アルゴリズム 67－4
（1999．3．15）

安全領域問題

シュバス C. ナンディ　バルガム B. バタッチャリヤ
Indian Statistical Institute
カルカッタ 700 035 インド

アブストラクト：単純な多角形 P を与えたとき、その (幅δの) 安全領域 (*safety zone*) G とは、線分と (半径δの) 円弧からなる閉領域で、多角形 Pの境界上の点 p と G の境界上の点 qとのユークリッド距離 $d(p,q)$ がδ未満となる点対 (p,q) を持たずに多角形 Pをかこむ領域のことである。本稿では、任意の単純多角形の最小面積安全領域を線形時間で求めるアルゴリズムを述べる。

Safety Zone Problem

Subhas C. Nandy[1] and *Bhargab B. Bhattacharya*
Indian Statistical Institute
Calcutta 700 035, INDIA

Abstract : Given a simple polygon P, its *safety zone* G (of width δ) is a closed region consisting of straight line segments and circular arcs (of radius δ), bounding the polygon P such that there exists no pair of points p (on the boundary of P) and q (on the boundary of G), having their Euclidean distance $d(p,q)$ less than δ. In this paper, we present a linear time algorithm for finding the minimum area safety zone of an arbitrary shaped simple polygon.

Key words : Polygon triangulation, convex hull, resizing of VLSI circuits, algorithm, complexity.

1 Introduction

In this paper, we introduce a new problem, called *safety zone problem* which is as follows : given a simple polygon P and a fixed parameter δ, the safety zone (of width δ) of the polygon P is a closed region S_P of minimum area, such that P is completely inside S_P, and there exists no pair of points p and q, where p is on the boundary of P and q is on the boundary of S_P, such that $d(p,q)$, the Euclidean distance between p and q, is less than δ. Here S_P is not a polygonal region. It is composed of straight line segments and circular arcs of radius δ, where each straight line segment is parallel to an edge of the polygon at a distance δ from that edge, and each circular arc (of radius δ) is centered at an unique vertex of the polygon. The boundary of S_P describes a *simple* region in the sense that its no two edges intersect in their interior. It is easy to observe that for every point q on the boundary of the *safety zone* of the polygon P, there exists at least one point p on P such that $d(p,q) = \delta$. The safety zone of a simple polygon is demonstrated in Figure 1. In this context, It is worth to mention that the *safety zone* (of width δ) of a simple polygon P is the outer boundary of the Minkowski sum of P and a circle of radius $\frac{\delta}{2}$. It is already known that the Minkowski sum of a monotone polygon and a convex polygon can also be computed in time linear in the total number of vertices of both the polygons [1]. But to our knowledge, no linear time algorithm exists for computing the Minkowski sum of an arbitrary simple polygon and a circle.

We present an algorithm for finding the minimum area safety zone of an arbitrary shaped simple polygon. Using Chazelle's linear time polygon triangulation algorithm [2], we show that the time complexity of our algorithm is $O(N)$, where N is the number of vertices of the polygon. Our main motivation for defining this problem comes from the problem of resizing the circuit components in VLSI physical design [5]. It will also find its other applications in designing automatic metal cutting equipments, robot motion planning, to name a few.

[1] Currently at School of Information Sciences, Japan Advanced Institute of Science and Technology, JAPAN.

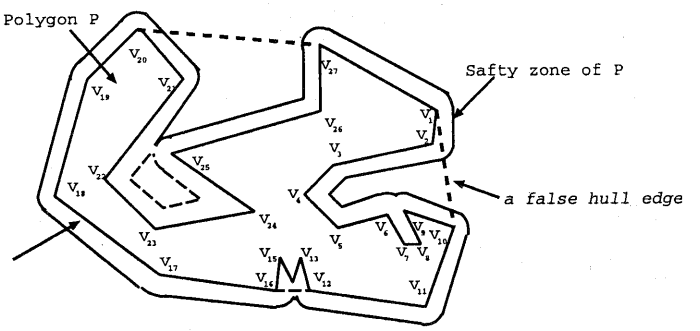

Figure 1 : Safety zone of a simple polygon

2 Preliminaries

We classify a vertex of the polygon P as *concave* and *convex* depending on whether the angle between its associated edges inside the polygon is greater than 180° or less than 180°. Consider the convex hull of the polygon P. Choose a vertex v on the convex hull and label the vertices of P as $v_1(=v), v_2, \ldots, v_n$, moving along its boundary in clockwise direction. Now consider a *PEN*, having two tips which are δ distance apart. Its one *tip* moves along the polygonal boundary from vertex v_1 in clockwise direction, and the other *tip* draws the safety zone staying orthogonal to the direction of motion of the former *tip*. Needless to say, the safety zone S_P of an N vertex convex polygon can easily be obtained in $O(N)$ time. So, we concentrate on describing the method of drawing the safety zone of an arbitrary simple polygon. Here, each hull edge may be classified as any of the following two types : (i) if it coincides with some edge of the polygon, it is called a *solid hull edge*, and (ii) if it does not coincide with any of the polygonal edges, it is called a *false hull edge*.

