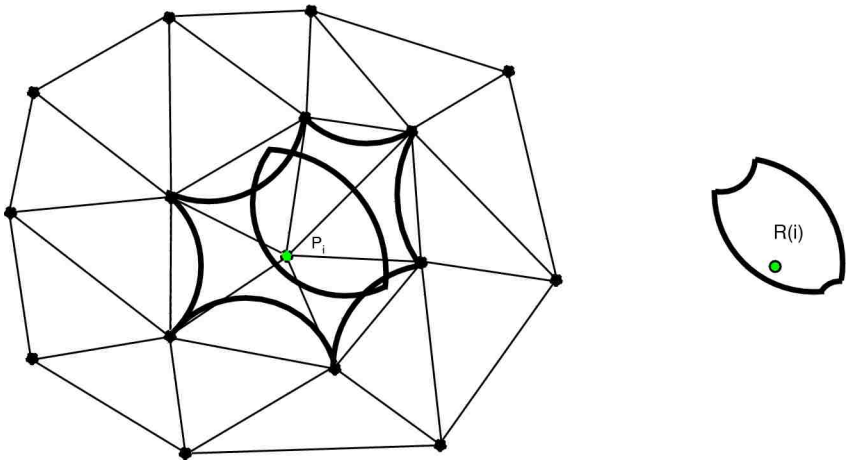
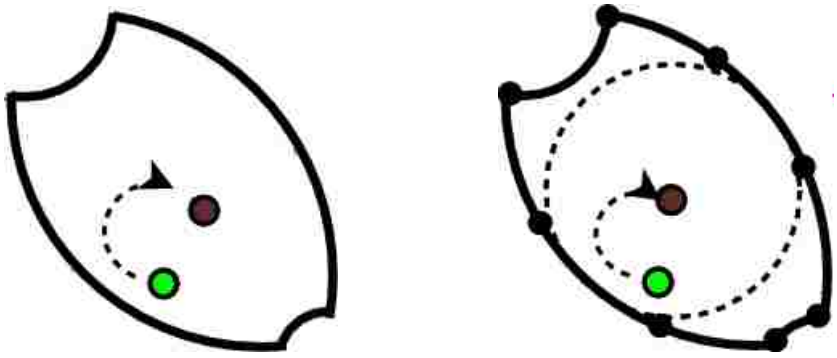


Figure 3.7: Illustrating formation of Free Roaming Region

3.7 Reliable Delaunay Refinement

As reviewed in Chapter 2, the refinement of a Delaunay triangulation is done by inserting a new node at the center of the circumcircle of an existing triangle. If the position of the new node P_i happens to be near the boundary of the corresponding free region R_i, then a slight change in the position of P_i would result in a change of the connectivity of the nodes. Such a position of P_i is an unstable position. One question that normally arises here is how to relocate the position of a candidate node P_i such that the resulting refined Delaunay triangulation is stable i.e., a slight change in the position of P_i would not affect the connectivity of the Delaunay triangulation.

The idea here is to first compute the free-region R_i for P_i and relocate the position P_i near the center of R_i. We propose two approaches for relocating P_i to a more reliable position. The first approach is to place P_i in the center of gravity of R_i as shown in Figure 3.8a and the second is to place it at the center of the largest empty circle inside R_i as shown in Figure 3.8b.



(a) shifting to CG of R(i)

(b) largest empty circle of R(i)

Figure 3.8: Reliable shifting of P_i

3.8 Center of Gravity of a Simple Polygon

The center of gravity CG of a polygon can be conceptualized by viewing the interior of the polygon to be made up of a material of uniform thickness. With such a notion, the center of gravity of the material is the CG of the polygon. To compute the center of gravity of a polygon, we can use the coordinates of the vertices and the area of the polygon as follows [8].

$$\mathrm{A} = 1/2 \sum_{i=0}^{n} (x_i y_{i+1} - x_{i+1} y_i) \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad (1)$$

where x_i, y_i are the coordinates of the vertex v_i. Let c_x, c_y denote the x and y coordinates of the center of gravity of the polygon. Then, c_x and c_y can be expressed as [8].

$$C_x = 1/6\mathrm{A} \sum_{i=0}^{n} (x_i + x_{i+1})(x_i y_{i+1} - x_{i+1} y_i) \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad (2)$$

$$C_y = 1/6\mathrm{A} \sum_{i=0}^{n} (y_i + y_{i+1})(x_i y_{i+1} - x_{i+1} y_i) \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad (3)$$

Using the above formulas, the center of gravity of a polygon can be calculated in a straightforward manner. For convex shapes, the center of gravity always lies inside the polygon. However, there are some class of non-convex polygons in which the center of gravity may lie outside the polygon, as illustrated in the following diagrams.

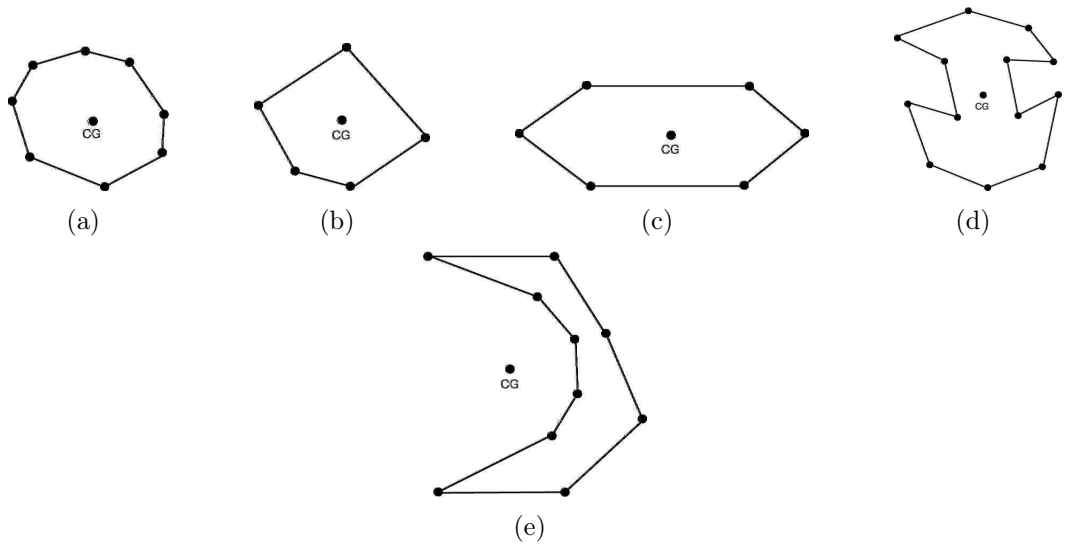


Figure 3.9: Illustration of CG of 2D shapes

Now, we have the ingredients to describe a CG-based relocation algorithm. The set of input point sites is triangulated by using Fortune's algorithm [3]. The free region of candidate nodes inside the convex hull is computed by using free region algorithm given in [9]. The center of gravity of the polygonal shape representing the free region is computed. The candidate node is relocated at the center of gravity of the free region. A formal sketch of the algorithm is listed as Algorithm 2.

3.8.1 CG based Relocation algorithm

Time complexity of CG based relocation algorithm is as follows :

Delaunay triangulation of n input point sites can be done in O(nlogn) time by using Fortune's algorithm [3]. Hence, Step 1 takes O(nlogn). Free region of a node can be done in O(n^2) time by using algorithm given in [9]. Hence, Step 2 takes O(n^2) time. Step 3 can be done in O(n) time by scanning the boundary of R_i and replacing arcs with polygonal chain. Center of gravity of polygonal shapes can be computed in

Algorithm 2 CG based Relocation Algorithm

Input : (i) Set of point sites $S = \{p_0,p_1,p_2,p_3,...p_n\}$ in 2D
(ii) An internal point $p_i \in S$

Output : Relocation position q_i for p_i

Step 1 : Compute the Delaunay triangulation DT for S

Step 2 : Compute Free Region R_i for P_i.

Step 3 : Approximate R_i with a simple polygon Q_i by introducing points on arcs.

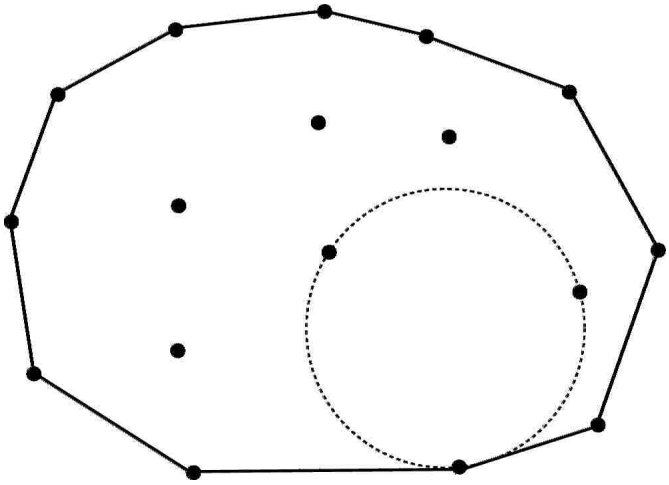
Step 4 : Compute CG of Q_i by using formula (2) and (3). Let the CG be q_i

Step 5 : Output q_i as the relocated position for p_i

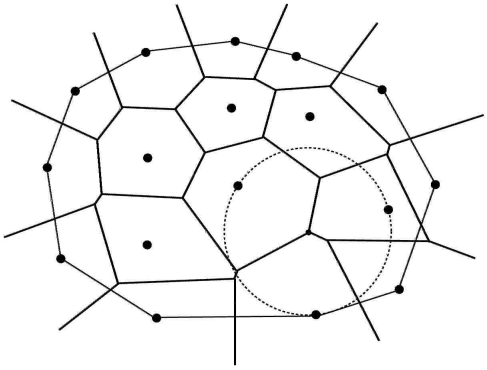
O(n) time by using standard method reported in literature [8]. Intersection of two polygonal shapes can be achieved in O(n) time [13]. Hence the overall time complexity of the algorithm is O(n^2).

