Computation of Straight Skeleton of Simple Polygon

A dissertation submitted in partial fulfillment
of the requirements of M.Tech.(Computer Science)
degree of Indian Statistical Institute, Kolkata

by

Sangameshwar Patil

under the supervision of

Dr. Subhas C. Nandy & Dr. Sandip Das

Indian Statistical Institute Kolkata-700 108.

15^{th} July 2002

Indian Statistical Institute

203, Barrackpore Trunk Road,

Kolkata-700 108.

Certificate of Approval

This is to certify that this thesis titled **"Computation of Straight Skeleton of Simple Polygon"** submitted by **Sangameshwar Patil** towards partial fulfillment of requirements for the degree of M.Tech in Computer Science at Indian Statistical Institute, Kolkata embodies the work done under my supervision.

Nandy 23/7/02

Dr. Subhas C. Nandy,
ACM Unit,
Indian Statistical Institute,
Kolkata-700 108.

Sandip Das 23/7/02

Dr. Sandip Das,
ACM Unit,
Indian Statistical Institute,
Kolkata-700 108.

S. Chattopadhyay 23/7/02

(External Examiner)

Acknowledgements

I take pleasure in thanking Dr. Subhas Chandra Nandy and Dr. Sandip Das for their guidance throughout the dissertation period. Their encouraging words have always kept my spirits up.

I express my deep gratitude to my teachers in the course. I would like to thank Members of ACM Unit, ISI-Kolkata. Finally I take this opportunity to thank my classmates and friends for their help, support and lighter moments throughout the course.

Sangameshwar Patil

Abstract

In 1995 Aichholzer et al. [2] introduced a new kind of skeleton for a polygon. It is defined as the trace of the vertices when the initial polygon is shrunken in self-parallel manner. Straight skeleton has only straight line segments which could be useful when parabolic edges of medial axis need to be avoided. Straight skeleton provides a canonical solution to the problem of roof construction in the architecture field. The straight skeleton may prove useful for other applications of the medial axis, for instance, shape recognition and terrain reconstruction from a river network as well.

The geometry of the straight skeleton is still not well understood and it is unclear whether the best known algorithm is close to optimal. We implement an $O(n^2)$ time algorithm for drawing straight skeleton of an arbitrary simple polygon. Next, we studied the special case where the polygon is monotone with respect to a coordinate axis. On the basis of some important observations, we conjecture that possibly the worst case time complexity of computing the straight skeleton for a monotone polygon is $O(n\log n)$. We executed our algorithm on randomly generated monotone polygons. Experimental results justify our conjecture.

Contents

Chapter 1

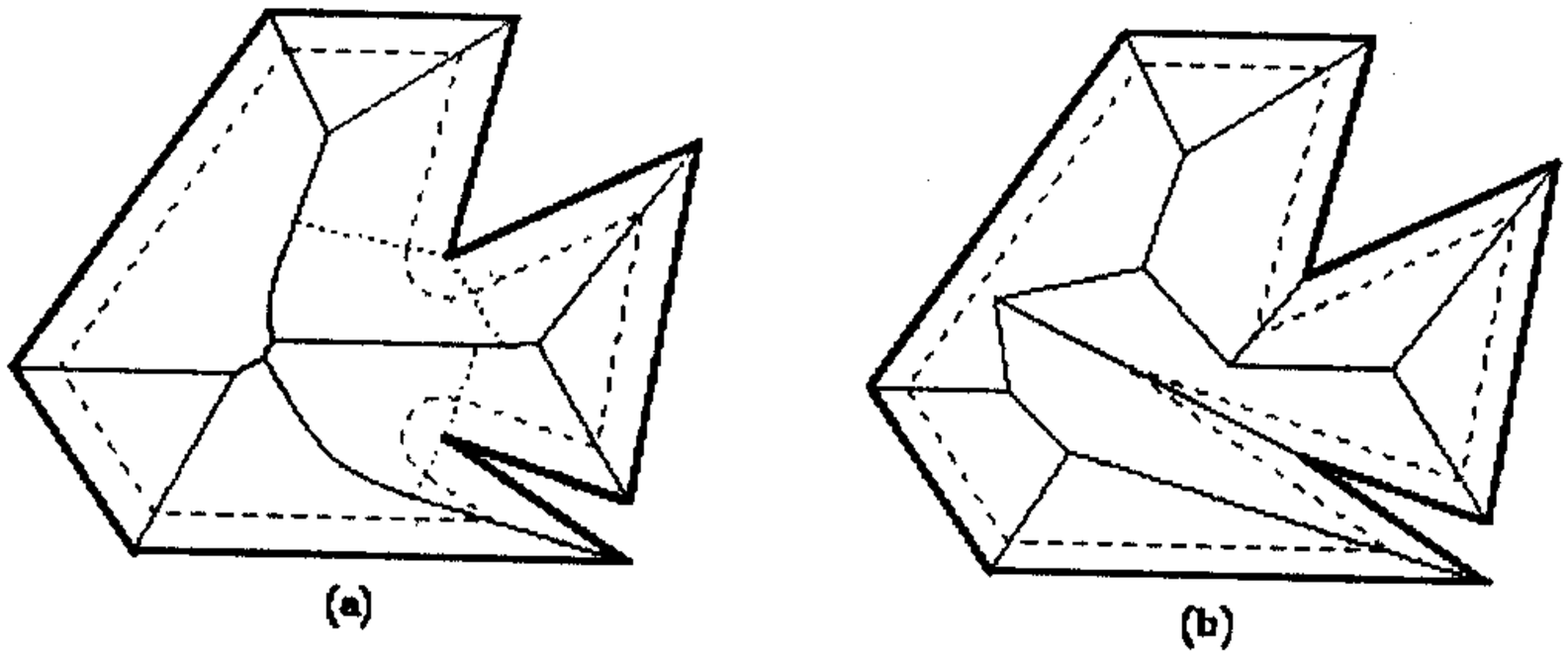
Introduction

Skeleton like structure is often used for the description of basic topological characteristics of a 2D object. Skeletons have numerous applications inside and outside computer science as is documented in Kirkpatrick [10]. In the literature of image processing and the computer vision, the skeleton of a geometric shape is informally defined as a set of discrete points in raster environment located in the centers of circles that touch more than one points of the object boundary. Medial axis serves the similar goal in the continuous vector environment.

1.1 Straight skeleton and medial axis

The medial axis of a simple plane polygon P may also be referred as symmetric axis or skeleton. One of the more picturesque is the grass-fire analogy: *imagine igniting all boundary points of P. If the flame burns inward at a uniform rate, then the quench points where the flame meets and extinguishes itself define the medial axis.* Equivalently, the medial axis is the locus of all centers of circles inside P that touch the boundary of P in two or more points. The medial axis was proposed and named by Blum in 1967 [5]. The pattern recognition literature uses it heavily as a one-dimensional structure that represents two-dimensional shape. It finds wide applications in solid modelling, mesh generation, pocket machining etc [8].

The medial axis of an arbitrary simple polygon consists of line segments and parabolic arcs, where each parabolic arc corresponds to a reflex vertex. The medial axis of a simple polygon describes the Voronoi diagram whose sites are the open edges and vertices of the polygon boundary. Inspite of many uses of medial axes, their curved arcs have been considered to be a shortcoming in different applications; e.g. computer representation of arcs and actual construction of arc segments. This led several researchers to investigate and form piecewise-linear approximations.



(a) (b)

Figure 1. (a) Medial Axis (b) Straight skeleton of same polygon

In 1995 Aichholzer et al. [2] introduced a new internal structure of a simple polygon, called *the straight skeleton.* It is solely made up of straight line segments which are pieces of angular bisectors of polygon edges. The straight skeleton of a simple polygon is defined by shrinking the polygon, i.e. translating each of its edges at a fixed rate, keeping sharp corners at the reflex vertices, and watching where the vertices go. The straight skeleton is quite similar to the medial axis; in fact, the two are equivalent for convex polygons. However, the straight skeleton of a nonconvex polygon has fewer edges than the medial axis. Unlike the medial axis, the straight skeleton is not a Voronoi diagram in any sense.

The straight skeleton can be used to construct a polygonal roof over a set of ground walls [2], reconstruct a geographical terrain from a river map, or define a pattern of folds that will make all the edges of a paper polygon colinear (although this last property is true of other *bisector graphs* [2] as well). The definition of straight skeleton can be generalized to arbitrary planar straight line graphs [1], where it has (potential) applications to planar motion planning, and to three-dimensional polyhedra, where it has potential applications to solid modelling.

Chapter 2

Straight Skeleton and its properties

2.1 Straight Skeleton

While the medial axis is a Voronoi-diagram-like concept, the straight skeleton is not defined using a distance function but rather by an appropriate shrinking process for P. Imagine that the boundary of P is contracted towards P 's interior, in a self-parallel manner and at the same speed for all edges. Lengths of edges might decrease or increase in this process. Each vertex of P moves along the angular bisector of its incident edges. This situation continues as long as the boundary does not change topologically. There are two possible types of changes:

1. **Edge event :** An edge shrinks to zero, making its two neighboring edges adjacent now.

2. **Split event :** An edge is split, i.e. a reflex vertex runs into this edge, thus splitting the whole polygon. New adjacencies occur between the split edge and each of the two edges incident to the reflex vertex.