Definition : A *notch* is a polygonal region outside the polygon P which is formed with a chain of edges of P initiating and terminating at the two vertices of a *false hull edge*. Clearly the notches indicated by different *false hull edges* will have disjoint sets of vertices.

The boundary of the *safety zone* S_P consists of the safety zones of the *solid hull edges* and hull vertices, and the safety zone of all the *notches*. As the safety zone of *solid hull edges* and hull vertices are easy to obtain, our problem now reduces to designing an efficient algorithm for drawing the safety zone inside a *notch*.

3 Safety zone inside a notch

During traversal along the boundary of the convex hull, the *PEN* identifies a *notch* when it encounters a *false hull edge*. Consider such a *notch* attached to a *false hull edge* v_iv_{i+k} and having $k+1$ vertices, labeled by $v_i, v_{i+1}, \ldots, v_{i+k}$. First of all, we triangulate the *notch* using a linear time algorithm due to Chazelle [2]. An edge of a triangle is termed as *triangulation edge* if it is generated due to the triangulation, otherwise it will be referred to as a *polygonal edge*. Each triangle must have at least one *triangulation edge*. It is easy to see that the graph constructed with each triangle as a node, is a tree [3], referred to as *triangulation tree*. Each edge of the tree can be mapped to a *triangulation edge*. We classify the triangles into three categories, *type-A*, *type-B* and *type-C*, depending on whether the number of its *triangulation edge*(s) is one, two or three. The *root* node of the tree corresponds to the triangle adjacent to the *false hull edge*, and directions are assigned to the edges by traversing the tree in depth first manner. It is easy to observe that the *type-A* triangles correspond to the leaf nodes of the tree, and each internal node may have one or two out-degree(s) depending on whether the corresponding triangle is *type-B* or *type-C*. A triangle with vertices v_i, v_j and v_k will be referred to as $\Delta v_iv_jv_k$. The triangulation edge of $\Delta v_iv_jv_k$, through which the control reaches this triangle during the forward traversal, will be referred to as the

incoming edge of $\Delta v_i v_j v_k$. Its other triangulation edges (if any) will be referred to as *outgoing edges.*

While drawing safety zone inside a *notch*, the tree is processed in post-order. After traversing the subtree(s) of a node, when the node is processed, the safety zones for all the elements (vertices and *solid triangulation edges*, if any) of the corresponding triangle are drawn. The safety zone of an element c_i will be referred to as *component of safety zone* ($CSZ(c_i)$). Next, a merge pass is performed to check for possible intersection among CSZs' of already drawn elements inside that triangle. Now note that, the CSZs' that are present inside a triangle may also intersect with the CSZs', that will be generated while processing the predecessor or the other sibling of that triangle. So, after processing the current triangle, we need to propagate a subset of these CSZs' to the predecessor of the current triangle. Below we describe the concept of visibility list that will aid the merge pass inside a triangle and the propagation of CSZs' from one triangle to its predecessor.

3.1 Visibility list

Let $\Delta v_i v_j v_k$ be the triangle under current processing, whose incoming edge is $v_i v_k$. In order to detect possible intersections among the CSZs' generated after processing the tree rooted at the current node (triangle) with some other CSZs' generated in its predecessor or the other sibling, we introduce the concept of *visibility list* as follows.

Consider a pair of lines ℓ and ℓ', both parallel to $v_i v_k$ at a distance δ from it. Let ℓ crosses the interior of $\Delta v_i v_j v_k$ and ℓ' stays completely outside of $\Delta v_i v_j v_k$. The component of the safety zone of an element, say CSZ^*, which is present inside $\Delta v_i v_j v_k$ may intersect another CSZ, say CSZ^{**}, belonging to its predecessor, if CSZ^* spans above ℓ. The reason is that CSZ^{**} can not penetrate inside the triangle beyond ℓ due to the width constraint δ. Surely, a CSZ^* can not also span above the line ℓ', from the interior of $\Delta v_i v_j v_k$.

Definition : The *active zone* of a triangulation edge $v_i v_k$ is a connected region inside the polygon bounded by ℓ, ℓ' and which contains $v_i v_k$ in its interior. In Figure 2, the shaded area is the active zone of the triangulation edge $v_i v_k$.

