An Algorithm to Solve 3D Guard Zone Computation Problem

Ranjan Mehera, Piyali Datta, Arpan Chakraborty and Rajat Kumar Pal

Abstract The guard zone computation problem finds vast applications in the field of VLSI physical design automation and design of embedded systems, where one of the major purposes is to find an optimized way to place a set of 2D blocks on a chip floor. 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. In this paper, we introduce the problem in its 3D version. Considering 3D simple solid objects makes the guard zone computation problem more complex and helps to solve many real life problems like VLSI physical design, Geographical Information System, motion control in robotics, and embedded systems. In this paper, we develop an algorithm to compute guard zone of a 3D solid object detecting and excluding overlapped regions among the guard zonal regions, if any.

Keywords Simple polygon · Safety zone · Notch · Convex hull · False hull edge · Convolution · Minkowski sum · Extreme points of a curve · 3D simple solid object · Computational geometry · 3D coordinate geometry

R. Mehera (✉) · P. Datta · A. Chakraborty · R.K. Pal
Department of Computer Science and Engineering, University of Calcutta, Acharya Prafulla Chandra Roy Siksha Prangan, JD – 2, Sector – III, Saltlake City, Kolkata 700098, West Bengal, India
e-mail: ranjan.mehera@gmail.com

P. Datta
e-mail: piyalidatta150888@gmail.com

A. Chakraborty
e-mail: arpanc250506@gmail.com

R.K. Pal
e-mail: pal.rajatk@gmail.com

R. Chaki et al. (eds.), *Advanced Computing and Systems for Security*,
Advances in Intelligent Systems and Computing 396,
DOI 10.1007/978-81-322-2653-6_18

1 Introduction

Guard zone computation problem is well defined in literature as an application of Computational geometry. Often, this problem is known as safety zone problem [1]. In case of 2D guard zone computation problem, 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 case of VLSI layout design as well as in embedded system, a chip may contain several million transistors. The goal of placement is to find a minimum area arrangement for the blocks that helps to complete interconnections among them. A good routing and circuit performance heavily depend on a good placement algorithm. Placement of modules is an NP-complete problem.

Each circuit component P_i is associated with a parameter p such that a minimum clearance zone of width p must be maintained around that circuit component. The location of the safety zone of specified width for a simple polygon is an important problem for resizing the circuit components. If more than one polygonal region is close enough, their safety zones overlap, violating the minimum separation constraint among them. Thus, with respect to resizing problems in VLSI, this is the motivation of defining the safety zone of a polygon [2].

We have developed a number of algorithms to solve the guard zone computation problem for only 2D simple polygons or objects. Now, the question arises whether the problem can be visualized and solved for its 3D version. A 3D simple solid object is surrounded by a number of planes such that no two planes cut each other except at their edges. The pair of planes meeting at an edge is the neighboring planes. In this paper, we develop an algorithm to compute guard zone of a 3D solid object detecting and excluding overlapped regions among the computed guard zonal regions, if any.

If two or more objects are close enough so that their guard zones overlap, indicating the violation of the minimum separation constraint among them, the intersecting regions are to be detected such that the guard zone can be computed eliminating those intersecting regions. As our inclination in doing the task is in the domain of computational geometry for a given 3D simple object, we like to detect the part(s) of G that overlap(s) using the concept of analytical and coordinate geometry.

In this paper, we have solved the problem of computation of the guard zone for a simple solid object as well as detection of the overlapped regions (if any). While computing the initial guard zone G, we enclose the solid object P by G, which is essentially a collection of O(n) planar segments, cylindrical segments, and spherical components corresponding to the planar surfaces, convex edges, and convex solid vertices of the input simple solid object; we explain it in the subsequent sections. After computing the guard zone trivially, we detect the overlapped regions of the guard zone to find the union of all individual guard zonal regions of the object. Again, the 3D guard zone computation algorithm proposed in this paper is output

sensitive in nature, i.e., the computational complexity varies with the number of overlapped regions in the guard zone of the solid object. Hence, the complexity depends on the shape of the simple solid object provided.

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 the 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. The problem originates in the context of resizing the VLSI layout design [3].

In the context of guard zone computation, several different algorithms have been proposed so far. The most discussed tool for guard zone computation is the Minkowski sum. Essentially, Minkowski sum between a line (as polygonal segment) and a point (perpendicularly at a distance r apart) with same x- and y-coordinates gives a line parallel to the given one. But a question arises whether the parallel line is inside or outside the polygon. Here, the definition of Minkowski sum [4] can be extended as follow: 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 to $2A$, and $A–A$ does not equal to 'zero' in any sense. Apart from Minkowski sum, convolution can also be used as a tool for guard zone computation.

A linear time algorithm is developed for finding the boundary of the minimum area guard zone of an arbitrarily shaped simple polygon in [3]. This method uses the idea of Chazelle's linear time triangulation algorithm [5]. After having the triangulation step, this algorithm uses only dynamic linear and binary tree data structures.

Again, the problem of locating guard zone of a simple polygon has been solved and a time-optimal sequential algorithm for computing a boundary of guard zone that uses simple analytical and coordinate geometric concepts have been presented in [6, 7]. It uses three different procedures to compute guard zone at convex, concave, and linear regions of the polygon. The algorithm can easily be modified to compute the regions of width r outside the polygon (as guard zone), and also inside the polygon.

In paper [8], the authors have developed algorithm for detection of guard zonal overlapping in case of a 2D simple polygon and the algorithm uses the concept of line sweep algorithm for a set of parallel line segments. Most interestingly, the algorithm is output sensitive, i.e., its behavior changes with the input. Thus, the guard zone computation is easier for a polygon without notches than that of a polygon with notches as well as overlapped guard zonal regions.

