A Cost-Optimal Algorithm for Guard Zone Problem

Ranjan Mehera[1] **and** ***Rajat K. Pal***[2]
1 Subex Limited, Adarsh Tech Park, Outer Ring Road, Devarabisannalli, Bangalore – 560037, India
2 Department of Computer Science and Engineering, University of Calcutta, 92, A. P. C. Road Kolkata 700 009, India
ranjan.mehera@gmail.com and rajatkp@vsnl.net

Abstract

Given a simple polygon P, its guard zone G (of width r) is a closed region consisting of straight line segments and circular arcs (of radius r) 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 r. In this paper we have designed a time-optimal sequential algorithm to solve the guard zone problem, and developed a cost-optimal parallel counterpart of the same problem for solving it in distributed environment.

Keywords: Guard zone problem, Minkowski sum, Convolution, Analytical geometry, Coordinate geometry, Computational geometry, Resizing of VLSI circuits.

1. Introduction

Guard zone problem is well defined in literature as an application of computational geometry. Often this problem is also known as safety zone problem [9]. Given a simple polygon P, its *guard zone* G (of width r) is a closed region consisting of straight line segments and circular arcs (of radius r) 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 r.

In this paper we design a time-optimal sequential algorithm for finding the guard zone of an arbitrarily shaped simple polygon, as shown in Figure 1. A *simple polygon* is defined as the polygon in which no two boundary edges cross each other. In addition, in this paper we also develop a cost-optimal parallel algorithm for the guard zone problem for solving the problem in a distributed computing environment.

In this context, it is worth mentioning that, given two polygons P and Q in R^2, their *Minkowski sum* [11] is defined as $P \oplus Q = (p+q \mid p \in P, q \in Q)$, where $p+q$ denotes the vector sum of the vectors p and q, i.e., if $p = (p_x, p_y)$ and $q = (q_x, q_y)$, then we have $p+q = (p_x+q_x, p_y+q_y)$. The *guard zone* (of width r) of a convex polygon P is surely obtained by taking the Minkowski sum of the polygon P and a circle C of radius r. But for a simple nonconvex polygon (i.e., a simple polygon with some concave vertices), the *guard zone is* a super set of the region A obtained by taking the Minkowski sum of the polygon P and the circle C. Here the area indicated by the Minkowski sum may be composed of the guard zone of the polygon P with some holes inside it such that the boundaries of the holes also satisfy the guard zone property [9].

The paper is organized as follows. In Section 2, we survey some important work done on the problem concerned. In Section 3, we formulate the guard zone problem. A time-optimal sequential algorithm for the guard zone problem is also developed and analyzed in this section. In Section 4, we formulate the problem for distributed computing environment, design a cost-optimal (parallel) algorithm [12] to implement the same in such an environment, and analyze. In Section 5, we discuss some important applications of guard zone problem, and conclude the paper, with a few remarks.

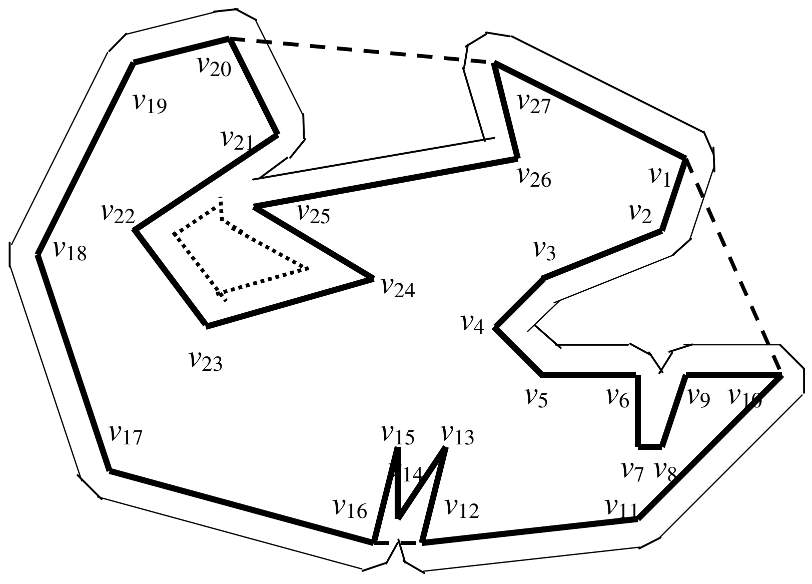

Figure 1: Guard zone of a simple polygon.

2. Literature Survey

If *P* is a simple polygon and *G* is its guard zone of width *r*, then the boundary of *G* is composed of straight-line segments and circular arcs of radius *r*, where each straight-line segment is parallel to an edge of the polygon at a distance *r* apart from that edge, and each circular arc of radius *r* is centered at a (convex) vertex of the polygon. The boundary of a guard zone describes a *simple* region in the sense that no two edges (straight-line segment(s) and/or circular arc(s)) on its boundary intersect in (or pass through) their interior. This has been explained in Figure 1. The problem originates in the context of resizing of VLSI layout design [14], as described later on (see Section 5).

The computational complexity of the Minkowski sum of two arbitrary simple polygons *P* and *Q* is $O(m^2n^2)$ [2], where *m* and *n* are the number of vertices of these two polygons, respectively. In particular, if one of the two polygons is convex, the complexity of the Minkowski sum reduces to $O(mn)$. In [4], a number of results are proposed on the Minkowski sum problem when one of the polygons is monotone.

An algorithm for finding the outer face of the Minkowski sum of two simple polygons is presented in [13]. It uses the concept of convolution, and the running time of the algorithm is $O((k+(m+n)\sqrt{l})\log^2(m+n))$, where *m* and *n* are the number of vertices of the two polygons, *k* and *l* represent the size of the convolution and the number of cycles in the convolution, respectively. In the worst case, *k* may be $O(mn)$. If one of the polygons is convex, the algorithm runs in $O(k\log^2(m+n))$ time, and there exists no algorithm that can compute the boundary defined by the Minkowski sum of an arbitrary simple polygon and a circle or a convex polygon in time, linear in the worst case of the problem.