A CSZ is said to be inside the active zone of an edge if it (or a part of it) spans inside that region, and this can easily be tested by comparing the CSZ with ℓ or ℓ' corresponding to that edge.

Definition : A *component of the safety zone* (CSZ) is said to be *visible* to a *triangulation edge* ($v_i v_k$) if it spans in the *active zone* of $v_i v_k$, and a line drawn from any point of $v_i v_k$ and perpendicular to it, cuts the component before cutting any other component of the safety zone or some edge of the polygon.

Lemma 1 *The projections (foot of perpendiculars) of the end points of the members of V-LIST($v_i v_k$) on $v_i v_k$ are linearly ordered along it.* □

Definition : The *visibility list V-LIST($v_i v_k$)*, attached to a *triangulation edge* $v_i v_k$, is a doubly connected link list containing a set of CSZs' which are drawn while processing the tree rooted at the current node, and are *visible* to that *triangulation edge.* Its each member is attached with a *count* field. The role of *count* field will be clear in due course.

It needs to mention that the *V-LIST* of a *polygonal edge* will consist two members : (i) CSZ of the *polygonal edge*, which is a line segment parallel to that edge, and (ii) the CSZ of the vertex, which is common to that *polygonal edge* and the incoming edge of the triangle.

While processing the triangle $\Delta v_i v_j v_k$, it is assumed that *V-LIST*s of its outgoing edges ($v_j v_i$) and ($v_j v_k$) are already available; they are either inherited from its successor(s), or are prepared by drawing the CSZs' of the elements (vertices/edges) of the triangle at the beginning of processing this triangle.

3.2 Data structure

While processing a *notch*, the algorithm maintains the following data structures.

poly_chain : It is an array of vertices and edges of the notch stored in clockwise order.

$\mathbf{DS}_1$: The triangulation tree of the *notch.* With each edge of the tree a *V-LIST* is maintained as described in the previous section.

$\mathbf{DS_2}$: A list containing CSZ for the elements (vertices and polygonal edges) inside a *notch*. Each element in this list has a pointer to its neighboring CSZ in clockwise order. This will be recursively constructed while traversing the tree. At the time of processing a triangle, the CSZ of its edge(s) and vertices inside the triangle are created in the $\mathbf{DS_2}$ list if they are not already present. During merge pass, if an intersection between two CSZs' (say CSZ^* and CSZ^{**}) is detected, these CSZs' are updated by removing the portions behind the the intersection. An appropriate pointer is established among CSZ^* and CSZ^{**}. Finally, after processing the root node, the $\mathbf{DS_2}$ list gives the safety zone of the *notch*.

3.3 Merging a pair of *V-LISTs*

Let $\Delta v_i v_j v_k$ be a triangle under process, whose incoming edge is $v_i v_k$ and two outgoing edges are $v_j v_i$ and $v_j v_k$ (see Figure 2). In order to detect the intersection(s) among CSZs' present in *V-LIST*$(v_j v_i)$ and *V-LIST*$(v_j v_k)$, they are merged from their end corresponding to v_j with the help of two pointers indicating the current elements of the respective lists. By Lemma 1, we can fix a point π_{ji} on $v_j v_i$. Our choice of π_{ji} is such that, during the merge pass inside $\Delta v_i v_j v_k$, a $CSZ \in$ *V-LIST*$(v_j v_i)$ can never intersect with any CSZ of *V-LIST*$(v_j v_k)$ if the projection of the former one on $v_j v_i$ lies completely outside the line segment $v_j \pi_{ji}$.

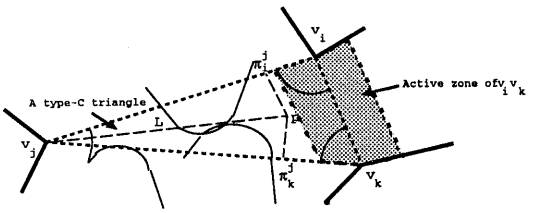

Figure 2 : Demonstration of merge pass

Definition : During the processing of $\Delta v_i v_j v_k$, a $CSZ \in$ *V-LIST*$(v_j v_i)$ is said to be *favorable*, if the projection of its at least one end point on $v_j v_i$ lies on the line segment $v_j \pi_{ji}$.