3 Formulation of the Problem and the Algorithm

In case of 2D guard zone computation problem [8], our objective is to derive a 2D imaginary region outside the polygon such that each point p on the polygon maintains at least a distance r, which is predefined from each point q on that region. Similarly, for the 3D version of the guard zone computation problem, we derive a 3D imaginary region bounding the solid object such that it maintains at least distance r between them. For the sake of simplicity, we consider only that kind of solid objects, which consist of only planes, i.e., no curved surfaces are there. Hence, two neighboring planes meet at their edges, i.e., at a straight line where the corresponding planes make an angle that may either be convex or concave as it is formed outside the object. If the planes meet at a convex angle outside the object, such an edge is considered to be a convex edge; otherwise, it is considered as a concave edge if the planes meet at a concave angle outside the object.

On the other hand, a simple solid object may contain both convex and concave vertices in it; at such a vertex, we call a solid vertex, several planes of the solid object intersect. It is important to observe that at each such vertex of a simple solid object, the number of planes intersects is at least three. The vertices of a simple solid object are defined as follows. A (solid) vertex v of a solid object S is defined as concave, if for each pair of intersectional lines (for the associated planes) incident at v form an angle outside the object (i.e., an external angle at vertex v) less than 180º; otherwise, it is defined as convex.

As in Fig. 1a, which is a portion of solid object S where two adjacent planes A and B intersect, the normal vector for plane B is n_1 and that of A is n_2. Now, the angle between planes A and B is same as the angle between the normal vectors to planes A and B. Hence, $\theta = \cos{-1}(|n_1 \cdot n_2|/(|n_1||n_2|))$. This is how all external angles of the solid object S can be computed (as concave or convex) in time $O(\ell)$, where ℓ is the number of intersection lines between adjacent planes of S. Now, it can be said that, if at a solid vertex the number of convex edges meeting at the vertex is greater than the number of concave edges, the vertex is a convex solid vertex; whereas, it is

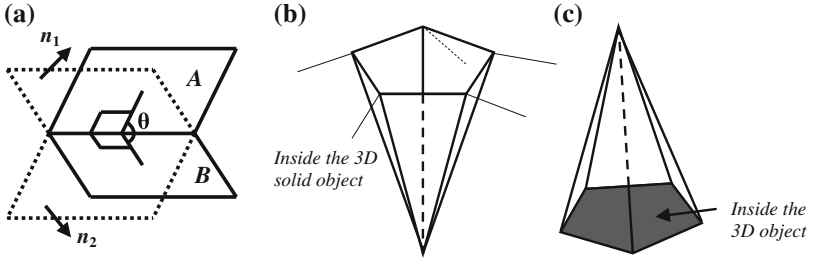

Fig. 1 **a** Part of a solid object with planes A and B, and a concave (external) angle θ between them. The *dotted lines* indicate the imaginary portion of the planes A and B (inside the solid object); n_1 and n_2 indicate the normal vector to the planes B and A, respectively. **b** Deep or concave solid vertex in a 3D solid object. **c** Peak or convex solid vertex in a 3D solid object

said to be a concave solid vertex if the number of concave edges meeting at the vertex is greater than that of convex edges meeting at the vertex. A concave and a convex solid vertex have been depicted in Fig. 1b, c, respectively.

A guard zone G of a 3D simple solid object S with n solid vertices can eventually be obtained as follows. Let intersection lines created due to the intersection of adjacent planes of the object be labeled as l_1, l_2, …, l_ℓ in some order. For each intersection line l_i, $1 \le i \le \ell$, where two adjacent planes of S intersect, we bisect the external angle. Then we draw a plane parallel to the plane (l_i, l_{i+1}), $1 \le i < \ell$, at a distance r outside the solid object that may be a portion of the desired guard zone G that is being computed, assuming that l_i and l_{i+1} are forming the plane under consideration. To be precise, successive guard zonal planes must meet (or intersect) each other only at an angle bisector of line l_i, formed due to the intersection of allied planes of the given 3D simple solid object. At the edges of the simple object, the guard zonal region is cylindrical in shape. Furthermore, the parallel planes that are guard zone of the neighboring planes are tangent to the cylindrical surface whose axis is the line of intersection of the two assumed planes. At the peak, where more than two planes coincide, the guard zonal regions result a spherical shape and the guard zonal planes of the given object's planes that meet at the peak are tangent to the spherical surface.

As the input 3D simple solid object is made of a set of planes (by assumption), for computing the guard zone of individual plane, we first need to compute planes parallel to the ones specified in the form of 3D simple solid object, at a distance r outside the object. For example, we have considered two planes for which we would like to compute parallel planes at distance r outside the two believed planes. Let $ABCD$ and $BEFC$ are two planes adjacent through the edge BC as shown in Fig. 2a. Since we always consider a simple solid object as input, in reality these two planes are also adjacent to some other planes of the input solid object through different edges.

For $ABCD$, we draw four perpendicular line segments at A, B, C, and D of length r. Thus, we get Aa, Bb, Cc, and Dd, respectively, and obtain the plane $abcd$

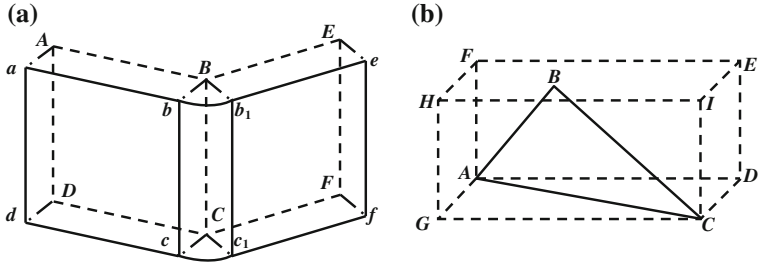

Fig. 2 **a** Two plane segments ($ABCD$ and $BEFC$) meet at a line segment (BC) and their guard zones ($abcd$ and b_1efc_1) meet at a cylindrical segment (bb_1c_1c). **b** Planar surface ABC bounded by the 3D box $AGCDFHIE$

which is parallel to the plane *ABCD* at distance *r*. Similarly, for the plane *BEFC* we get plane b_1efc_1 as its parallel one.