In this context, a linear time algorithm is developed for finding the boundary of the minimum area guard zone of an arbitrarily shaped simple polygon in [9]. This algorithm uses the idea of Chazelle's linear time triangulation algorithm [3], and requires space complexity of $O(n)$ as well, where *n* is the number of vertices of the polygon. After having the triangulation step, this algorithm uses only dynamic linear and binary tree data structures.

Lastly, we wish to mention the work on an approximation of Minkowski sum boundary curves [8]. Given two planar curves of any shape, their convolution curve is defined as the set of all vector sums generated by all pairs of curve points that have the same curve normal direction. The Minkowski sum of two planar objects is closely related to the convolution curve of the two object boundary curves. That is, the convolution curve is a super set of the Minkowski sum boundary. By this method after eliminating all redundant parts in the convolution curve, one can generate the Minkowski sum boundary. But the convolution curve of two rational curves is not rational, in general. Therefore, in practice, it is required to approximate the convolution curves with polynomial / rational curves. This work also discusses on various other important issues in the boundary construction of the Minkowski sum.

3. Formulation of the Problem and the Algorithm

To solve the guard zone problem we can use Minkowski sum to draw a parallel line with respect to a given polygonal edge. Actually, Minkowski sum between a line and a point with same x and y coordinates gives the line parallel to the given one. But the question here arises is whether the parallel line will be inside or outside the polygon. Here we can extend the definition of Minkowski sum [11] as below.

If A and B are subsets of R^n, and $\lambda \in R$, then $A+B = \{x+y \mid x \in A, y \in B\}$, $A-B = \{x-y \mid x \in A, y \in B\}$, and λA $\{\lambda x \mid x \in A\}$.

Note that $A+A$ does not equal $2A$, and $A-A$ does not equal 'zero' in any sense.

We can also use convolution [1] to solve the guard zone problem. The convolution between the polygon and circle of radius r gives us the desired solution. But here we have to draw the circles in every possible points of the polygon and consequently the time complexity of the problem increases. Minkowski sum and convolution theories find their vast applications in mathematics, computational geometry, resizing of VLSI circuit components, and in many other problems.

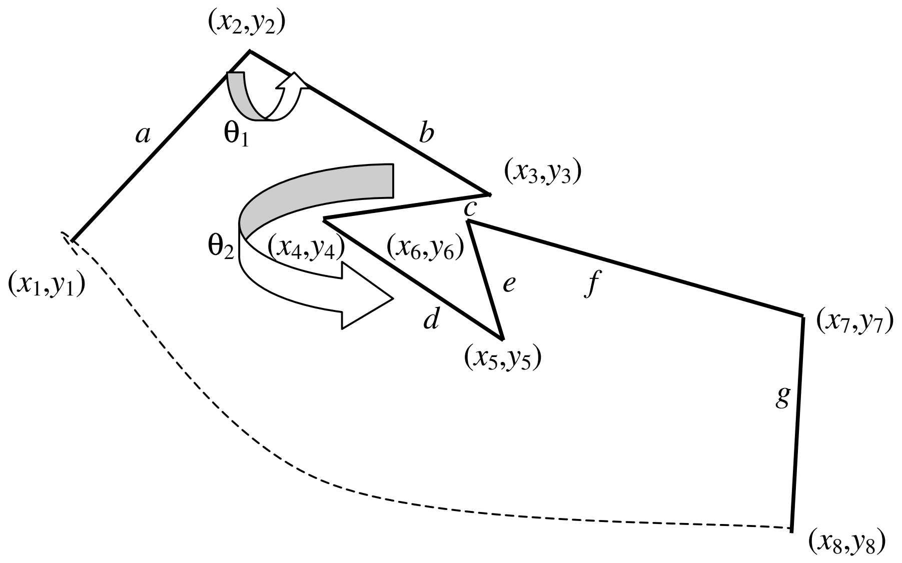

Figure 2: Part of a polygon with vertices (x_1,y_1) through (x_8,y_8), and edges a through g; the dotted line indicates the inner portion of the polygon.

A linear time algorithm is developed for finding the boundary of a minimum area safety zone for an arbitrarily shaped simple polygon, based on computational geometry [9]. The algorithm triangulates the concave regions outside the polygon, and uses dynamic linear and binary tree data structures to provide safety zones for those regions. In this paper, we develop another linear time algorithm, based on analytical and coordinate geometry only, for solving the guard zone problem.

A simple polygon may contain both convex and concave vertices on it. We define these vertices as follows: A vertex v of a polygon P is defined as *convex* (*concave*), if the angle between its associated edges inside the polygon, i.e., the internal angle at vertex v, is less than or equal to (greater than) 180°. Now consider Figure 2. Here angle θ_1, between edges a and b at vertex (x_2,y_2), is convex whereas angle θ_2, between edges c and d at vertex (x_4,y_4), is concave. In figure 4, we will categorize all the vertices as convex or concave for a portion of a given polygon.

