8-2004

Corrective Greedy Algorithm for Finding the Straight Line Skeleton of a Simple Polygon

Amit Parnerkar

CORRECTIVE GREEDY ALGORITHM FOR FINDING THE STRAIGHT LINE SKELETON OF A SIMPLE POLYGON

by

Amit Parnerkar

B.E., Devi Ahilya University, Indore, India, 1999

A Thesis

Submitted to the Graduate Faculty

of

St. Cloud State University

in Partial Fulfillment of the Requirements

for the Degree

Master of Science

St. Cloud, Minnesota

August, 2004

This thesis submitted by Amit Parnerkar in partial fulfillment of the requirements for the Degree of Master of Science at St. Cloud State University is hereby approved by the final evaluation committee.

[illegible]
Chairperson

[illegible]

[illegible]

[illegible]
Dean
School of Graduate Studies

CORRECTIVE GREEDY ALGORITHM FOR FINDING THE STRAIGHT LINE SKELETON OF A SIMPLE POLYGON

Amit Parnerkar

The purpose of this paper is to introduce an algorithm for finding the ***Straight Line Skeleton of Simple Polygons*** that uses a "*corrective greedy*" approach. The thesis explores the behavior of a simple algorithm that attempts to find the Straight Skeleton using local information only. Since it is well known that local information alone is insufficient to find the Straight Skeleton [1, 9], the algorithm is at times forced to rollback/backtrack some of the computation done. Since it is hard to get an exact theoretical bound on the number of backtrack operations, we have tested the algorithm extensively on randomly generated star-shaped polygons. Our experimental results show that for star-shaped polygons, this approach gives us an $O(n \log n)$ behavior. Implementation for this algorithm is available through an applet at http://web.stcloudstate.edu/rsarnath/skeleton/straightSkeleton.htm.

Straight Skeleton has several applications: layered manufacturing [10], drawing free trees inside rectilinear polygons [11], origami polygon cutting theorem (solve the fold-and-cut problem) [12, 13], label placement algorithms that locate a label at a centroid [14], to construct a polygonal roof over a set of ground walls [7] (see Figure 1.6), reconstruct a geographical terrain from a river map, character recognition (see Figure 1.5), etc.

Straight Skeleton is a concept that was first introduced by Aichholzer in 1995 [1] and is similar to medial axis. Finding medial axis can be achieved in linear time [15], but there are no known linear or nearly-linear algorithms for finding the Straight Skeleton.

Several researchers have proposed algorithms for this problem. The fastest deterministic algorithm is the one by Eppstein and Erickson [5] which runs in $O(n^{1+e} + n^{8/11+e} r^{9/11+e})$ where n is the total number of vertices, r is the number of reflex (non-convex) vertices, and e is an arbitrarily small positive constant. More recently, a slightly faster randomized algorithm using $O(n\sqrt{n} \log n)$ time and O(n) space was proposed by Cheng and Vigneron [6]. Our algorithm is an experimental algorithm and

runs faster than any of the known algorithms for computing Straight Skeleton of Star-Shaped polygon. Figure 5.3 shows the comparison chart for all available algorithms at this time.

Aug 2004
Month Year

Approved by Research Committee:

[illegible]

Ramnath Sarnath Chairperson

ACKNOWLEDGMENT

I am thankful for having the unique experience of working on my Masters thesis at St. Cloud State University and for successfully implementing the project. First and foremost, I would like to recognize the unrelenting support from my advisor Dr. Sarnath Ramnath during all of the work towards this thesis.

Secondly, I wish to thank Dr. Peiyi Zhao, for her timely help, especially for geometry. I would also like to thank Dr. Andrew Anda for his kind help in guiding me in writing the report. Finally, my parents and my brother are always my firmest support through all of my life. I would also give many thanks to them. I sincerely express my gratitude to Dr. Sarnath for giving me an opportunity to work on the excellent concept of straight skeleton.

TABLE OF CONTENTS

LIST OF FIGURES

Figure Page

Chapter 1

INTRODUCTION

The purpose of this paper is to introduce an algorithm for finding the ***Straight Line Skeleton of Simple Polygons*** that uses a "*corrective greedy*" approach. The thesis explores the behavior of a simple algorithm that attempts to find the *Straight Skeleton* using local information only. Since it is well known that local information alone is insufficient to find the Straight Skeleton [1, 9], the algorithm is at times forced to rollback/backtrack some of the computation done. Since it is hard to get an exact theoretical bound on the number of backtrack operations, we have tested the algorithm extensively on randomly generated star-shaped polygons. Our experimental results show that for star-shaped polygons, this approach gives us *O(n log n)* behavior.

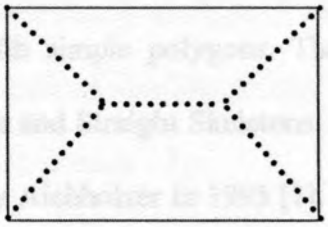
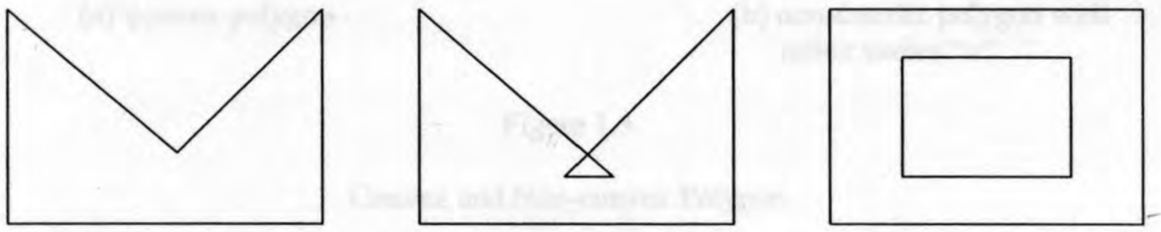


Figure 1.1

A Simple Example of Straight Skeleton for Polygon

A general polygon *P*, is defined by the line segments spanned by *n* points in the Euclidean plane; it is possible that a polygon could have self-crossing segments which is considered to be a non-simple polygon. Furthermore a simple polygon is a polygon without any hole (see Figure 1.2).

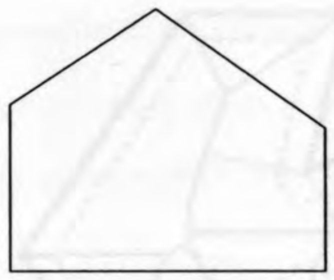


(a) simple polygon (b) a non-simple polygon (c) non-simple polygon (polygon with hole)

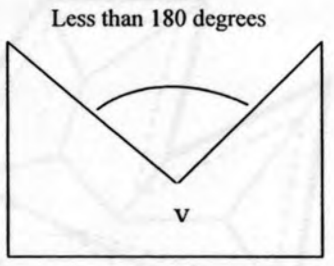
Figure 1.2
Three Polygons

SKELETONS

A Skeleton is a structure used to represent basic two dimensional structures. In this paper we are dealing with simple polygons. There are mainly two types of skeleton structures; medial axis and Straight Skeletons. Straight Skeleton is a concept that was recently introduced by Aichholzer in 1995 [1]. The Straight Skeleton is quite similar to the medial axis; the two are equivalent for convex polygons [16] (see Figure 1.3). However, the Straight Skeleton of a non-convex polygon has lower combinatorial complexity than the medial axis, hence it is preferable over the medial axis.



(a) convex polygon



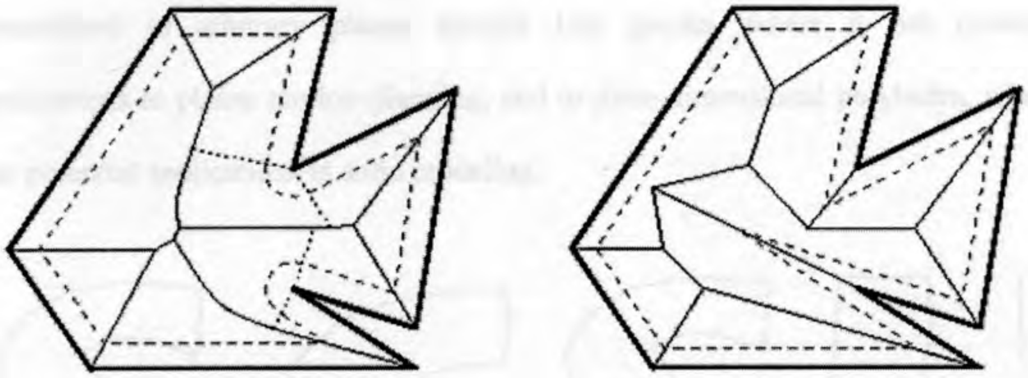
(b) non-convex polygon with reflex vertex "v"

Figure 1.3

Convex and Non-convex Polygon

Medial axis is the most widely used skeleton, which consists of all interior points whose closest point on the polygon's boundary is not unique. The collection of all these points will produce not only straight line segments but also curved arcs (parabolic arcs) in the vicinity of reflex vertices of the polygon (see Figure 1.4a). If the exterior angle for any vertex in a polygon is less than *180* degree then it is called *reflex* vertex (see Figure 1.3b).

The *Straight Skeleton* for a simple polygon, *P*, denoted *S(P)*, can be defined by shrinking all of its edges at a constant rate and tracing the vertices' path (see Figure 1.4b, formal definition is given in section 1.3).



(a) medial axis skeleton (b) Straight Skeleton

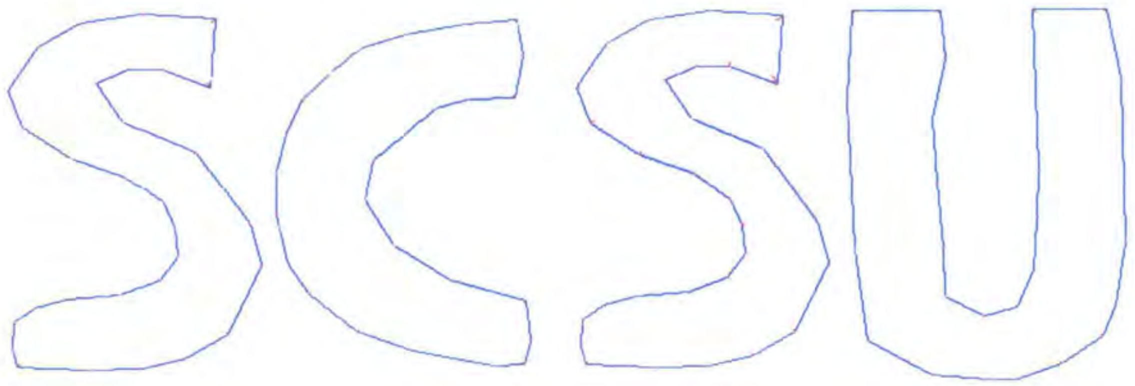
Figure 1.4

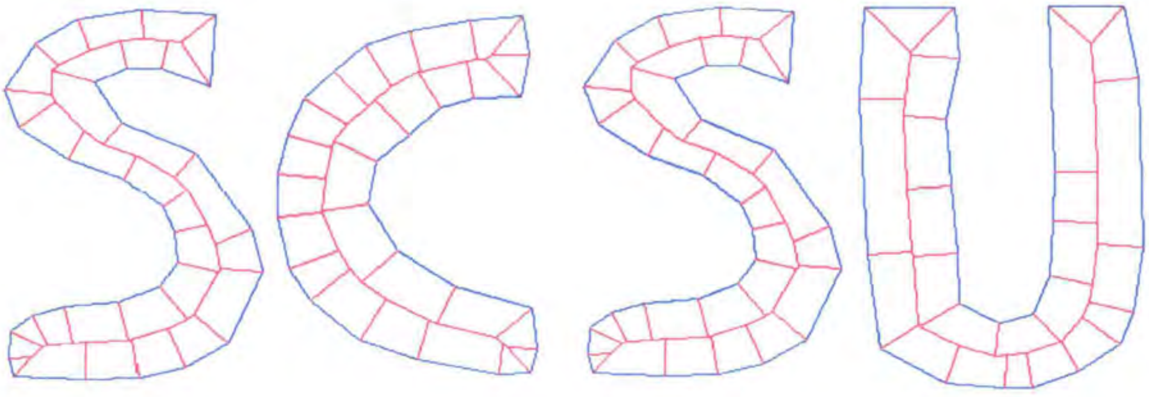
Two Types of Skeletons of the Same Polygon

The advantage of Straight Skeleton, i.e., lower combinatorial complexity, is offset by the fact that fastest known algorithms for finding the Straight Skeleton are much slower and complex than those for finding the medial axis. Finding medial axis can be achieved in linear time [15], but there are no known linear or nearly-linear algorithms for finding the Straight Skeleton.

