Study of Guard Zone Problem

Rajarshi Bhattacharya

97/CSM/211012

M.Tech. Computer Science and Engineering, Semester 3

University of Calcutta

Abstract

In this paper I am presenting two linear time algorithm for finding the minimum area safety zone or guard zone of an arbitrarily shaped simple polygon. It can be also shown that these methods can easily be modified to compute the Minkowski sum of a simple polygon and a convex polygon in O(MN) time, where M and N are the number of vertices of both the polygons.

Table of Contents

1. Introduction

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. A simple polygon is defined as the polygon in which no two boundary edges cross each other.

In this context, it is worth mentioning that, given two polygons P and Q in R^2 , their Minkowski sum [10] 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 = (px , py) and q = (qx , qy), then I 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].

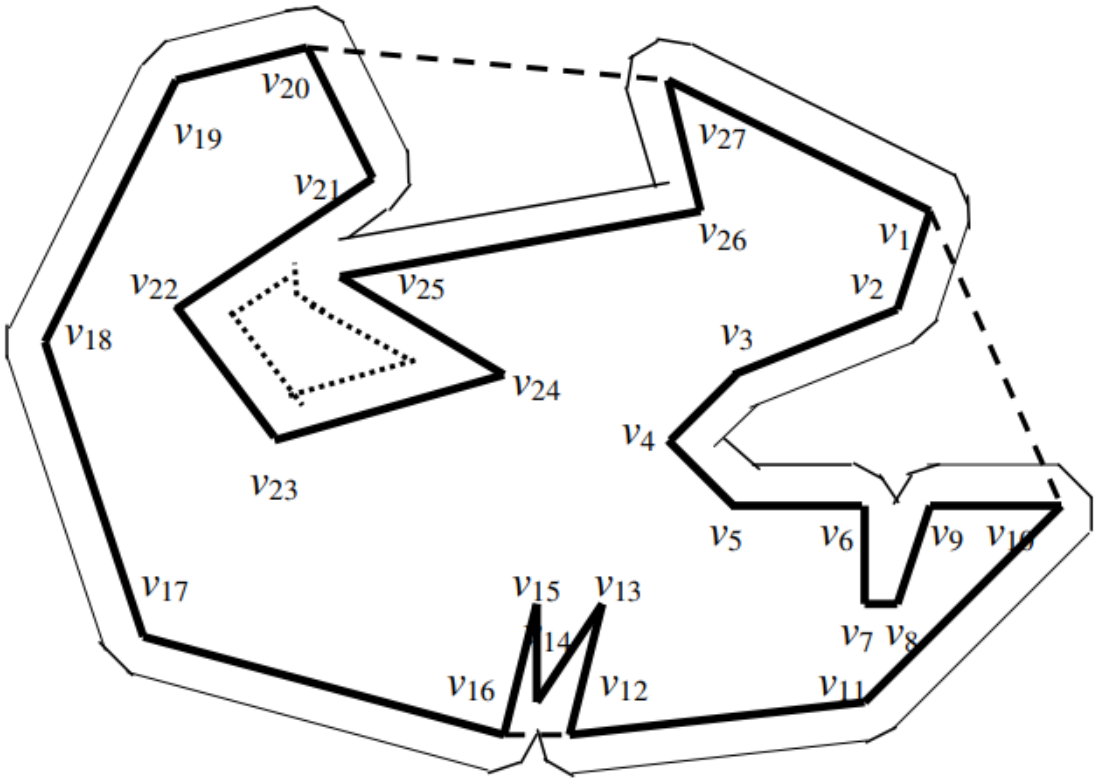

Figure 1: Guard zone of a simple polygon

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 computational complexity of the Minkowski sum of two arbitrary simple polygons P and Q is $O(m^2n^2)$ [5], 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 [7], 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 [4]. 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 [6], 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, I 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. Algorithms

Here, I am going to discuss about two algorithms for computing the Guard Zone of a simple polygon.

Algorithm 1: ALGO_GUARD_ZONE [1].
Input: A simple polygon P.
Output: A guard zone of polygon P.
Steps:
Begin
Step 1: Clockwise label the vertices v1, v2, …, vn, of polygon P.
Step 2: For i = 1 to n−1 do
Step 2.1: If the internal angle at vi is convex Then
Step 2.1.1: Draw a circular arc (outside the polygon) of radius r centered at vi .
Step 2.1.2: Find the internal angle at vi+1, and consider polygonal edge (vi ,vi+1).
Step 2.1.3: If the internal angle at vi+1 is convex Then
Step 2.1.3.1: Draw a circular arc (outside the polygon) of radius r centered at vi+1.
Step 2.1.3.2: Draw a line parallel to (vi ,vi+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 vi and vi+1. Else
Step 2.1.3.3: Bisect the internal angle at vi+1, denote the bisection bsi+1.
Step 2.1.3.4: Draw a line parallel to (vi ,vi+1) at a distance r apart from the polygonal edge (outside the polygon) that is a tangent to the arc drawn at vi and intersects bsi+1 at a point, say pi+1.
Step 2.1.4: Assign i ← i+1.
Step 2.1.5: If vi = vn, then vi+1 = v1. Else
Step 2.1.6: Bisect the internal angle at vi , denote the bisection bsi .
Step 2.1.7: Find the internal angle at vertex vi+1, and consider polygonal edge (vi,vi+1).
Step 2.1.8: If the internal angle at vi+1 is convex Then
Step 2.1.8.1: Draw a circular arc (outside the polygon) of radius r centered at vi+1.
Step 2.1.8.2: Draw a line parallel to (vi ,vi+1) at a distance r apart from the polygonal edge (outside the polygon) that intersects bsi at a point, say pi and is a tangent to the arc drawn at vi+1. Else
Step 2.1.8.3: Bisect the internal angle at vi+1, denote the bisection bsi+1.
Step 2.1.8.4: Draw a line parallel to (vi ,vi+1) at a distance r apart from the

polygonal edge (outside the polygon) that intersects bsi at a point, say pi and also intersects bsi+1 at a point, say pi+1.