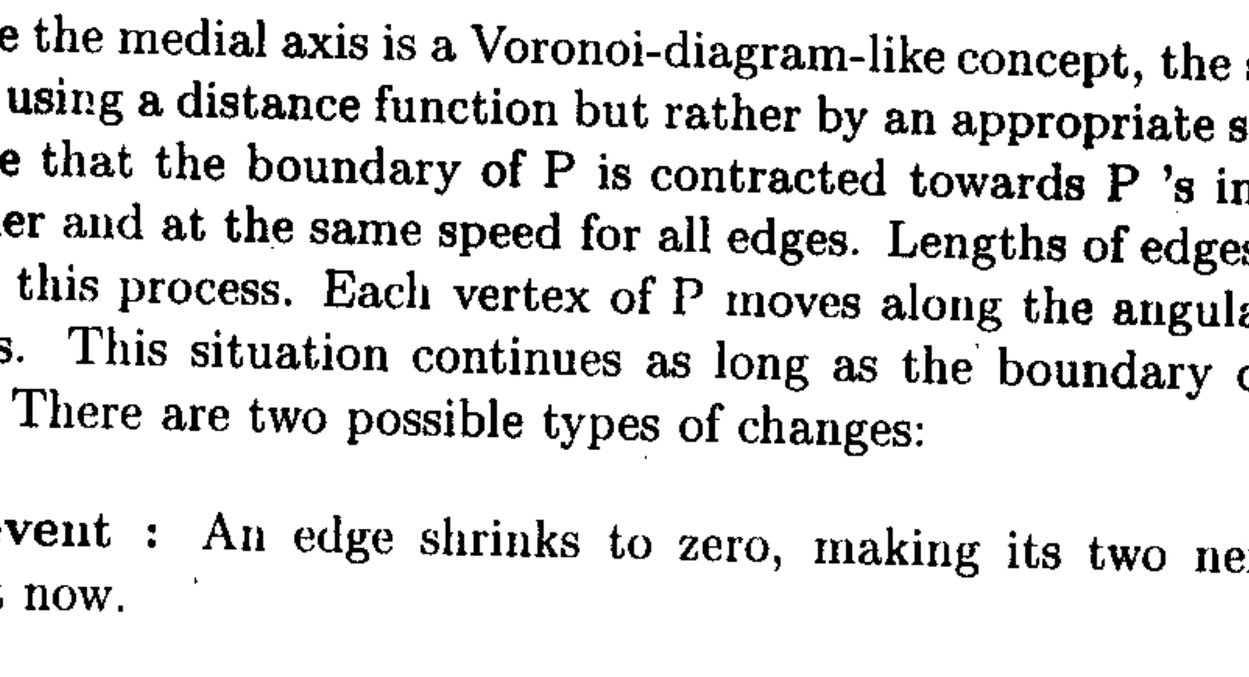


Figure 2. (a) Two simultaneous edge events (b) A split event
(name the vertices in fig 2a, 2b & use those names in above description)

After either type of event, we are left with a new, or two new, polygons which are shrunk recursively if they have non-zero area. Note that certain events

will occur simultaneously even if P is in general position, namely three edge events letting a triangle collapse to a point. The shrinking process gives a hierarchy of nested polygons. The figure 3a shows the polygon during various stages of shrinking. Each inside polygon is obtained after an event (edge/split) occurs in the polygon immediately encloses it. The outermost polygon is the original polygon.

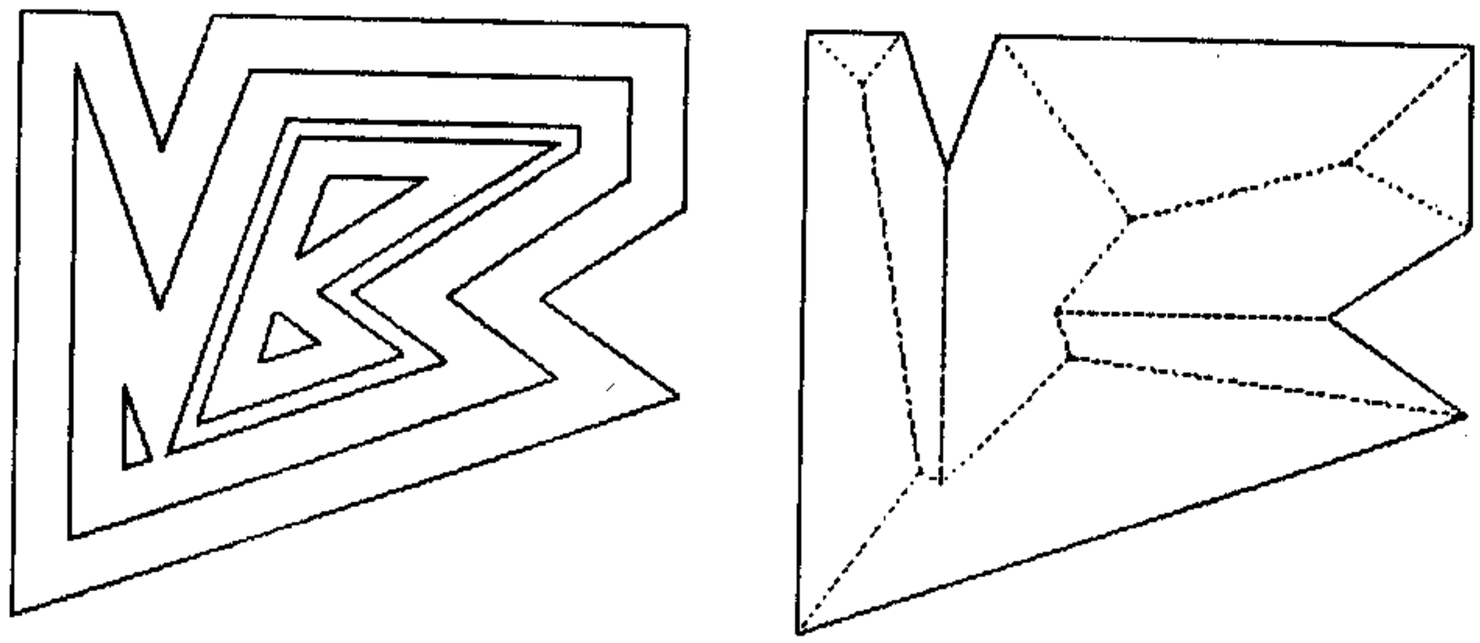


Figure 3: (a) Polygon hierarchy during shrinking, (b) its straight skeleton

The straight skeleton, S(P) is defined as the union of the pieces of angular bisectors traced out by polygon vertices during the shrinking process. S(P) is a unique structure defining a polygonal partition of P. As mentioned earlier, we will use the terms arcs and nodes for denoting the objects that form the boundaries of skeleton regions, in order to distinguish them from the objects forming the polygon, which will be called edges and vertices. Each edge e of P sweeps out a certain area which we call the "face" of e. Bisector pieces are called "arcs", and their endpoints which are not vertices of P are called "nodes" of S(P). Each edge/split event introduces a node of degree three into the evolving straight skeleton.

In degenerate cases, the straight skeleton can have nodes of degree higher than three, introduced by simultaneous events at the same location. In most cases, we can handle these events one at a time using standard perturbation techniques, replacing the high-degree node with several nodes of degree three, connected by zero-length edges. The only exception occurs when two or more reflex vertices (and nothing else) reach the same point simultaneously. We call this a **vertex event** (See Figure 4a). Unlike edge or split events, a vertex event can introduce a new reflex vertex into the shrinking polygon, although the total number of reflex vertices always decreases. Any perturbation of the polygon that removes a vertex event radically changes the structure of the polygon's straight skeleton. See Figure 4(b).

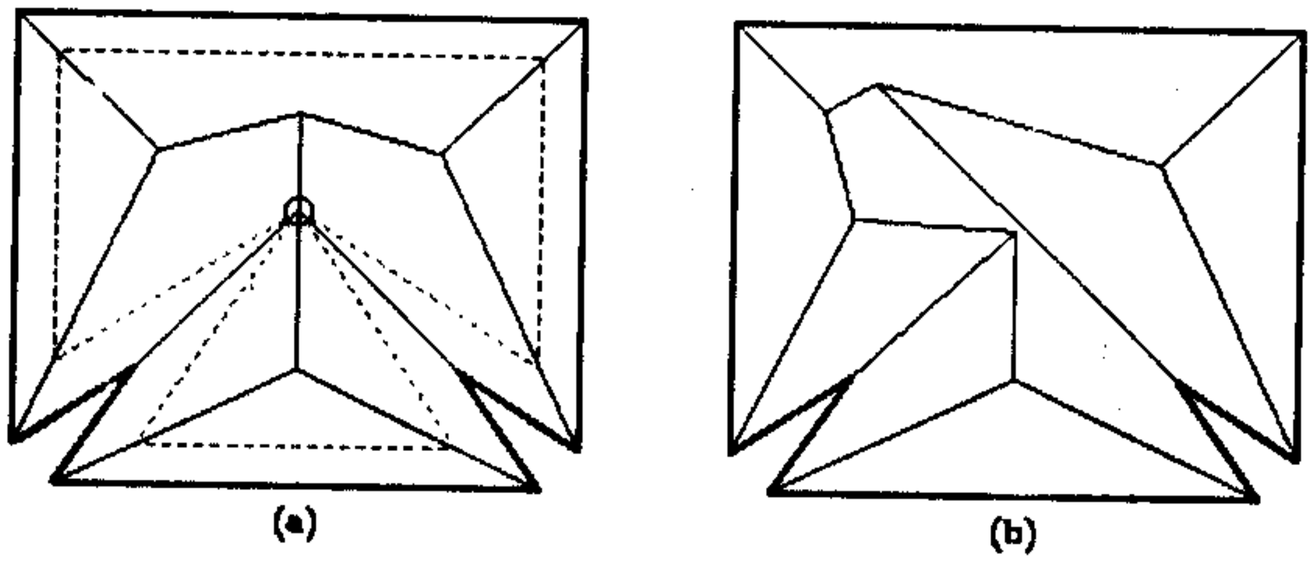


Figure 4. (a) A vertex event.
(b) Perturbing a degenerate polygon can radically alter its skeleton.