3.9 Largest Empty Circle

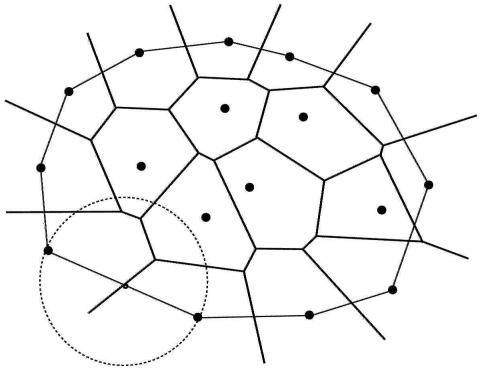
Let $Q = \{q_1,q_2,...,q_n\}$ be a set of n points on the plane and let *CH(Q)* denote the convex hull of Q. The *largest empty circle* (LEC) problem asks to find an empty circle whose center is inside the convex hull of Q [4]. It is known that the *largest empty circle* [4] is either centered on the vertex of the Voronoi diagram or on the intersection of a Voronoi edge with the convex hull boundary. Figure 3.10 shows different cases of empty circle formation. In Figure 3.10a, a convex hull of a set of points with its *largest empty circle* is shown. Similarly, Figure 3.10b and Figure 3.10c show two of the cases when the *largest empty circle* is centered on a vertex of the *voronoi* diagram and when the *largest empty circle* is not centered on the *voronoi* vertex. Toussaint [4] has shown that the computation of the largest empty circle with location constraints can be done in O(n logn) time.



(a) Largest Empty Circle of points



(b) Circle centered on voronoi vertex



(c) Circle not centered on voronoi vertex

Figure 3.10: Largest empty circle

3.9.1 Largest Empty Circle based Relocation algorithm

Algorithm 3 Largest Empty Circle based Relocation Algorithm

Input : (i) Set of point sites $S = \{p_0, p_1, p_2, p_3, \ldots p_n\}$ in 2D
(ii) An internal point $p_i \in S$

Output : Relocation position q_i for p_i

Step 1 : Compute the Delaunay triangulation DT for S

Step 2 : Compute Free Region R_i for P_i.

Step 3 : Approximate R_i with a simple polygon Q_i by introducing points on arcs.

Step 4 : Compute Largest Empty Circle of Q_i by using Toussiant algorithm. Let the center of largest empty circle be q_i

Step 5 : Output q_i as the relocated position for p_i

Now, the relocation algorithm based on empty circles can be described as follows :

Free region R_i is approximated by a polygon shape. The largest empty circle inside the polygon representing R_i is computed by using Toussiant's algorithm [4]. The node is relocated to the center of the largest empty circle. A formal sketch of the algorithm is listed as Algorithm 3.

The time complexity of Algorithm 3 can be done in the same way as for Algorithm 2, which leads to the fact that the execution time of Algorithm 3 is $O(n^2)$.

Chapter 4

IMPLEMENTATION

In this chapter, we present an implementation of the algorithms that were reviewed in Chapter 2 for refining triangular meshes. The program was implemented in JavaSE-1.7. The implemented algorithms include (i) Ruppert's Delaunay refinement algorithm, and (ii) Refinement by quadrangulation. We also present a variation of Ruppert's algorithm by incorporating the idea of center of gravity of Delaunay triangles.

4.1 GUI Description

The main graphical interface of the implementation is developed by using Java Swing class. As shown in Figure 4.1, there are four panels present in the main frame. The top panel is used to contain the menu bar. The middle portion of the jFrame consists of two panels: *center panel* and the *right panel*. The *center panel* is the main area to display graphics whereas the *right panel* contains check boxes and buttons to allow user to select appropriate choices. The checkboxes on the *right panel* are used to select operations such as *draw nodes*, *edit nodes*, *split segment*, *split triangle*, *Delaunay triangulation*, *circum-circles*, *encroached segment*, *quadrangulation* and *roaming region*.

The *right panel* also contains three buttons *Ruppert*, *CG refinement* and *CC refinement* which are used for selecting various mesh refinement methods. Besides these, two more buttons *clear canvas* and *test points* buttons, are present. The bottom panel displays coordinates of the position of the mouse. All of the panels are implemented by extending the jPanel class.

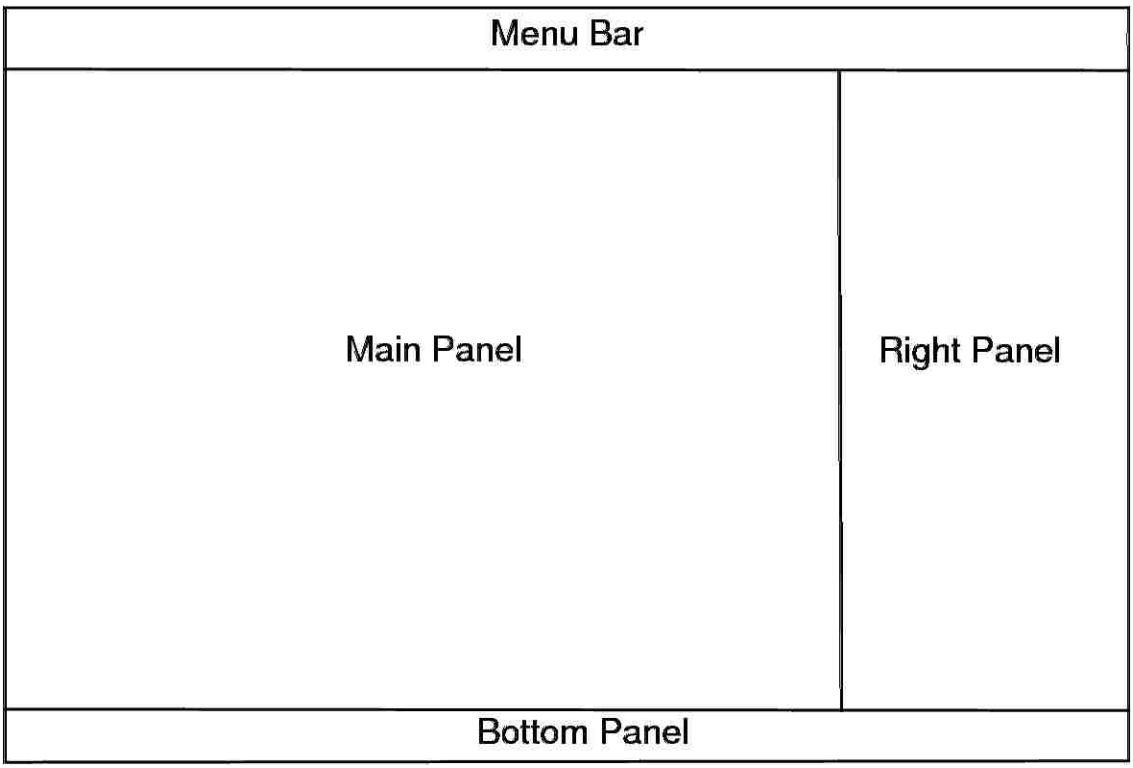


Figure 4.1: Layout of main user interface

4.2 Interface Description

Figure 4.2 shows a snap-shot of the actual interface of the program. The file menu on the top panel allows user to (i) read an existing dcel file, (ii) save the dcel file to filesystem and (iii) exit the application. A brief description of the functionalities of the file menu items is provided in Table 4.1. Users can plot nodes by enabling the *draw node* checkbox. The nodes can be edited and a mesh can be drawn with it. Table 4.2 gives an overview of the functionality of all checkboxes in the GUI. Similarly, Table 4.3 gives an overview of the functionality of all the buttons in the GUI.

Figure 4.2: Graphical User Interface

S.N.	Menu Item	Functionalities
1	Read Dcel File	Brings up a pop up to allow the user to select a pre-saved file
2	Save Dcel File	Brings up a pop up to allow the user to save the diagram
3	Exit	Exits the application

Table 4.1: File Menu Items Description

4.3 Illustrating Refinement

Delaunay mesh refinement starts with a given Delaunay triangulation. Figure 4.3 shows the Delaunay triangulation of a set of points. The mesh is unrefined due to the presence of low quality skinny triangles as shown in Figure 4.6. Generally, mesh refinement is done by splitting segments and by splitting skinny triangles present in the mesh. Figure 4.5 and Figure 4.7 show the segment split and triangle split operations.

S.N.	Menu Item	Functionalities
1	Draw Vertex	Allows users to draw vertices on the mainPanel
2	Edit Vertex	Allows users to edit drawn vertices
3	Delaunay Triangulate	Triangulates vertices by using Delaunay triangulation algorithm
4	Encroached Segment	Displays the segments which are encroached upon by any other points
5	Refine Triangle	Refines existing triangles in the mesh
6	Circum Circles	Draws circumcircles around the triangles in the mesh
7	Cartesian Grid	Draws cartesian grid in the main Panel
8	Split Segment	Splits the segment into half
9	Skinny Triangle	Displays skinny triangles in the triangular mesh
10	Split Triangle	Splits skinny triangles in the mesh
11	Split Triangle at CG	Splits the triangle at center of gravity of the triangle
12	Quadrangulate Points	Changes the existing triangular mesh into a quadrangular mesh
13	Refine Quadrangulation	Refines the quadrangular mesh by adding diagonals
14	Roaming Region	Displays the radial and lateral roaming regions
15	Radial Triangles	Displays the Radial discs and Radial Roaming Region
16	Lateral Triangles	Displays the Lateral discs and the lateral roaming region

Table 4.2: Checkbox Items Description

One of the simplest approaches to refine a mesh is through the quadrangulation of the triangles as shown in Figure 4.8, and then joining the diagonals of the thus formed quadrilaterals. Figure 4.10 shows a second approach to mesh refinement. In this approach, we refine the mesh iteratively by inserting a new point at the center of gravity of each triangle in the mesh.