While processing $\Delta v_i v_j v_k$, our merge pass progresses along *V-LIST*$(v_j v_i)$ and *V-LIST*$(v_j v_k)$ starting from their ends corresponding to v_j. This merge pass terminates as soon as it finds a CSZ in either *V-LIST*$(v_j v_i)$ or *V-LIST*$(v_j v_k)$ which is not *favorable* inside $\Delta v_i v_j v_k$. The choice of π_{ji} and π_{jk} follows from the following observation :

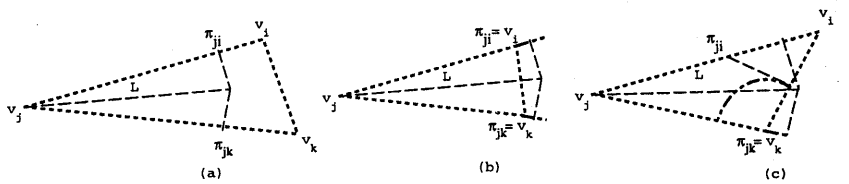

Figure 3 : Selection of the members of two V-LISTs corresponding to the outgoing edges of a triangle which are favourable for the merge pass inside the triangle.

Observation 1 Consider the bisector L of the $\angle v_i v_j v_k$ and choose a point p on it such that the length of the perpendiculars on $v_j v_i$ and $v_j v_k$ from p is equal to δ. Let the foot of perpendiculars of p on $v_j v_i$ and $v_j v_k$ be α and β respectively. Now the following cases arise.

Case 1 : Both of α and β fall on the closed segment $v_j v_i$ and $v_j v_k$ respectively. In this case, a CSZs' of *V-LIST*$(v_j v_i)$ (*V-LIST*$(v_j v_k)$) becomes favorable if the projection (foot of perpendicular) of its at least one end point on $v_j v_i$ ($v_j v_k$) is closer to v_j than α (β) (see Figure 3a). Thus, in this case, $\pi_{ji} = \alpha$ and $\pi_{jk} = \beta$.

Case 2 : Both of α and β fall outside the closed segment $v_j v_i$ and $v_j v_k$ respectively, and $\Delta v_i v_j v_k$ is an acute angle triangle. Here the $CSZ(v_i)$ and $CSZ(v_k)$ will intersect. But, there may exist some CSZ who can intersect both of $CSZ(v_i)$ and $CSZ(v_k)$. So, a merge pass inside this triangle is needed, and $\pi_{ji} = v_i$ and $\pi_{jk} = v_k$ in this case (see Figure 3b).

Case 3 : α lies inside the closed segment v_jv_i but β is outside the closed segment v_jv_k. Here $CSZ(v_k)$ must participate in the merge pass. Now consider a line perpendicular to v_iv_k which touches (not intersects) $CSZ(v_k)$ inside $\Delta v_iv_jv_k$. Let it be meeting v_jv_i at a point γ. Note that, as the vertex v_k lies in one side of the line v_jv_i, and the centers of all the CSZs' in $V\text{-}LIST(v_jv_i)$ lie in the other side of v_jv_i, if any one of them intersects $CSZ(v_k)$, its projection on v_jv_i must overlap the line segment $v_j\gamma$ (as shown in Figure 3c). So, in this case a $CSZ \in V\text{-}LIST(v_jv_i)$ remains favorable if its projection on v_jv_i overlaps the line segment $v_j\gamma$. Thus, we have $\pi_{ji} = \gamma$ and $\pi_{jk} = v_k$ in this case. A similar case arises if α lies outside the closed segment v_jv_i but β is inside the closed segment v_jv_k.

Case 4 : Both of α and β fall outside the closed segment v_jv_i and v_jv_k respectively, and $\Delta v_iv_jv_k$ is an obtuse angle triangle whose $\angle v_jv_kv_i > 90^o$. As $CSZ(v_k)$ must participate in the merge pass, here $\pi_{jk} = v_k$ and π_{ji} is chosen in a similar manner as in the earlier two cases. □

During the merge inside $\Delta v_iv_jv_k$, let $CSZ^* \in V\text{-}LIST(v_jv_i)$ be a favorable candidate. First of all, we test whether CSZ^* penetrates inside the active zone of v_jv_k or not. In case of negative answer, we skip CSZ^* and consider the next element of $V\text{-}LIST(v_jv_i)$. But in case of affirmative answer, we need to check CSZ^* with the members of $V\text{-}LIST(v_jv_k)$ for possible intersection. We draw perpendiculars from the end points of CSZ^* on L, which hit L at the points a_1 and a_2. Now, the favorable members of $V\text{-}LIST(v_jv_k)$, whose projections on L overlap a_1a_2, will be considered one by one to detect for possible intersection(s) with CSZ^*, if any. The cost of each comparison, excepting the last one, is charged by incrementing the *count* field of the participating element of $V\text{-}LIST(v_jv_k)$. For the last comparison, we charge its cost to CSZ^*. The merge pass then proceeds considering the next element of $V\text{-}LIST(v_jv_i)$. As soon as a member in either of $V\text{-}LIST(v_jv_i)$ or $V\text{-}LIST(v_jv_k)$ is reached which is not *favorable* inside $\Delta v_iv_jv_k$, the merge pass terminates.