APPLICATIONS OF STRAIGHT SKELETON

The Straight Skeleton has several applications: layered manufacturing [10], drawing free trees inside rectilinear polygons [11], origami polygon cutting theorem (solve the fold-and-cut problem) [12, 13], label placement algorithms that locate a label at a centroid [14], to construct a polygonal roof over a set of ground walls [7] (see Figure 1.6), reconstruct a geographical terrain from a river map, character recognition (see Figure 1.5), etc. The definition of the Straight Skeleton can be generalized to arbitrary planar straight line graphs, where it has (potential) applications to planar motion planning, and to three-dimensional polyhedra, where it has potential applications in solid modeling.


(a) "SCSU" polygon


(b) skeleton inside polygon

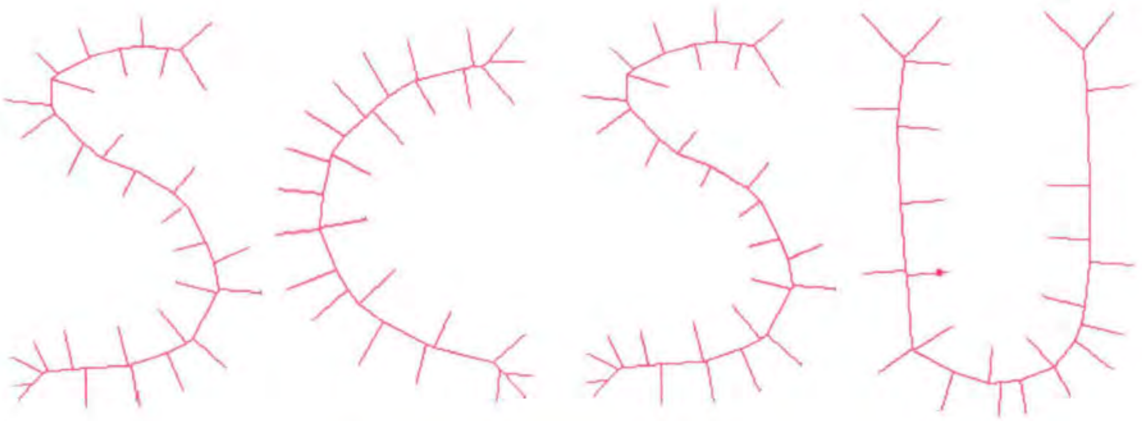
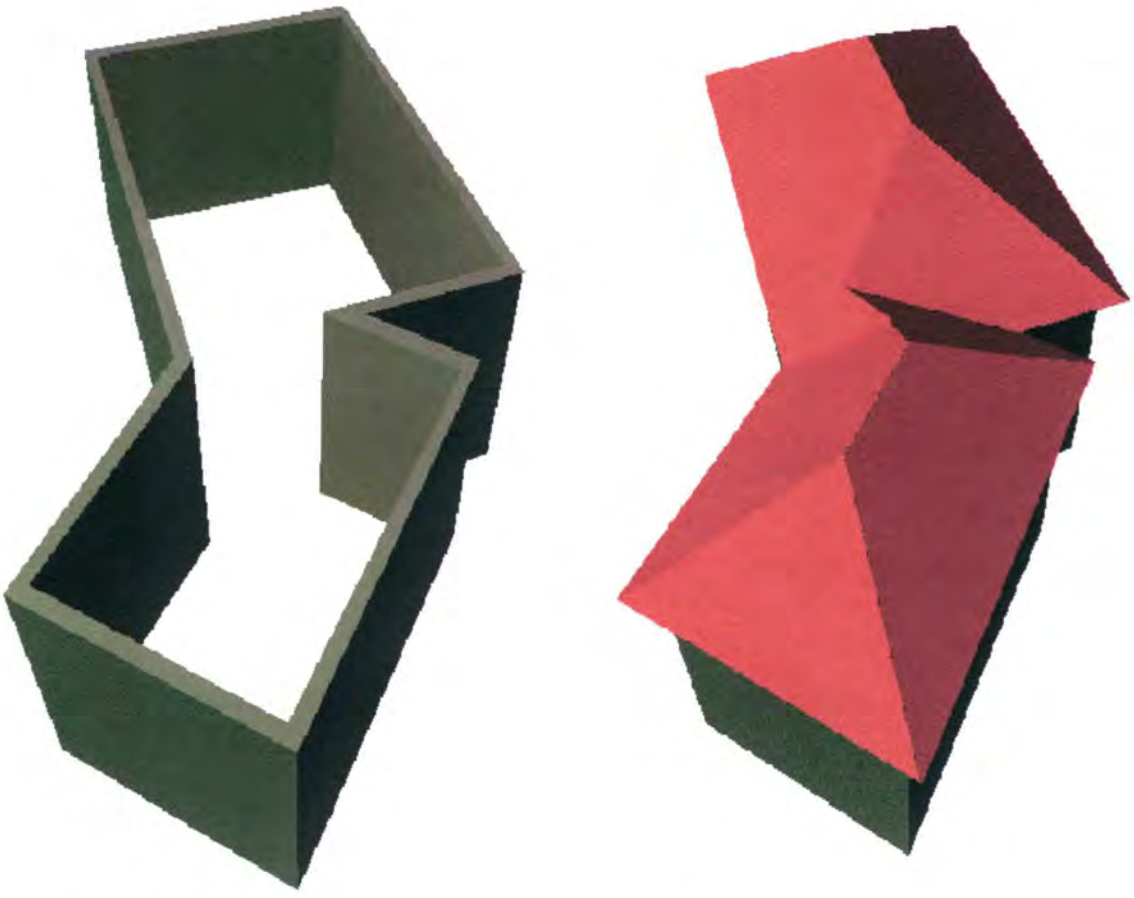

(c) Straight Skeleton for "SCSU" polygon letters

Figure 1.5

Straight Skeleton for the Letters "SCSU"



(a) walls of a house

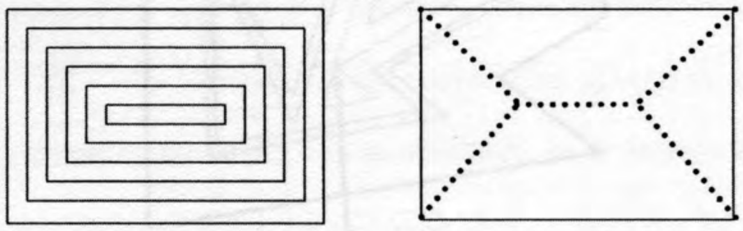
(b) Unique roof such that all faces have *same slope*

Figure 1.6

Ground Walls and Corresponding Roof

STRAIGHT SKELETON CONSTRUCTION PROCESS AND PROPERTIES

The formal definition of Straight Skeleton *S(P)* of polygon *P* is the union of the pieces of angular bisectors found by polygon vertices during the shrinking process (see Figure 1.7). Each edge *e* sweeps out a certain area which is called a plane or face of *e*. Bisector pieces are called *rays*, and their endpoints which are not vertices of *P* are called nodes of *S(P)*. It uniquely partitions the interior of given n-gon *P* into *n* monotone polygons, one for each edge of *P*.



(a) shrinking process (b) Straight Skeleton in dashed line

Figure 1.7

Shrinking Process

Shrinking Process

The shrinking process creates two kinds of events (see Figure 1.8).

1) An *edge event* occurs when an edge length decreases to *0*. The edges neighboring that edge (if they still have a positive length), become adjacent. As a result we get a polygon with one fewer edge.
2) A *split event* occurs when a reflex vertex collides with an edge. This splits the edge and changes adjacencies. As a result we get two polygons which again shrink recursively.

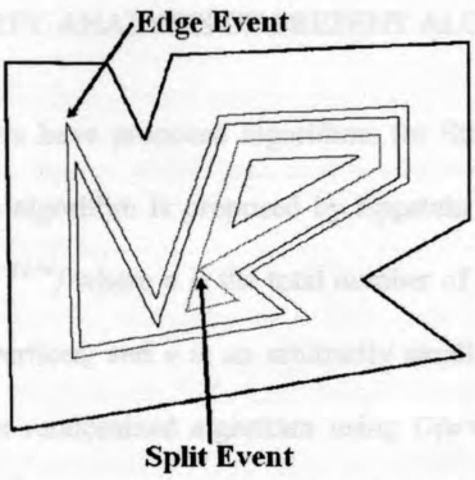


Figure 1.8

Shrinking Process, Edge Event and Split Event

COMPARISON WITH MEDIAL AXIS

If *P* is an n-gon with *r* reflex vertices then *S(P)* realizes in *2n-3* arcs whereas medial axis of *P* realizes *2n–3+r* arcs, *r* of which are parabolically curved. The part of *S(P)* interior to *P* has only *n-2* nodes, whereas medial axis of *P* has *n+r-2* nodes. Thus Straight Skeleton gives lower combinatorial complexity over medial axis representation. For convex polygons both medial axis and Straight Skeleton are the same.

Although the medial axis can be constructed in linear time, the fastest known algorithms for Straight Skeletons are much slower. The main difficulty is that changing the positions or angles of reflex vertices has a significant non-local effect on the skeleton. This non locality makes techniques such as incremental construction or divide-and-conquer fail.

COMPLEXITY ANALYSIS OF PRESENT ALGORITHMS

Several researchers have proposed algorithms for Straight Skeleton problem. The fastest deterministic algorithm is proposed by Eppstein and Erickson [5] which runs in $O(n^{1+e} + n^{8/11+e}r^{9/11+e})$ where n is the total number of vertices, r is the number of reflex (non-convex) vertices, and e is an arbitrarily small positive constant. More recently, a slightly faster randomized algorithm using $O(n\sqrt{n} \log n)$ time and $O(n)$ space was proposed by Cheng and Vigneron [6]. The following table summarizes these results [9].

Time	**Space**	**Reference**
$O(n^2 \log n)$	$O(n)$	[2]
$O(nr \log n)$	$O(nr)$	[4]
$O(n \log n + nr \log(n/r))$	$O(nr)$	[1, 5, 8]
$O(n \log n + nr + r^2 \log r)$	$O(n)$	[2, 5]
$O(n \log n + nr)$	$O(n + r^2)$	[3]
$O(n^{1+e} + n^{8/11+e}r^{9/11+e})$	$O(n^{1+e} + n^{8/11+e}r^{9/11+e})$	[5]
$O(n\sqrt{n} \log n)$	$O(n)$	[6]

The Algorithm presented here employs a "*corrective greedy approach*." This approach was tried earlier by Huang [16], but the implementation encountered anomalies that were not fully addressed.

ANOMALIES IN PREVIOUS APPROACH

The implementation used a roof model (3 dimensional) to construct Straight Skeleton. It considered each edge of the polygon as a plane and considered the lowest height of the three adjacent planes having a constant slope and intersecting locally at the point where the plane or edge can be collapsed. This was enough for general cases, but for some polygons direction of rays plays a key role. Huang's work did not take into consideration a lot of factors which gave rise to two anomalies, (i) Infinite loop and (ii) Self-Intersecting skeleton.

In *Infinite Loop*, the rays will cause an infinite loop in which same set or sets of edges keep getting committed and backtracked because the algorithm has incorrectly determined ray's directions (i.e., growing direction of Straight Skeleton) *Self- Intersection* is caused by directional problem and results in the self-intersection of the Straight Skeleton, where the arcs of skeleton intersect with itself (see Figure 1.9) and results in incorrect Straight Skeleton.

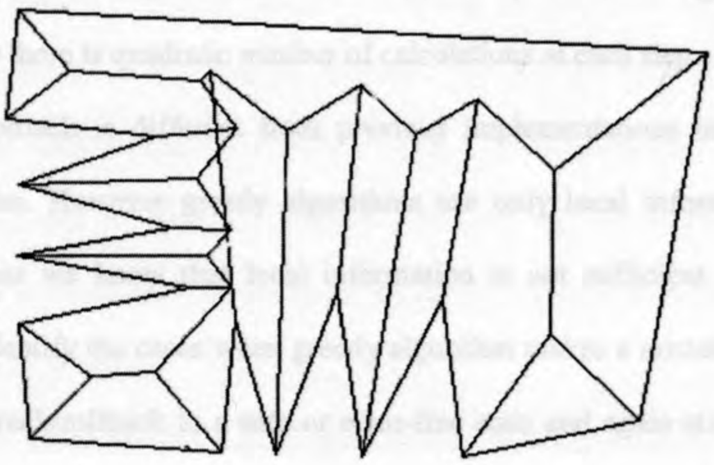


Figure 1.9

Self-intersecting Skeleton Anomaly

(Incorrect Straight Skeleton)

Chapter 2

GOALS OF THE PROJECT

There are very few papers dedicated to the implementation of Straight Skeleton problem, and all these algorithms are complicated since they need to simulate edge events and split events. The split events introduce the element of non-locality into the problem. As can be seen in Figure 2.1 split events can involve an edge and a vertex that are not adjacent. Researchers have tried to attack this problem by tracking all the information simultaneously (i.e., global computation). Such an approach is more expensive since there is quadratic number of calculations at each step.