2.2 Properties of Straight skeleton

In straight skeleton, S(P) of a polygon P, arcs can be classified into two types: convex arcs and reflex arcs.

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

Property 2.1 *([2]) : Reflex arcs of S(P) only emanate from reflex vertices of P. (This implies that each new vertex generated during the shrinking process is convex.)*

Property 2.2 *([1]) : The faces of S(P) are monotone polygons.*

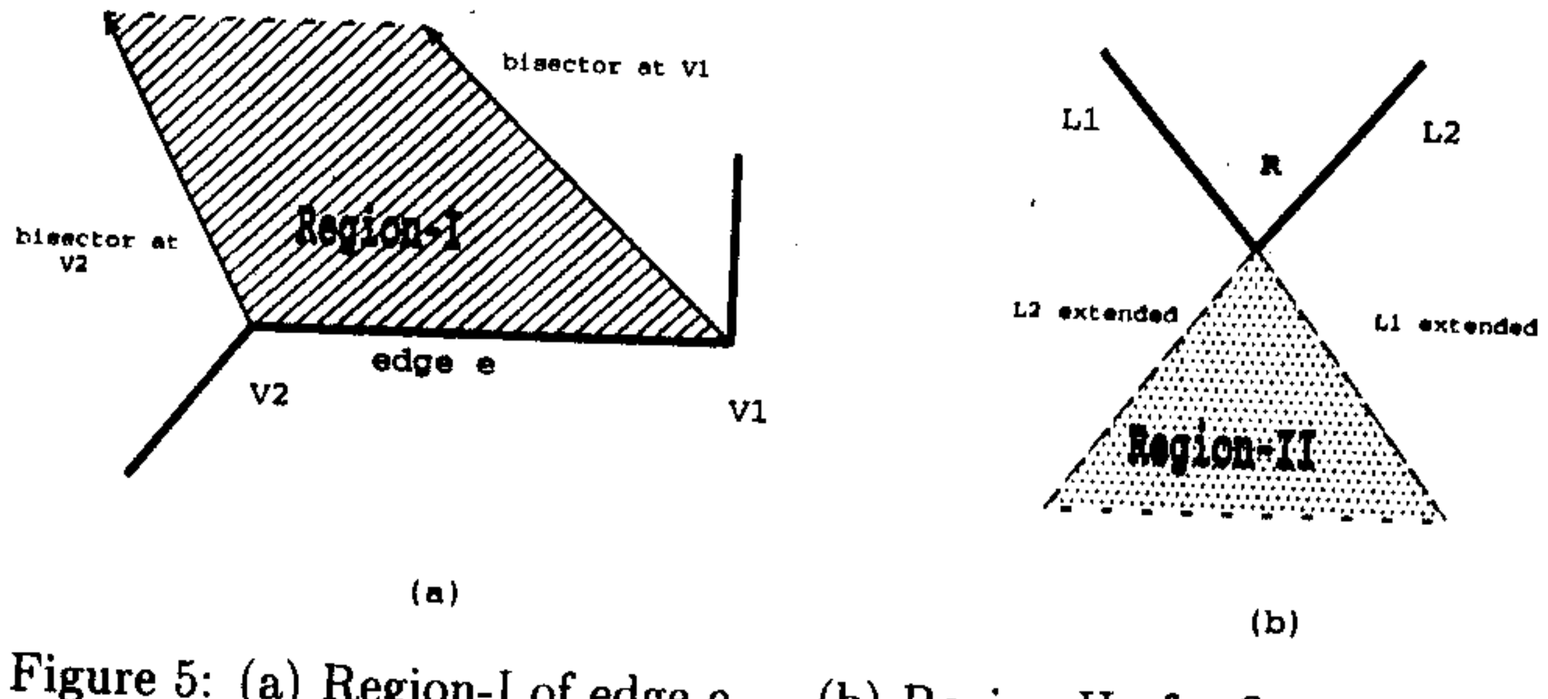
Property 2.3 *([2]) : S(P) is a tree and consists of exactly n connected faces, n-2 nodes and 2n-3 arcs.*

2.3 Characterizing the split point

Before going for the determination of split point coordinates, we will define two regions.

Region-I: Area bounded by the currently tested edge e and by the bisectors leading from the vertices at both ends of this line segment (see Fig. 5).

Region-II: Area enclosed by the "supporting lines" of edges forming the reflex vertex R. (Observe that the bisector at reflex vertex R always lies in the Region-II).



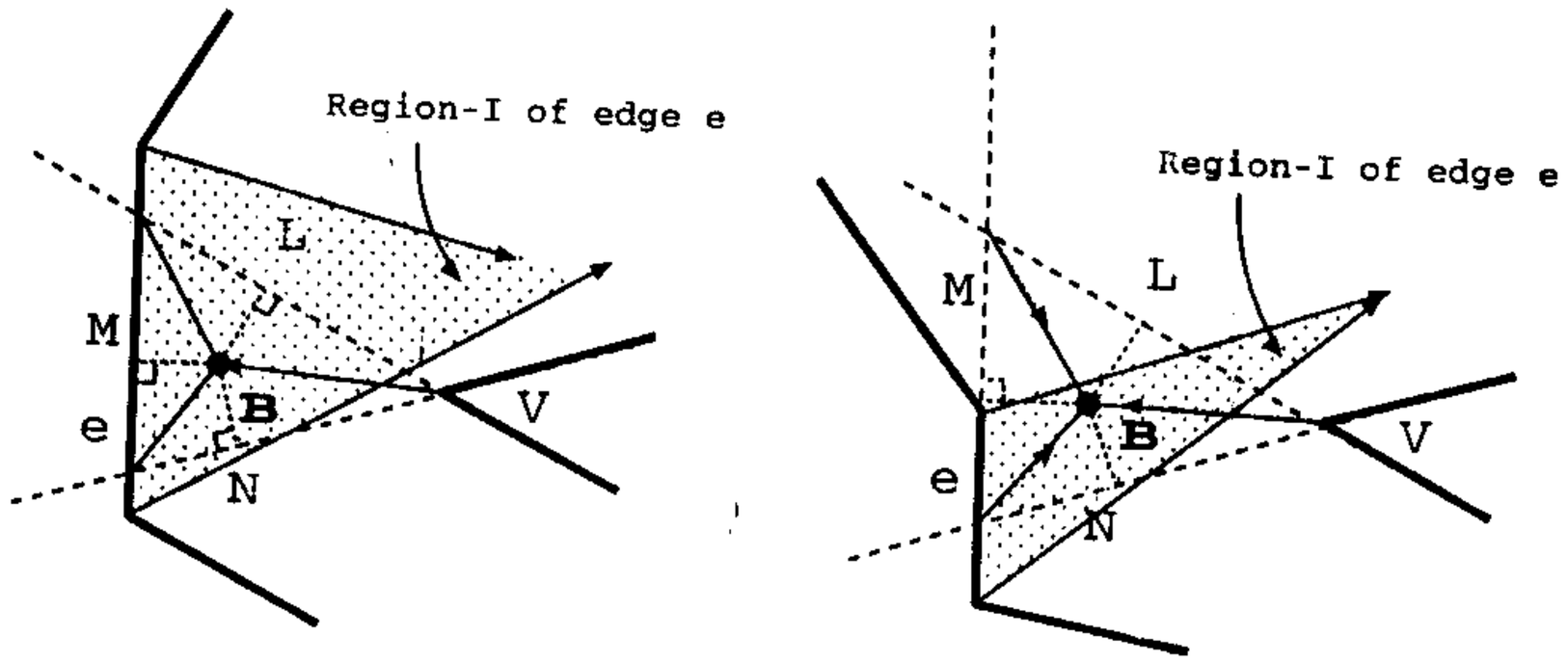
(a) (b)

Figure 5: (a) Region-I of edge e, (b) Region-II of reflex vertex R

Let B be the split-point corresponding to reflex vertex V. The edge under consideration is edge 'e' (see Fig. 6). This point B is intersection of

- the bisector of V and
- bisectors of the angle between supporting line of edge 'e' and the supporting lines of the edges incident at reflex vertex V

The point B is equidistant from the supporting lines of edge 'e' and the edges incident at reflex vertex V (i.e. $BL = BM = BN$; see Fig. 6). Point B must lie in Region-II of V.



(a) (b)

Figure 6: Intersection point computation for a reflex vertex V which yields a split event(point- B) and the currently tested line segment e cannot be used (see Fig. 6b).

Simple intersection test between the bisector starting at V and the supporting line of currently tested edge 'e' rejects the line segments laying "behind" the vertex V. Simple check should be performed to properly handle the case when one of the line segments starting at V is parallel to e. The resulting point B is selected from all the candidate B points as the nearest point to vertex V.