During the merge pass, the intersection (if any) which is observed most recently, are preserved. Let the participating members be $CSZ^* \in V\text{-}LIST(v_jv_i)$ and $CSZ^{**} \in V\text{-}LIST(v_jv_k)$, at the end of the merge pass. We update CSZ^* and CSZ^{**} by deleting the portions behind the point of intersection, and establish an appropriate pointer among CSZ^* and CSZ^{**} in the $\mathbf{DS_2}$ list.

The processing of the current triangle ends by decrementing the *count* field of last two compared element during the merge pass inside it. Conceptually, the cost of last comparison is charged to the triangle itself. As the merge pass inside a triangle is performed at most once, such a charging to a triangle may also be done at most once during the entire execution process of the *notch.*

Lemma 2 *The merge pass inside a triangle requires time linear to the number of elements in the V-LISTs of both of its outgoing edges.* □

Lemma 3 *If $\angle v_iv_jv_k > 90^o$, members of $V\text{-}LIST(v_jv_i)$ and $V\text{-}LIST(v_jv_k)$ will not intersect.*

Proof : Let us draw the projections of the point $p \in L$ on v_jv_i and v_jv_k (as defined in the first paragraph of this section) which touch the respective lines at α and β respectively. As $\angle pv_jv_i$ ($\angle pv_jv_k$) $\geq 45^o$, both $v_j\alpha$ and $v_j\beta$ will be less than δ. So all the CSZs' of $V\text{-}LIST(v_jv_i)$ ($V\text{-}LIST(v_jv_k)$) spanned over α (β) have been be covered by $CSZ(v_j)$, and are not present in the respective $V\text{-}LIST$s. Again as π_{ji} and π_{jk} is determined by α and β respectively in this case, the merge pass need not be executed in such a triangle. Hence the result follows. □

3.4 Creation of new *V-LIST*

After the completion of the merge pass inside a triangle $\Delta v_iv_jv_k$, the *V-LIST* of its incoming edge v_iv_k is created by the selected members of $V\text{-}LIST(v_jv_i)$ and $V\text{-}LIST(v_jv_k)$. In this context, the following results are important.

Lemma 4 *If $\angle v_jv_iv_k > 90^o$ then, the members of $V\text{-}LIST(v_jv_i)$ will not intersect with the CSZs' generated inside the other sibling or in the predecessor triangle of the current node. So, no element of $V\text{-}LIST(v_jv_i)$ need to be propagated to $V\text{-}LIST(v_iv_k)$.* □

Lemma 5 *If $\angle v_iv_kv_j > 90^o$, there exists at most one element of $V\text{-}LIST(v_jv_i)$ which will be compared with members of $V\text{-}LIST(v_jv_k)$ as well as propagated to $V\text{-}LIST(v_iv_k)$.*

Proof : Consider a pair of elements CSZ^* and $CSZ^{**} \in$ *V-LIST*(v_jv_i) which are *favorable* with respect to the merge pass inside $\Delta v_iv_jv_k$. Let CSZ^* appears after CSZ^{**} during the above merge pass. The question of propagation of CSZ^{**} to *V-LIST*(v_iv_k) arise if CSZ^{**} is inside the active zone of v_iv_k. Surely, CSZ^* will also be inside the active zone of (v_iv_k) in this case, and will be compared with $CSZ(v_k)$. Now if CSZ^* intersects $CSZ(v_k)$, then CSZ^{**} will immediately be deleted from the $\mathbf{DS}_2$ list. Otherwise, by Case 3 of Observation 1, as π_{ji} is the point of intersection of v_jv_i, and a vertical line which is the tangent of $CSZ(v_k)$, the orthogonal visibility of CSZ^{**} from v_iv_k is lost. So, CSZ^{**} can not belong to *V-LIST*(v_iv_k). Thus it is sufficient to propagate CSZ^* to *V-LIST*(v_iv_k). □

Thus Lemmata 3, 4 and 5 lead to the fact that if the triangle $\Delta v_iv_jv_k$ is an obtuse angle triangle, there may exist at most one element of *V-LIST*(v_jv_i) which (i) will be compared with members of *V-LIST*(v_jv_k) during the merge pass inside $\Delta v_iv_jv_k$, and (ii) will be propagated to *V-LIST*(v_iv_k). So, it remains to study the situation if $\Delta v_iv_jv_k$ is an acute angle triangle. As $\angle v_iv_jv_k < 90^o$, the merge pass has already been performed inside $\Delta v_iv_jv_k$. Similar to the Observation 1, let us consider the point p on the bisector of the angle $\angle v_iv_jv_k$, and its projection α on v_jv_i. By Case 1 of Observation 1, α determines π_{ji}. Now,

if α lies outside the active region of v_iv_k (see Figure 3a) then the members of *V-LIST*(v_jv_i), that have participated in the merge pass inside $\Delta v_iv_jv_k$, will fail to belong inside the active zone of v_iv_k. So, they need not be propagated to *V-LIST*(v_iv_k).