Our approach is different from previous implementations in that it uses a greedy algorithm. However greedy algorithms use only local information to make decision whereas we know that local information is not sufficient to address this problem. We identify the cases when greedy algorithm makes a mistake and for those cases we backtrack/rollback to a safe or error-free state and again start working in a direction where we will avoid those cases. Thus at the end we get fully constructed Straight Skeleton.

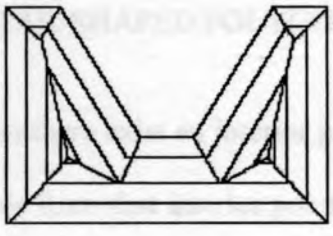


Figure 2.1

Split Event and Edge Event in Shrinking Process

In order to understand the behavior of the *Corrective Greedy Algorithm for Straight Skeleton* better, we have to use fairly complicated polygons which have a large number of reflex vertices. We chose star-shape polygons to test our algorithm for the following reasons:

1) Star-shaped polygons can have as many as *n/2* reflex vertices. Since the complexity of the Straight Skeleton depends heavily on the number of reflex vertices, these polygons serve as a good representative sample.
2) It is easy to generate random instances of star-shaped polygons, as described below. It is not clear that there is such a procedure for arbitrary simple polygons.
3) Star-shaped polygons have been studied in other contexts and have some interesting properties. Although our algorithm does not use any of these properties, it would be interesting to examine, at a later stage, algorithms tailored to computing the skeleton of a star-shaped polygon.

STAR-SHAPED POLYGON

In a star-shaped polygon there exist an interior point such that all the boundary points of the polygon are visible from that interior point. Two points' *p* and *q* are said to be visible if the straight line segment between them does not intersect any edges (see Figure 2.2).

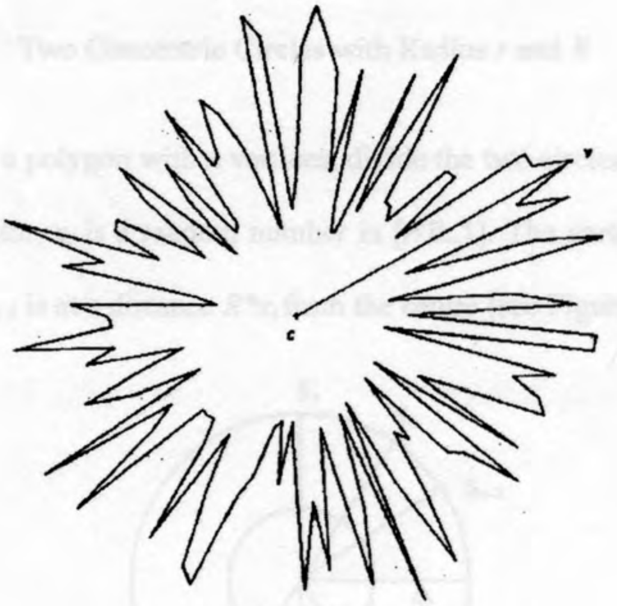


Figure 2.2

Star-shaped Polygon

The star-shaped polygons are randomly generated as follows; consider two concentric circles of radius *R* and *r* respectively. The ratio *r/R*, called the spike value of the polygon, is what decides how sharp the reflex vertices will be.

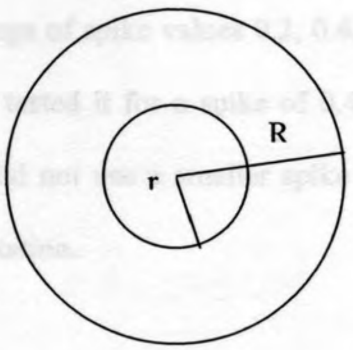


Figure 2.3

Two Concentric Circles with Radius *r* and *R*

To generate a polygon with *n* vertices, divide the two circles into *n* sectors. Let S_i denote the i^{th} sector, x_i is a random number in $[r/R..1]$. The vertex v_0 on the radius separating S_i and S_{i+1} is at a distance $R*x_i$ from the centre (see Figure 2.4).

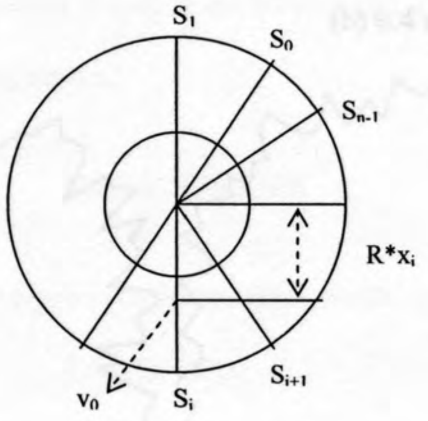


Figure 2.4

Two Concentric Circles with *n* Sectors for Polygon Size *n*

We have built the star-shape polygon generation algorithm in a way that it can generate random star-shape polygon for given polygon size and spike. Though we tested all our tests for the range of spike values 0.2, 0.4, 0.6, 0.8 (see Figure 2.5); for running time calculation we tested it for a spike of 0.4 only, which generates fairly complicated polygons. We did not use a smaller spike since it sometimes generates rounding errors in the computation.

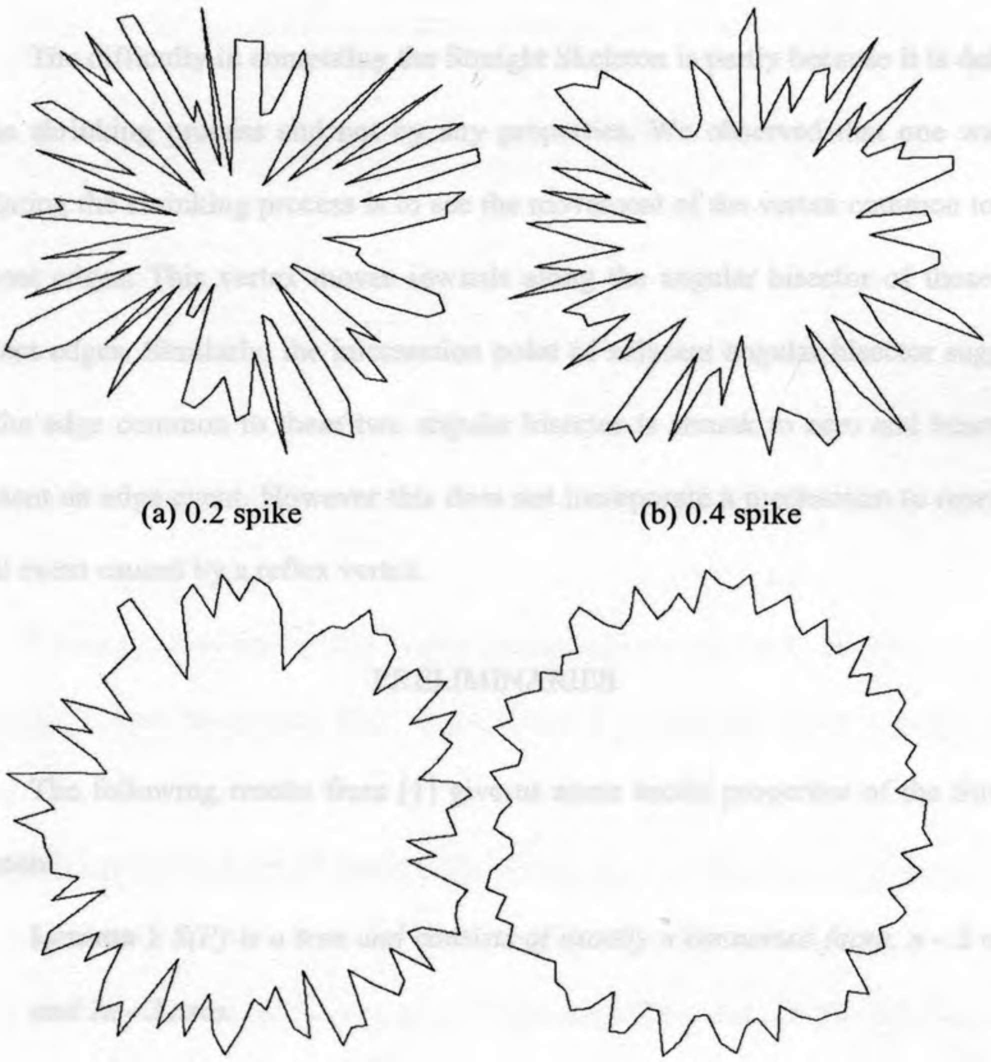


Figure 2.5

Star-shaped Polygon with Different Spikes

Chapter 3

THE APPROACH

The difficulty in computing the Straight Skeleton is partly because it is defined by the shrinking process and not by any properties. We observed that one way of simulating the shrinking process is to see the movement of the vertex common to two adjacent edges. This vertex moves inwards along the angular bisector of those two adjacent edges. Similarly, the intersection point of adjacent angular bisector suggests that the edge common to these two angular bisector is shrunk to zero and hence we represent an edge event. However this does not incorporate a mechanism to represent a split event caused by a reflex vertex.

PRELIMINARIES

The following results from [1] give us some useful properties of the Straight Skeleton:

Lemma 1 *S(P) is a tree and consists of exactly n connected faces, n - 2 nodes and 2n - 3 arcs.*

The construction of a face *f(e)* starts at its edge, *e*, of *P*. The construction of *f(e)* is completed when every part of *e* has shrunk to zero. As *e* cannot reappear again, *f(e)* is connected, and *S(P)* is acyclic.

Two types of arcs of *S(P)* can be distinguished. Each arc is a piece of the angular bisector of two edges *e* and *e'* of *P* or, more precisely, of the lines *l(e)* and *l(e')* supporting these edges. Note that the angular bisector of *l(e)* and *l(e')* actually consists of two lines that intersect at *l(e)∩l(e')*. We single out the one relevant for *S(P)* as follows. Each line *l(e)* defines a halfplane *h(e)* that contains *P* near *e*. One of the bisector lines intersects the wedge *h(e)∩h(e')* while the other avoids it. We call the former the bisector of the edges *e* and *e'*. An arc *a* defined by this bisector is called a convex arc or a reflex arc depending on whether its wedge contains *e* and *e'* in its boundary or not. We also consider *a* as labeled by the ordered pair (*e, e'*). The order reflects the side of a where *l(e)* contributes to the boundary of the wedge.

Each convex (reflex) vertex of *P* obviously gives rise to a convex (reflex) arc of *S(P)*. While convex arcs can also connect two nodes of *S(P)* this is impossible for reflex arcs.

Lemma 2 *Reflex arcs of S(P) only emanate from reflex vertices of P.*

Let *vu* be an arc emanating from some vertex *v* of *P*. Then *u* is a node which corresponds either to an edge event or to a split event. It suffices to show that, after the event, *S(P)* continues at *u* with convex arcs only.

In the former case, let *vw* be the vanishing edge. Since the arc *wu* meets *vu* at *u*, *u* is a convex vertex of the shrunk polygon after the event. In the latter case the polygon splits at *u*. It is obvious that, after that event *u* is a convex vertex of both new polygons.

Thus each new vertex generated during the shrinking process is convex.

INTRODUCTION TO THE CORRECTIVE GREEDY ALGORITHM

As discussed before the intersection point of adjacent angular bisector of two edge sequence suggests that the edge common to these two bisector is shrunk to zero. The idea for the Corrective Greedy Algorithm is simple; it looks for these intersection points to see which edge should collapse first (find local minimum) and at the end connect all these points together to form the Straight Skeleton. However since greedy algorithm uses only local information we can represent only edge events, in the case of split events the algorithm has to backtrack or undo the work it has done previously until it reaches a safe state and then start again.

Local Minimum

A *local minimum* is defined as the pair of edge sequence which can be combined to represent edge event. To find the local minimum for a region, start with an edge sequence say E_{12} and its angular bisector b_{12} (see Figure 3.1); now consider the intersection of adjacent angular bisectors b_{n1}, b_{23} corresponding with edge sequence E_{n1} and E_{23}. Check to see which neighbor is intersecting b_{12} closer to its origin (say b_{23} in this case). Now consider b_{23} and its neighbor sequences b_{12} and b_{34}. Note in Figure 3.1 b_{12} is still intersecting b_{23} closer to its origin when compared with the intersection point formed by b_{34} and b_{23}, so report [E_{12}, E_{23}] as local minimum, else if b_{34} intersects b_{23} closer to its origin then check recursively local minimum using E_{34} as start sequence.

Corrective Greedy Algorithm

The algorithm starts with n rays, one from each vertex of the polygon. The algorithm has to find all nodes of the skeleton. At each step, one more node is added by combining two adjacent rays. The rays to be combined are chosen so that their point of intersection is a local minimum. (In Figure 3.1 b_{12} & b_{23} intersect at a local minimum since rays b_{12} & b_{23} reach P before intersecting any other ray.) P becomes a node and we replace b_{12} & b_{23} by b_{13} and continue the process.