We end this section by emphasizing the fact that a reflex vertex does *NOT* imply that it can cause only split events. It can cause either edge event or a split event. (see fig. 7)

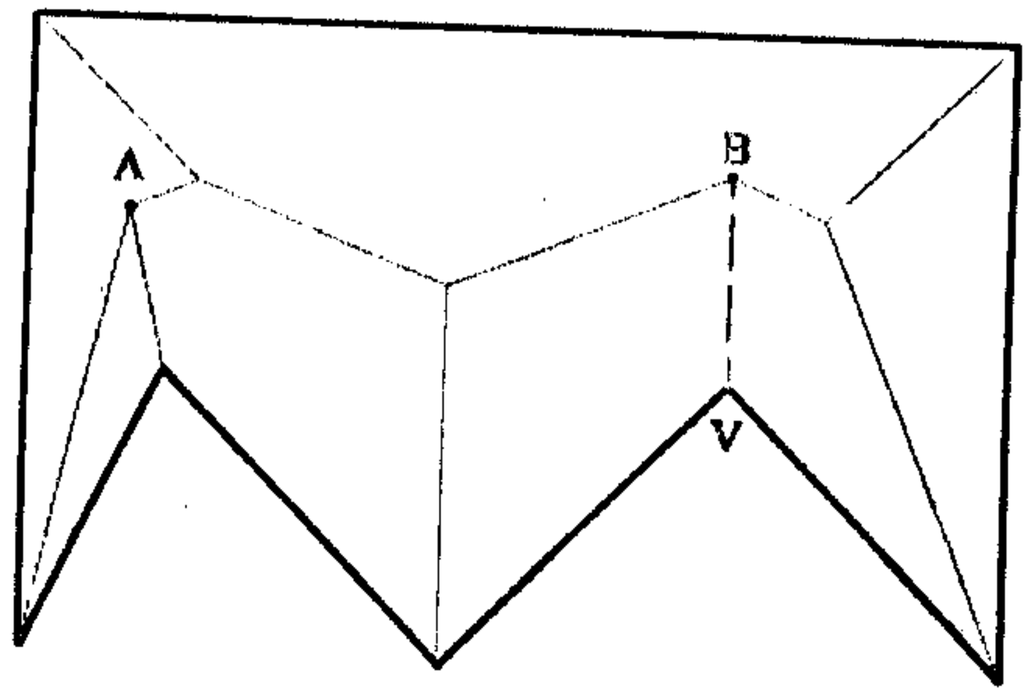


Figure 7: Reflex vertex can yield to both cases:
an edge event (point A) or a split event (point B)

Chapter 3

Straight skeleton of monotone polygon

A simple polygon is said to be monotone w.r.t. a line L, if its intersection with any line perpendicular to L is connected. Without loss of generality, we assume that unless specified otherwise, all monotone polygons considered in this report are monotone w.r.t. Y-axis. We investigate the effect of monotonicity on computation of the straight skeleton.

3.1 Relation between distance along bisector and perpendicular distance from edge, for a vertex

Let V be a vertex of the polygon. Let it be a common end-point for edges e and e'. V' be its position at any instant during the shrinking process (before V vanishes, due to edge/split event). (see Fig. 8).

r = dist(V, V'), θ = internal angle at vertex V

d = distance through which e, e' have moved in self-parallel manner to reach V'

$$d = r \ sin \ \theta/2 \tag{3.1}$$

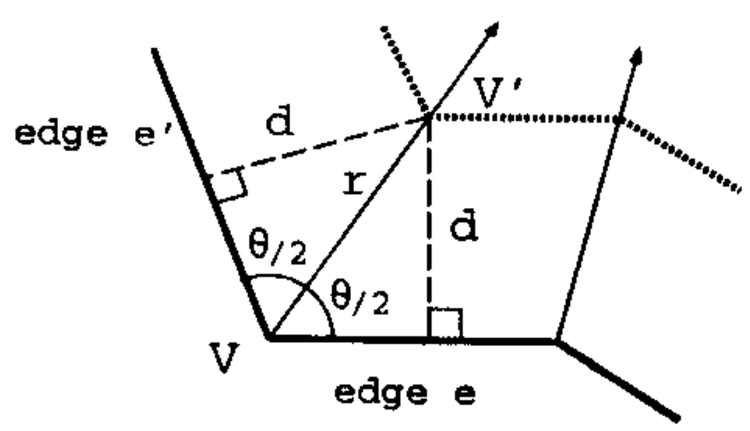


Figure 8: relation between d and r

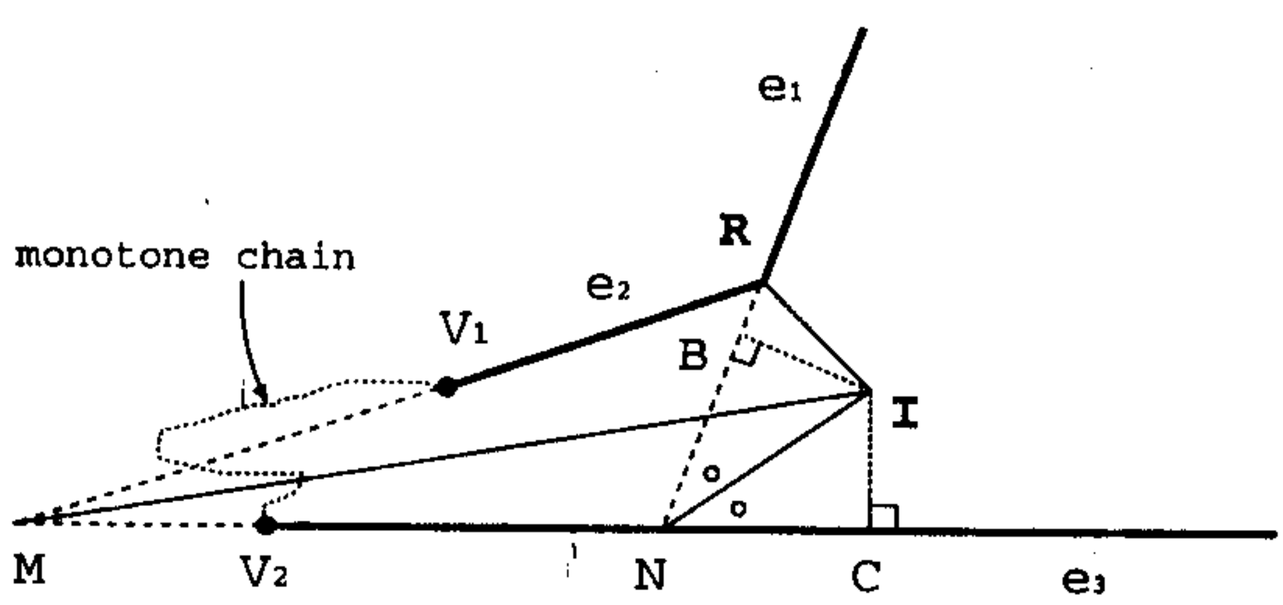
3.2 Properties of Straight Skeleton of Monotone Polygon

Observation 3.1 *During the shrinking process, monotonicity of a polygon is preserved. (Note that this implies, the two newly generated sub-polygons after a split event in a* monotone *polygon are again monotone.)*

Observation 3.2 *(a) The sub-polygons created due to each split event are disjoint. (b) A reflex vertex belonging to a sub-polygon can split only those edges which belong to same sub-polygon.*[1]

Lemma 3.1 *For any split event in a monotone polygon, the reflex vertex and the edge being split always belong to the different monotone chains.*

Proof : We will prove it by contradiction. Assume that the reflex vertex and the edge being split belong to the same monotone chain.
Let R be the reflex vertex formed by the incident edges e_1 and e_2. Let e_3 be the edge belonging to the same monotone chain as that of R and split by R. Let M be the intersection point of supporting lines of edges e_2 and e_3; N be the intersection point of supporting lines of edges e_1 and e_3. Let I be the split point where R will split the edge e_3.



From section 2.3, we know that I is intersection of bisectors at M and R. Join points I and N. Draw IB $\perp e_2$ and IC $\perp e_3$. Since I is split point, IB = IC.
$IN.sin(\angle INB) = IN.sin(\angle INC) \Rightarrow \angle INB = \angle INC$
Hence IN is bisector of $\angle BNC$.

Note that, during the shrinking process, the edge e_2 moves in self-parallel manner and its end vertices move along the bisectors of corresponding internal angles. Consider an instant just prior to meeting R and the edge e_3 at point I. Whatever polygonal edges may be between V_1 and V_2, both V_1 and V_2 move along the bisector of the $\angle V_1MV_2$. At the moment node I is created, both end-vertices of e_2 are coincident (at I). Length of edge e_2 is reduced to zero at point I. Thus an edge event and a split event occur simultaneously at point I. Out of the two sub-polygons generated due to the split event, one has zero area.

[1] Although it seems obvious, this observation will be important when we prove one of the lemmas.

We may observe the same thing from a different viewpoint as follows – at I, e_2, one of the incident edges at R, vanishes. So R is no longer a reflex vertex. Due to edge event of e_2, a new node of straight skeleton is created at I. As per Property 2.1, this is a convex vertex. So no split-event can happen due to R. ■

Lemma 3.2 *Let A and B be two reflex vertices belonging to same monotone chain. If A is above B (i.e. having larger Y coordinate value), then – the edge split by the split-event due to A is either same edge split by the split-event due to B or is above it.*
(w.l.g. we have assumed the polygon to be Y-monotone.)