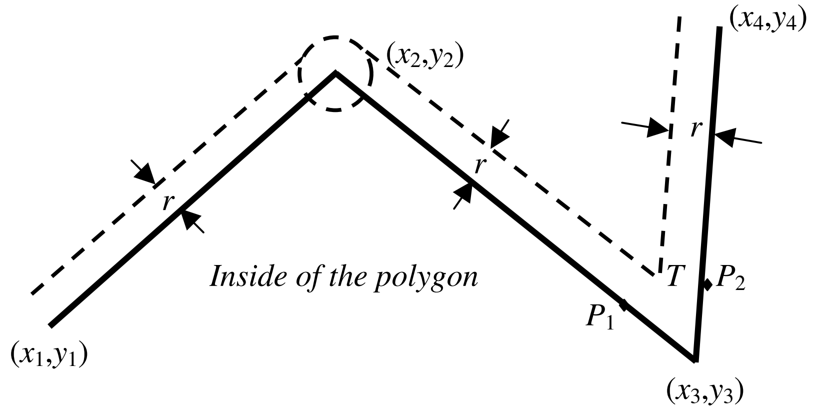

Figure 3: Lines of a guard zone (dotted) corresponding to polygonal edges (firm), a convex vertex at (x_2,y_2), and a concave vertex at (x_3,y_3).

A guard zone G of a simple polygon P can easily be obtained with the help of two pens, say Pen_1 and Pen_2, whose tips are always at distance r apart, as stated below. Let the vertices of polygon P are labeled as v_1, v_2, …, v_n, moving along its boundary in clockwise direction. Let the tip of Pen_1 moves along the polygonal boundary from vertex v_1 through v_n (to v_1), and the tip of Pen_2 draws the boundary of the guard zone, staying orthogonal to the direction of motion of Pen_1.While drawing the boundary of the guard zone, two cases arise.

(i) If Pen_1 finds a convex vertex (as at (x_2,y_2) in Figure 3), then the direction of Pen_2 is changed by drawing a circular arc of radius r, with center at that convex vertex.

(ii) If Pen_1 residing at point P_1 (on the polygon) finds another point P_2 on the boundary of the polygon such that $P_1T+TP_2 = 2r$ (where Pen_2 resides at point T), and P_1T and TP_2 are completely outside the polygon (see Figure 3), then Pen_1 moves to P_2 and Pen_2 remains at its present position T. Then Pen_1 starts moving toward the next (higher numbered) vertex attached to that edge; the movement of Pen_2 will be guided by that of Pen_1, as described earlier. This is happened only when a concave vertex (as at (x_3,y_3) in Figure 3) of the polygon is found, where P_1P_2, i.e., the distance between points P_1 and P_2, is less than $2r$.

So, Pen_1 moves through the boundary of the polygon, shown by the firm line in Figure 3, and Pen_2 moves through the dotted line, always at distance r apart from (and outside) the polygon, which is actually the guard zone.

Now, let us consider a simple polygon P with n vertices and n edges. A polygon P is given implies that the coordinates of the successive vertices of the polygon are given; where no two polygonal edges cross each other rather two consecutive polygonal edges intersect only at a polygonal vertex. We can assume that a portion of this polygon is as shown in Figure 2. To know whether an angle θ, inside the polygon, is either convex or concave at vertex v, we do the following constant time computation of determining the value of θ between two consecutive polygonal edges intersected at vertex v.

Let the slope of edge a be m_a and that of edge b be m_b. Therefore, $m_a = (y_2 - y_1) / (x_2 - x_1)$ and $m_b = (y_3 - y_2) / (x_3 - x_2)$. Hence, $\tan\theta_1 = (m_a - m_b) / (1 + m_a m_b)$ gives $\theta_1 = \tan^{-1}\{(m_a - m_b) / (1 + m_a m_b)\}$. In this way, all the n internal angles of P are identified as convex or concave in $O(n)$ time.

At this point, we can conclude that the guard zone is computed only with the help of n straight-line segments and n circular arcs, if all the n internal angles of the polygon are convex. But for a given simple polygon, we may have concave internal angles as well, in P. Problems may arise in computing guard zone for those portions of polygon P with concave internal angles. In this context, we introduce the concept of notch as defined below.

A *notch* is a polygonal region outside polygon P that is formed with a chain of edges of P initiating and terminating at two vertices of a false hull edge [9]. A *false hull edge* is a hull edge introduced in obtaining a convex hull, shown in Figure 1. Here we have three false hull edges drawn by dotted lines, between vertices v_1 and v_{10}, v_{12} and v_{16}, and v_{20} and v_{27}. A *convex hull* is a convex polygon (having no concave internal angle) of minimum area with all the points residing on the boundary or inside the polygon for a given set of arbitrary points on a plane.

Clearly, if P is a given simple polygon and CH(P) denotes the convex hull of polygon P, then the area CH(P) – P consists of a number of disjoint notches outside polygon P. According to this definition, a notch is formed outside the polygon in Figure 4, below the dotted line v_2v_8, as this edge is a false hull edge.