Backtrack occur in situations like one in Figure 3.2, when b_{34} and b_{45} were first combined to get b_{35}. Later, b_{12} & b_{23} were combined and we find that the combination of b_{34} & b_{45} has to be undone, because the origin of b_{35} and right end point of edge e_3 (common to b_{13} & b_{35}) are on different sides of b_{13}. Backtracking is further discussed in Chapter 4.

Clearly the time taken by this process depends on how often we have to undo these intersections. The current research has fully implemented this algorithm and attempted to find a relationship between the number of undo (backtrack) operations m, and the number of vertices n by testing the algorithm on several instances.

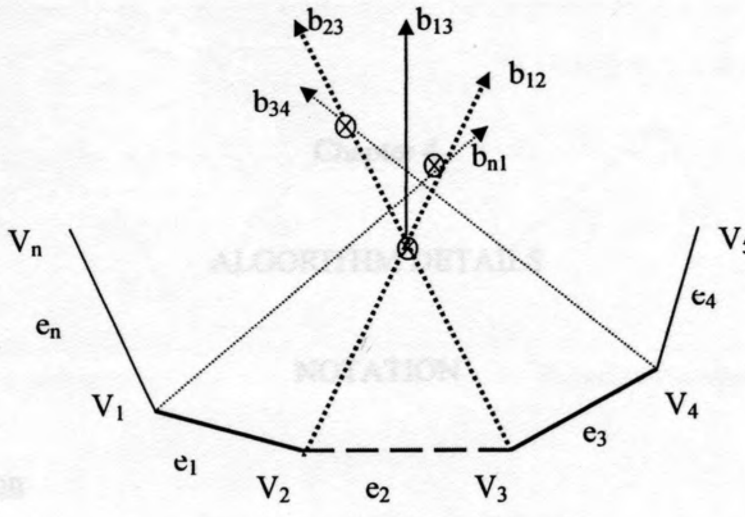


Figure 3.1

Construction Process

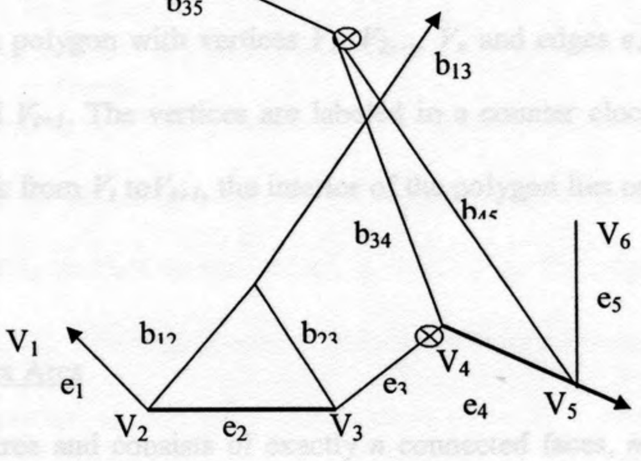


Figure 3.2

Backtrack Condition

Chapter 4

ALGORITHM DETAILS

NOTATION

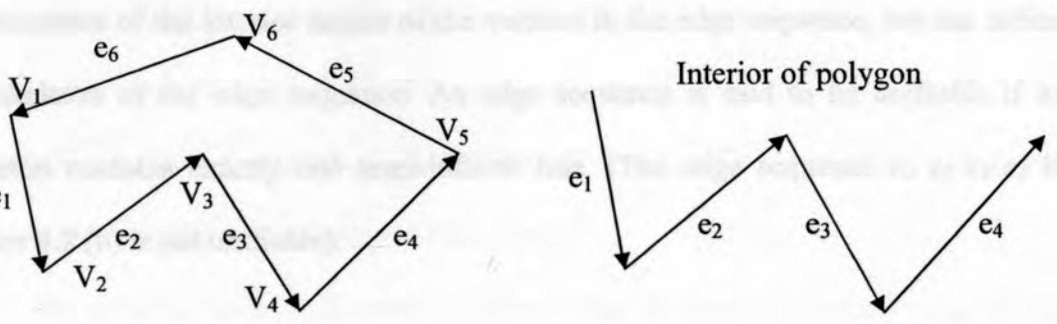
Simple Polygon

A polygon P in which no two non-consecutive edges intersect is called simple polygon. Simple polygons can be further classified as convex and non-convex polygon. A polygon P is convex if and only if all the exterior angles are greater than *180 degrees*, otherwise it is a non-convex or concave polygon.

Let P be a polygon with vertices $V_1, V_2..., V_n$ and edges $e_1, e_2..., e_n$ where e_i connecting V_i and V_{i+1}. The vertices are labeled in a counter clockwise direction so that when we walk from V_i to V_{i+1}, the interior of the polygon lies on the left hand side Figure 4.1 (b).

Convex and Reflex Arcs

$S(P)$ is a tree and consists of exactly n connected faces, n-2 nodes and $2n$-3 arcs, Lemma 1[1]. Arc is a piece of angular bisector of two edges e and $e`$ of P. $S(P)$ can have two types of arcs convex and reflex. Reflex arcs of $S(P)$ only emanate from reflex vertices of P, Lemma 2[1].



(a) polygon (b) edge sequence

Figure 4.1

Polygon and Edge Sequence

Edge Sequence

An edge sequence of the polygon is any subset of contiguous edges. At the start, we have *n* edge sequences e_1e_2, e_2e_3..., e_ie_{i+1}..., e_ne_1. Associated with any edge sequence e_ie_{i+1}... e_j, we have a bisector b_{ij}, which is the bisector of the angle defined by the rays $\overrightarrow{e_{i+1}e_i}$ and $\overrightarrow{e_{j-1}e_j}$. At each iteration we try to reduce appropriately the number of edge sequences by choosing some pair of adjacent edge sequences, say, $e_ie_{i+1}...e_j$ and e_je_{j+1}... e_k which are then merged as e_i... $e_{j-1}e_je_{j+1}$... e_k. The edge sequence $e_i...e_j$ is also represented as E_{ij}. The edge sequence is called *ground edge sequence* if the origin of the angular bisector of that edge sequence is same as a vertex of the original polygon.

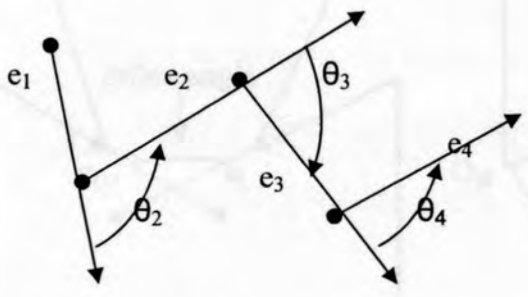
Unifiability of Edge Sequences

Consider the edges e_i, e_{i+1}..., e_j; if the edges e_i and e_j are extended (either as semi infinite line or until they intersect) we induce a partitioning of the plane. Using the bisectors of the interior angles of the vertices in the edge sequence, we can define the skeleton of the edge sequence. An edge sequence is said to be *unifiable* if its skeleton contains exactly one semi-infinite line. (The edge sequence $e_1\ e_2\ e_3\ e_4$ in Figure 4.2 (b) is not unifiable).

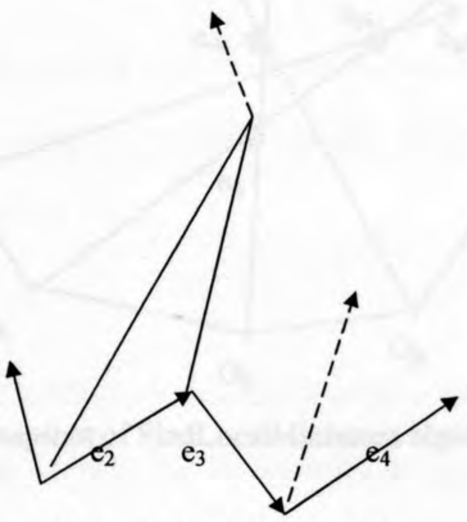
Wrap of an Edge Sequence

The wrap of an edge sequence $e_i ..., e_j$ is defined as the sum $\sum^{j}_{k=i+1} \theta_k$, where θ_k is the signed exterior angle at vertex V_k. The sign of θ_k is positive if the angle as shown in the figure is counter clockwise and negative otherwise.

Lemma 4.1. Let α be any arc of the Straight Skeleton, such that α bisects the angle formed by e_i and $e_j, j>i$, then either e_i and e_j are adjacent edges or the wrap of the sequence $e_i ... e_j$ lies between 0 and 2π.



(a) edge sequence and wrap



(b) Straight Skeleton of (a)

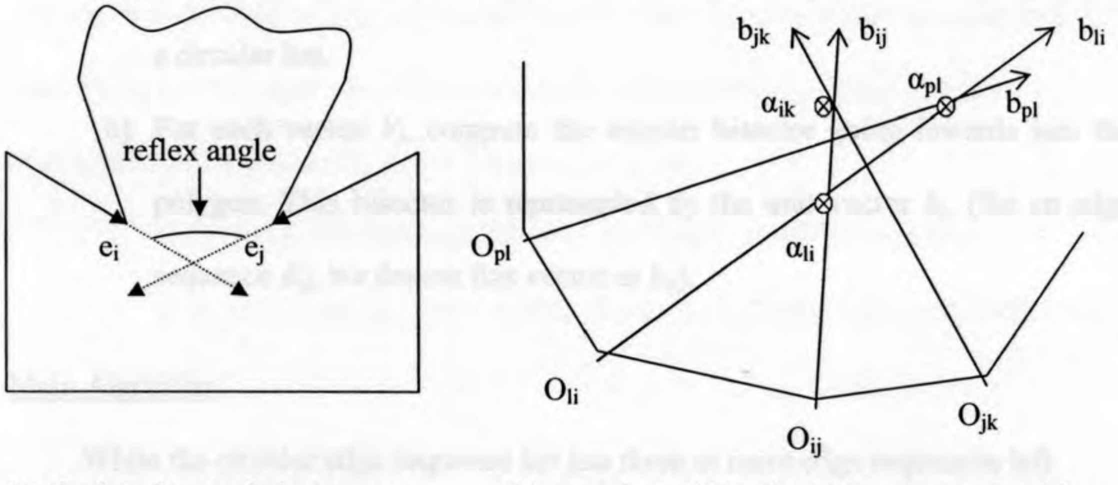
Table 4.2

Edge Sequence and Partial Straight Skeleton of Polygon

Proof. If e_i and e_j are adjacent then α obviously belongs to the Straight Skeleton. It is easy to verify that if the wrap is less than 0, the interior angle defined by the rays $\overrightarrow{V_{i+1}V_i}$ and $\overrightarrow{V_jV_{j+1}}$ is reflex. Thus it follows from Lemma 2 that the bisector of e_i and e_j can not belong to the Straight Skeleton.

On the other hand if the wrap is greater than 2π we have situation like one in Figure 4.3 (a) In this case the wrap of the edge sequence $e_j, e_{j+k}..e_i$ is less than 0, i.e., the interior angle formed by the rays $V_{i+1}V_i$ and V_jV_{j+1} is reflex and therefore this bisector can not belong to this partial Straight Skeleton **(end of proof).**

The above lemma shows that edge sequence E_{ij} is unifiable if the wrap lies in the range (i.e., lies between 0 and 2π). We say that two adjacent edge sequences E_{ij} and E_{jk} are unifiable if E_{ik} is unifiable.



(a) the interior angle between e_i and e_j is reflex

(b) snapshot of FindLocalMinimum algorithm

Figure 4.3

The Interior Angle Between e_i and e_j is Reflex

The entire algorithm can therefore be summarized as follows:

1) At the start we have n edge sequences e_1e_2, e_2e_3..., e_ie_{i+1}..., e_ne_1 each consisting of two edges.
2) At each step we identify two adjacent edge sequences that can be combined, taking into account the wrap and any backtracks that need to be performed.
3) The algorithm terminates when we are left with *three unifiable* edge sequences.

DETAILS OF THE ALGORITHM

Initialize Data Structures

a) Store edge sequences E_{12}, E_{23}..., E_{n1} corresponding to vertex V_1, V_2..., V_n in a circular list.
b) For each vertex V_i, compute the angular bisector going inwards into the polygon. This bisector is represented by the unit vector b_i. (for an edge sequence E_{ij}, we denote this vector as b_{ij}).

Main Algorithm

While the circular edge sequence list has three or more edge sequences left

a) Find a pair of adjacent edge sequences $[E_{ij}, E_{jk}]$ such that the intersection of the half-lines b_{ij} and b_{jk} forms a local minimum, as explained in Algorithm FindLocalMinimum (4.2.3).

b) Check if the combination of b_{ij} and b_{jk} causes self-intersection, as explained in Algorithm Remove Self-intersection (4.2.4).

c) Check if the resultant bisector b_{ik} causes backtrack to its neighboring edge sequences as explained in Algorithm Check Backtrack (4.2.5).

d) Combine E_{ij} and E_{jk} to form one edge sequence E_{jk}. Remove E_{ij} and E_{jk} and insert E_{ik} into list of edge sequence.