Proof : If both A and B cause split events, then either these split events will occur simultaneously or one will occur before other. Now consider the following three cases
case 1: *'A' causes split event earlier than 'B'*

We know that the two sub-polygons generated due to the split event are monotone (see Observation 3.1). The split event due to A will put vertex *B' (the position of vertex B when the split event occurs)* into the the lower sub-polygon (since, B lies monotonically below A). As the two polygons are disjoint, if B is to split an edge, it must belong to this lower sub-polygon. Hence, if A causes a split event earlier than B, then B can't split an edge above the one split by A.
case 2: *'B' causes split event earlier than 'A'*

Following similar reasoning as above, A will belong to the upper sub-polygon and hence cannot split an edge below the one split by B.
case 3: *Both split events occur simultaneously.*

If the bisector of reflex vertex B is to split an edge above the one split by A then the bisectors of A and B must **converge** (i.e. the bisector rays must meet at some point). So they can't be end-points of same edge. (otherwise monotonicity and convergence are not simultaneously possible. see Fig. 9)

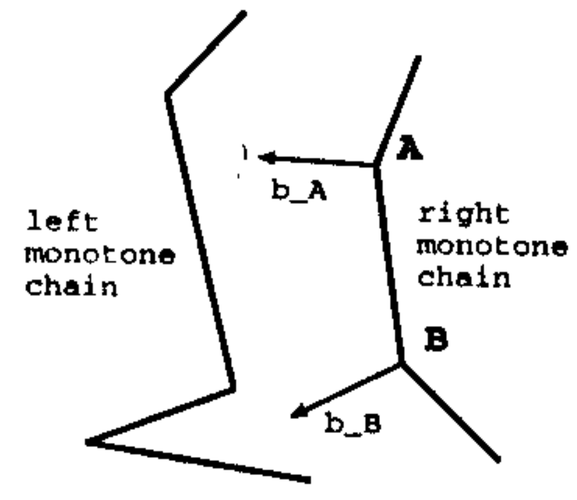


Figure 9: Both end points of an edge can't be reflex vertices (otherwise divergent bisectors)

Hence the edges incident on A are different from the edges incident on B. Let I be the point where the simultaneous split events occur. I must lie on both the reflex bisectors b_A and b_B.

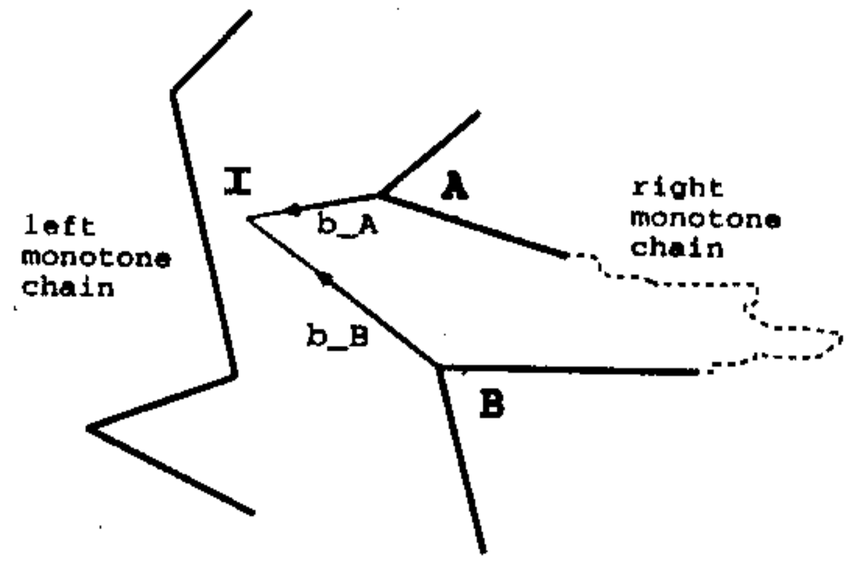


Figure 10: Simultaneous split events

Note that, for a valid split event at I, only one edge can be split at that particular point. (since, two edges at a point implies that it is their common end-point and in that case, no question of occurrence of a split event as the other part of the split edge will have length zero.) So if split events due to A and B occur simultaneously, they split the same edge. ■

Before proceeding further, we highlight two points of special importance which will be used in the further discussions. We assume that the vertices (hence the edges) of the polygon are listed in clockwise order. Let R be a reflex vertex. Let e_{prev} and e_{next} be the previous and next (in clockwise sense) edges incident at R. (see Fig. 11).

$Next_R$ = intersection of supporting line of e_{next} and the supporting line of edge which we want to test whether it can be split by R

$Prev_R$ = intersection of supporting line of e_{prev} and the supporting line of edge which we want to test whether it can be split by R

Lemma 3.3 *Reflex vertex R can not split an edge V_1V_2, if (a) $Next_R$ and $Prev_R$ lie outside the segment V_1V_2 and (b) both these intersection points are on the same side of V_1 and on the same side of V_2. However the converse is not true.*

Proof :

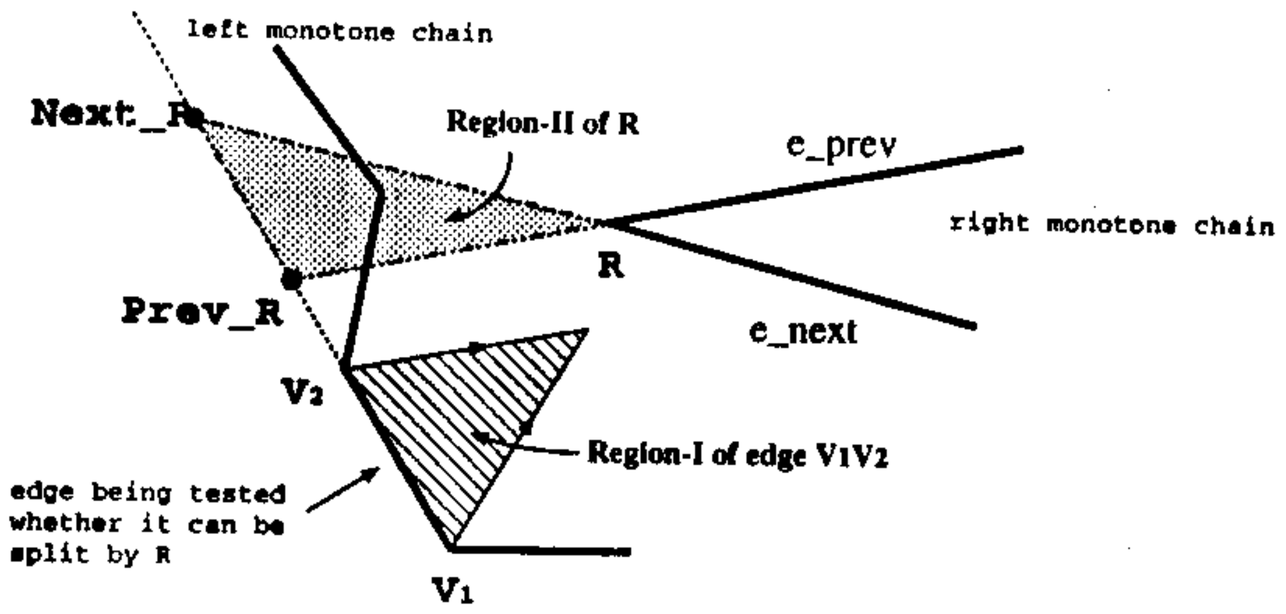


Figure 11: $Next_R$ and $Prev_R$

If $Next_R$ and $Prev_R$ satisfy the conditions, then there is no overlap of Region-I of edge being tested and Region-II of R. (see section 2.3). (Observe that in the **Fig. 11**, the shaded regions don't overlap.) It is obvious that if these two regions don't overlap, then R can't split the edge V_1V_2.

In order to disprove the converse, consider an example in Fig. 12. Here, $Next_R$ and $Prev_R$ lie on different sides of V_2. But the split point I lies outside Region-I of the edge V_1V_2. Hence R cannot split edge V_1V_2.