Step 2.1.9: Assign vi ← vi+1.

Step 2.1.10: If vi = vn, then vi+1 = v1.

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).

Stop

Algorithm 2: ALGO_SAFE_BOUNDARY [2].

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

poly_chain	An array of vertices and edges of the notch stored in clockwise order.
T	The triangulation tree of the notch. With each edge of this tree a V-LIST is maintained, as described in the previous subsection.
L	A list containing SB for the elements (vertices and polygonal edges) inside a notch. Each element of L has a pointer to its neighboring SB in clockwise order. This will be recursively constructed while traversing the tree. At the time of processing a triangle, the SB of its edge(s) and vertices inside the triangle is created in L if it is not already present. During a merge pass, if an intersection between two SBs (say SB* and SB**) is detected, these SBs are updated by removing the portions to the left of the point of the intersection. An appropriate pointer is also established among SB* and SB**. Finally, after processing the root node, the list L gives the safe boundary of the notch.

Begin
Input: A simple polygon P.
Output: A guard zone of polygon P.
Steps:
Step 1. Compute the convex hull of the given polygon.
Step 2. For each notch do:

Step 2.1. Triangulate the notch to prepare the triangulation tree T.
Step 2.2. (*Traverse the tree in post-order manner *) .
//Call Post_Order(root)
(* This procedure recursively traverses the tree T. At each node after the traversal of both its children, it calls the procedure Process_Triangle which processes the triangles.*)

Step 3. Draw the safe boundaries for the solid hull edges and for the hull ertices of the convex hull.
Step 4. Connect the SB of each solid hull edge to the SB of its attached hull vertices.
Step 5. For each notch do Merge each end of the L list of that notch to the SB of the hull vertex attached to its false hull edge.
Step 6. Traverse the L list from any element to output the safe boundary of the polygon.
Stop

4. Complexity of the Algorithm

Complexity of Algorithm 1 [1]:

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, I 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 vi and edge ei , I 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. (I may use some colour C of some specific intensity I for this purpose.) I update this flag for (x,y) (that has already been included for vi and/or ei), if these coordinates are also included in a newly computed region of G for some other vertex vj and/or edge ej .

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 ei and ej and/or polygonal vertices vi and vj 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 vj and ej , I 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 ej , for the region of G (outside the polygon). (Or, again colour C of intensity I may be assigned to the region, up to edge ej , inside G and outside P.)

Complexity of Algorithm 2 [2]:

The convex hull of an N vertex simple polygon can be drawn in O(N) time using the algorithm proposed in [3]. Let it give birth to K notches where the ith notch is assumed to have N vertices. The time and space complexities of this proposed algorithm are both O(N). where N is the total number of ertices of the polygon.

Below I have described a few results related to the analysis of the time complexity of processing the notches.

Lemma 1: Time required for drawing the safe boundary of a notch of Ni ertices is $O(N_i)$.

Lemma 2: At any point of time, the total space occupied by the V-LISTs of all the triangulation edges is O(N).

Lemmas 1 and 2 lead to the fact that the time required for the generation of the safe boundaries 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 safe boundaries of all the notches is also linear in the total number of vertices of all the notches.

Therefore, It can be said that the time and space complexities of our proposed algorithm are both O(N), where N is the total number of ertices of the polygon.

5. Conclusion

As conclusion I would like to point out a few important applications of computing a guard zone of a simple polygon.

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. 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) Ci is associated with a parameter δi such that a minimum clearance zone of width δi is to be maintained around Ci . 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. It also finds application in the polygon containment problem and in computing the buffer zone in geographical information systems, to name only a few.

6. References

1. Mehera, R., Pal, R.K. (2008). A Cost-Optimal Algorithm for Guard Zone Problem. In: Garg, V., Wattenhofer, R., Kothapalli, K. (eds) Distributed Computing and Networking. ICDCN 2009.

2. Subhas C. Nandy, Bhargab B. Bhattacharya, Antonio Hernández-Barrera, Safety Zone Problem, Journal of Algorithms, Volume 37, Issue 2, 2000, Pages 538-569, ISSN 0196-6774,

3. D. T. Lee, On finding the convex hull of a simple polygon, Internat. J. Comp. Inform. Sci. 12, No. 2 (1983) , 87-98.

4. 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.

5. M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, Computational Geometry: Algorithms and Applications. Springer-Verlag, Berlin, 1997.

6. B. Chazelle, Triangulating a Simple Polygon in Linear Time, Discrete Computational Geometry, vol. 6, pp. 485-524, 1991.

7. A. Hernandez-Barrera, Computing the Minkowski Sum of Monotone Polygons, IEICE Trans. on Information Systems, vol. E80-D, No. 2, pp. 218-222, 1996.

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. H. Pottmann and J. Wallner, Computational Line Geometry. Springer-Verlag, Berlin, 1997.