Now, the guard zonal planes of two neighboring object planes meet at a cylindrical surface which is considered to be the intermediate curved surface between the two planes said above. To compute this surface, we have drawn a cylindrical surface considering *BC* as its axis and making an angle, $\angle bBb1$ at the axis. Now, the guard zonal planes of the object planes, i.e., abcd and $b1efc1$, are tangent to this cylindrical surface.

Now, there is a set of guard zonal components in the search space from which we have to find out the pair of intersecting components. In case of 2D, i.e., the sample space contains only a set of line segments, we could use plane sweep algorithm to find out the interesting pairs of line segments. However, in 3D, space sweep algorithm is only applied for a set of orthogonal planes though the guard zonal plane segments are not necessarily orthogonal [9]. Furthermore, the search space also contains cylindrical and spherical regions. Hence, space sweep algorithm cannot be applied directly.

Let us take a different view toward the problem. If we could bind each guard zonal component within its minimum possible orthogonal 3D box, then the problem reduces to find the overlapping pairs of those boxes only. Again, a 3D orthogonal box consists of six bounding surfaces parallel to one of the three coordinate planes and the problem is reduced in finding overlaps among these boxes which are regular in shape.

Now, we can imagine that in the search space there are only O(n) 3D orthogonal boxes, where n is the total number of planes in a given 3D simple solid object. Exhaustively, O(n^2) checking needs to be performed to find all the intersections among the set of boxes. Instead of the exhaustive method just described, we like to use the space sweep method [9], which solves the problem in O(n log n) time. This three-phase sequential algorithm computes O(n) number of 3D bounded boxes in its first phase and in the second phase it checks for overlapping among the boxes, whereas in the third phase it deals only with the corresponding guard zonal regions for which overlapping has been detected in the second phase. Thus, if there is no overlapping between any two boxes, the third phase of the algorithm is skipped and the results we obtain are reported accordingly.

Phase I: Construction of Bounded Boxes for Individual Guard Zonal Surfaces
As the guard zone of a 3D simple solid object may consist of a set of planar, cylindrical, and spherical surfaces, we compute bounded boxes for each such surface individually. To bind a planar or a curved surface, we take orthogonal projection of each surface on xy, yz, and zx planes of the coordinate system. Hence, in each plane we obtain either a line segment or a curved segment or a bounded region which are not necessarily identical. Thus, we get three 2D counterparts of each surface of the 3D guard zone. For each of the 2D segments, we compute its 2D bounded box as described in [8]. Now, we obtain three mutually perpendicular planes, i.e., three surfaces of the 3D bounded box. The three other surfaces also need to be constructed to complete the 3D bounded box of the guard zonal portion

taken into consideration. We like to illustrate this concept with the help of an example discussed below.

Let us consider a guard zonal planar surface *ABC* as shown in Fig. 2b, of which we would like to construct the bounded box. At first, we take orthogonal projection of *ABC* on xz, xy, and yz planes. This creates three 2D regions on the three planes for which we find three 2D bounded rectangles. Here, *AGHF*, *ADCG*, and *ADEF* are three bounded rectangles on the three coordinate axis planes, respectively. Hence, the three adjacent planes are computed of the 3D box and other three planes are yet to be constructed. The width of the 2D box on yz plane indicates the width of the 3D box along y axis and z axis; similarly, we get the width along x axis from the 3D box formed on planes xz and xy.

Now, to construct the remaining surfaces of the 3D box, we have to draw three planes parallel to xz, xy, and yz planes. For an example, the plane *DCIE* is drawn parallel to the plane *AGBF* maintaining a distance *AD* which is the width of the 2D box on yz (or xy) plane as well as the width of the 3D box along y axis. Similarly, the face *GCIH* and *HIEF* of the 3D box are drawn parallel to *ADEF* and *ADCG*, respectively. Thus, we get the 3D box *AGCDEFHI*, which bounds the guard zonal plane *ABC*. We perform similar task to get 3D bounded boxes for each of the guard zonal (planar or curved) surfaces. The 3D bounded boxes computed in this fashion are not necessarily disjoint to each other, i.e., they may have overlaps. Figure 3a, b shows the projection of a spherical surface on three orthogonal coordinate planes.

Phase II: Detection of Overlapping among the Bounded Boxes

At the end of the first phase, there are $O(\ell)$ number of 3D boxes in the search space, in which there may be overlap among the boxes, where ℓ denotes the number of intersection lines at which the consecutive plane segments of the object meet. In the second phase, we use the space sweep algorithm [4] that checks for intersection among the boxes and report it accordingly.

In the search space, an infinite plane parallel to each of the xz, xy, and yz planes is moved along its perpendicular direction (i.e., along y-, z-, and x-axis) consecutively. For each sweep, we get information regarding overlapping of the boxes along the corresponding direction. Suppose two boxes overlap while sweeping

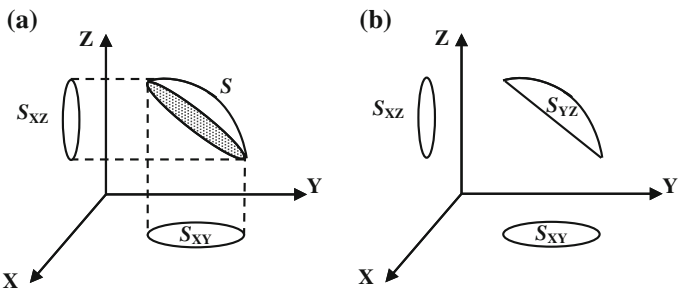

Fig. 3 **a** Projections are taken on the orthogonal coordinate planes **X**. **b** Projection of the spherical surface *S* on xy plane (S_{XY}), yz plane (S_{YZ}), and xz plane (S_{XZ})

through y axis; however, it does not necessitate having overlapping along other two axes. It means that there was no overlapping between them; rather, they share their y-interval. By Lemma 1, we can conclude that two boxes overlap if and only if overlapping has been detected along all the three directions.