On the contrary, if α lies inside the active region of v_iv_k (as in Figure 3b), few elements of *V-LIST*(v_jv_i) which has been compared in $\Delta v_iv_jv_k$ may need to be propagated to *V-LIST*(v_iv_k).

During the progress of the merge pass inside a triangle $\Delta v_iv_jv_k$, two pointers are maintained corresponding to the *V-LIST*s of its two outgoing edges v_jv_i and v_jv_k. At the end of the merge pass, each of them will contain the address of an element of the respective *V-LIST* which has been compared last during the current merge pass. Let $CSZ' \in$ *V-LIST*(v_jv_i) and $CSZ'' \in$ *V-LIST*(v_jv_k) be the above two elements. The *V-LIST* of the incoming edge v_iv_k will be constructed by following the steps described below.

- If the projection of both the end points of CSZ' is closer to v_j than π_{ji}, the pointer along *V-LIST*(v_jv_i) advances to get its first member, say CSZ^*, which lies inside the active zone of v_iv_k.
- If the point π_{ji} lies in the interior of the projection of CSZ' on v_jv_i, then CSZ^*, the first member which lies to the active zone of v_iv_k, is CSZ' itself.
- If the projection of both the end points of CSZ' is far from v_j than π_{ji}, the pointer along *V-LIST*(v_jv_i) backtracks to get its first member CSZ^*, lying inside the active zone of v_iv_k.
- Any one of the aforesaid three situations may appear regarding the selection of CSZ^{**}, first member of *V-LIST*(v_jv_k), which lies inside the active zone of v_iv_k.
- Finally, CSZ^* and CSZ^{**} are connected using bidirectional pointers. Now, the *V-LIST* of the incoming edge v_iv_k is a list of CSZs' whose two terminal members correspond to the last element of *V-LIST*(v_jv_i) towards v_i and the last element of *V-LIST*(v_jv_k) towards v_k respectively.

Now note that, the elements of *V-LIST*(v_jv_i) (*V-LIST*(v_jv_k)) which are encountered during back track, are considered during current merge pass and their *count* fields have been incremented. But the *count* field of CSZ' (CSZ'') remains unchanged as it is considered last during current merge pass.

¿From the earlier discussions we notice that while processing a *type-B* and a *type-C* triangle, there may exist some situation when some CSZ of *V-LIST*(v_jv_i) may be considered in the merge pass of both $\Delta v_iv_jv_k$ and in its predecessor triangle. This gives an impression that after the generation of a CSZ, it may participate in the merge pass of different triangles arising along a path of $\mathbf{DS}_1$, which indicates an $O(n^2)$ time complexity of our algorithm. Below we prove that a CSZ, after its generation, may participate in the merge pass of at most two triangles.

Theorem 1 *After the generation of a CSZ, its count field may be incremented to at most 2 during its propagation while backtrack along* $\mathbf{DS}_1$.

Proof : In order to prove this result, we have to study two triangles along a path of $\mathbf{DS}_1$. We assume that currently the control is in the successor triangle, and the merge pass has already been performed inside the current triangle. Our aim is to prove that, a CSZ whose *count* field is incremented during

the current merge pass, will participate to at most one more merge pass inside some other predecessor triangle incrementing its *count* field. For the sake of simplicity in the proof, we assume that the two triangles under consideration share a common triangulation edge.

Let $\Delta v_i v_j v_k$ be the current triangle under consideration and $\Delta v_i v_k v_\ell$ be its predecessor triangle. If $\angle v_i v_j v_k > 90^o$, then by Lemma 3, the merge pass need not be performed inside $\Delta v_i v_j v_k$. So, we assume $\angle v_i v_j v_k < 90^o$. Surely, at least one of the other three angles of the quadrilateral, formed by concatenating $\Delta v_i v_j v_k$ and $\Delta v_i v_k v_\ell$, must be greater than 90^o. This gives birth to three different cases as follows :

Case 1 : $v_j v_i v_\ell > 90^o$. Here, by Lemma 3, the members of *V-LIST*$(v_j v_i)$ will not be able to enter in the active zone of the edge $v_i v_\ell$. Thus the members of *V-LIST*$(v_j v_i)$ will either not be compared with the members of *V-LIST*$(v_i v_\ell)$, or will not participate in the *V-LIST*$(v_i v_\ell)$, depending on whether $v_k v_\ell$ or $v_i v_\ell$ is the incoming edge of $\Delta v_i v_k v_\ell$. Thus in either case, their *count* fields will remain unchanged inside $\Delta v_i v_k v_\ell$.