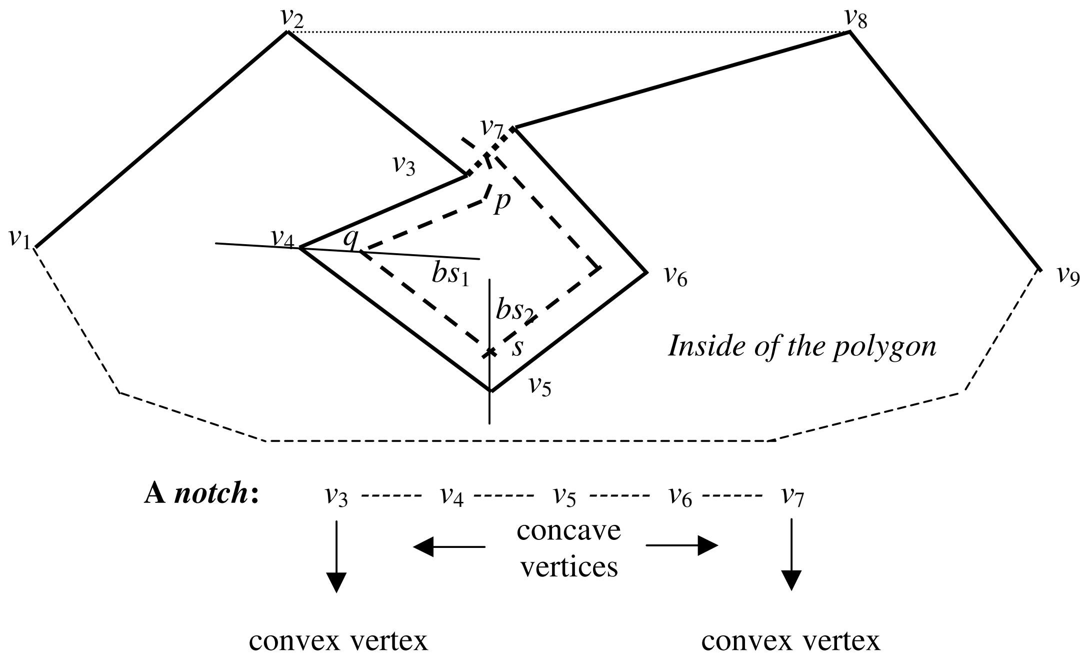

Figure 4: A notch is formed between vertices v_3 and v_7, and a guard zone is obtained for this notch.

Now for the sake of convenience, in order to develop our algorithm, we redefine notch as the following. A *notch* is a polygonal region outside the polygon, which is initiating and terminating between two consecutive convex vertices of the polygon that are not adjacent. That means a notch starts with a convex vertex, goes through one or more concave vertices, and terminates with a convex vertex, along the edges (and vertices) of the polygon (in one particular direction). According to this definition, in Figure 4, a notch is formed between polygonal vertices v_3 and v_7, as v_4, v_5, and v_6 are concave vertices whereas v_3 and v_7 are convex.

Now we develop a time-optimal sequential algorithm for computing a guard zone G of a simple polygon P as described below. Usually, a guard zone contains line segments that are parallel to the edges of P. A circular arc shaped portion of a guard zone is only obtained at a convex vertex of the polygon. So, in developing the algorithm, we consider edges one after another and do the following.

Consider edge v_2v_3 in Figure 4. Here both the vertices v_2 and v_3 are convex. So a line segment parallel to v_2v_3 and of length same as v_2v_3, for G is computed at a distance r apart the polygonal edge with a circular arc of radius r centered at vertex v_2. Clearly, the line segment parallel to v_2v_3 is a tangent to the circular arc centered at v_2.

Now consider edge v_3v_4. Here v_3 is a convex vertex whereas v_4 is concave. So, the length of the line segment of G parallel to v_3v_4 is less than that of v_3v_4. Surely the line segment of G touches (as a tangent) the circular arc centered at v_3 at point p (in Figure 4), but on the other end (near v_4, which is a concave vertex) the line segment of G is prolonged up to point q, as stated below. Point q is the intersection point of the bisection bs_1, of the angle between polygonal edges v_3v_4 and v_4v_5 (i.e., the angle at vertex v_4) and the line segment of G passes through p and parallel to v_3v_4. The cause of doing this has been clearly explained in Figure 3, for a concave vertex at (x_3,y_3), where $P_1T = TP_2 = r$.

Now consider polygonal edge v_4v_5, where both polygonal vertices v_4 and v_5 are concave. So it is not required to draw any circular arc here for this portion of G. The line segment of G is obtained by computing a line going through q up to s, where s is the intersection of the line segment of G parallel to v_4v_5 and the bisection bs_2 of the internal angle at vertex v_5. Surely, the length of qs is less than that of v_4v_5.

It may so happen that the length of v_4v_5 is sufficiently small and the internal angles at v_4 and v_5 are large enough such that the distance between the intersection point f of bs_1 and bs_2 to v_4v_5 is less than the distance between the intersection point g of the two line segments of G parallel to v_3v_4 and v_5v_6 to v_4v_5 (see Figure 5). In this case, there is no line segment of G parallel to polygonal edge v_4v_5. In general, instead of a single edge like v_4v_5, several edges or even several notches of a simple polygon may be there for which no guard zone is distinguishingly computed.

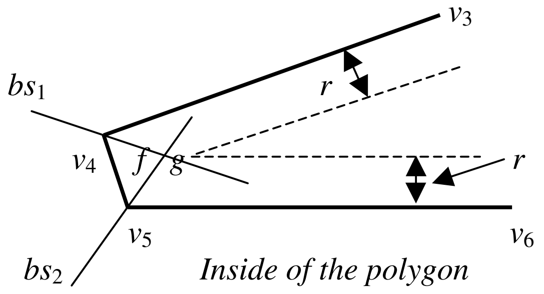

Figure 5: A portion of the guard zone without any line segment parallel to the polygonal edge v_4v_5.

Needless to mention that all these computations could be performed with the help of simple analytical and coordinate geometry. Besides, all these computations required to obtain a portion of the guard zone for a polygonal edge or for a polygonal vertex could be performed in constant time. We establish this result as a theorem, at the end of this section. Before that we briefly state the steps of the algorithm *Guard_Zone*, for computing a guard zone of a simple polygon, as given below.