Algorithm FindLocalMinimum (E_{ij})

Edge sequence E_{ij} has bisector b_{ij} and O_{ij} as the origin of bisector. Consider the left and right neighbor sequences of E_{ij} as E_{li} and E_{jk} with b_{li}, b_{jk} as bisectors and O_{li}, O_{jk} as their origin (see Figure 4.3 b).

Compute the intersection point of bisectors b_{ij} and b_{jk} as α_{ik} and b_{li} & b_{ij} as α_{li} Assume without loss of generality that distance from O_{ij} to α_{li} is less than distance from O_{ij} to α_{ik} (the other case can be solved in similar fashion) which can also be represented as *Distance* $(O_{ij}, \alpha_{li}) <$ *Distance* (O_{ij}, α_{ik}).

1. If E_{ij} and E_{li} are not unifiable then call FindLocalMinimum (E_{li})
2. If E_{ij} and E_{li} are unifiable, let E_{pl} denote the left edge sequence of E_{li}.
    - 2.1 E_{pl} and E_{li} are not unifiable then report the combination [E_{li}, E_{ij}] as local minimum.
    - 2.2 Otherwise, if E_{pl} and E_{li} are unifiable then check to see if *Distance* $(O_{li}, \alpha_{pl}) >$ *Distance* (O_{li}, α_{li}) if it is, then report the combination [E_{li}, E_{ij}] as local minimum, else call FindLocalMinimum (E_{pl}).

<u>Algorithm RemoveSelfIntersection</u> (E_{ij})

Let $e_3, e_1, ..., X, e_2, e_4$ be the edge vectors, and the bisector of E_{13} causes self intersection (see Figure 4.6).

Unify the edge vectors $e_3, e_1 ..., X$ before we include e_2

a) Check to see if edge sequence formed by $e_1, ..., X$ is unifiable or not, If they are not unifiable then undo X and repeat this process until they are unifiable

b) Try to unify in between e_3 and X until e_1 gets included (at this point we are out of self intersection)

<u>Algorithm CheckBacktrack</u> (E_{ij})

Check if edge sequence E_{ij} is backtracking its neighbors E_{li} and E_{jk} by performing two tests given below, if both are successful then backtrack detected. Below are the steps for checking if E_{ij} is backtracking E_{jk} or not. (Similar procedure can be followed for checking backtrack for E_{li})

a) Perform the half-plane test

i. Take left most edge vector of E_{jk} which is the edge common between E_{ij} and E_{jk} and take its right point, we call it V_m. Consider the two half planes defined by the line containing b_{ij}. If the origin of b_{jk} and V_m lie in different half planes, we say the half plane test is successful.

b) Local minimum test

   i. Find out the local minimum with respect to E_{ij} if it returns E_{li} and E_{ij} then check to see if the resultant of $[E_{li}, E_{ij}]$ is intersecting with b_{jk}, if it is intersecting, then calculate the intersection point of $[E_{li}, E_{ij}]$ bisector with b_{ij} and b_{jk}. Checks to see which one is closer to origin of $[E_{li}, E_{ij}]$; if b_{jk} is closer then there is no backtrack. If any of the above condition fails then backtrack is detected.

   ii. If FindLocalMinimum(E_{ij}) does not return $[E_{li}, E_{ij}]$ as combination then backtrack detected.

<u>Algorithm UndoEdgeSequence</u> (E_{ij}, d)

Undo edge sequence E_{ij} to depth d. While we reach depth d do the following steps,

a) Remove edge sequence E_{ij} from the list and take out its left and right children as E_{li} and E_{jk} and insert them in the list.

b) Check to see if E_{li} and E_{jk} is causing any backtrack to their neighbor

c) Decrement depth d

d) Repeat this process until d reaches *zero*.

<u>Algorithm RemoveInfiniteLoop</u> ()

Sometimes infinite loop is detected when findLocalMinimum fails to return a combination of edge sequence and goes to infinite recursive mode. It usually happens because we get a combination of alternate reflex and convex arcs (always an even number of arcs e.g., *4, 6, 8*). Once this scenario is detected, the following steps can be used to remove it.

1. Combine the first reflex arc with corresponding convex arc and repeat this process until all the combination of reflex and convex arc got combined.

Algorithm is UnifiableEdgeSequence (E_{ij}, E_{jk})

This process will test the unifiability of two edge sequences E_{ij} and E_{jk} based on two tests given below.

a) Check to see if the bisectors (which are unit vector going inwards in polygon) b_{ij} and b_{jk} meets in forward direction of there origin.

b) Check to see if the wrap or angle θ is in between valid limits (i.e., lies between 0 *and* 2π) for E_{ij} and E_{jk}.

BACKTRACK EXAMPLE

Let E_{24} is the newly combined edge sequence and hence b_{24} is the newly formed bisector. As we are maintaining backtrack free invariant, after every combination of edge sequence the algorithm will check if the newly formed bisector is backtracking any of its immediate neighbor, in this case left and right bisectors are denoted by b_{12} and b_{46}. The left edge sequence is *ground sequence* hence there is no need to check for backtrack but for right edge sequence b_{46} the algorithm will work as explained in Algorithm Check Backtrack (4.2.5).

Origin of b_{46} bisector and the right point V_5 of leftmost edge vector e_4 of edge sequence E_{46} lies on different half planes of bisector b_{24}, so half-plane test is successful (Figure 4.4).

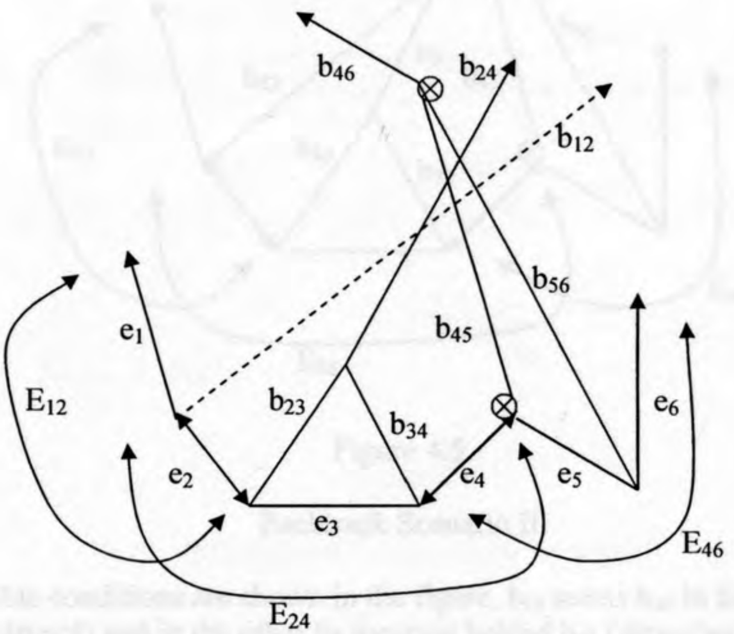


Figure 4.4

Backtrack Scenario I

(Shows half-plane test is successful in this scenario, as origin of b_{46} and right point of leftmost edge vector of E_{46} lies on different half-plane of b_{24})

After half-plane test the algorithm test for the second condition is shown in Figure 4.5 The local minimum corresponding to b_{24} returns b_{14} as the resultant. Now check to see if b_{14} and b_{46} are intersecting or not. If b_{14} and b_{46} formed valid intersection then no backtrack detected otherwise backtrack is detected and b_{46} needs to be undone till we get b_{45} and b_{56}.

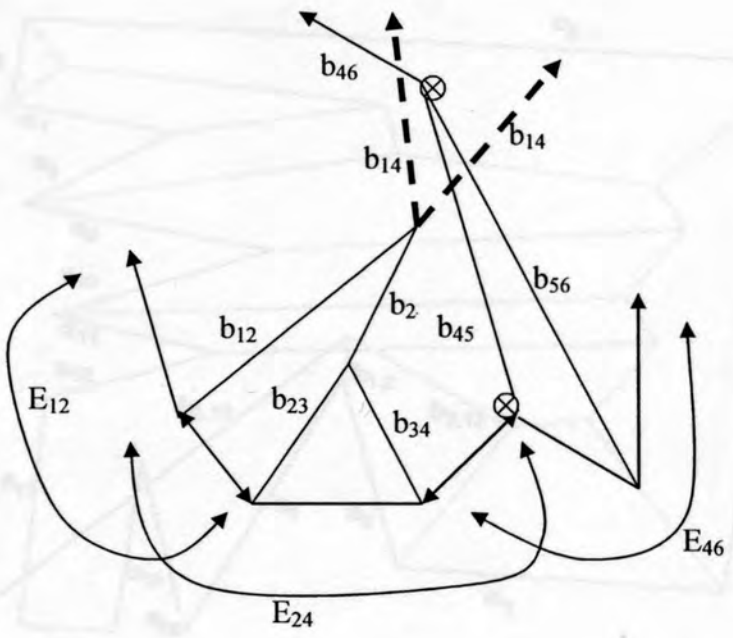


Figure 4.5

Backtrack Scenario II

(Two possible conditions are shown in the figure, b_{14} meets b_{46} in forward direction (no *backtrack*) and in the other its meeting behind b_{46} (*detecting backtrack*))

SELF-INTERSECTION EXAMPLE

A *self-intersection* occurs in a situation like the one in Figure 4.6. Here $b_{2,12}$ is ray emerging from the skeleton of $E_{2,12}$ and $b_{1,2}$ is the ray emerging from the skeleton of $E_{1,2}$. $b_{2,12}$ intersects the skeleton of $E_{2,12}$ before it meets $b_{1,2}$. Here part of the skeleton of $E_{2,12}$ has to be undone. This situation suggests that edge e_{12} should be included after we include edge e_1 in the skeleton for $E_{1,12}$.

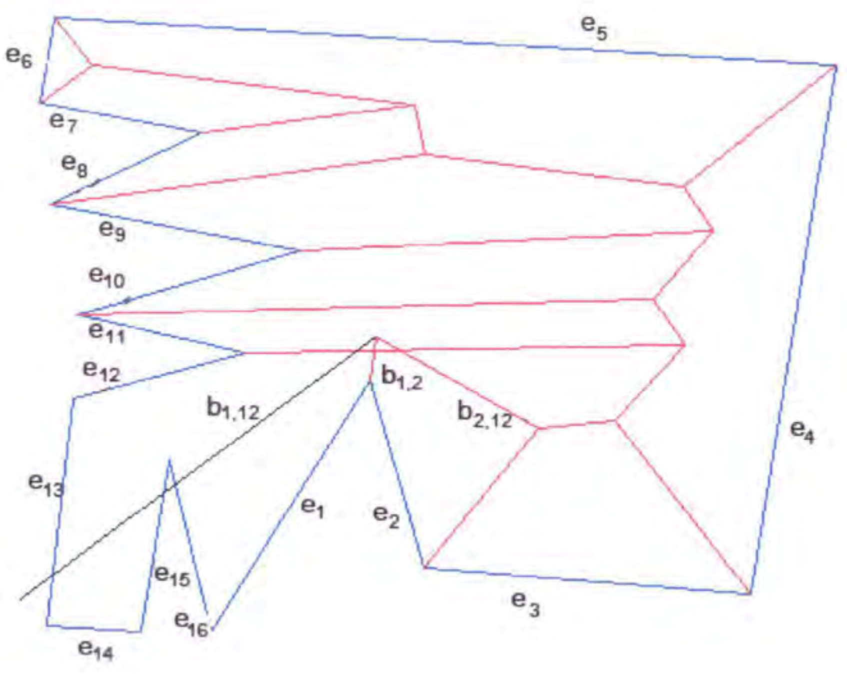


Figure 4.6

Self-intersecting Skeleton Example

(bisector $b_{2,12}$ is intersecting one of its own bisectors arcs previously formed)

SNAPSHOTS OF CONSTRUCTION PROCESS

This section explains the skeleton construction process by taking snapshots of a simple polygon. The polygon is labeled in counter clock-wise direction as a sequence of edge vectors $e_1,..e_{16}$.