Lemma 1 Two boxes overlap if and only if they share x-, y- and z-interval.

Proof Let B1 and B2 be any two boxes in the search space. The x-, y-, and z-span of B_1 and B_2 are $\{(x_{11} - x_{12}), (y_{11} - y_{12}), (z_{11} - z_{12})\}$ and $\{(x_{21} - x_{22}), (y_{21} - y_{22}), (z_{21} - z_{22})\}$, respectively. In Fig. 4, we observe that $x_{11} < x_{21}$, $y_{11} < y_{21}$, and $z_{11} < z_{21}$. Now in a case, if any two boxes overlap, then the point(s) belonging to the overlapped region is (are) also belonging to the individual boxes, and hence, if we that consider boxes B_1 and B_2 have overlapped and an arbitrary point A(x, y, z) is a point within the overlapped region, then the following inequalities hold.

$$x_{i1} < x < x_{i2},\ y_{i1} < y < y_{i1},\ \text{and}\ z_{i1} < z < z_{i1},\ \text{where}\ i = 1, 2. \tag{1}$$

Considering the fact stated in the inequation (1), we can conclude that $x_{21} < x_{12}$, $y_{21} < y_{12}$, and $z_{21} < z_{12}$, i.e., there are overlaps along all x-, y-, and z-direction. On the flip side, if there is no overlapping between boxes B_1 and B_2, the inequation (1) stated above would not be satisfied and must deny the existence of any such point A, which in turn ensures that there is no overlapping between the mentioned boxes.

Now, in our algorithm for overlapping detection among the 3D bounded boxes, we store the information of overlapping for three different directions and boxes by maintaining three different Binary Search Tree (BST) data structures. Again, we extract the final information regarding overlapping by combining the results of these three BSTs.

We illustrate the process through an example. Let us consider that there are three 3D boxes, namely 1, 2, and 3 on which we perform space sweep operations along three coordinate axes. The surfaces of the boxes which are parallel to the sweep plane are considered to be the event points during sweeping. For an example, when we move the xz plane, the surfaces of the boxes parallel to xz plane are considered to be the event points and the surface with lower y value is the starting point of the allied box while the surface with higher y value is considered to be the end point. At the beginning of the sweeping process, the event points are sorted depending on the values of that coordinate along which the sweeping is being performed. Through

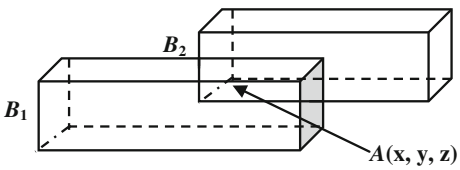

Fig. 4 Two boxes B_1 and B_2 are overlapped and A(x, y, z) is a point at the overlapped region

the sweeping process whenever the sweep plane is at a starting point, the corresponding box is inserted in the query tree and the overlapping list is updated by inserting overlapping information of the newly inserted box and the existing boxes in the query tree. On the other hand, at an end point the corresponding box is deleted from the tree. Overlapping information for two boxes is stored in the form of a dipole $\{box_1, box_2\}$.

Let us consider the example depicted in Fig. 5a–c. In Fig. 5a, xz plane is moving along y axis and the event points have been denoted as 1s, 1e, 2s, 2e, 3s, and 3e. Now, the event points are sorted according to their y-coordinate values and we obtain 2s, 3s, 1s, 2e, 1e, and 3e. At each event point, the query tree is updated through insertion or deletion and the event point is deleted from the event point list. Hence, the sweeping ends when the event point list is empty and we obtain a list of overlapping 3D boxes.

In Fig. 5a, when the sweep plane is at the starting plane 2s, the corresponding box_2 is inserted in query tree. Next event point is 3s and it is also a starting point; hence, box_3 is inserted as the right child of box_2, as its y-coordinate value is greater than that of box_2. The overlapping list which was initially empty is now updated by inserting the pair of boxes {2, 3} or {3, 2}. To remove ambiguity, we prefer to store the lower numbered box first, i.e., {2, 3} is inserted into the overlapping list. The next event point is 1s and box_1 is inserted into the query tree as the right child of box_3.

After the insertion, as the tree becomes height imbalanced, AVL rotation is performed to balance the tree. Now the overlapping list is updated by inserting event points {1, 3} and {1, 2}. At the next three event points, as these are the end points, the allied boxes are deleted from the query tree. Finally, the query tree as well as the event point list becomes empty and we obtain a set of overlapping pairs along y axis, say A. Here, A = {{1, 2}, {2, 3}, {1, 3}}. The query tree after each update has been shown in Fig. 6a.

Next, the sweep plane is moved through z axis and the surfaces of the boxes that are parallel to xy plane are considered to be the event points. Now, the starting and ending event points are shown in Fig. 5b; after sorting we obtain the sequence of the event points as 3s, 2s, 3e, 1s, 2e, and 1e and the query trees are updated at each event point. After completion of the sweeping, the set of overlapping pairs of boxes, say set B, becomes {{2, 3}, {1, 2}}. The series of query trees formed after each operation performed for each event point has been shown in Fig. 6b. At the last step of the space sweep phase, yz plane is swept along x axis where the sorted list contains 1s, 2s, 1e, 2e, 3s, and 3e, the event points based on Fig. 5c.

At the end of sweeping, it provides the overlapping information through a set of overlapping pairs. If the set is named as C, C = {{1, 2}}. The series of query trees formed after each operation performed for each event point has been shown in Fig. 6c.