Algorithm *Guard_Zone.*

Input: A simple polygon P.
Output: A guard zone of polygon P.

```
Step 1: Clockwise label the vertices $v_1, v_2, \ldots, v_n$, of polygon $P$.
Step 2: For $i = 1$ to $n-1$ do
    Step 2.1: If the internal angle at $v_i$ is convex
        Then  Step 2.1.1: Draw a circular arc (outside the polygon) of radius $r$ centered at $v_i$.
              Step 2.1.2: Find the internal angle at $v_{i+1}$, and consider polygonal edge $(v_i,v_{i+1})$.
              Step 2.1.3: If the internal angle at $v_{i+1}$ is convex
                  Then  Step 2.1.3.1: Draw a circular arc (outside the polygon) of radius $r$ centered at $v_{i+1}$.
                        Step 2.1.3.2: Draw a line parallel to $(v_i,v_{i+1})$ at a distance $r$ apart from the polygonal edge (outside the polygon) that is a simple common tangent to both the arcs drawn at $v_i$ and $v_{i+1}$.
                  Else  Step 2.1.3.3: Bisect the internal angle at $v_{i+1}$, denote the bisection $bs_{i+1}$.
                        Step 2.1.3.4: Draw a line parallel to $(v_i,v_{i+1})$ at a distance $r$ apart from the polygonal edge (outside the polygon) that is a tangent to the arc drawn at $v_i$ and intersects $bs_{i+1}$ at a point, say $p_{i+1}$.
              Step 2.1.4: Assign $i \leftarrow i+1$.
              Step 2.1.5: If $v_i = v_n$, then $v_{i+1} = v_1$.
        Else    Step 2.1.6: Bisect the internal angle at $v_i$, denote the bisection $bs_i$.
                Step 2.1.7: Find the internal angle at vertex $v_{i+1}$, and consider polygonal
                edge $(v_i,v_{i+1})$.
                Step 2.1.8: If the internal angle at $v_{i+1}$ is convex
                    Then Step 2.1.8.1: Draw a circular arc (outside the polygon) of
                            radius $r$ centered at $v_{i+1}$.
                         Step 2.1.8.2: Draw a line parallel to $(v_i,v_{i+1})$ at a
                            distance $r$ apart from the polygonal edge (outside the
                            polygon) that intersects $bs_i$ at a point, say $p_i$ and is a
                            tangent to the arc drawn at $v_{i+1}$.
                    Else Step 2.1.8.3: Bisect the internal angle at $v_{i+1}$, denote the
                            bisection $bs_{i+1}$.
                         Step 2.1.8.4: Draw a line parallel to $(v_i,v_{i+1})$ at a
                            distance $r$ apart from the polygonal edge (outside the
                            polygon) that intersects $bs_i$ at a point, say $p_i$ and also
                            intersects $bs_{i+1}$ at a point, say $p_{i+1}$.
                Step 2.1.9: Assign $v_i \leftarrow v_{i+1}$.
                Step 2.1.10: If $v_i = v_n$, then $v_{i+1} = v_1$.
    End for.
Step 3: If two line segments or a line segment and a circular arc or two circular arcs of the guard
zone intersect, then exclude the portions of the line segment(s) and/or the circular arc(s) that are at
a distance less than $r$ apart from a polygonal edge or a polygonal vertex (outside the polygon). ♦
```

Computational Complexity of Algorithm *Guard_Zone*

It is easy to observe that the guard zone of an n-vertex convex polygon is a convex region with n straight-line segments and n circular arcs only. The straight-line segments of the guard zone are parallel to the edges of the polygon at a distance r outside the polygon, and two consecutive line segments of the guard zone are joined by a circular arc of radius r centered at the corresponding polygonal vertex. As a result, the time required for computing the guard zone of a convex polygon is $O(n)$.

The situation is little bit complicated, if notches rather concave vertices belong to a simple polygon. Anyway, for the presence of a concave polygonal vertex, we bisect the polygonal angle (instead of introducing a circular arc outside the polygon), and draw a line parallel to the polygonal edge under consideration. For a polygonal edge or an angle of the polygon, either convex or concave, all these computations take constant time. Therefore, for a polygon of n vertices (and n edges), the overall worst-case time required in computing a guard zone of a simple polygon is $O(n)$.

The remaining part of our algorithm is to exclude the portions of guard zone G that are at a distance less than r apart from some polygonal edge or vertex other than the polygonal edge or vertex for which the segments of G were computed (outside the polygon). This could be realized in time $O(n)$, by the way stated below. During computation of G for vertex v_i and edge e_i, we identify the coordinates, say (x,y), of the region as flag one, inside the guard zone and outside the polygon. This could be performed in constant time. (We may use some colour ***C*** of some specific intensity ***I*** for this purpose.) We update this flag for (x,y) (that has already been included for v_i and/or e_i), if these coordinates are also included in a newly computed region of G for some other vertex v_j and/or edge e_j.

Clearly, from each of the points (x,y) there are polygonal edge(s) and/or vertex (vertices), one of which is at a distance less than r, within the region of G (outside the polygon). Subsequently, the flag for points (x,y) is incremented to two. (Here the intensity of colour ***C*** changes from ***I*** to some other intensity ***I′***.) This signifies that the nearby polygonal edges e_i and e_j and/or polygonal vertices v_i and v_j must not have a continuous segment of guard zone, as the segments are computed at a distance r apart from each of the points of P. So, after obtaining the guard zone for v_j and e_j, we remove the portions of G that are intersected each other and forming a region of points (x,y) with flag two (excluding their intersection points). The flag is reassigned to one again, up to edge

e_j, for the region of G (outside the polygon). (Or, again colour ***C*** of intensity ***I*** may be assigned to the region, up to edge e_j, inside G and outside P.)

This is also worthwhile to mention that the lower bound of computing a guard zone of a simple polygon of n vertices is $\Omega(n)$, as we have to consider each of the n vertices (and n edges) once. This signifies that the algorithm we have developed is time-optimal. This result is established in the following theorem.

Theorem 1: *Algorithm Guard_Zone computes a guard zone of a simple polygon of n vertices in $\Theta(n)$ time. This algorithm is time-optimal, as the lower bound of the problem of computing guard zone of a simple polygon of n vertices is $\Omega(n)$ and the worst-case computational complexity of algorithm Guard_Zone is $O(n)$.*

4. A Cost Optimal Algorithm for Distributed Environment

In this section, we develop a constant time, cost-optimal algorithm *Parallel_Guard_Zone* for computing a guard zone of a simple polygon with n vertices in a distributed environment. This algorithm is based on the sequential algorithm; we have developed in Section 3. Here we use $O(n)$ processors, instead of a single processor as in the case of its sequential counterpart, on a CRCW PRAM [10].