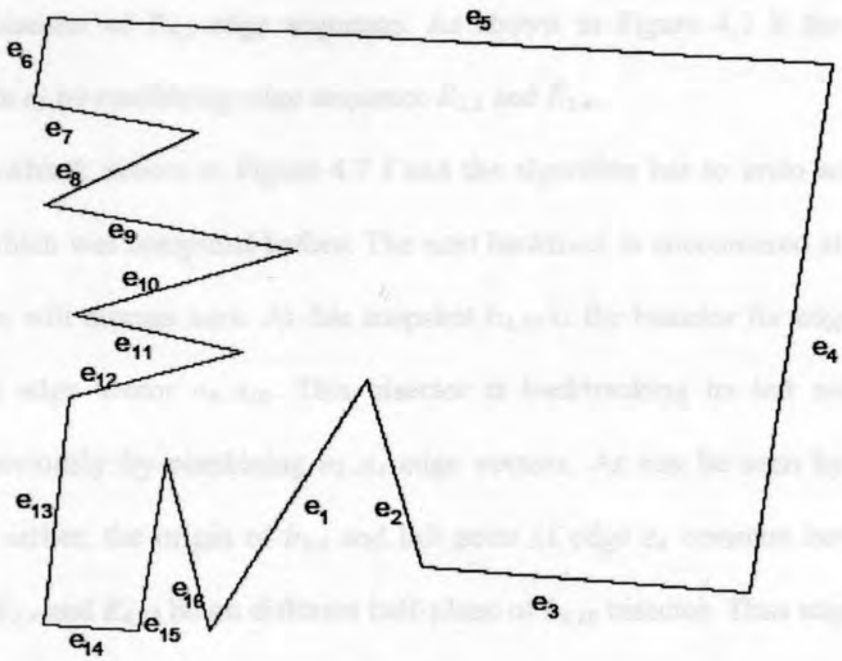


Figure 4.7 a
Polygon Snapshot
(construction process snapshots are shown below)

The corrective greedy algorithm starts finding local minimum using $E_{1,2}$ (edge vector e_1, e_2) as reference edge sequence. $E_{1,2}$ is not unifiable with its left neighbor ($E_{16,1}$) and right neighbor ($E_{2,3}$), so findLocalMinimum algorithm shifts one edge sequence in right (counter clock-wise) direction with $E_{2,3}$ as new reference edge sequence. Again the left neighbor of $E_{2,3}$ is not unifiable but right neighbor $E_{3,4}$ is unifiable. To find the local minimum check to see if $E_{2,3}$ and $E_{3,4}$ are local minimum, this can be done by recursively calling findLocalMinimum with $E_{3,4}$ as reference edge sequence, if it still returns $E_{2,3}$ and $E_{3,4}$ then it is indeed the local minimum. In this case as we can see that angular bisector of $E_{2,3}$ is intersecting $E_{3,4}$ closer to its origin than the bisector of $E_{4,5}$ edge sequence. As shown in Figure 4.7 b the first edge collapsed is e_3 by combining edge sequence $E_{2,3}$ and $E_{3,4}$.

Backtrack occurs in Figure 4.7 f and the algorithm has to undo some part of skeleton which was computed before. The next backtrack is encountered at Figure 4.7 j which we will discuss here. At this snapshot $b_{4,10}$ is the bisector for edge sequence combining edge vector $e_4..e_{10}$. This bisector is backtracking its left neighbor $b_{2,4}$ formed previously by combining $e_2..e_4$ edge vectors. As can be seen by backtrack definition earlier, the origin of $b_{2,4}$ and left point of edge e_4 common between edge sequence $E_{2,4}$ and $E_{4,10}$ lie on different half-plane of $b_{4,10}$ bisector. Thus edge sequence $E_{2,4}$ has to be undone as shown in Figure 4.7 k

Figure 4.7 o shows self-intersection which was discussed earlier. Thus the algorithm in this case computed 7 extra computation because of two backtracks and one self-intersection. The last Figure 4.7 s shows fully constructed Straight Skeleton.