We have already proved the phenomenon that if two 3D boxes have overlapped with each other, they must overlap along all the three axes. Now, to detect the boxes which overlap indeed, we have to find the common pair(s) in these three sets A, B, and C. In our example, if we perform $A \cap B \cap C$, we obtain only one pair {1, 2},

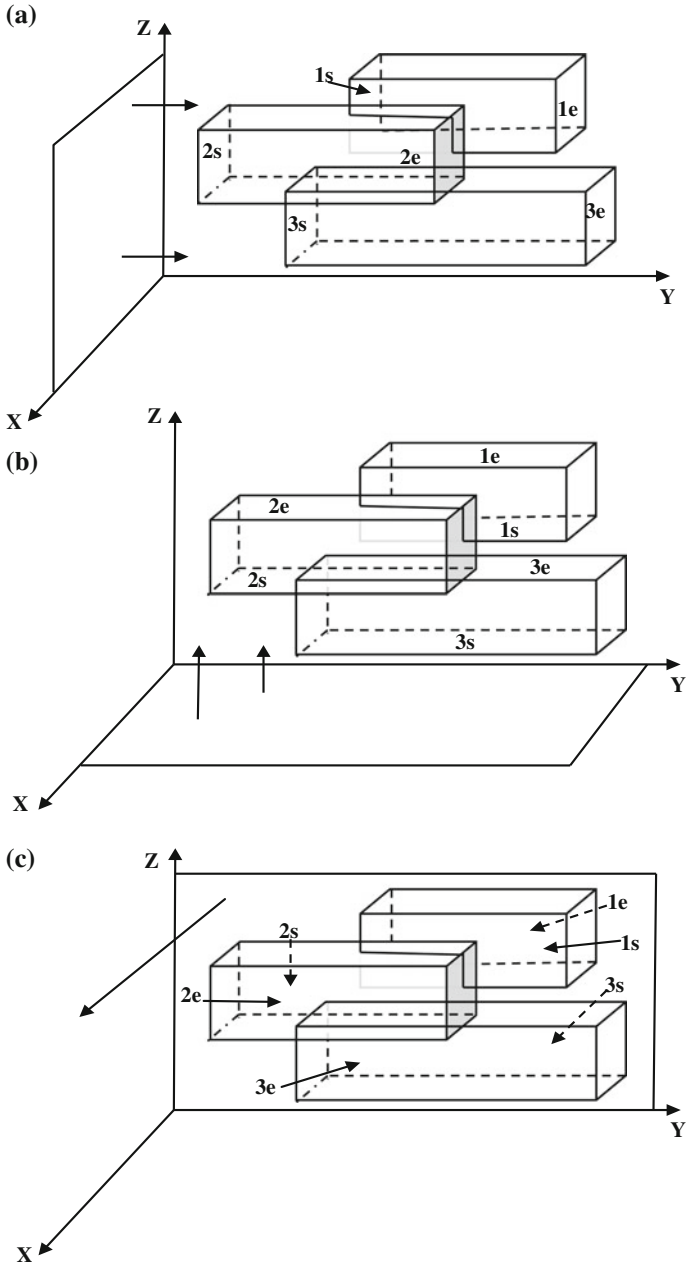

Fig. 5 Sweep of the plane along **a** y axis, **b** z axis, and **c** x axis

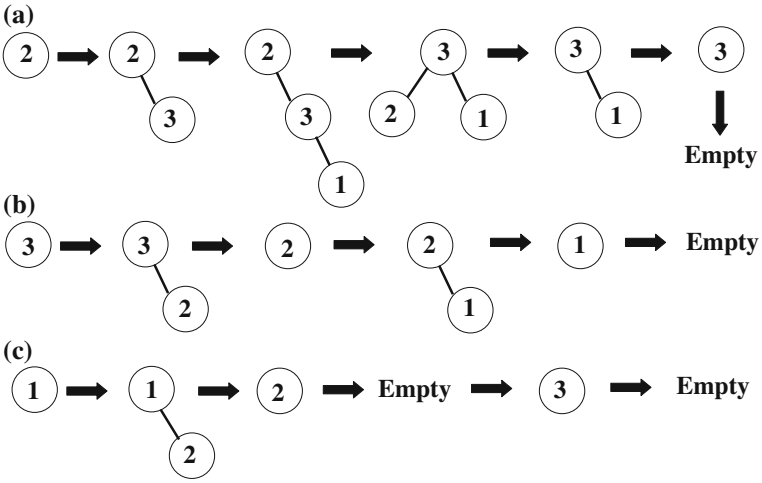

Fig. 6 Query trees during the sweep of the plane along **a** y axis, **b** z axis, and **c** x axis

though there are three entries in set A and two entries in set B. Hence, box1 and box2 have intersection and we deal with only these boxes in the third phase, as only these boxes contain the probable intersecting guard zonal surfaces.

Phase III: Detection of Intersection among the Guard Zonal Components Contained in the Overlapped Boxes

The third phase of our algorithm deals with those guard zonal surfaces whose bounded boxes are found overlapped in the second phase. As there are only three types of guard zonal surfaces, any pair of them may intersect with the other, and there are only six types of possible intersections, that are planar–planar, planar-spherical, planar-cylindrical, spherical-spherical, spherical-cylindrical, and cylindrical-cylindrical.

Planar–Planar Intersection If we like to check intersection between two plane segments, we notice that two plane segments always cut at a line segment satisfying the equation of both the planes [10]. Two plane segments intersect in two ways; either one of them fully passes through the other or they intersect partially. If *A* and *B* are two plane segments, the possible intersections are depicted in Fig. 7.