Figure 4.10 also shows the formation of skinny triangles in a local region near the boundary of the triangular mesh. Similarly, Figure 4.11 shows another approach for refining the mesh. In this approach, we insert new points at the circumcircle of the

triangle instead of at its center of gravity. Figure 4.14 shows the radial discs and the radial roaming region for a point. Similarly, Figure 4.15 shows the lateral discs and the corresponding lateral roaming region formed with the intersection of lateral triangles. The intersection of radial roaming region and lateral roaming region forms the free roaming region as shown in Figure 4.16.

S.N.	**Menu Item**	**Functionalities**
1	Clear Canvas	Clears whatever graph is drawn on the main panel
2	Predefined Points :5	Draws a point set consisting of five nodes
3	Predefined Points :10	Draws a point set consisting of ten nodes
4	Ruppert	Opens up a new popup and refines the mesh using Ruppert's algorithm
5	Random	Draws random set of points on the main panel
6	CG Refinement	Refines the existing mesh by inserting new node at the center of gravity of the triangles
7	Circumcenter Refinement	Refines the existing mesh by inserting a new node at the circumcenter of the triangles

Table 4.3: Buttons Description

Snapshots of meshes and their refinements produced by the implemented algorithms are displayed in Figure 4.3 to Figure 4.13.

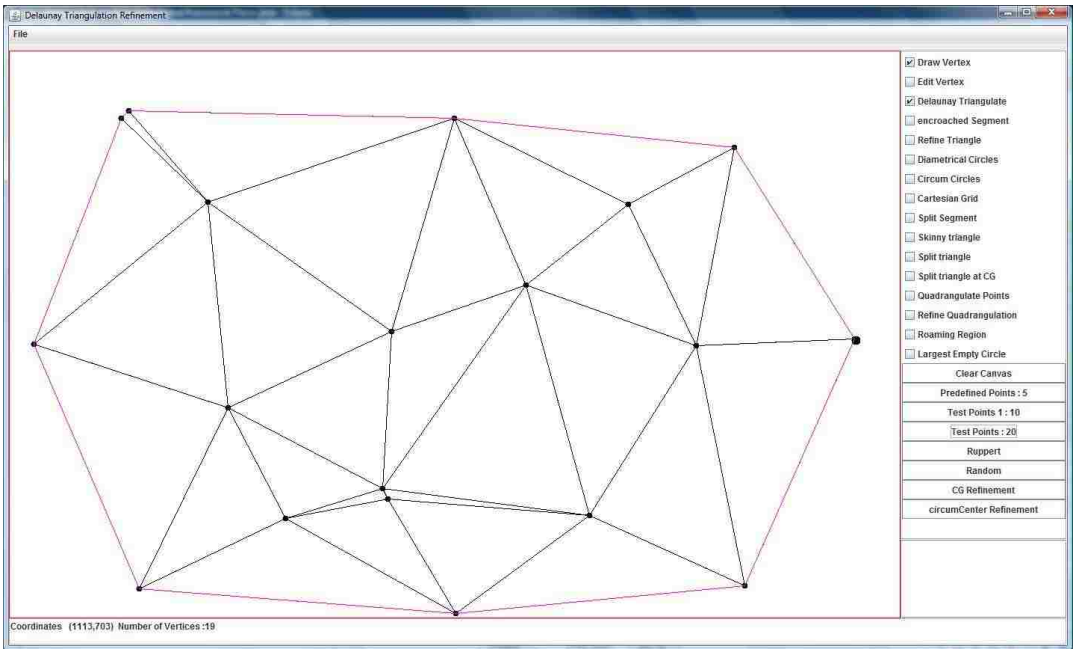


Figure 4.3: Snapshot of Delaunay Triangulation

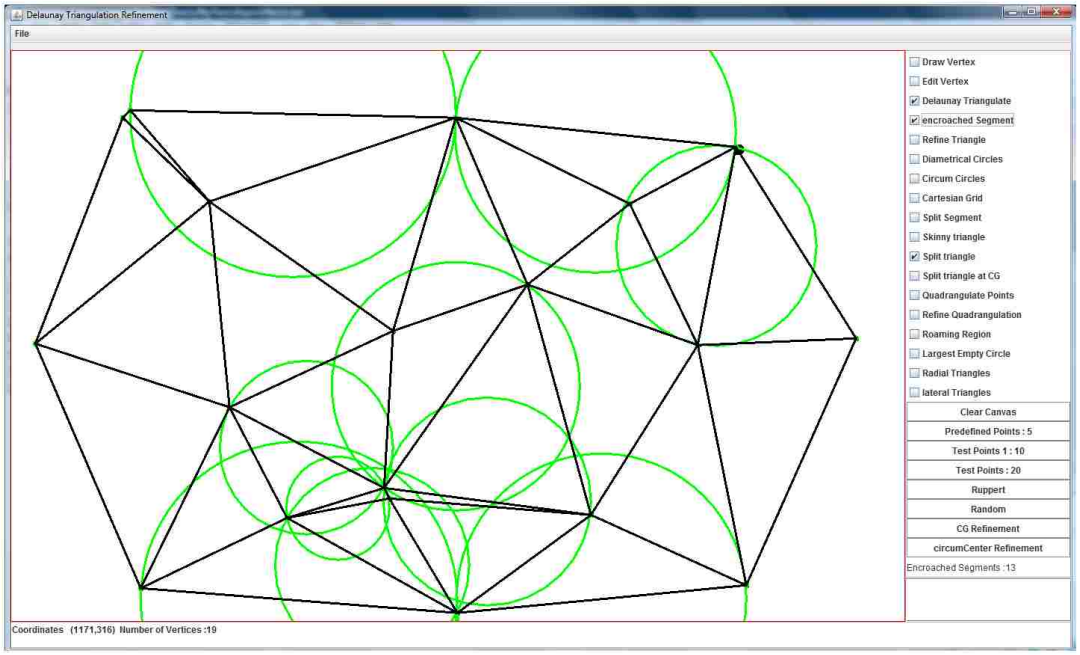


Figure 4.4: Illustrating Encroached Segments

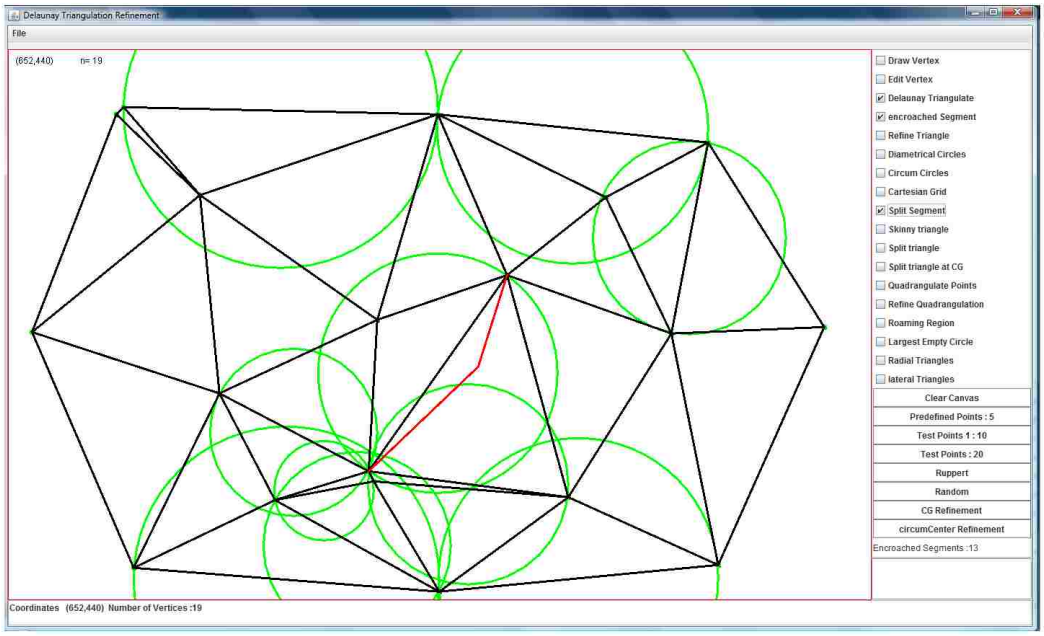


Figure 4.5: Illustrating Splitted Segment

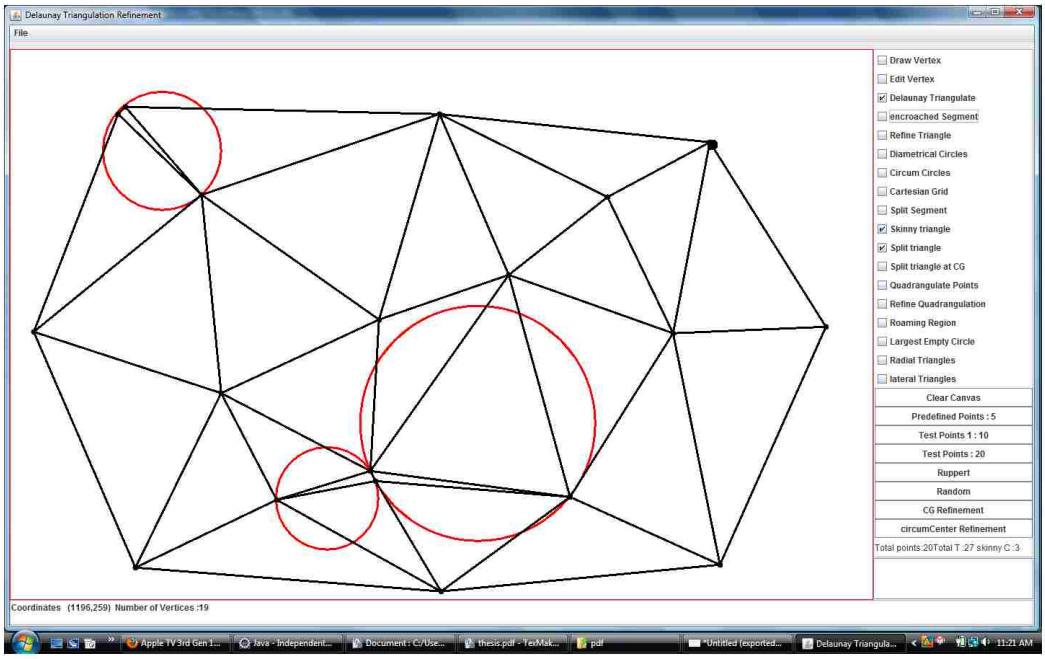


Figure 4.6: Illustrating Skinny Triangles

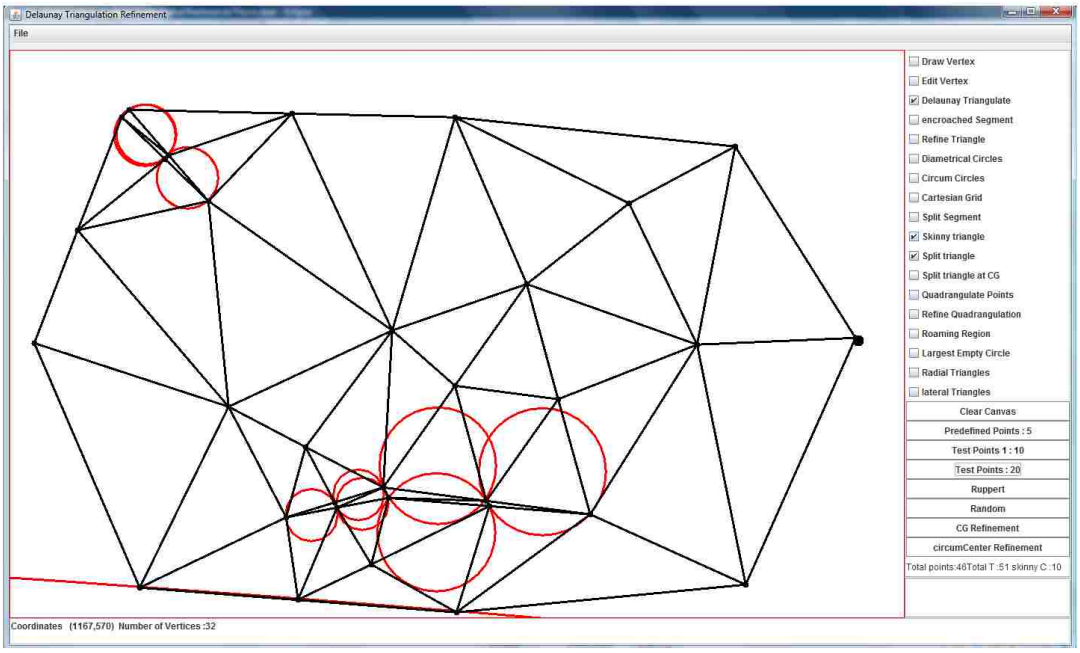


Figure 4.7: Splitting Skinny Triangles

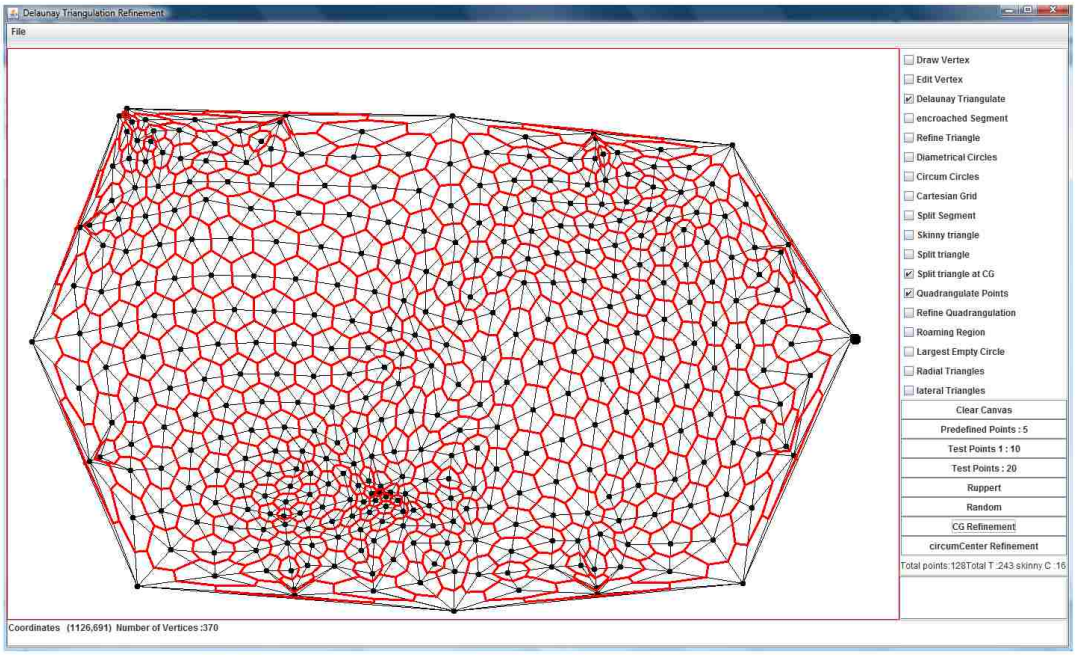


Figure 4.8: Refinement by Quadrangulation (3 iterations)