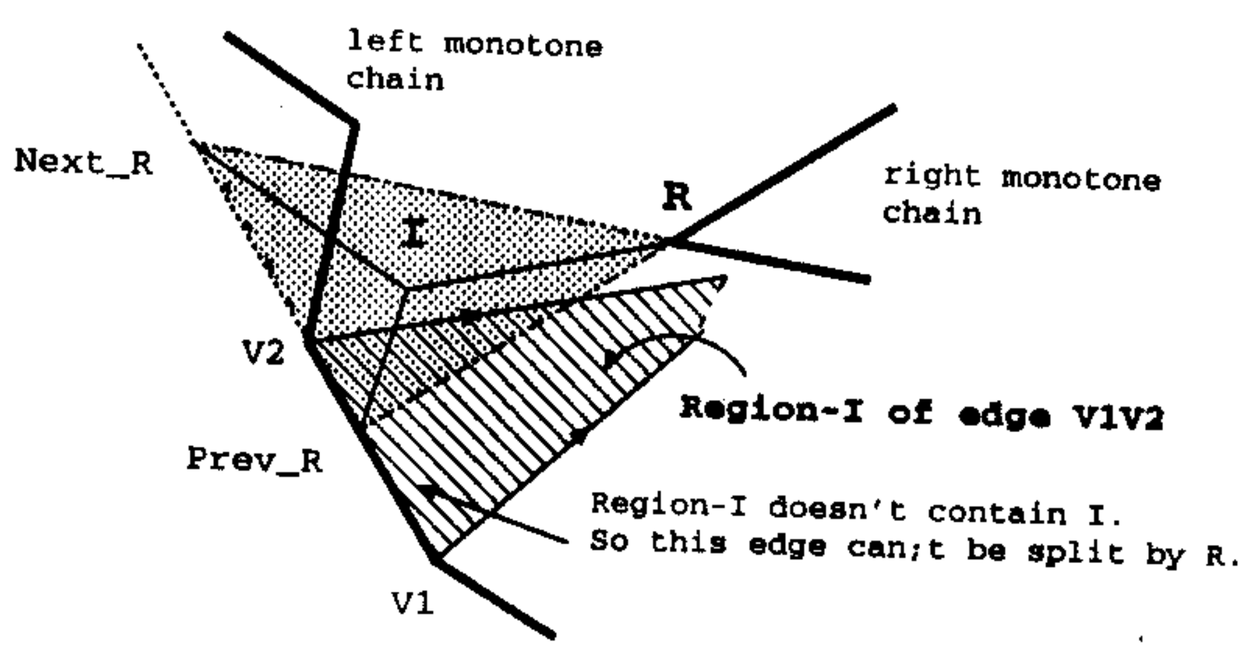


Figure 12: example to show that above condition is *sufficient*, but *not necessary*.

■

Observation 3.3 *A reflex vertex R can split an edge (say e = V_1V_2) and only that edge, if R lies in Region-I of e,* **and** *either of the following is true –*
(a) the bisector at R intersects the segment V_1V_2, **or**
(b) both $Next_R$ and $Prev_R$ intersection points lie on the segment V_1V_2.

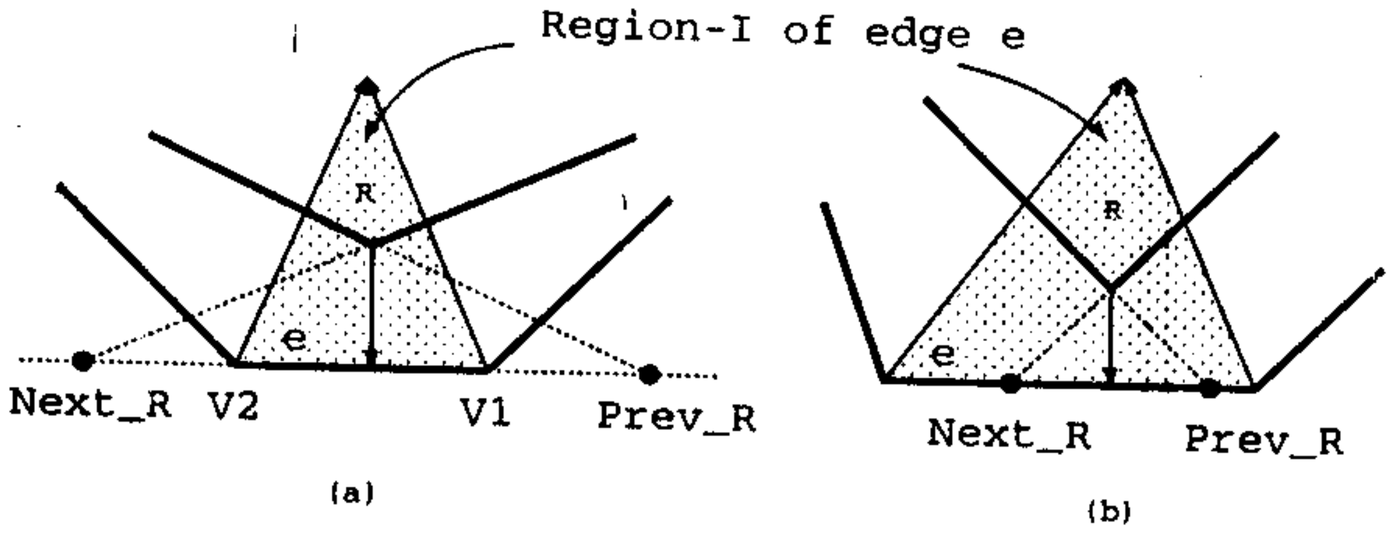
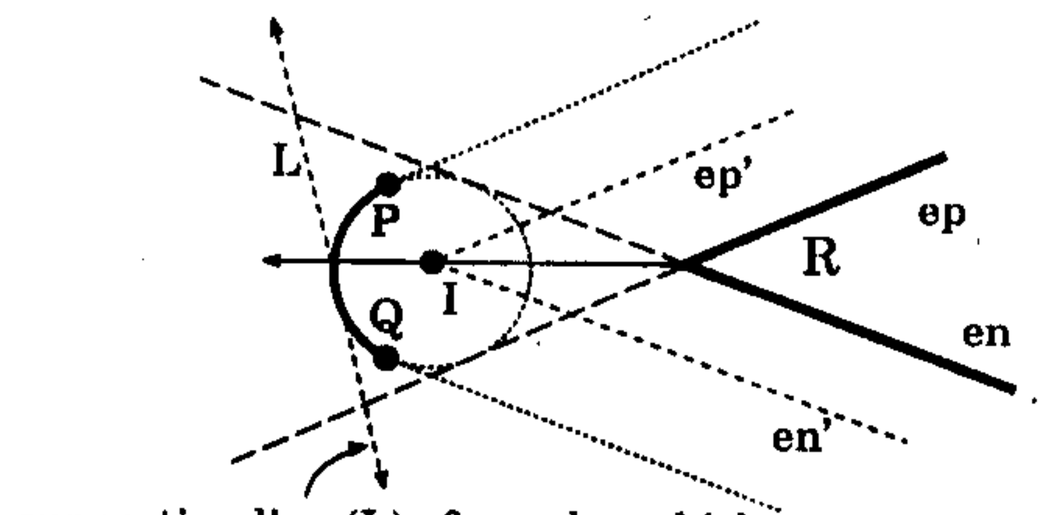


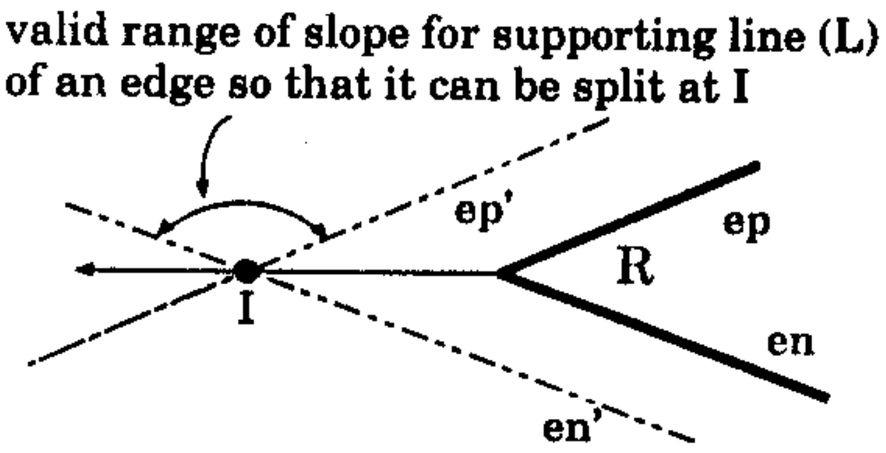
Figure 13: case (a) and (b) of Observation 3.3

Observation 3.4 *Valid range of slope for supporting line of an edge to be split is limited by the slopes of the edges incident at reflex vertex R. (See Fig. 14).*



supporting line (L) of an edge which is to be split at I, must be tangent to the circle somewhere in arc PQ only

(a)



(b)

Figure 14: conditions for an edge (and its supporting line) to be split by a reflex vertex R

Valid range of slope for the supporting line L is bounded by the slopes of supporting lines of e_p and e_n. Observe that if slope of L exceeds the limit set by e_p, then edge e_p will be reduced to length zero at I. Reasoning for this is similar to that given in proof for Lemma 3.1. Case for e_n is analogous to e_p.

Let I be any valid split point due to reflex vertex R. Let e_p and e_n be the edges incident at R. This observation(3.4) gives the conditions which must be satisfied by the edge which can be split at I by R. We know that point I must be equidistant from the supporting lines of e_p, e_n and the edge to be split(See section 2.3). Hence the supporting line of the edge which can be split at I must be tangent to the circle centered at I and which touches the supporting lines of e_p, e_n. The portion of the circle in which this supporting line can be tangent is limited by its valid range of slope (See Fig. 14b). In Fig. 14a –

P : point where a line parallel to edge e_p is tangent to the circle

Q : point where a line parallel to edge e_n is tangent to the circle

Thus the supporting line L must be tangent to the circle somewhere in arc PQ. Of course, point I must lie in the Region-I of this edge being split (See section 2.3).

Conjecture 3.1 ***A reflex vertex of a monotone polygon can split only one edge.***

We have tried to find out how many edges of monotone polygon can be split by a reflex vertex. Running the tests on several typical instances of monotone polygons generated interactively, we found that the candidate edges which give rise to a valid split-point in a split-event by a reflex vertex, during the shrinking process, is at most one. (The polygons are generated using a JAVA applet. User can generate any arbitrary polygon as he/she wishes.) This experiment strengthens our conjecture. In appendix C, we show an instance of a monotone polygon and along with the

trace of computing the straight skeleton. This demonstrates the justification of our conjecture, i.e., for a monotone polygon, each reflex vertex splits at most one edge.