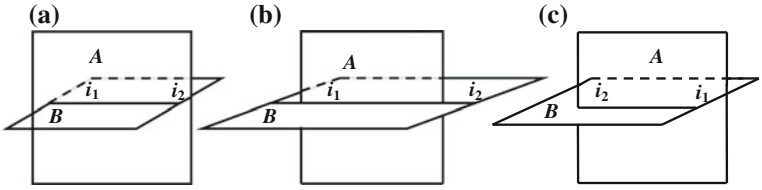

Fig. 7 **a** *B* passes through the plane segment *A*. **b** *A* passes through the plane segment *B*. **c** The planes partially cut each other

In Fig. 7a, *B* passes through A resulting in two intersection points within the boundary of *A*, whereas in Fig. 7b, *A* passes through *B* and results two intersection points within the boundary of *B*. In Fig. 7c, they cut partially; hence, one of the intersection points is within the boundary of *B* and the other one is within the boundary of *A*.

As a planar surface is always adjacent to either a spherical or a cylindrical surface, a partial cut refers to the fact that there may be intersection between the plane segments that partially cut and the adjacent surface of the other plane, i.e., one plane segment may cut the other in such a way that it cuts its neighboring cylindrical or spherical segment as well. In each of the cases, the planar–planar intersection is a line segment. But there is a difference in the above three cases; the first one has the intersection line segment starting from one point on the plane segment and ending at another point on the same plane segment having the intersection line segment fully on the plane segment, the second one has the intersection line segment beyond the plane segment, and in the third case the intersection line segment starts at a point on the plane segment while ending at a point beyond the plane segment. We discuss all the three cases below.

We define each plane segment by the equation of the plane and its boundary line segments, and therefore, the equations of lines. Now, we consider a plane and check for intersection between the line segments of the other planes. If we consider plane *A*, i.e., its equation and take the line segments of plane *B*, it results in i_1 and i_2. The points may either reside on or within the boundary of *A*. For each point, we traverse the boundary of *A* clockwise (or anti-clockwise) if the point is always on the right (or left) side, the point is within the boundary; otherwise, if the point satisfies one of the boundary line segments of *A*, the intersection point is on the boundary of the plane.

Planar—Spherical Intersection Whenever a plane cuts a spherical region once, the shape of the intersection is of a circular curve [10, 11]. Now, if the plane intersects a spherical region it results a circle, as the intersection curve has been depicted in Fig. 8a. As a spherical surface is adjacent to a set of cylindrical and planar surfaces, we may conclude that the planar surfaces that cut the spherical

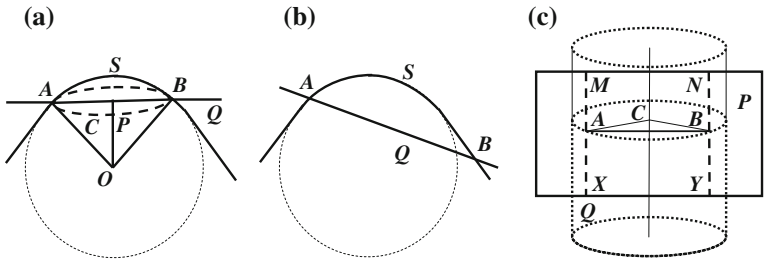

Fig. 8 The surface *Q* intersects the spherical surface *S*. **a** It results a circle *C* as the intersection *curve*. **b** When the surface (*Q*) partially intersects the spherical surface (*S*) that results a circular *curve* which is extended through the junction of the spherical and cylindrical or planar surfaces. **c** When a plane *P* intersects a cylinder *Q* resulting in two intersection line segments *MX* and *NY*

surface once, cut one or more adjacent surfaces as well, as shown in Fig. 8b, and has been discussed previously in case of planar–planar intersection.

Let *S* and *Q* be the spherical and planar surfaces. In Fig. 8a, a circle (*C*) is produced after the plane cuts the spherical surface. We draw a perpendicular *OP* on the plane. As we obtain the point *P* and the length of *OP*, we find the length of *AP*. Thus, the center (*P*) and the radius (*AP*) are known to us, and hence, we get the equation of the circle. Now, it may so happen that the plane does not cut the spherical surface at all. In that case, we check *OP* with the radius of the sphere. If *OP* is greater than the radius, there is no intersection.

Planar—Cylindrical Intersection When a plane segment cuts a cylinder, the intersection is a line segment satisfying both the equations of the cylinder and the plane [10, 11]. If the plane cuts the cylinder twice, we obtain two such line segments which are parallel to the axis of the cylinder. When the plane cuts the cylinder once, it means that there may be intersection with the adjacent surfaces of the cylindrical surface.

Let the plane *P* cut the cylinder *Q* twice, as shown in Fig. 8c. Then, we have to find two line segments MX and *NY*. We have the equation of cylinder as $(ny\text{-}mz)^2 + (lz\text{-}nx)^2 + (mx\text{-}ly)^2 = r^2(l^2 + m^2 + n^2)$ and its axis as (x/l) = (y/m) = (z/n). Also the equation of the plane is ax + by + cz = d. We draw a perpendicular *CO* from *C*, a point on the axis, on the plane. As we know, the radius of the cylinder (*CA* or *CB*), *AO* (*BO*) can be directly found. Hence, we find the coordinate of point *A*. As the intersecting line segment passes through *A* and is parallel to the axis, we can derive its equation.

After knowing the equation, we check for intersection between the boundary line segments of planar surface, and obtain two intersection points on the boundary of the plane, here *M* and *X*. The line joining the two gives the line segment *MX*. Similarly, for point *B*, we get the line segment *NY*.