Case 2 : $v_i v_\ell v_k > 90^o$. By Lemma 5, the two subsets of members of *V-LIST*$(v_i v_k)$, which will participate in the merge pass inside $\Delta v_i v_k v_\ell$ and which needs to be propagated to *V-LIST* of the incoming edge of $\Delta v_i v_k v_\ell$, may have at most one element common, whose *count* field will not be incremented during the merge pass inside $\Delta v_i v_k v_\ell$. Thus if there exists any member(s) of *V-LIST*$(v_j v_i)$, which also need(s) to be propagated to the *V-LIST* of the incoming edge of $\Delta v_i v_k v_\ell$, will have its *count* field unincremented during the merge pass of $\Delta v_i v_k v_\ell$.

Case 3 : $v_\ell v_k v_j > 90^o$. Here three situations, stated below, need to be considered separately.

Case 3.1 : $\angle v_i v_k v_\ell \geq 90^o$. If $v_i v_\ell$ is the incoming edge, the elements of *V-LIST*$(v_i v_k)$ will not participate in the merge pass with *V-LIST*$(v_k v_\ell)$ (by Lemma 3). So, the elements of *V-LIST*$(v_i v_k)$ which are inherited from *V-LIST*$(v_j v_i)$ and needs to be propagated to *V-LIST*$(v_i v_\ell)$ will have their *count* field unchanged inside $\Delta v_i v_k v_\ell$. Similarly, if $v_k v_\ell$ is the incoming edge, the elements of *V-LIST*$(v_i v_k)$ will not be propagated to *V-LIST*$(v_k v_\ell)$.

Case 3.2 : $\angle v_j v_k v_i \geq 90^o$. Here the elements of the *V-LIST*$(v_i v_k)$ which have been propagated from *V-LIST*$(v_j v_i)$ will have their *count* field unchanged inside $\Delta v_i v_j v_k$ (by Lemma 4). Among them, if some one participates in the merge pass inside $\Delta v_i v_k v_\ell$, its *count* field may be incremented to at most 2. In Case 3.3, we show that, the *CSZ*s' of $v_j v_i$ whose *count* field have been incremented during the merge pass inside two adjacent triangles, say $\Delta v_i v_j v_k$ and $\Delta v_i v_k v_\ell$, will not be propagated further. The same result is true if the *count* field of a *CSZ* is incremented during the merge pass of two triangles, not necessarily adjacent.

Case 3.3 : Both $\angle v_j v_k v_i$ and $\angle v_i v_k v_\ell$ are less than 90^o. As we are discussing the nature of propagation of the *CSZ*s' of *V-LIST*$(v_j v_i)$, we assume that $\angle v_j v_i v_k < 90^o$.

Case 3.3.1 : The incoming edge of $\Delta v_i v_k v_\ell$ is $v_k v_\ell$. We study this case by considering a hypothetical triangle $\Delta v_i v_j v_\ell$ whose incoming edge is $v_j v_\ell$ and the predecessor triangle is $\Delta v_j v_i v_\ell$. In other words, the triangulation edge $v_i v_k$ is assumed to be replaced by $v_j v_\ell$. After the merge pass among *V-LIST*$(v_j v_i)$ and *V-LIST*$(v_i v_\ell)$ the elements in *V-LIST*$(v_j v_\ell)$ will be linearly ordered on $v_j v_\ell$. Now arguing in a similar manner as in Case 2 of this proof, the elements in *V-LIST*$(v_j v_\ell)$ can be splitted into two subsets. One of these subsets will participate in the merge pass with *V-LIST*$(v_j v_k)$ and the other subset will be propagated to *V-LIST*$(v_k v_\ell)$. Among these two subsets, we may have at most one element common, whose *count* field will not be incremented during the merge pass of this hypothetical triangle $\Delta v_j v_k v_\ell$. This leads to the conclusion that there may exists at most one element of *V-LIST*$(v_j v_i)$ which will be considered in the merge passes of both the *actual* triangles $\Delta v_i v_j v_k$ and $\Delta v_i v_k v_\ell$.