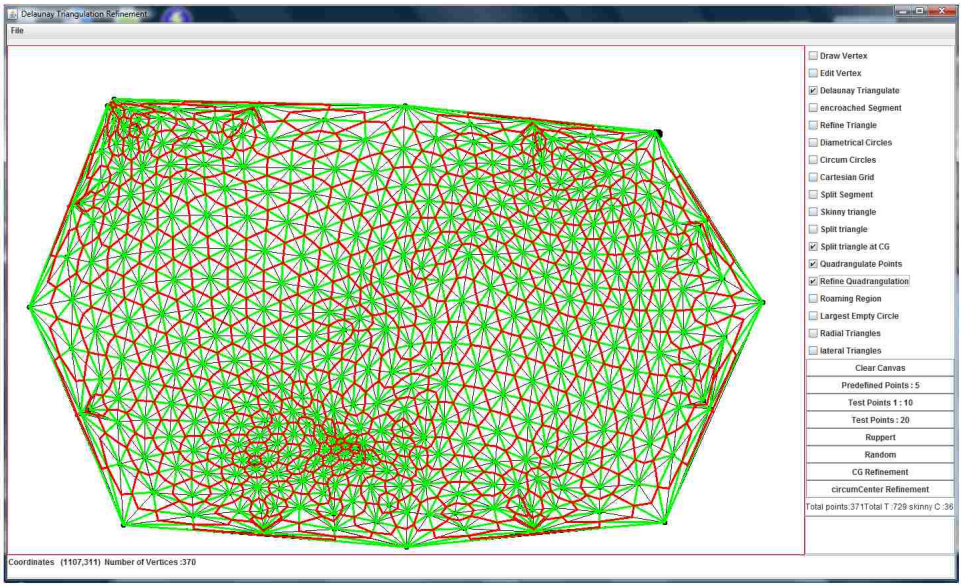


Figure 4.9: Further Refinement by Quadrangulation

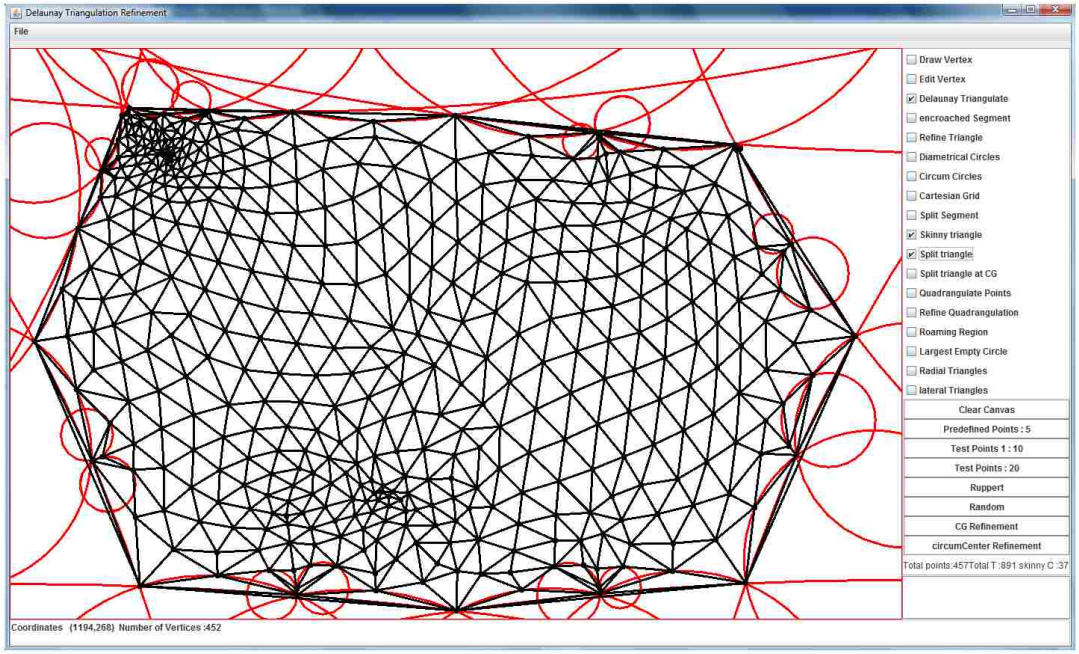


Figure 4.10: Result of CG based Refinement

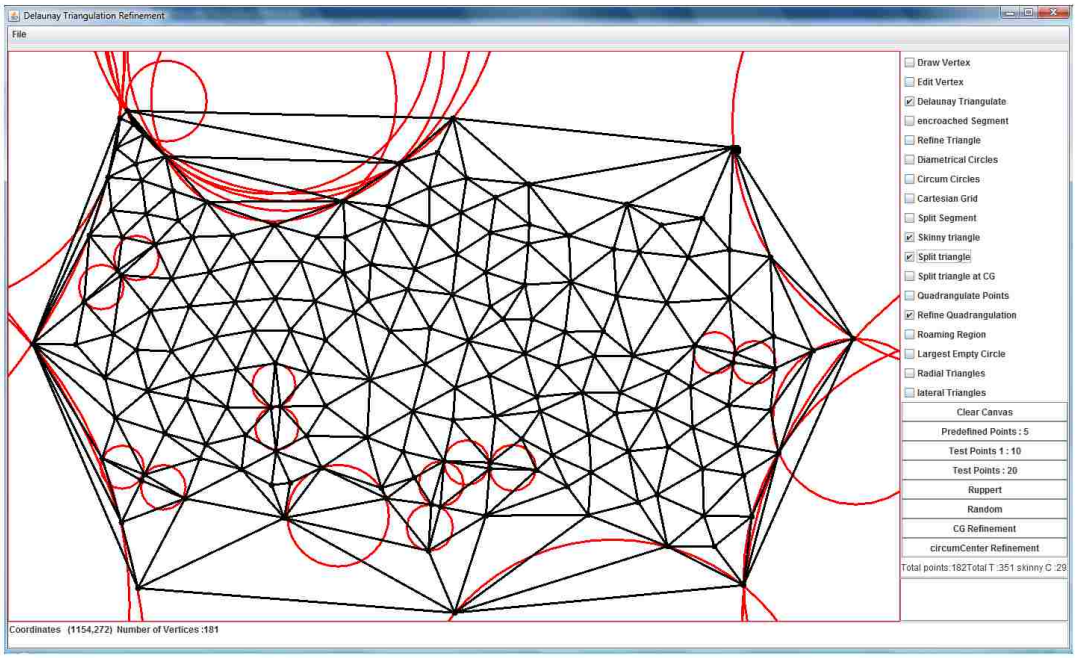


Figure 4.11: Circumcircle Refinement for Internal Elements

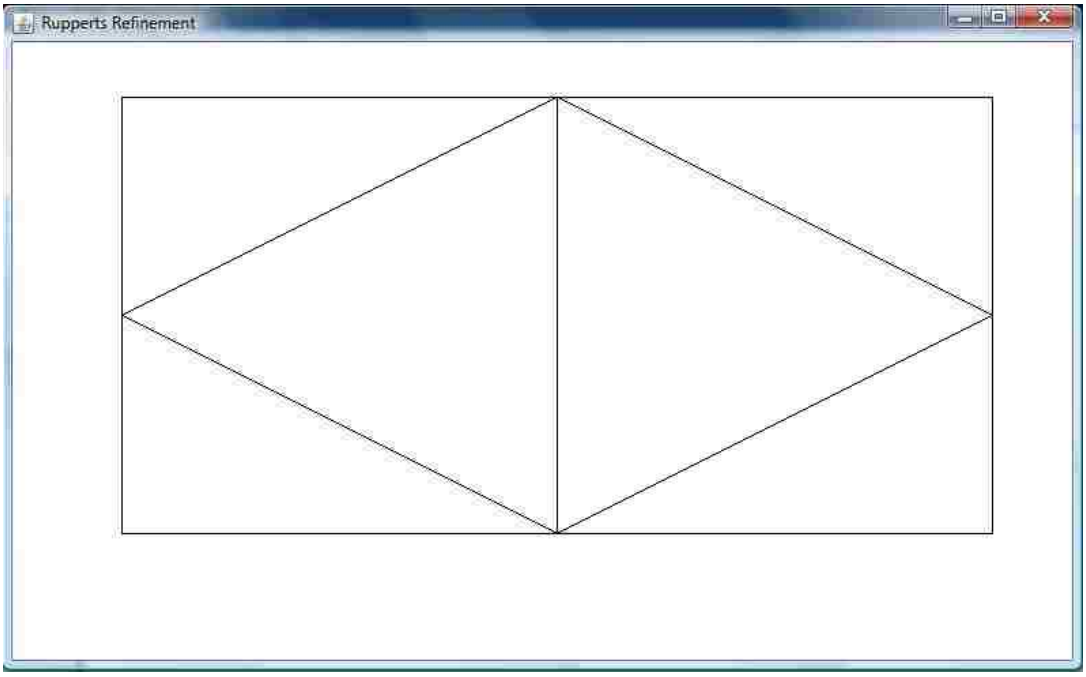


Figure 4.12: Input to Ruppert's algorithm

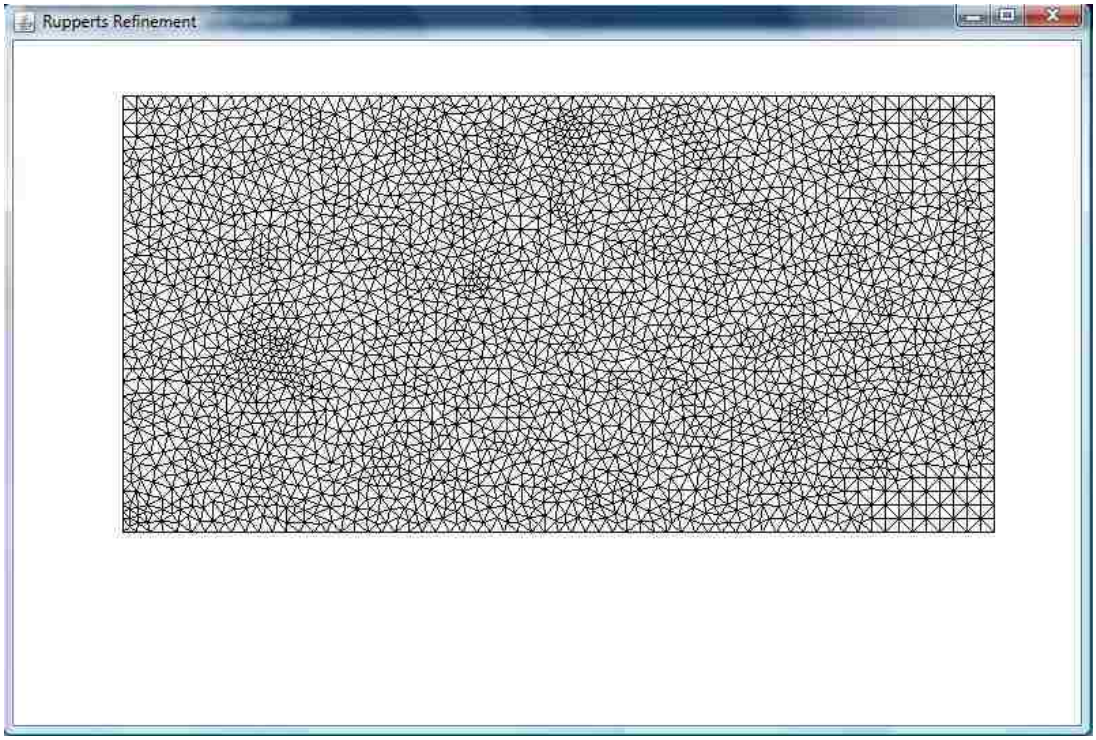


Figure 4.13: Refined mesh obtained by Ruppert's Algorithm

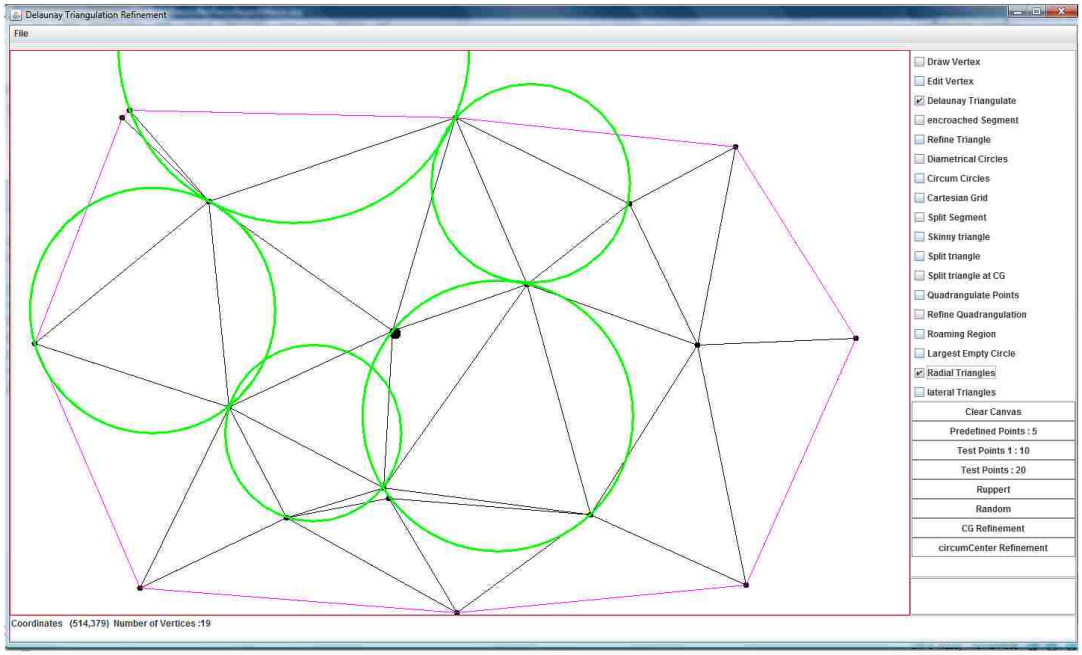


Figure 4.14: Illustrating Radial Roaming Region and Radial discs

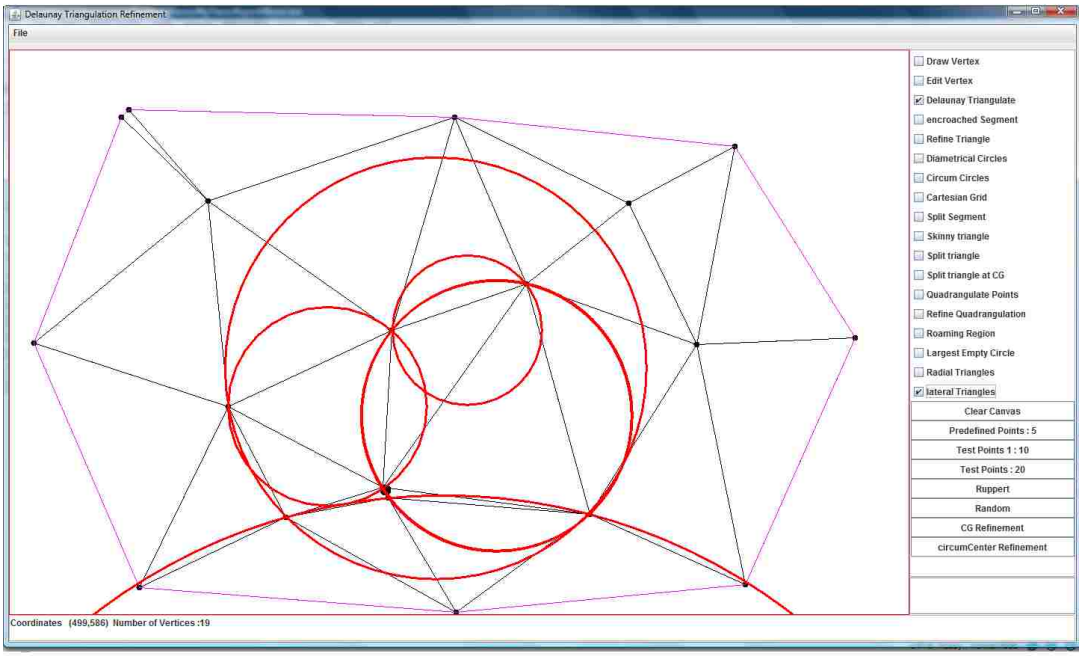


Figure 4.15: Illustrating Lateral Roaming Region and Lateral discs

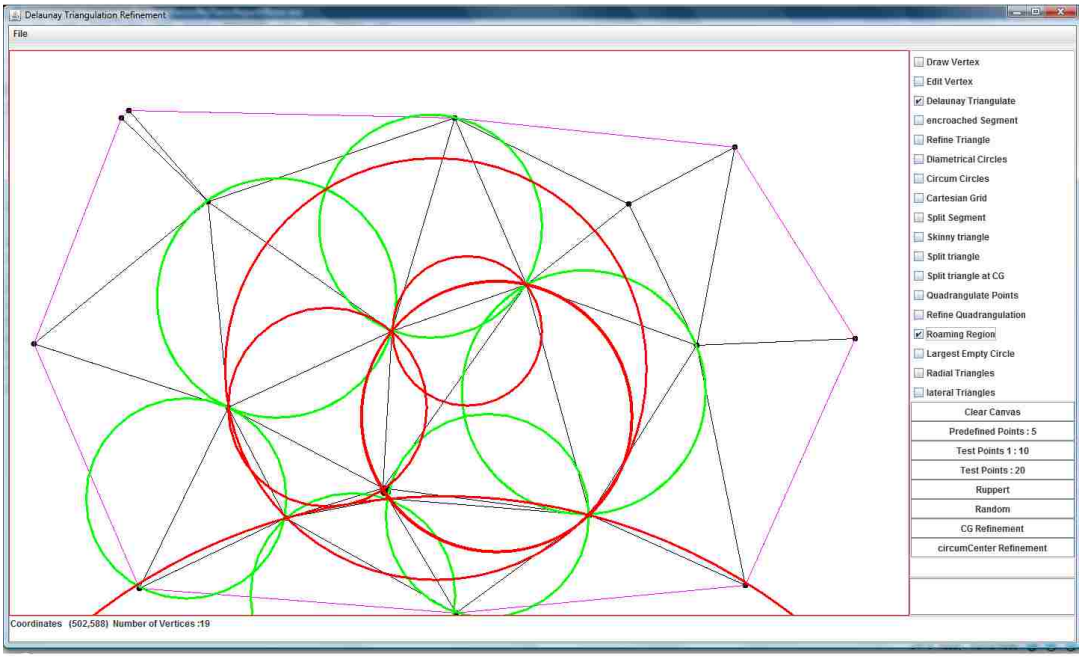


Figure 4.16: Roaming Region showing Radial and Lateral regions

4.4 Results and Statistics

It is clear from Figure 4.9 that the refinement done by quadrangulation increased the number of skinny triangles extensively. This is due to the fact that at each iteration of quadrangulation, generated skinny triangles were further partitioned into more skinny triangles. Center of gravity refinement was able to refine the interior of the mesh in a few iterations and the skinny triangles were only present mostly at the boundary of the mesh, no matter how we altered the minimum angle. Circumcircle refinement, unlike Center of gravity refinement, did not show localized behavior and the skinny circles could be found at any place in the mesh irrespective of the number of iterations. Ruppert's algorithm refined the Delaunay mesh more or less uniformly. However, how stable the nodes are after performing refinement by using Ruppert's algorithm is not known.

Figure 4.17, Figure 4.18, Figure 4.19, and Figure 4.20 shows the behavior of the center of gravity refinement and the circumcircle refinement in terms of the number of nodes, number of triangles, number of skinny triangles and number of iterations. Results on various values of minimum selected angles is shown