Spherical—Spherical Intersection If two spherical surfaces intersect, we obtain a circle as the intersecting curve and the equation of the circle satisfies equations of both the spheres [10, 11]. Let two spheres *S* and *S′* with their centers *A* and *B*, respectively, intersect with each other as depicted in Fig. 9a. If their equations are

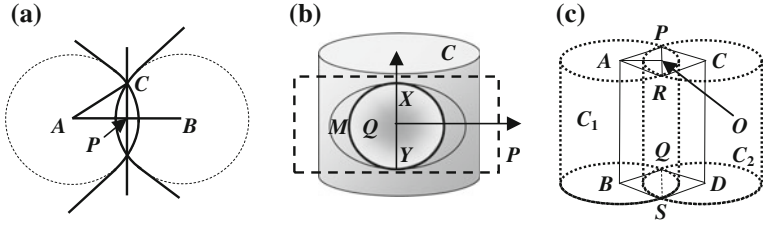

Fig. 9 **a** Two spheres with *A* and *B* as their centers intersect each other resulting in a circle as the intersection *curve* with *P* as the center and *PC* as the radius of the circle. **b** A sphere intersects the cylinder *C* resulting in *M* as the intersection curve in 3D, to find plane *P* that is drawn and the sphere cuts it along a circle *Q*. *X* and *Y* be the common points on *M* and *Q*. **c** Two cylinders C_1 and C_2 cut each other keeping their axes *AB* and *CD* parallel, results in two intersection line segments *PQ* and *RS*

$S \equiv x^2 + y^2 + z^2 + 2gx + 2fy + 2hz + c = 0$ and $S' \equiv x^2 + y^2 + z^2 + 2g'x + 2f'y + 2h'z + c' = 0$, the coordinate of the points of their intersection satisfying the equation $S–S' \equiv 2(g–g')x + 2(f–f')y + 2(h–h')z + (c–c') = 0$, which is the equation of the plane of intersection of the two spheres. This plane cuts either of the spheres in a circle. Our objective is to find out the circle thus obtained.

We draw a perpendicular from the center of one of the spheres on the plane. Here in the figure, the perpendicular *AP* is drawn on the plane. From the length of *AP* (as we can find the length of a perpendicular from a point outside a plane) and *AC* (the radius of the sphere), *PC* is calculated which is the radius of the intersecting circle. Again, as we know the coordinate of the center (*P*) and the radius (*PC*), the equation of the circle is immediately derived.

Spherical—Cylindrical Intersection When a spherical surface intersects a cylindrical surface, the intersecting curve does not lie in 2D plane [11]. It can be visualized by drawing a circle on a plane and then the plane is wrapped over the cylinder. The circle is not on the 2D plane now; rather, it is on the surface of the cylinder.

Here, in Fig. 9b, a sphere intersects a cylinder *C* resulting in the intersecting curve *M*. We have to derive the equation for *M*. At first, we draw a plane tangent to the cylinder, on which the line segment normal to the axis of the cylinder from the center of the sphere, is perpendicular. Now the sphere cuts the plane and results a circle *Q* as their intersecting curve. We derive the equation for this circle as we did in the planar-spherical intersection. Now, as we know the circle *Q*, we have its center and radius.

At this moment, to find *M* we need to know a point on it and then the locus of the point on the cylindrical surface. So, we draw a line passing through the center of the circle *Q* and parallel to the axis of the cylinder. This line cuts the circle at two points *X* and *Y*. The locus of either of the points satisfying the equation of the cylinder and maintaining the distance from the center of the circle as a constant provides the equation for *M*.

Cylindrical—Cylindrical Intersection If two cylinders intersect each other as in Fig. 9c, we obtain two intersecting line segments satisfying equations of both the cylinders. From the equations of the cylinders, we obtain the equation of the plane satisfying both the equations of the cylinders.

Let C_1 and C_2 be two cylinders. We draw *AO*, perpendicular on the intersecting plane from *A*. From *AO* and *AP*, we derive *OP* and hence obtain *P*. Also we find in the same way the point *Q*. Then *PQ* is attained as one of the intersecting line segments. Similarly, the other intersecting line segment *RS* is derived.

4 Computational Complexity

It is easy to observe that the guard zone of an n-vertex, ℓ-intersection line and p-plane convex solid object is a convex (3D) region with p planes, ℓ cylindrical arcs, and n spherical arcs only, when there is no intersection, and with intersection there

might be p planes only, where $n = O(\ell)$ as well as $p = O(\ell)$. The planes of the guard zone are parallel to the planes of the solid object at a distance r apart, outside the solid object, and two adjacent planes of the guard zone are joined by a cylindrical arc of radius r centered at the associated intersection line of the solid object. Spherical arcs of radius r each are introduced as parts of the computed guard zone at the vertices of the solid object, where the associated planes of the guard zone are tangent to the cylindrical as well as spherical arcs and the cylindrical portions of the guard zone are also ending with spherical arcs of the guard zone near the vertices of the solid object. As a result, the time required for computing a 3D guard zone of a convex solid object is $O(n+\ell+p) = O(\ell)$.

Now, as per the next step, we need to draw the orthogonal projections of the individual guard zonal components, including planar, spherical, and cylindrical surfaces onto xy, yz, and zx planes. The projections can be drawn in linear time with respect to the number of guard zonal components, which is $O(\ell)$, where ℓ is the number of intersection lines present in the given 3D simple solid object. Therefore, for each guard zonal component, we obtain three different 2D objects at each of the xy, yz, and zx plane. Now, we need to merge the individual 2D components belonging to a specific 3D guard zonal component in such a way that the 2D components collectively form a 3D orthogonal box that encloses the corresponding 3D guard zonal component to its entirety. This step can be achieved in constant time and hence, the complexity of this step is O(1).