If we can prove this conjecture, then an algorithm with $O(n\log n)$ time complexity to compute straight skeleton of monotone polygon is easy to develop. The algorithm would be just a simple modification of the algorithm implemented in this dissertation (see Appendix B). The modification would consist of –

(a) candidate edges for split-event due to a reflex vertex will be from a monotone chain different than that of this reflex vertex *(reason: Lemma 3.1)*.

(b) search for candidate edges for split-point can be stopped when we find the first edge which can be validly split (see section 2.3) by the given reflex vertex (say R). *(reason: conjecture 3.1)*[2]

(c) for any reflex vertex V belonging to same chain as that of R (in above step b), the search for candidate edge (which may be split by V) will consist of the edge split by R or edges below(above) it, if V is below(above) R *(reason: Lemma 3.2)*.

We conclude the chapter by pointing out that Lemma 3.3 and Observations 3.3, 3.4 may prove important in proving the conjecture.

[2]of course, we need to prove the conjecture first

Chapter 4

Previous Results and Conclusion

4.1 Previous Results

Although the medial axis can be constructed in linear time [6], the fastest known algorithms for straight skeletons are much slower. The main difficulty is that changing the positions of reflex vertices has a significant non-local effect on the skeleton. This nonlocality makes techniques such as incremental construction or divide-and-conquer fail. The following table lists the time and space bounds of various algorithms. Here n is the total number of vertices, r is the number of reflex (nonconvex) vertices, and ϵ is an arbitrarily small positive constant. Note that $r = O(n)$ in worst case.

Table 4.1: Time and space complexity of earlier algorithms

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

Though the subquadratic algorithm [3] is a significant theoretical improvement over earlier results, the authors of [3] observe that, because of the complexity of the range-searching algorithms involved, they would be less efficient (slower) in practice.

4.2 Future Work

An obvious way ahead is to try and prove the conjecture 3.1. Proving it would give us a $O(nlogn)$ algorithm for computing straight skeleton of monotone polygon. The geometry of the straight skeleton is still not well understood and none of the time bounds has proven to be tight. It is unclear whether the best known algorithm is close to optimal. Finding algorithms with better time complexity for general cases of computing straight skeleton of any arbitrary simple polygon and a polygon with polygonal holes is still an open problem.

Some of the approaches which can be tried are – (a) For monotone polygon, extend the bisectors of reflex vertices till they meet an edge of the polygon. Thus the polygon is divided into convex pieces. Compute the straight skeletons of these individual pieces and try to merge them. This idea is analogous to finding medial axis of a polygon in linear time [6]. (b) Using duality to check whether it can help to reduce the time complexity of computing straight skeleton.

Appendix A

A few false starts[1]

Let us see few approaches which we have tried and found that they don't improve the complexity of straight skeleton computation[1].

A.1 Triangulating a monotone polygon

First triangulate the monotone polygon. The bisector of each vertex will lie in a unique triangle. (for the time being, disregard the degenerate cases.) Then in the dual graph of triangulation, process the nodes with degree one in successive iterations.

At start of the iteration, the nodes of degree one are placed into a list. During the iteration, each node (i.e. the corresponding triangle) of the list is processed as discussed below. At the end of iteration, all the nodes belonging to the list are deleted from the graph. We iterate like this till all the nodes in the graph are processed.

Processing a node in the dual graph:

If the bisectors of some vertices lie in the current triangle, compute intersection of neighbouring bisectors. It is also possible that some bisectors might enter from the adjoining triangles are also taken into account while computing the intersections. Out of those intersection points which lie in this triangle, the one having least distance to its corresponding edge becomes the new node of straight skeleton. Compute the bisector at this new node. As before, check for intersection of remaining with this new bisector. When none of the neighbouring bisector pairs intersect inside this triangle, we mark the node corresponding to this triangle as processed. The bisectors which remain un-intersected at end of processing current triangle are carried forward to the adjacent triangle.

Objective: A monotone polygon can be triangulated in linear time [11]. By processing each triangle only once, we attempted to get an algorithm with linear time complexity.

Problem: Our wrong assumption was that the number of bisectors per triangle is

[1]This part is included in the report so that in future, time is not wasted in following these approaches.

constant. Also, triangulation of a polygon is not unique. This will have a significant impact on the implementation of such a procedure.

Counter-example: A regular polygon's straight skeleton will take $O(n^2)$ time using the above procedure. Observe that all bisectors of regular polygon will meet at one point.

A.2 Successive neighbouring bisectors' intersections for case of Convex Polygon

The idea was to compute the intersections between all pairs of neighbouring bisectors. The intersection point having minimum distance to its corresponding edge becomes the new node of straight skeleton. Compute bisector at new node and its intersections with its neighbouring bisectors. The nearer of these intersections to its becomes the next node of straight skeleton. Continuing like this, compute the straight skeleton of given convex polygon.

Objective: Objective was to compute straight skeleton of convex polygon in linear time.

Problem: Because of non-local nature of events, the straight skeleton computed using above method may not be correct. A counter-example can be easily constructed such that after finding the first correct node of straight skeleton, the next node is not given by the bisector at this latest created node, but by some other pair of the bisectors. So it results in wrong computation. To correct it, we have to make a linear scan after each computation of new node. This modification for correct computation makes the method $O(n^2)$.

Appendix B

Implementation of Straight Skeleton computation algorithm

For the sake of completeness, the algorithm implemented for computation of straight skeleton is given below. The algorithm given by [4] has been implemented in C during the dissertation. The results are available in two formats-

1. an interactive GUI which displays the computed straight skeleton graphically and allows chronological tracing of the events during the shrinking process
2. a text-file listing the nodes (with coordinates) and the arcs of the computed straight skeleton.

The basic data structure used by the algorithm is a set of circular lists of active vertices (SLAV). This structure stores a loop of vertices for polygon boundary and for sub-polygons created during the straight skeleton computation. In the case of convex polygon, it always contains only one list(since no split events occur for convex polygon). In the case of a simple non-convex polygon, it stores a list for every sub-polygon created during the shrinking process.

B.1 Convex Polygon Skeleton Ccmputation

Given a simple convex polygon P, only the edge events occur and the straight skeleton S(P) is computed in the following steps (We suppose the polygon vertices and edges are oriented counter-clockwise and the polygon interior is on the left-hand side of its boundary):

1. Initialization:

   (a) Organize given vertices $V_1, V_2, \ldots, V_n$ into one double connected circular list of active vertices (LAV) stored in SLAV. The vertices in LAV are all active at this moment.

(b) for each vertex V_i in LAV add the pointers to two incident edges $e_{i-1} = V_{i-1}V_i$ and $ei = V_iV_{i+1}$, and compute the vertex angle bisector (ray) b_i,

(c) for each vertex V_i compute the nearer intersection of the bisector b_i with adjacent vertex bisectors b_{i-1} and b_{i+1} starting at the neighboring vertices V_{i-1}, V_{i+1} and (if it exists) store it into a priority queue according to the distance to the line $L(e_i)$ which holds the edge e_i. For each intersection point I_i store also two pointers to the vertices V_a, V_b, that means two origins of bisectors which have created the intersection point I. They are necessary for the identification of appropriate edges (e_a and e_b) during the bisector computation in later steps of the algorithm.

2. While the priority queue with the intersection points is not empty do:

   (a) Pop the intersection point I from the front of the priority queue,

   (b) if the vertices/nodes V_a and V_b, pointed by I, are marked as processed then continue on the step 2, else the edge e between the vertices/nodes V_a, V_b shrinks to zero (edge event),

   (c) if the predecessor of the predecessor of V_a is equal to V_b (peak of the roof) then output three straight skeleton arcs V_aI, V_bI and V_cI, where V_c is the predecessor of V_a and the successor of V_b in the LAV simultaneously, and continue on the step 2,

   (d) output two skeleton arcs of the straight skeleton V_aI and V_bI,

   (e) modify the list of active vertices/nodes

   - Mark the vertices/nodes V_a; V_b (pointed to by I) as processed,
   - create a new node V with the coordinates of the intersection I,
   - insert this new node V into the LAV. That means connect it with the predecessor of V_a and the successor of V_b in the LAV,
   - link the new node V with appropriate edges e_a and e_b (pointed to by the vertices V_a and V_b),

   (f) for the new node V , created from I, compute:

   - a new angle bisector b between the line segments e_a and e_b, and
   - the intersections of this bisector with the bisectors starting from the neighbour vertices in the LAV in the same way as in the step 1c,
   - store the nearer intersection (if it exists) to the priority queue.

It can be seen that in the steps 1c and 2f, there are duplicities among the intersection points in the priority queue. The algorithm always computes one intersection for one vertex. Step 2b removes these duplicities.

B.2 The algorithm for non-convex polygons

Algorithm for non-convex polygons is in principle similar to the algorithm described in the section B.1. As an extension, it handles the intersections that may generate polygon splits (split events):

1. Initialization:

   (a) Generate one LAV as in the convex case, store it in SLAV,

   (b) compute the vertex bisectors as in the convex case,

   (c) compute intersections with the bisectors from the previous and the following vertices as in the convex case and for reflex vertices compute also the intersections with the "opposite" edges. Store the nearest intersection point I of these three ones into the priority queue. In addition store also the type of the intersection point (edge event or split event).