in tables Table 4.4 to Table 4.12. Figure 4.17 and Figure 4.18 show the plot of refinement results for circumcenter refinement for various values of minimum angle. Similarly, Figure 4.19 and Figure 4.20 show the plot of refinement results for center of gravity refinement. In these plots, x-axis represents the number of iterations and y-axis represents the number of generated triangles.

Iteration No	Total points	Total triangles	skinny triangles
0	10	13	3
1	12	17	3
2	15	23	3
3	19	31	3
4	25	43	3
5	35	63	3
6	54	101	3
7	97	187	3
8	190	371	3

Table 4.4: CC Refinement Minimum angle 5

Iteration No	Total points	Total triangles	skinny triangles
0	10	13	5
1	12	17	4
2	15	23	4
3	19	31	4
4	25	43	4
5	35	63	4
6	54	101	4
7	97	187	7
8	190	371	10

Table 4.5: CC Refinement Minimum angle 10

Iteration No	Total points	Total triangles	skinny triangles
0	10	13	5
1	12	17	6
2	15	23	6
3	19	31	6
4	25	43	6
5	35	63	6
6	54	101	8
7	97	187	19
8	190	371	31

Table 4.6: CC Refinement Minimum angle 15

Iteration No	Total points	Total triangles	skinny triangles
0	10	13	9
1	12	17	12
2	15	23	15
3	19	31	19
4	25	43	26
5	35	63	32
6	54	101	44
7	97	187	75
8	190	371	121

Table 4.7: CC Refinement Minimum angle 30

Iteration No	Total points	Total triangles	skinny triangles
0	10	13	3
1	23	39	4
2	62	116	9
3	178	348	16
4	526	1041	31

Table 4.8: CG Refinement Minimum angle 5

Iteration No	Total points	Total triangles	skinny triangles
0	10	13	5
1	23	39	6
2	62	116	14
3	178	348	27
4	526	1041	50

Table 4.9: CG Refinement Minimum angle 10

Iteration No	Total points	Total triangles	skinny triangles
0	10	13	9
1	23	40	15
2	62	116	24
3	178	348	52
4	526	1024	91

Table 4.10: CG Refinement Minimum angle 20

Iteration No	Total points	Total triangles	skinny triangles
0	10	13	11
1	23	39	26
2	62	116	47
3	178	348	82
4	526	1041	158

Table 4.11: CG Refinement Minimum angle 30

Initial triangles	Area	Minimum Angle	Final no of triangles	Skinny triangles
6	8	30	12	0
6	3.73	30	14	0
6	2.7	30	18	0
6	1.99	30	34	0
6	1.05	30	46	0
6	0.48	30	110	0