At the very beginning of our algorithm, we keep the n vertices, $v_1, v_2, \ldots, v_n$, as input of a given polygon P in n (shared) memory locations [6]. A processor P_i, $1 \le i \le n$, concurrently reads the coordinates of vertices v_{i-1}, v_i, v_{i+1}, and v_{i+2}, and computes the equations of edges (v_{i-1},v_i), (v_i,v_{i+1}), and (v_{i+1},v_{i+2}) along with their lengths. We may rename the edges as e_{i-1}, e_i, and e_{i+1}, respectively. Then processor P_i computes the internal angles flanking edge e_i, at vertices v_i and v_{i+1}. It is easy to understand that for $i = 1$, $i-1 = n$, and for $i = n$, $i+1 = 1$, etc. as we have indexed the vertices and kept into memory.

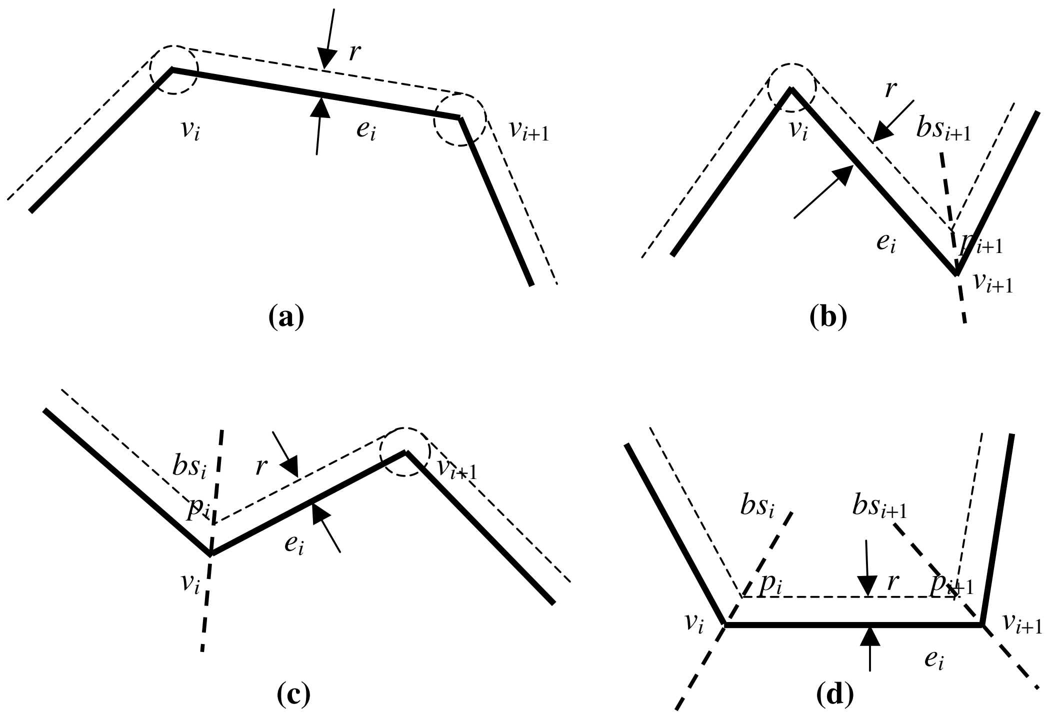

Figure 6: Computation of guard zone for a polygonal vertex v_i and a polygonal edge $e_i = (v_i,v_{i+1})$. (**a**) Edge e_i with both its end vertices v_i and v_{i+1} convex. (**b**) Edge e_i with its end vertex v_i convex and v_{i+1} concave. (**c**) Edge e_i with its end vertex v_i concave and v_{i+1} convex. (**d**) Edge e_i with both its end vertices v_i and v_{i+1} concave.

In our algorithm, primarily processor P_i is responsible for considering one of the following four possibilities, and draws portions of G for vertex v_i and edge e_i, if they exist. If line segment(s) and/or circular arc(s) of G intersect, the processor excludes these portions of G drawn by itself that are at a distance less than r apart from a polygonal edge or vertex (outside the polygon).

Consider a polygonal edge e_i with both its end vertices v_i and v_{i+1} convex (see Figure 6(a)). In developing a parallel algorithm, P_i draws circular arcs (outside the polygon) of radius r centered at v_i and v_{i+1}, and draws a line parallel to e_i at a distance r apart from the polygonal edge (outside the polygon) that is a simple common tangent to the arcs drawn at v_i and v_{i+1}, and length same as (v_i,v_{i+1}).

Consider a polygonal edge e_i with its end vertex v_i convex whereas v_{i+1} concave (see Figure 6(b)). In developing a parallel algorithm, P_i draws a circular arc (outside the polygon) of radius r centered at v_i, and bisects the internal angle at v_{i+1} and denotes the bisection bs_{i+1}. Then P_i draws a line parallel to e_i at a distance r apart from the polygonal edge (outside the polygon) that is a tangent to the arc drawn at v_i and intersects bs_{i+1} at a point, say p_{i+1}.

Consider a polygonal edge e_i with its end vertex v_i concave whereas v_{i+1} convex (see Figure 6(c)). In developing a parallel algorithm, P_i bisects the internal angle at v_i and denotes the bisection bs_i, and draws a circular arc (outside the polygon) of radius r centered at v_{i+1}. Then P_i draws a line parallel to e_i at a distance r apart from the polygonal edge (outside the polygon) that intersects bs_i at a point, say p_i, and is a tangent to the arc drawn at v_{i+1}.