Case 3.3.2 : The incoming edge of $\Delta v_i v_k v_\ell$ is $v_i v_\ell$. Let $CSZ^* \in$ *V-LIST*$(v_j v_i)$ be compared last during merge pass inside $\Delta v_i v_j v_k$. It may have been compared with $CSZ(v_k)$ or some element preceding $CSZ(v_k) \in$ *V-LIST*$(v_j v_k)$. The merge pass inside $\Delta v_i v_k v_\ell$ starts with $CSZ(v_k)$ as the first element of *V-LIST*$(v_k v_\ell)$. So, the subset of members (if any) of *V-LIST*$(v_j v_i)$ which need to be propagated to *V-LIST*$(v_i v_k)$ and the subset of *V-LIST*$(v_j v_i)$ that have participated in the current merge pass may have at most one element common, which is CSZ^*. Also note that, the *count* field of CSZ^* has not been incremented during the merge pass inside $\Delta v_i v_j v_k$. □

Based on the observations stated above related to the merging and creation of visibility lists, we are in a position to analyze the complexity of of our proposed method.

4 Complexity analysis

The convex hull of an N vertex simple polygon can be drawn in $O(N)$ time using the algorithm suggested in [4]. Let it gives birth to K *notches*, where the i-th *notch* is assumed to have N_i vertices. Below we describe a few results related to the analysis of the time complexity of processing the *notches.*

Lemma 6 *Time required for drawing the safety zone of a notch of N_i vertices is $O(N_i)$.*

Proof : The triangulation of a *notch* of N_i vertices can be done in $O(N_i)$ time [2]. It gives birth to $(N_i - 2)$ triangles. The total number of vertices and polygonal edges considering all the triangles are $3(N_i - 2)$ and $(N_i - 1)$ respectively. So, the time required for drawing of safety zone of $O(N_i)$ vertices and edges of the *notch* is $O(N_i)$. By Lemma 2, the time required to conduct the merge pass in each triangle is linear in the number of CSZs' attached to *V-LIST*s' of its outgoing edges. But a $CSZ(c_j)$ corresponding to an element c_j of the notch may participate in the merge pass more than one triangles. Each time when it is considered, either its *count* field is incremented or the triangles inside which the merge pass is conducted, is charged. Theorem 1 shows that for any CSZ generated inside a notch, its *count* field may be incremented to at most be 2. Moreover, a triangle is charged at most once, provided a merge pass takes place inside it. Thus, apart from generation of CSZs', the total time required for the merge passes of all the triangles is also $O(N_i)$. □

Lemma 7 *At any point of time, the total space occupied by the V-LISTs' of all the triangulation edges is $O(N)$.* □

Lemmata 6 and 7 lead to the fact that the time required for the generation of the *safety zone* of each notch is linear in the number of its vertices. Again, since the notches around the convex hull of the polygon are disjoint, the total time complexity for drawing safety zones of all the notches is also linear in the total number of vertices of all the notches.

The time required to draw the safety zone of all *solid hull edges* and the hull vertices may be $O(N)$ in the worst case. Next, we may have to concatenate safety zone of a notch to the safety zones of some other notch or *solid hull edge* which are attached with the two vertices of its corresponding *false hull edge.* This requires $O(K)$ time, where K is the number of notches. A further pass is required, for reporting the safety zone by traversing along the $\mathbf{DS}_2$ list. This requires time linear to the number of CSZs' describing the safety zone of the polygon. Thus we have the following theorem describing the time complexity of our proposed algorithm.

Theorem 2 *The time and space complexities of our proposed algorithm are both $O(N)$, where N is the total number of vertices of the polygon.* □

Acknowledgment : The first author acknowledges Dr. Antonio Hernnández Barrera for pointing out the relation of the safety zone problem with the Minkowski sum of a simple polygon and a circle of specified radius.

References

[1] A. Hernández-Barrera, *Computing the Minkowski sum of monotone polygons*, IEICE Transactions on Information and Systems, vol. E80-D, No. 2, pp. 218-222, 1996.

[2] B. Chazelle, *Triangulating a simple polygon in linear time*, Discrete and Computational Geometry, vol. 6, pp. 485-524, 1991.

[3] M. R. Garey, D. S. Johnson, F. P. Preparata and R. E. Tarjan, *Triangulating a simple polygon*, Information Processing Letters, vol. 7, pp. 175-179, 1978.

[4] D. T. Lee, *On finding the convex hull of a simple polygon*, Int'l Journal on Computer and Information Science, vol. 12(2), pp. 87-98, 1983.

[5] T. Ohtsuki, Layout Design and Verification, North-Holland, Amsterdam, 1986.