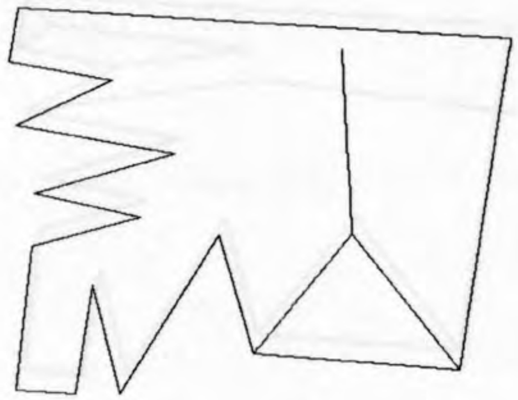


Figure 4.7 b

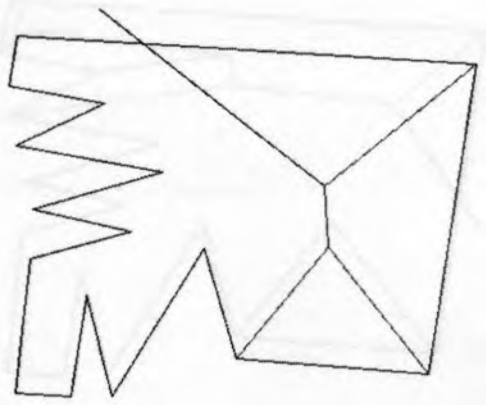


Figure 4.7 c

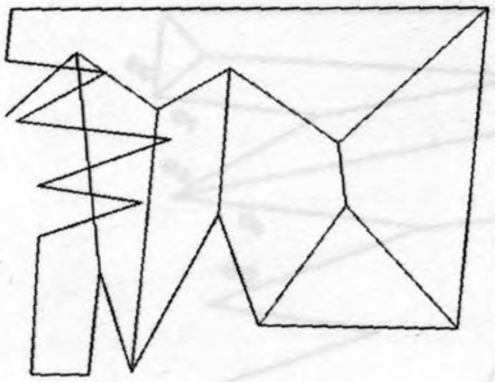


Figure 4.7 d

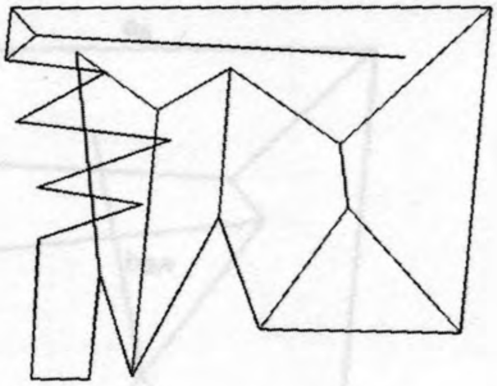


Figure 4.7 e

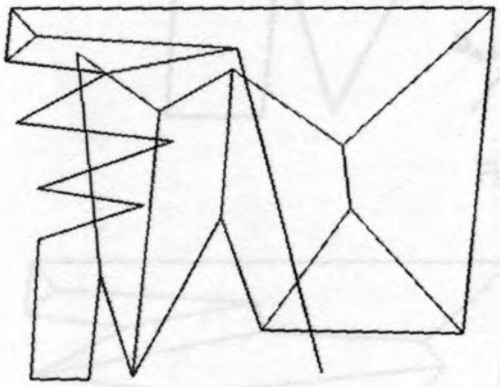


Figure 4.7 f

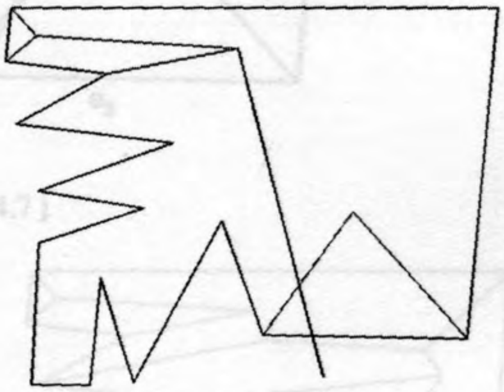


Figure 4.7 g

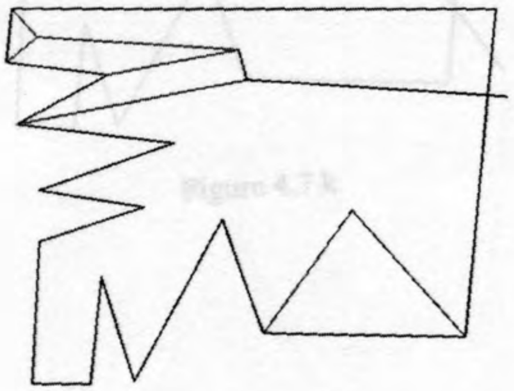


Figure 4.7 h

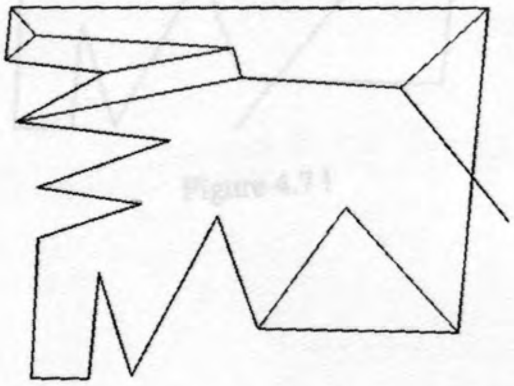


Figure 4.7 i

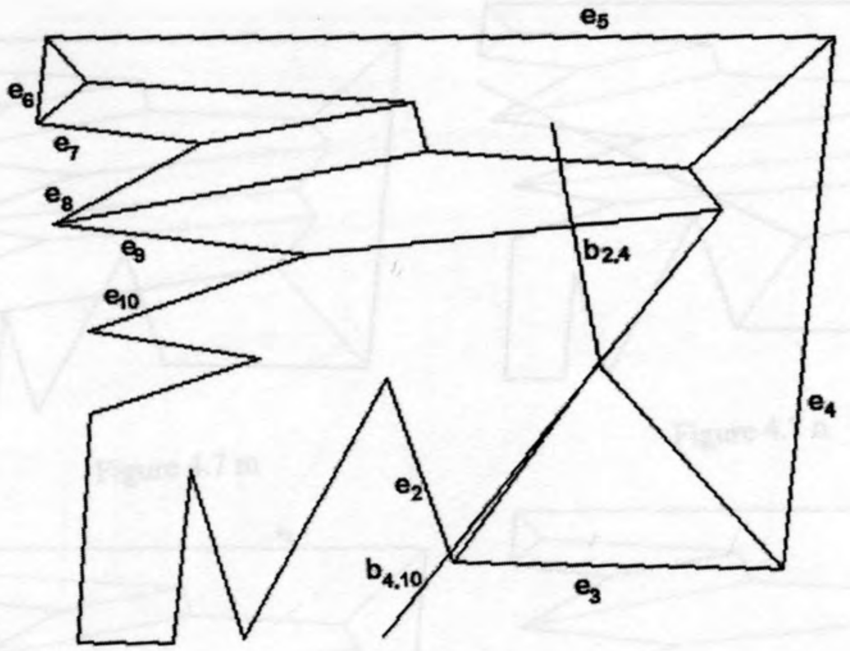


Figure 4.7 j

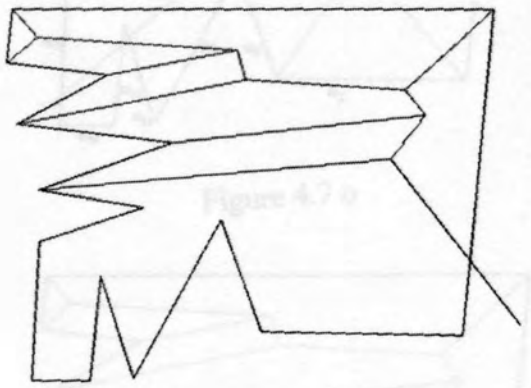


Figure 4.7 k

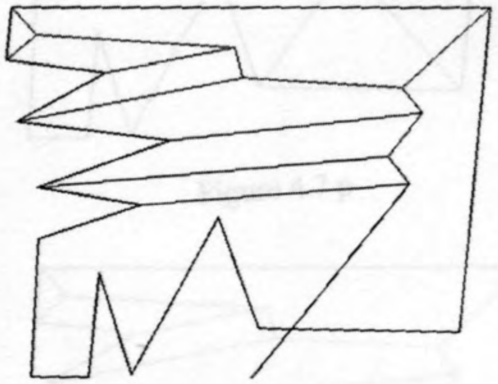


Figure 4.7 l

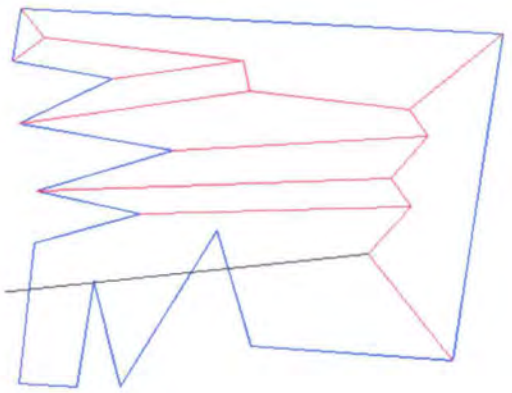


Figure 4.7 m

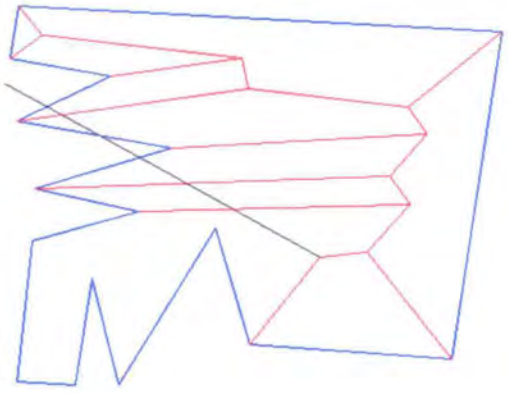


Figure 4.7 n

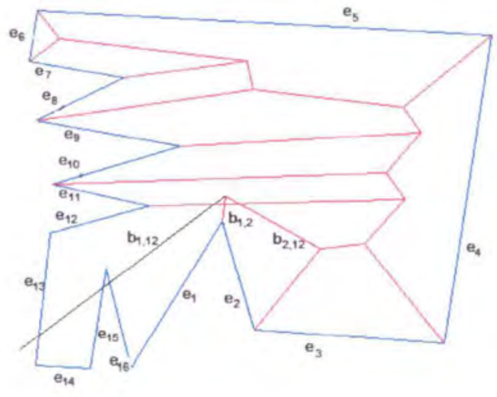


Figure 4.7 o

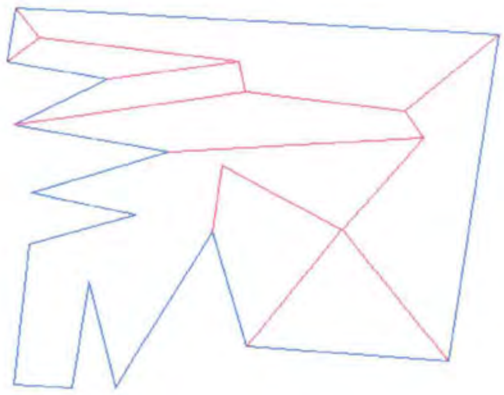


Figure 4.7 p

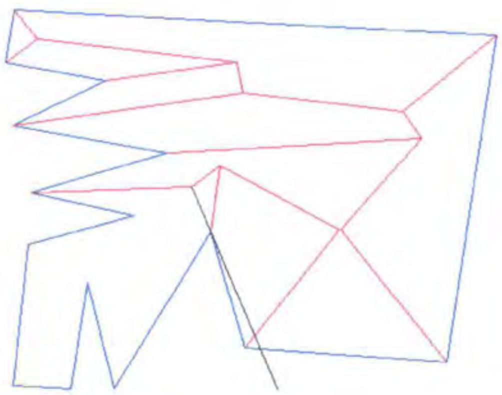


Figure 4.7 q

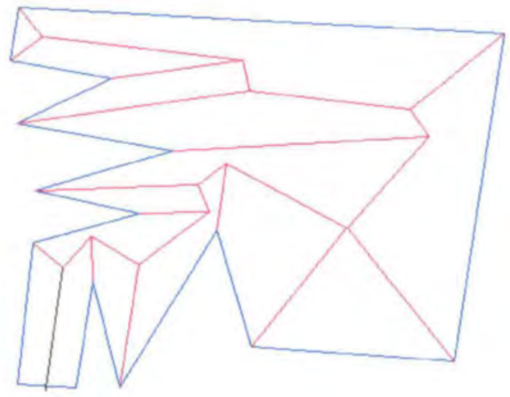


Figure 4.7 r

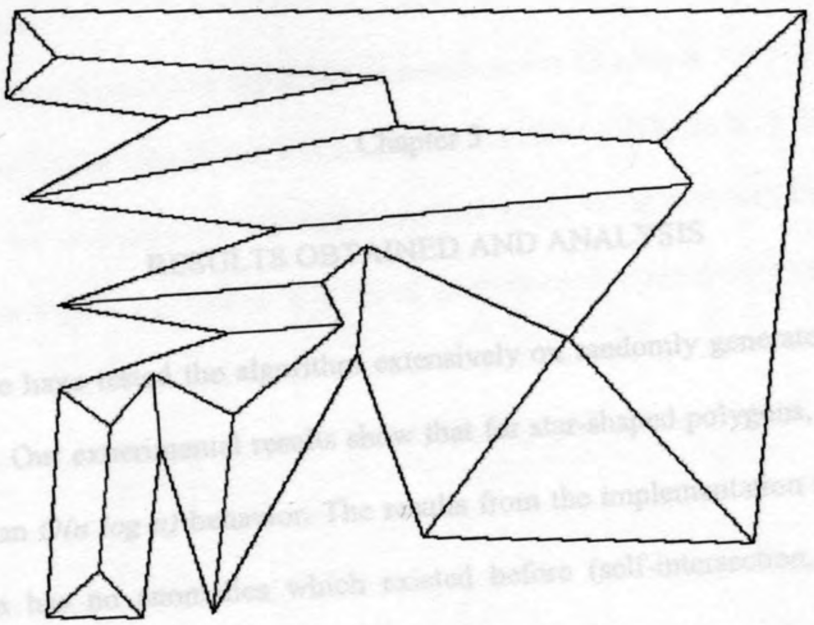


Figure 4.7 s

Chapter 5

RESULTS OBTAINED AND ANALYSIS

We have tested the algorithm extensively on randomly generated star-shaped polygons. Our experimental results show that for star-shaped polygons, this approach gives us an *O(n log n)* behavior. The results from the implementation shows that the algorithm has no anomalies which existed before (self-intersection, infinite loop, wrong backtrack).

This is an experimental algorithm and runs faster than any of the known algorithms for computing Straight Skeleton of star-shaped polygon.

STATISTICAL CHART

As mentioned earlier, *n* denotes the number of vertices and *m* denotes number of backtracks. Each backtrack can be computed in *log n* time using simple data structures, the algorithm runs in **O(*(n + m) log n)*** time. Since it is hard to get a good theoretical bound on *m*, we have done this experimentally. To do this, we have focused on the special case of star-shaped polygon since it is possible to randomly generate a large number of instances of such polygons. The information captured helped us in analyzing running time and backtrack counts for the algorithm.

Running Time

Running time of this algorithm as mentioned is O(*n log n* + *m log n*) where *m* is the number of backtracks and *n* is the number of vertices (Figure 5.1). Though in the worst case we will have *n/2*n/2* backtracks and hence worst case will be $O(n^2 \log n)$ as shown in Figure 5.2, though it is based on theoretical assumptions as in all of our tests on star-shaped polygons, we have not come across any case which has shown such a behavior. We experimented with randomly generated star shape polygons and plotted the running time with respect to vertex size *(n)* as shown in Figure 5.1. For each value of *n*, we generated *50* instances and took the average of all these running times.

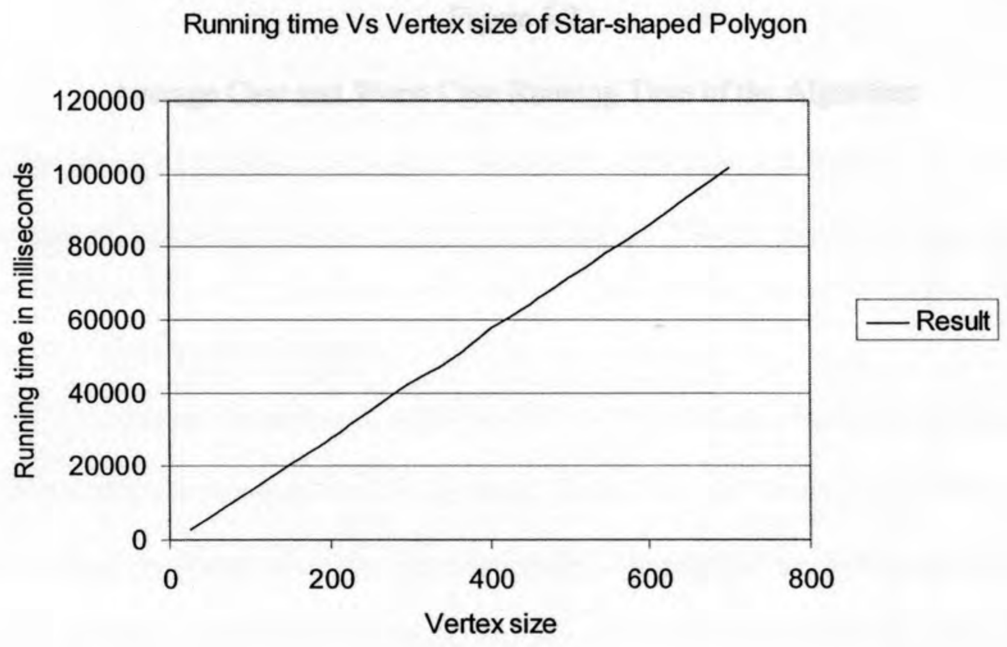


Figure 5.1

Running Time of Algorithm Plotted Against Number of Vertices in the Polygon

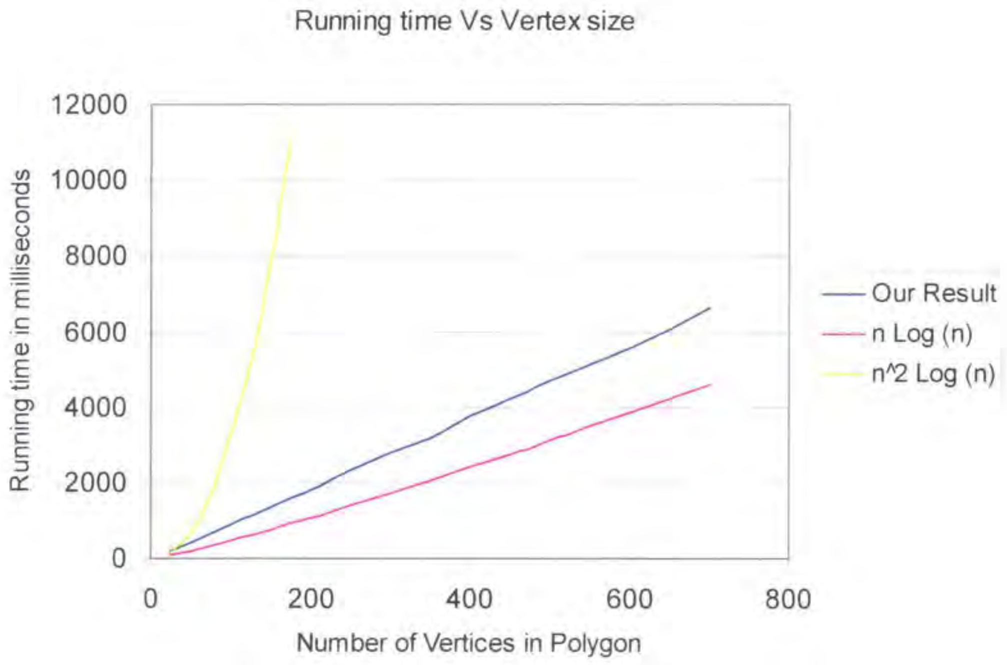


Figure 5.2

Average Case and Worst Case Running Time of the Algorithm

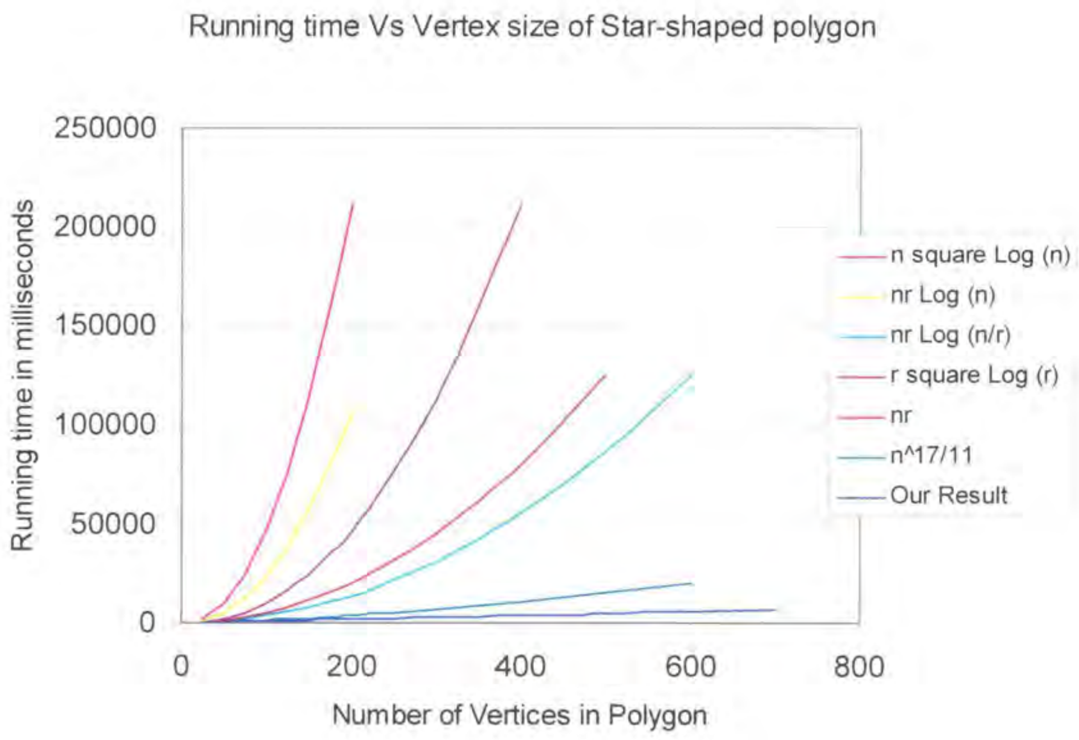


Figure 5.3

Comparison of Running Time of All Available Algorithms for Straight Skeletons

(our approach is at the bottom which shows it is more efficient than other algorithms)

Number of Backtrack per Vertex

The number of backtrack or intersections computed, *m*, plays an important role in determining the running time of algorithm. Since it is hard to get a good theoretical bound on *m*, we have done this experimentally. To estimate *m*, we generated *2000* random polygons of *100* vertices each. For each value of α (backtrack per vertex) we counted the number of input instances that had an $(m/n) < = \alpha$ (cumulative distribution function). This plot for $n = 100$ and $n = 200$ is given in figure 5.4. The graph shows that *90%* of the test result has *1.2* intersections computed per vertex.