Now consider a polygonal edge e_i with both its end vertices v_i and v_{i+1} concave (see Figure 6(d)). In developing a parallel algorithm, P_i bisects the internal angles at v_i and v_{i+1}, and denotes the bisections bs_i and bs_{i+1}, respectively. Then P_i draws a line parallel to e_i at a distance r apart from the polygonal edge (outside the polygon) that intersects bs_i at a point, say p_i and bs_{i+1} at a point, say p_{i+1}, if the distance from the intersection point of bs_i and bs_{i+1} to e_i is more than r.

Now it is easy to develop a parallel algorithm that computes a guard zone of a simple polygon in constant time, and that can be executed in a distributed computing environment. The remaining part of this algorithm is to exclude the portions of line segment(s) and/or circular arc(s) of G that are at a distance less than r apart from a polygonal edge or vertex (if such line segment(s) and/or circular arc(s) of G intersect, outside the polygon) as described in the sequential counterpart.

Computational Complexity of Algorithm *Parallel_Guard_Zone*

Now we analyze the time complexity of the algorithm *Parallel_Guard_Zone*, for computing a guard zone G of a simple polygon P in parallel. In order to do that we consider a *Concurrent Read Concurrent Write (CRCW) Parallel Random Access Machine (PRAM)* in a *Single Instruction stream Multiple Data stream (SIMD)* parallel computing environment [7,12], where a control unit issues an instruction to be executed simultaneously by all processors on their respective data.

In our algorithm we primarily perform the following accession of memory and computation required. We keep all the coordinates of n vertices of polygon P in $O(n)$ shared memory locations of the PRAM. A *Concurrent Read (CR)* instruction is executed by $O(n)$ processors, where two or more processors can read from the same memory location at the same time. This instruction is executed constant times as the coordinate of a vertex is read by four processors. As for example, the coordinate of vertex v_{i+2} is accessed by processors P_i through P_{i+3}.

Thus when instruction CR is executed, $O(n)$ processors simultaneously read the contents of $O(n)$ memory locations such that processor P_i reads the coordinates of vertices v_{i-1}, v_i, v_{i+1}, and v_{i+2}. This could also be executed using an *Exclusive Read (ER)* instruction constant times by $O(n)$ processors, where processors gain access to memory locations for the purpose of reading in a one-to-one fashion. Thus when this instruction is executed, $O(n)$ processors simultaneously read the contents of $O(n)$ distinct memory locations such that each of the $O(n)$ processors involved reads from exactly one memory location and each of the $O(n)$ memory locations involved is read by exactly one processor. Note that CR includes ER as a special case.

Then processor P_i computes the edges (v_{i-1},v_i), (v_i,v_{i+1}), and (v_{i+1},v_{i+2}) (i.e., e_{i-1}, e_i, and e_{i+1}, respectively), and computes the internal angles at vertices v_i and v_{i+1}. So, processor P_i finds one of the four possible situations, as explained in Figure 6. Accordingly, the processor draws circular arc(s), if the angle(s) is (are) convex and/or bisects the angle(s) if it is (they are) concave, as this has been described explicitly in the algorithm above. In addition, the processor draws a line parallel to e_i at a distance r apart from the polygonal edge (outside the polygon), as a portion of G.

Hence, a guard zone for polygon P is computed in constant time, which may contain several redundant portions of (overlapping) line segments due to presence of notch(es). To remove these undesired portions of G, we execute a *Concurrent Write (CW)* instruction in the form of *COMBINED SUM* CW by $O(n)$ processors, where two or more processors can write into the same memory location at the same time. Moreover, when this instruction is executed, in our algorithm, $O(n)$ processors simultaneously write into one memory location, whereas the memory location involved can be written into by all the processors.

Specifically, to exclude the portions of G that are at a distance less than r apart from a polygonal edge or vertex (outside the polygon), processor P_i flags the region (for which the processor is responsible), inside G and outside P, as one. Then a CW instruction, as stated above, is executed by $O(n)$ processors, over the regions inside G and outside P, that are computed simultaneously by all the processors, and written into the (n+1)th shared memory location of the PRAM.

So, some region of the guard zone (i.e., inside G and outside P) may now contain flag two or more, if they overlap. (If we use some colour $\boldsymbol{C}$ with intensity $\boldsymbol{I}$ to differentiate a region of G from the polygon, then after summing the n distinct portions of the computed guard zone (that is written into the (n+1)th shared memory location of the PRAM), the overlapped region will have colour $\boldsymbol{C}$ with intensity other than $\boldsymbol{I}$.) Now to exclude the boundaries of these regions with flag other than one (or colour $\boldsymbol{C}$ with intensity other than $\boldsymbol{I}$), we involve all the processors and execute a CR instruction, as stated below.

All the $O(n)$ processors, now concurrently read the content of the (n+1)th shared memory location of the PRAM, which is nothing but the guard zone as a whole that may contain some redundant portions in it. So, each of the processors reads the information (i.e., coordinates) about the specified region inside G and outside P, that are flagged either one or more. Anyway, processor P_i excludes the remaining portion of G, other than the ith portion, for which it is responsible. Processor P_i now checks whether the said region inside G (and outside P), computed by it, contains any coordinate with flag other than one. If it does not find any such region, it takes no action. Otherwise, P_i excludes the portion of G (computed by it), whose coordinates are with flag other than one. This can also be computed in constant time.

Ultimately, a CW instruction is executed to write the modified portions of G computed by all the processors concurrently to the (n+1)th shared memory location of the PRAM. Hence the desired solution of guard zone G computed for a simple polygon P is available at the (n+1)th location of the shared memory.

It is now straightforward to verify that the running time of this algorithm is $O(1)$. As $O(n)$ processors are involved in computing a guard zone, the cost of algorithm *Parallel_Guard_Zone* is $O(n)$, which is same as the worst-case computational complexity of the sequential version of this algorithm that is time-optimal (see Theorem 1). Hence we conclude the following theorem.