Table 4.12: Ruppert's Refinement

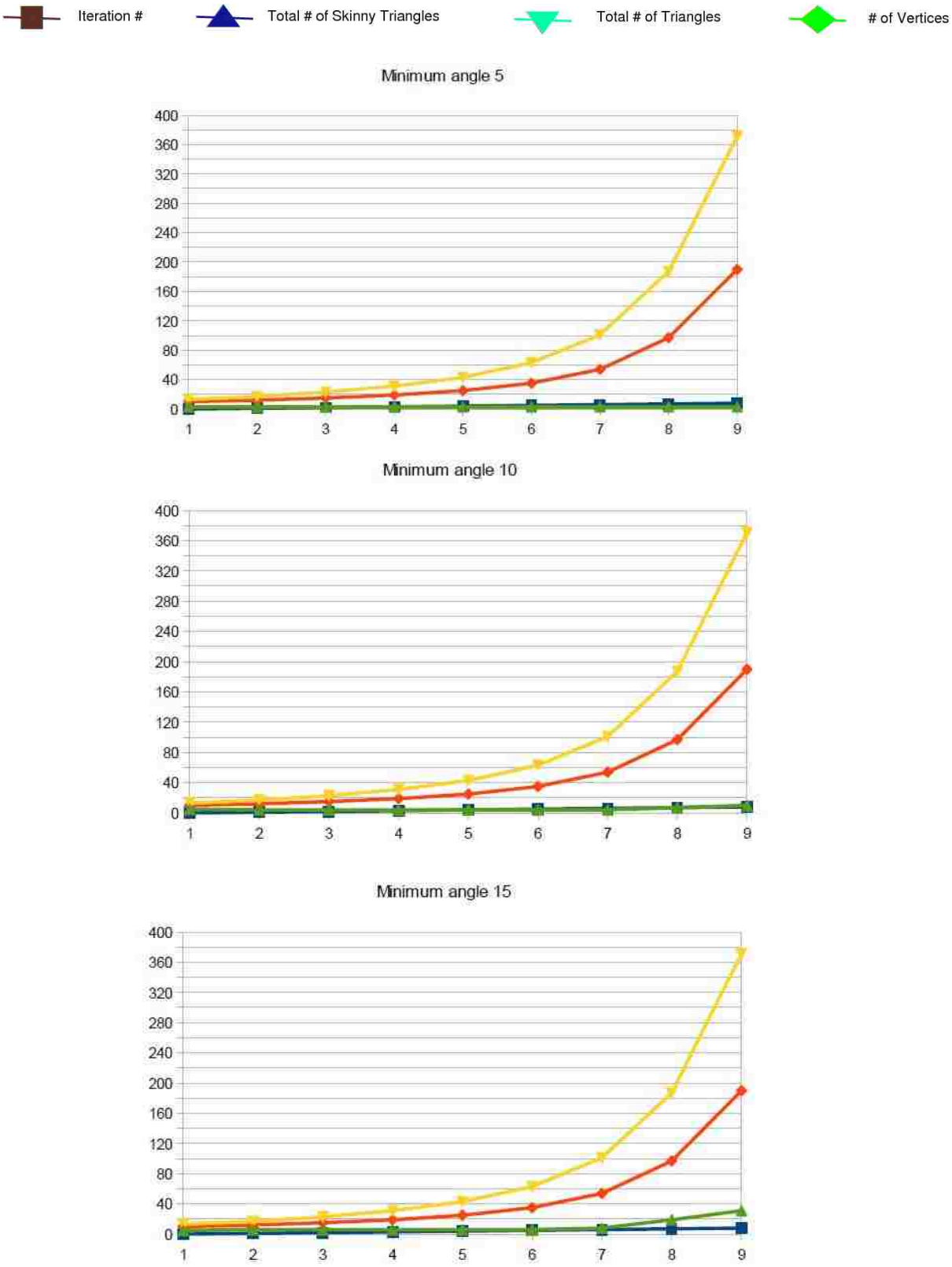


Figure 4.17: CircumCenter Refinement

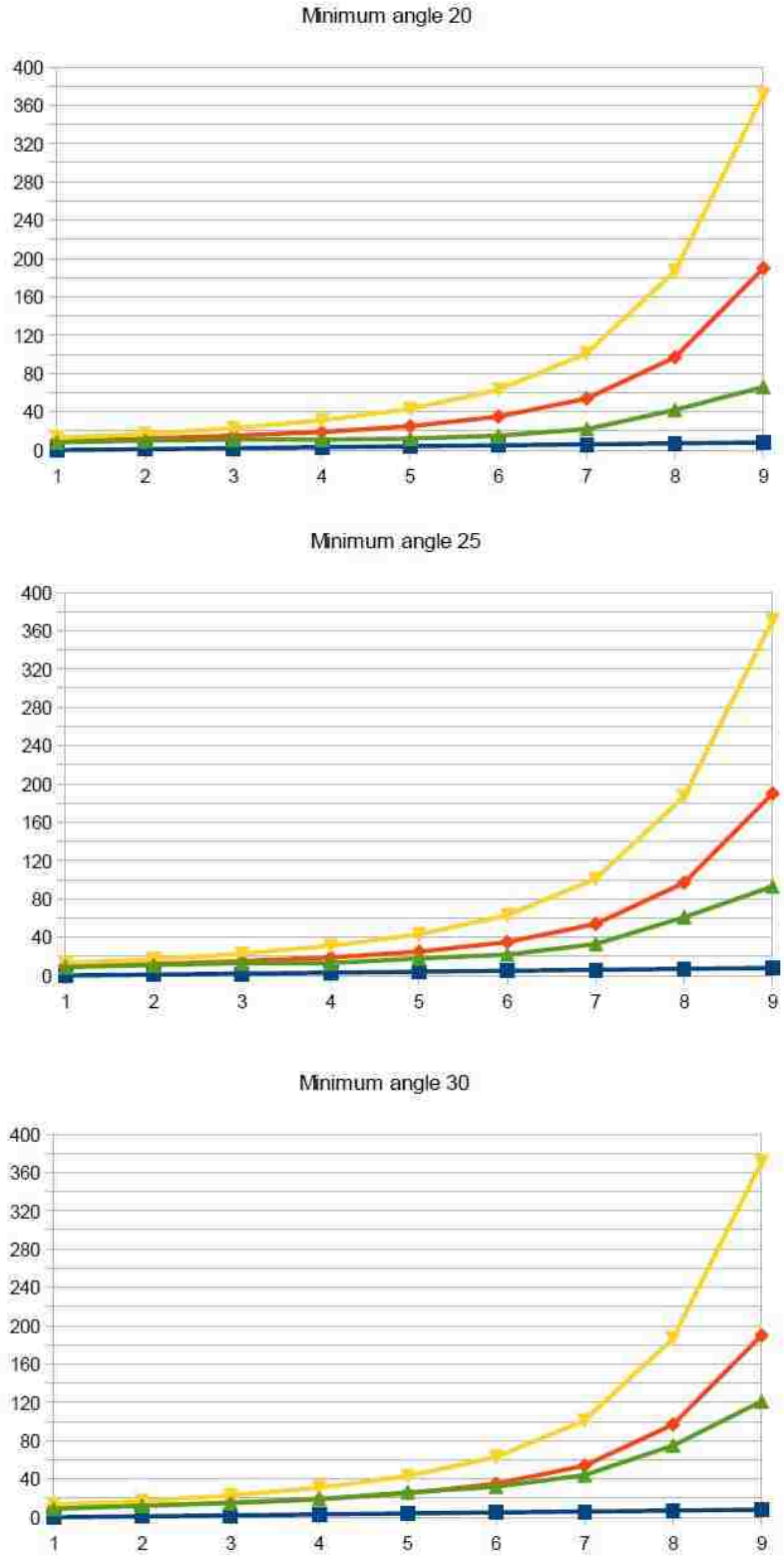


Figure 4.18: CircumCenter Refinement

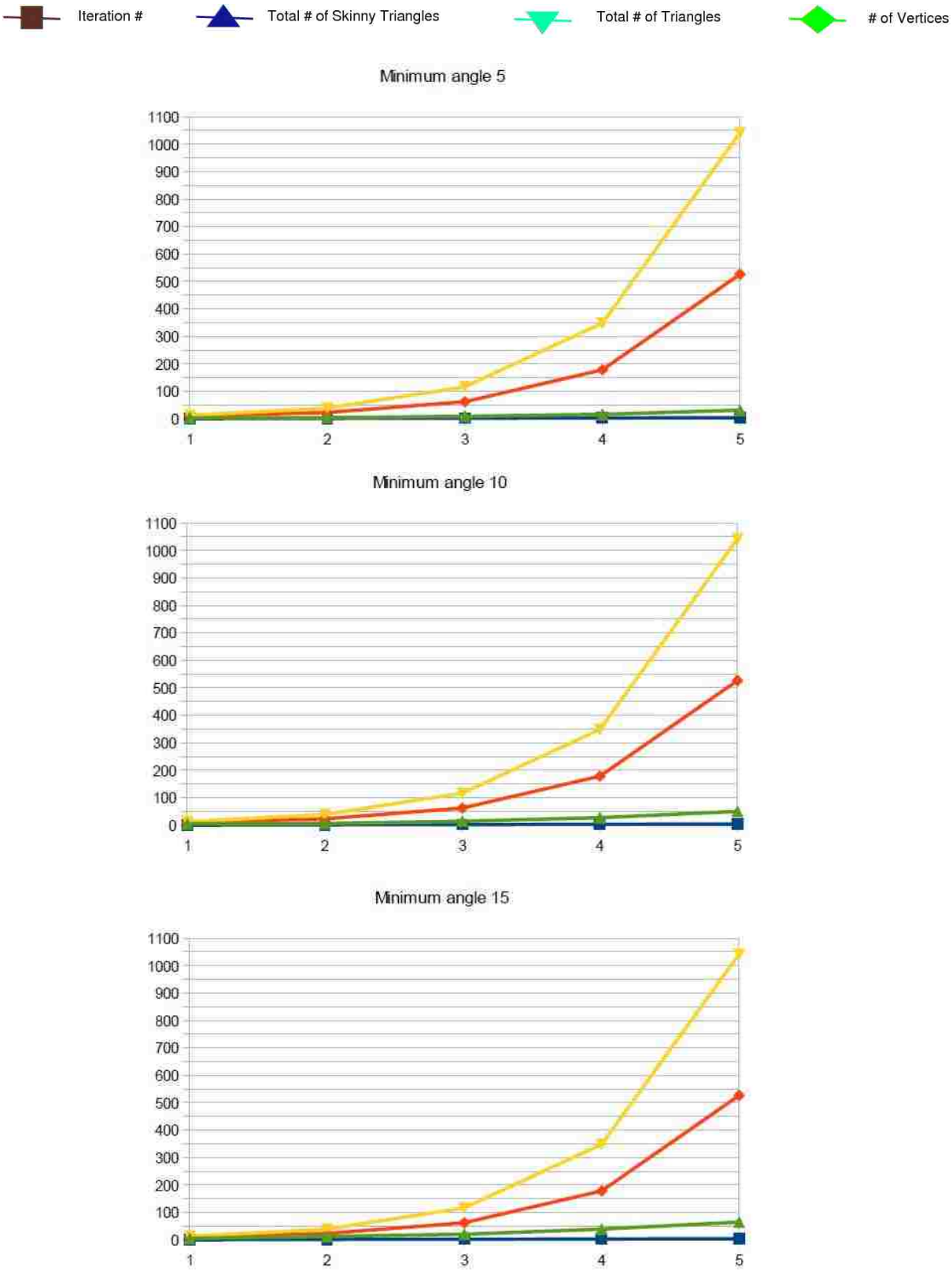


Figure 4.19: Center of Gravity Refinement

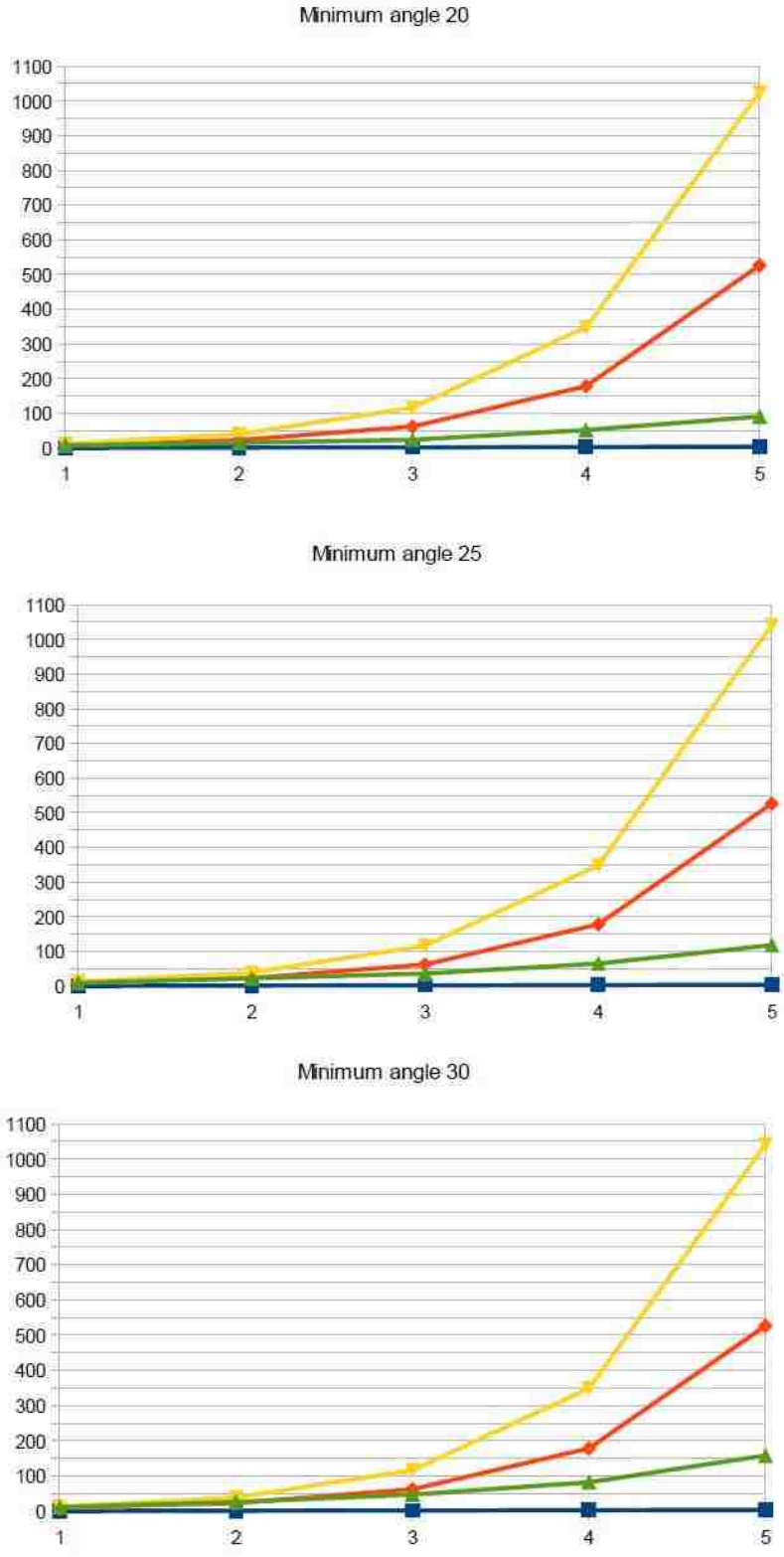


Figure 4.20: Center of Gravity Refinement

Chapter 5

CONCLUSION

In this thesis, we presented a comprehensive review of some of the mesh refinement algorithms which focus on generating quality meshes. Different mesh refinement techniques reviewed include (i) Ruppert's mesh refinement and (ii) quadrangulation guided refinement.

We introduced a new approach for stable mesh refinement by introducing the concept of roaming regions for Delaunay nodes. Based on this idea, we developed two algorithms for refining Delaunay triangulation to make internal nodes more stable. The first algorithm is based on the concept of center of gravity, and the second algorithm uses the notion of largest empty circle. Both algorithms run in $O(n^2)$ time, where n is the initial number of nodes.

We implemented some of the mesh refinement algorithms reviewed in Chapter 2 and 3, including Ruppert's algorithm, quadrangulation refinement, CG based refinement and circumcircle based refinement. The implementation also includes calculating the radial roaming region, lateral roaming region, and the intersection of the two types of roaming regions. The implemented algorithms are used for experimental investigation of quality of mesh elements produced. The experimental results show that the center of gravity approach is very effective in producing quality refinement near the interior region with respect to the convex hull of input nodes. We also found that Ruppert's algorithm produces refinement throughout the region of the convex hull.

Our presented algorithms work only on the interior nodes in the mesh. Due to time limitation, we could not generalize our presented algorithms for non-interior nodes.

It would be very interesting to extend our approach for external nodes as well. The presented algorithms run in $O(n^2)$ time and it may possible to reduce this time complexity by getting more insight into the structure of free regions.

REFERENCES

[1] De Oliveira, Sanderson L. Gonzaga. "A review on delaunay refinement techniques." Computational Science and Its Applications-ICCSA 2012. Springer Berlin Heidelberg, 2012. 172-187.

[2] Du Qiang, Vance Faber, and Max Gunzburger. "Centroidal Voronoi Tessellations: Applications and Algorithms." Society for Industrial and Applied Mathematics. (1999): 637-676.

[3] Fortune Steven. "Voronoi diagrams and Delaunay triangulations." *Computing in Euclidean geometry 4* (1995): 225-265.

[4] Godfried T. Toussaint, "Computing Largest Empty Circles with Location Constraints." *International journal of Computer and Information Science Vol 12 No. 5*, 1983.