2. While the priority queue is not empty do:

   (a) Pop the lowest intersection I from queue as in the convex case. If the type of I is the edge event, then process steps 2b to 2g of the algorithm for convex polygons, else (split event) continue within this algorithm,

   (b) if the intersection point points to already processed vertices continue on step 2 as in the convex case,

   (c) do the same as in the convex case, only the meaning is a bit different, because more local peaks of the roof exist,

   (d) output one arc V-I of the straight skeleton, where vertex/node V is the one pointed to by the intersection point I. Intersections of the split event type point exactly to one vertex in LAV/SLAV,

   (e) modify the set of lists of active vertices/nodes (SLAV):

   - Mark the vertex/node V (pointed to by I) as processed,
   - create two new nodes V_1 and V_2 with the same coordinates as the intersection point I,
   - search the opposite edge in SLAV (sequentially),
   - insert both new nodes into the SLAV (break one LAV into two parts. Vertex V_1 will be interconnected between the predecessor of V and the vertex/node which is an end point of the opposite line segment. V_2 will be connected between the successor of V and the vertex/node which is a starting point of the opposite line segment. This step actually splits the polygon shape into two parts (as discussed before in this section).
   - link the new nodes V1 and V2 with the appropriate edges.

   (f) for both nodes V_1 and V_2:

   - compute new angle bisectors between the line segments linked to them in step 2e,
   - compute the intersections of these bisectors with bisectors starting at their neighbor vertices according to the LAVs, the same way as in step 1c. New intersection points of both types may occur, and
   - store the nearest intersection into the priority queue.

Appendix C

An example for Conjecture 3.1

Figure 15 gives an instance of a monotone polygon generated interactively. The straight skeleton of this polygon is computed. The table below describes the shrinking process. It can be noticed that each reflex vertex marks exactly one edge for split during the shrinking process (while inserting in the priority queue).

Distance	Reflex Vertex			Edge split(v1 to v2)			format: (id) x y		
20.565	(2)	142	268	(28)	166	247	(29)	130	238
9.428	(4)	176	260	(27)	202	228	(28)	166	247
17.001	(5)	214	255	(26)	254	226	(27)	202	228
2.797	(7)	235	231	(26)	254	226	(27)	202	228
5.119	(8)	259	238	(25)	278	236	(26)	254	226
3.296	(10)	291	240	(23)	302	245	(24)	282	227
5.211	(12)	353	246	(19)	362	242	(20)	337	231
10.665	(21)	333	241	(11)	334	284	(12)	353	246

Priority Queue built

Split Events that actually take place:

Distance	Reflex Vertex			Edge split(v1 to v2)			format: (id) x y		
3.296	(10)	291	240	(23)	302	245	(24)	282	227
2.797	(7)	235	231	(26)	254	226	(27)	202	228
5.211	(12)	353	246	(19)	362	242	(20)	337	231
5.119	(8)	259	238	(25)	278	236	(26)	254	226
9.428	(4)	176	260	(27)	202	228	(28)	166	247
20.565	(2)	142	268	(28)	166	247	(29)	130	238
10.665	(21)	333	241	(11)	334	284	(37)	353	241

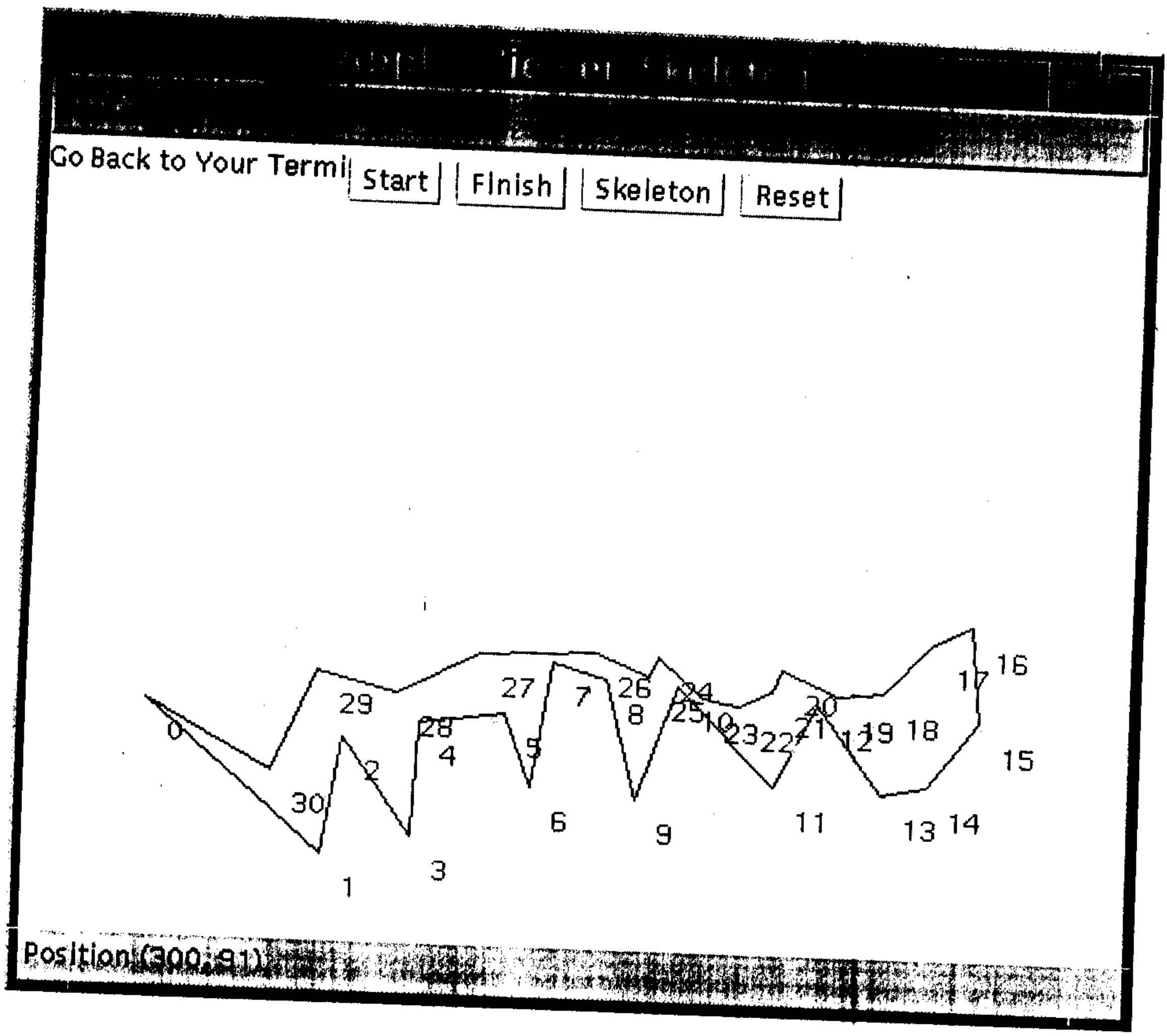


Figure 15. An example for conjecture 3.1

Bibliography

[1] O. Aichholzer and F. Aurenhammer. "Straight skeletons for general polygonal figures in the plane". Proc. 2nd Annu. Int'l. Computing and Combinatorics Conf., 1996, pp. 117-126.

[2] O. Aichholzer, F. Aurenhammer, D. Alberts, and B. Gartner. "A novel type of skeleton for polygons". The Journal of Universal Computer Science, 1(1995), pp. 752-761.

[3] D. Eppstein and J. Erickson. "Raising roofs, crashing cycles, and playing pool: Applications of a data structure for finding pairwise interactions". Discrete Comput. Geom., 22 (1999), pp. 569-592.

[4] P. Felkel and S. Obdrzlek. "Straight Skeleton Implementation". Proceedings of Spring Conference on Computer Graphics, Budmerice, Slovakia. ISBN 80-223-0837-4. pp. 210-218.

[5] H. Blum, "A transformation for extracting new descriptors of shape". Proc. Symp. Models for Perception of Speech and Visual Form. pp. 362-380, MIT Press, 1967.

[6] Francis Chin, Jack Snoeyink, and Cao An Wang. "Finding the Medial Axis of a Simple Polygon in Linear Time", Proc. 6th Ann. Int. Symp. Algorithms and Computation (ISAAC 95), Lecture Notes in Computer Science 1004, pp. 382-391, 1995.

[7] D.T. Lee. "Medial axis transformation of a planar shape". IEEE Trans. Pattern Analysis and Machine Intelligence, PAMI-4 (1982), pp. 363-369.

[8] F. Aurenhammer. "Voronoi diagrams - a survey of a fundamental geometric data structure", ACM Computing Surveys 23, 3 (1991), pp. 345-405.

[9] D. Eppstein. "Fast hierarchical clustering and other applications of dynamic closest pairs". Proc. 9th Annu. ACM-SIAM Sympos. Discrete Algorithms, pp. 619-628. 1998.

[10] D.G. Kirkpatrick. "Efficient computation of continuous skeletons". Proc. 20th Ann. IEEE Symp. FOCS (1979), pp. 18-27.

[11] M. de Berg, M. vam Kreveld, M. Overmars, and O. Schwarzkopf. "Computational Geometry, Algorithms and Applications". Springer Verlag Berlin Heidelberg New York, 1997.

[12] F. P. Preparata and M. I. Shamos. "Computational Geometry - An Introduction". Springer-Verlag,New york, 1985.