Theorem 2: *Algorithm Parallel_Guard_Zone computes a guard zone of a simple polygon of n vertices in constant time. This algorithm is cost-optimal, as the cost of the algorithm is O(n), which is same as the worst-case running time of a sequential algorithm for solving the problem of computing guard zone of a simple polygon of n vertices.*

5. Conclusion

In conclusion, first of all, we point out a few important applications of computing a guard zone of a simple polygon. Then, we conclude the paper with a few remarks.

Applications of Computing a Guard Zone

In VLSI layout design, the circuit components (or the functional units/modules or groups/blocks of different subcircuits) are not supposed to be placed much closer to each other in order to avoid electrical (parasitic) effects among them [14]. The (group of) circuit components on a chip-floor may be viewed as a set of polygonal regions on a two-dimensional plane. Each (group of) circuit component(s) C_i is associated with a parameter δ_i such that a minimum clearance zone of width δ_i is to be maintained around C_i. The regions representing the (groups of) circuit components are in general isothetic polygons, but may not always be limited to convex ones. The location of the guard zone (of specified width) for a simple polygon is a very important problem for resizing the (group of) circuit components.

Given a set of isothetic nonoverlapping polygonal regions and a common resizing parameter δ, the objective is to compute another set of closed regions resizing each polygon by an amount of δ. If two or more polygons are closed enough so that their guard zones overlap, indicating the violation of the minimum separation constraint among them, and the polygons have to move relatively to overcome this violation.

It may so happen that sometimes a small polygon that has been placed outside a large polygon with a sufficiently big notch in it. In this case the small polygon could be accommodated inside the notch of the large polygonal boundary. Often this sort of placement of a small polygon inside a notch of some other polygon may provide a compact design and subsequently, space is also saved. Thus resizing is an important problem in VLSI layout design, while accommodating the (groups of) circuit components on a chip floor, and this problem motivates us to compute a guard zone of a simple polygon.

The guard zone problem finds another important application in the automatic monitoring of metal cutting tools. Here a metal sheet is given and the problem is to cut a polygonal region of some specified shape from that sheet. The cutter is like a *ballpoint pen* whose tip is a small ball of radius δ, and it is monitored by a software program. If the holes inside the notch also need to be cut, our algorithm can easily be tailored to satisfy that requirement too.

The Minkowski sum is an essential tool for computing the free configuration space of translating a polygonal robot [1]. It also finds application in the polygon containment problem and in computing the buffer zone in geographical information systems [5], to name only a few.

Remarks

In this paper we state the problem of locating guard zone of a simple polygon and present a time-optimal sequential algorithm for computing a boundary of guard zone, which uses simple analytical and coordinate geometric concepts. Our algorithm can easily be modified to compute the regions of width r outside the polygon (as guard zone), and also inside the polygon, which may find several applications in practice.

In addition, in this paper we also develop a cost-optimal parallel algorithm for solving the guard zone problem that can easily be implemented in a distributed computing environment. Needless to mention that there are several alternative way outs in solving the guard zone problem but so far as complexity is concerned we can't do better (as our sequential and parallel versions of the algorithm are time-optimal and cost-optimal, respectively). Our work can be extended for computing guard zone of a three dimensional solid object.

In computing the running time of our parallel algorithm, we only consider the *computation complexity*, which is the time spent by a processor on local computations, and do not consider the *communication complexity*, that is the time spent in communicating data among processors [7,10]. We use the standard sequential model (in parallel) for estimating the computation complexity. However, an estimate of the communication complexity mainly depends, in general, on the data layout and the architecture of the machine under consideration. Whatsoever, in computing the computational complexity of our parallel algorithm, we assume that the communication complexity is no way worse than the computation complexity of the algorithm. Even, we may assume that the communication complexity is constant, and the complexity of the (parallel) algorithm fully depends upon the computation complexity only.

References

1. C. Bajaj and M.-S. Kim, Generation of Configuration Space Obstacles: The Case of a Moving Algebraic Curves, *Algorithmica*, vol. 4, No. 2, pp. 157-172, 1989.
2. M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, 1997.
3. B. Chazelle, *Triangulating a Simple Polygon in Linear Time*, Discrete Computational Geometry, vol. 6, pp. 485-524, 1991.
4. A. Hernandez-Barrera, Computing the Minkowski Sum of Monotone Polygons, *IEICE Trans. on Information Systems*, vol. E80-D, No. 2, pp. 218-222, 1996.
5. I. Heywood, S. Cornelius and S. Carver, *An Introduction to Geographical Information Systems*, Addison Wesley Longman, New York, 1998.
6. K. Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programmability*. McGraw Hill, New York, 1993.
7. K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing*. McGraw-Hill, New York, 1984.
8. I.-K. Lee, M.-S. Kimand, and G. Elber, Polynomial / Rational Approximation of Minkowski Sum Boundary Curves (Article No.: IP970464), *Graphical Models and Image Processing*, vol. 60, No. 2, pp. 136-165, 1998.
9. S. C. Nandy, B. B. Bhattacharya, and A. Hernandez-Barrera, Safety Zone Problem, *Journal of Algorithms*, vol. 37, pp. 538-569, 2000.
10. D. A. Patterson and J. L. Hennesy, *Computer Architecture: A Quantitative Approach* (Second Edition). Morgan Kaufman, 1996.
11. H. Pottmann and J. Wallner, *Computational Line Geometry*. Springer-Verlag, Berlin, 1997.
12. M. J. Quinn, *Parallel Computing: Theory and Practice* (Second Edition). McGraw-Hill, New York, 1994.
13. G. D. Ramkumar, An Algorithm to Compute the Minkowski Sum Outer Face of Two Simple Polygons, *Proc. of the 12th Annual ACM Symposium on Computational Geometry*, pp. 234-241, Association for Computing Machinery, New York, 1996.
14. N. A. Sherwani, *Algorithms for VLSI Physical Design Automation*. Kluwer Academic, Boston, 1993.