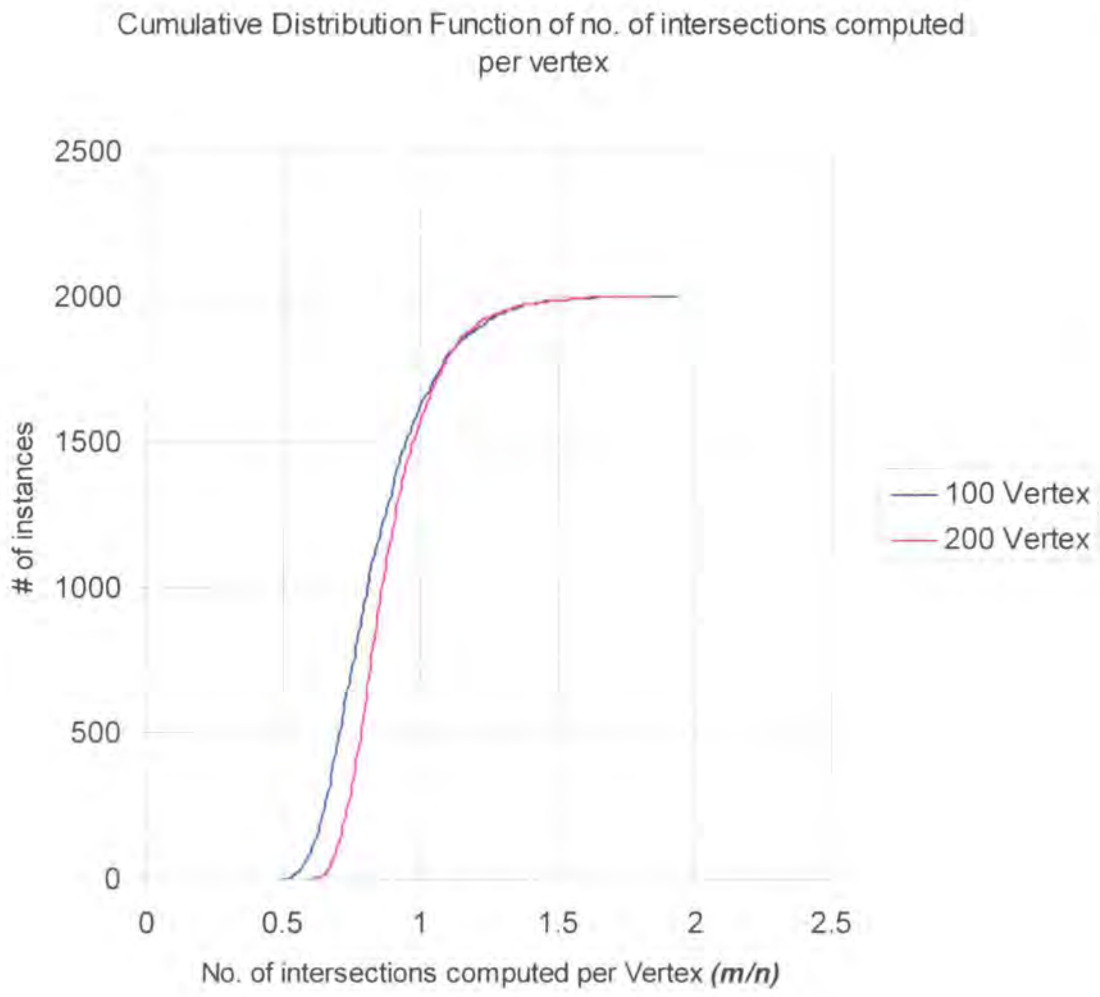


Figure 5.4

Cumulative Distribution Function of Number of Intersections Computed per Vertex

The experimental result in Figure 5.4 suggests that the value of *m* or the number of extra intersections calculated is not very high. This is further strengthened by the cumulative distribution function whose sigmoid shape suggests that we can expect a good average case performance and perhaps a good worst case behavior with high probability using this approach.

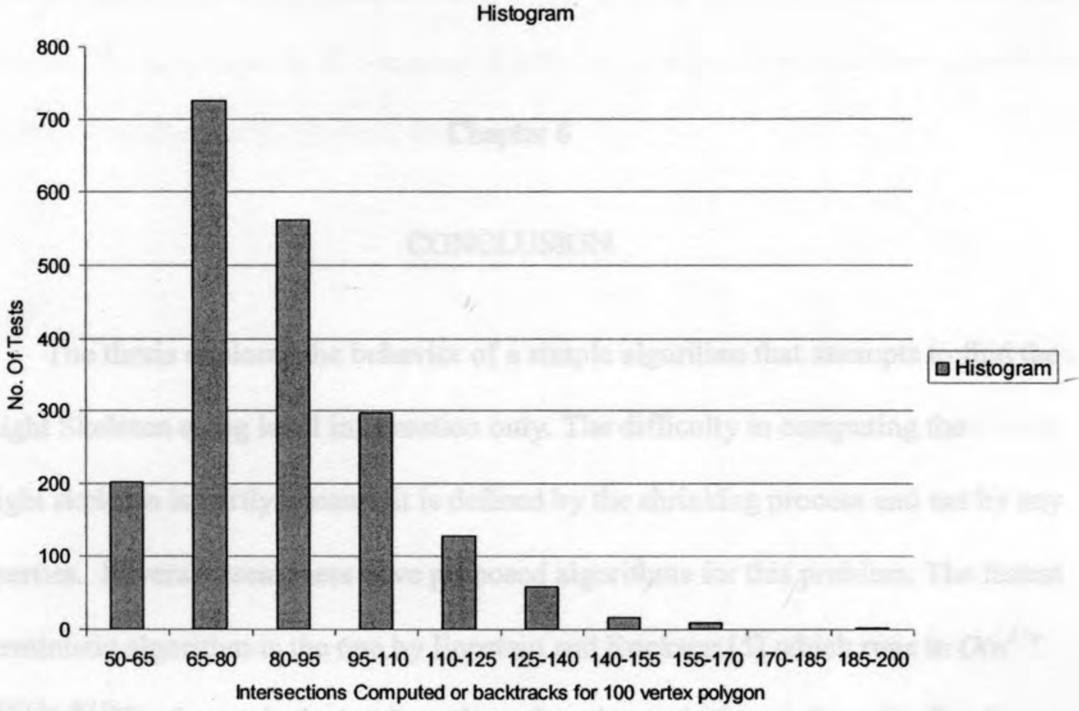


Figure 5.5

Histogram

The histogram shows the distribution of backtracks calculated for *2000* star-shaped polygons. These charts (CDF and Histogram) show that even if we increase the number of vertices in star shape polygon the number of intersection computed per vertex is less in general, although we might get some results where it is more but the probability for that is very less as we saw in the histogram there are only *2* results which has backtracks in *185* to *200* range.

Chapter 6

CONCLUSION

The thesis explores the behavior of a simple algorithm that attempts to find the Straight Skeleton using local information only. The difficulty in computing the straight skeleton is partly because it is defined by the shrinking process and not by any properties. Several researchers have proposed algorithms for this problem. The fastest deterministic algorithm is the one by Eppstein and Erickson [5] which runs in $O(n^{1+e} + n^{8/11+e}r^{9/11+e})$ where n is the total number of vertices, r is the number of reflex (non-convex) vertices, and e is an arbitrarily small positive constant. More recently, a slightly faster randomized algorithm using $O(n\sqrt{n} \log n)$ time and O(n) space was proposed by Cheng and Vigneron [6].

We have tested the algorithm extensively on randomly generated star-shaped polygons. Since it is hard to get a good theoretical bound on *m (backtracks)*, we have done this experimentally. Furthermore, star-shaped polygons have fair amount of reflex vertices and can be generated randomly using a program for testing. Our experimental results show that for star-shaped polygons, this approach gives us an $O(n \log n)$ behavior. The results from the implementation shows that the algorithm has no anomalies which existed before [16] (self-intersection, infinite loop, wrong backtrack).

This is an experimental algorithm and runs faster than any of the known algorithms for computing straight skeleton of star-shaped polygon (Figure 5.3 shows the comparison chart for all available algorithms at this time)

FUTURE WORK

1. Using *3D* model for detecting backtracks without any false alarm (as we observed in our approach which is *2D*, we sometimes get false backtracks).
2. This algorithm does not use the knowledge of where the *"centre"* of the star-shaped polygon lies. That may help improve the performance of the algorithm and also help to extend this approach to the case of general polygons by decomposing them into star-shaped polygons.
3. A star-shaped polygon can be covered by a single *"guard"* placed at the centre. Does the complexity of finding the skeleton depend on the number of *"guards"* needed to cover the polygon?

REFERENCES

REFERENCES

[1] O. Aichholzer, F. Aurenhammer, D. Alberts, and B. Gartner. *A Novel Type of Skeleton for Polygons*, J. Universal Comput. Sci., 1995.

[2] Oswin Aichholzer, Franz Aurenhammer, *Straight Skeletons for General Polygonal Figures in the Plane*. Proceedings of the Second Annual International Conference on Computing and Combinatorics, Springer-Verlag London, 1996.

[3] P. Felkel and S. Obdrzálek, *Straight Skeleton Implementation*. 14th Spring Conference on Computer Graphics, Slovakia,1998, *pages 210-218.*

[4] Franz Aurenhammer et al., *Straight skeleton of a simple polygon*, Symposium of LIESMARS, China, 1995, *pages 114-124.*

[5] David Eppstein and Jeff Erickson, *Raising roofs, crashing cycles, and playing pool: applications of a data structure for finding pairwise interactions*, ACM Symposium on Computational Geometry. Press New York, 1998, p*ages 58–67.*

[6] Siu-Wing Cheng, Antoine Vigneron, *Motorcycle graphs and straight skeletons*, ACM-SIAM symposium on Discrete algorithms, 2002, *pages 156-165.*

[7] David Bélanger, *Designing Roofs of Buildings,* http://www.sable.mcgill.ca/~dbelan2/roofs/roofs.html.

[8] David Eppstein, *Fast Hierarchical Clustering and Other Applications of Dynamic Closest Pairs*, ACM and SIAM, 1998, *pages 619-628.*

[9] Jeff Erickson, *Straight skeleton of a simple polygon*, http://compgeom.cs.uiuc.edu/~jeffe/open/skeleton.html.

[10] Ravi Janardan, *Geometric Algorithms for Rapid Physical Prototyping*, http://www-users.cs.umn.edu/~janardan/rpp-res-overview_files/frame.htm.

[11] A.Bagheri, M.Razzazi, *Drawing Free Trees Inside Rectilinear polygons Using Polygon Skeleton*, 18th European Workshop on Computational Geometry, 2002.

[12] Eric Biunno, The *Origami Polygon Cutting Theorem*, http://transform.tv/EricBiunno/welcome.html.

[13] Erik Demaine, The Fold-and-Cut Problem, http://theory.lcs.mit.edu/~edemaine/foldcut/#skeleton.

[14] Hoseok Kang, Shoreh Elhami, *Using Shape Analyses for Placement of Polygon Labels*, http://gis.esri.com/library/userconf/proc01/ professional/papers/pap388/p388.htm.

[15] Francis Chin, Jack Snoeyink, Cao An Wang, *Finding the Medial Axis of a Simple Polygon in Linear Time*, Discrete & Computational Geometry 21(3): 405-420 (1999).

[16] Yen-Chang Huang and Sarnath Ramnath, *Corrective Greedy Algorithm for Straight Skeleton,* MS Project 2001, Dept of Computer Science, St Cloud State University.