[5] Guy E. Blelloch, Gary L. Miller, and Dafna Talmor, "Developing a Practical Projection-Based Parallel Delaunay Algorithm." *Proceedings of the Twelfth Annual Symposium on Computation Geometry (Philadelphia, Pennsylvania)*, pp. 186-195, ACM, May 1996.

[6] Hada Romas James, "Roaming region for Delaunay triangulation" (2011). UNLV Theses/Dissertations/Professional Papers/Capstones. Paper 1242.

[7] Henry Selvaraj, Navin Rongratana, Laxmi Gewali. "Characterizing free-regions of sensor nodes." International Conference on Systems Engineering, pages 375379, 2008.

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

[9] Jan B. Pedersen, N. Rongratana, Laxmi Gewali, "Estimating the free region of a sensor node." JCMCC 74:5364, 2010.

[10] Jim Ruppert, "A Delaunay Refinement Algorithm for Quality 2-Dimensional Mesh Generation." *NASA Ames Research Center, M/S T045(Moffet Field, California)*, January 31, 1994.

[11] L. P. Chew, "Constrained Delaunay triangulations." *Proceedings of the third annual symposium on Computational geometry Pages 215-222 , ACM* May 1987.

[12] Lee Der-Tsai, and Bruce J. Schachter. "Two algorithms for constructing a Delaunay triangulation." International Journal of Computer and Information Sciences 9.3 (1980): 219-242.

[13] M. de berg, M. Van Kreveld, M. overmays, O. Schwarzkopf, *Computational Geometry : Algorithms and Applications*, Springer, 1997

[14] Panagiotis Foteinos, Nikos Chrisochoides. "High-Quality Multi-Tissue Mesh Generation for Finite Element Analysis"

[15] Pav Steven, Walkington, Noel, "Delaunay refinement by corner lopping." *Proceedings of the 14th International Meshing Roundtable*, 2005

[16] Ramaswami Suneeta, Pedro Ramos, and Godfried Toussaint. "Converting triangulations to quadrangulations." Computational Geometry 9.4 (1998): 257-276.

[17] Roylance, David. "Finite Element Analysis." Department of Materials Science and Engineering Massachusetts Institute of Technology, Cambridge, MA 2139 (2001).

[18] Shewchuk, Jonathan, "Delaunay refinement algorithms for triangular mesh generation." *Computational Geometry: Theory and Applications* 22, 2002

[19] Shewchuk, Jonathan R. "Delaunay refinement mesh generation." Carnegie-Mellon University Pittsburgh PA School of Computer Science, 1997.

[20] Shewchuk, Jonathan Richard. "Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator." Applied computational geometry towards geometric engineering. Springer Berlin Heidelberg, 1996. 203-222.

VITA

Graduate College

University of Nevada, Las Vegas

Bishal Acharya

Home Address :

4247 Chatham Cir apt 4

Las Vegas, NV, 89119

Degree : Bachelor of Computer Engineering

Kathmandu Engineering college, Tribhuvan University

Thesis Title : Stability aware Delaunay Refinement

Thesis Examination Committee: Chairperson, Dr. Laxmi Gewali, Ph.D.

Committee Member, Dr. John T. Minor, Ph.D.

Committee Member, Dr. Ajoy K. Datta, Ph.D.

Graduate Faculty Representative, Dr. Rama Venkat, Ph.D.