Once the orthogonal boxes are ready, we feed these boxes into our customized space sweep algorithm for further processing. During the execution of the space sweep algorithm, the faces of each orthogonal boxes parallel to the xy, yz, and zx plane have been considered as event points. The event points are maintained in a dynamic list data structure, whereas the 3D boxes corresponding to the event points are maintained in a separate BST data structure for subsequent processing. The algorithm terminates when the event list becomes empty. Hence, the operations like creation, insertion, and deletion from the BST take $O(\ell \log \ell)$ time, whereas the similar operations for the dynamic list data structure consumes $O(\ell)$ time, for each of the axis planes. Therefore, the overall time complexity to perform this particular step requires $O(\ell \log \ell)$.

Now, let us assume that the execution of the previous step yields the following result, where the sweep plane xy has produced a set I containing all the overlapped box pairs while sweeping; afterward the yz plane operates on the pair of boxes contained in set I and produces a reduced set named J. Similarly, zx plane operates on the pair of boxes contained in set J and yields the final set K containing all the possible boxes that needs further investigation. Thus, xy $\rightarrow I$, yz $\rightarrow J$, and zx $\rightarrow K$.

All the operations explained above can be executed in $O(\ell \log \ell) + O(I \log I) + O(J \log J) \equiv O(\ell \log \ell + I \log I)$ time, since I dominates J.

As per the final step, we are left with the detection of intersection among the guard zonal components contained within the 3D boxes registered in set K. This operation can be executed in constant time for each pair of such guard zonal component, and hence, the time complexity for the intersection detection among the guard zonal components is $O(K)$.

The overall time complexity for the 3D guard zone computation for a given simple object is $O(\ell) + O(\ell \log \ell) + O(K) \equiv O(\ell \log \ell + I \log I)$.

5 Application

The guard zone computation problem occupies vast place of interest in the field of VLSI physical design automation and design of embedded systems, robotic motion control, geographic information system, etc. In VLSI physical design for optimized placement on the chip, we have to consider 3D subcircuits and their 3D guard zone for avoiding any parasitic effect. In this context, we deal each 3D subcircuit as a 3D simple solid object and accordingly we compute its guard zone. As cited for its 2D variation, we may achieve the goal in 3D counterpart.

In Robotics, it is important to have the motion planning feature built within the robot itself. This motion planning feature ensures that the robot does not collide with any obstacles while it is in motion unless it is programmed to do so. As we have noticed that Minkowski sum finds a tremendous application in motion planning of an object among obstacles [3], the guard zone computation can also be used to solve similar problems. Once a robot identifies the obstacles that it needed to bypass and transforms them into 2D simple polygons, then it can use the computed guard zone to avoid any possible collision. As the robot moves along its way, it keeps checking whether it encounters the already computed guard zone of any of the obstacles and if it finds one, then immediately it changes its direction unless it reaches to its destination. This process can further be enhanced by incorporating a learning mechanism within the robot where the robot records the objects found as an obstacle so far along with the computed guard zone and later when the robot encounters the similar objects, then it applies the already computed guard zone to avoid any possible collisions.

3D guard zone computation plays a significant role in robot motion planning, as the real life scenario conveys that the robots and obstacles faced are 3D in nature. Considering this fact it is obvious to take into account the 3D guard zone computing for efficient motion planning for 3D robots. Now, the problem of motion planning for robots can further be simplified if the robot somehow acquires the information of all the obstacles it is going to encounter from its start to final destination. Then the robot can compute its own simplified guard zone along with the guard zone for all the obstacles that it is going to encounter and then determines the path it is supposed to take to reach its destination. Another important application of the 3D guard zone computation problem is in the medical field. Consider the treatment of cancer cells in human body; now the biggest unsolved mystery till date is finding a targeted treatment for the cancer affected cells. There can be two different approaches of applying the guard zone computation in this regard: (1) A guard zone defining the growth of cancer affected cells in human body, definitely the medicine that should target these cells should have a faster healing power compared to the growth of the cancerous cells and also the medicines should have a predefined

minimum guard zone which completely encompasses the affected cancerous cells. (2) A comparative study (with respect to the guard zonal effects) of various medicines that target the cancerous cells to determine which medicine needs to be applied on human body to reduce the effect of medicine on the healthy cells.

It also finds application in computing the buffer zone in geographical information systems [1], to name only a few.

6 Conclusion

As discussed earlier, resizing of electrical circuits is an important problem in VLSI layout design as well as in embedded system design, while accommodating the (groups of) circuit components on a chip floor. This problem motivates us to compute a guard zone of a simple polygon. In robot motion planning, geographic information system, embedded system 3D guard zone computation takes an important role. In this paper, we have considered the problem of computing a guard zone of a (3D) simple polygon, and developed a sequential algorithm for computing the same that uses the concepts of analytical and coordinate geometry to detect overlapped region(s) within the guard zone (if any) and accordingly exclude that region to report the resulting outer guard zone. Our algorithm can easily be modified to compute the regions of width r (as guard zonal distance) outside the polygon, and also inside the polygon (if necessary), which may find several applications in practice. This work can also be extended for computing a guard zone of a three-dimensional solid object that may not be a simple one, as a problem of probable future work.

References

1. Heywood, I., Cornelius, S., Carver, S.: An Introduction to Geographical Information Systems. Addison Wesley Longman, New York (1998)
2. Pottmann, H., Wallner, J.: Computational Line Geometry. Springer, Berlin (1997)
3. Lee, I.-K., Kimand, M.-S., Elber, G.: Polynomial/rational approximation of minkowski sum boundary curves (Article No.: IP970464). Graph. Models Image Process. **60**(2), 136–165 (1998)
4. Mehlhorn, K.: Data Structures and Algorithms 3: Multi-Dimensional Searching and Computational Geometry. Springer (1984); Bajaj, C., Kim, M.-S.: Generation of configuration space obstacles: the case of a moving algebraic curves. Algorithmica **4**(2), 157–172 (1989)
5. Chazelle, B.: Triangulating a simple polygon in linear time. Discrete Comput. Geom. **6**, 485–524 (1991)
6. Mehera, R., Chatterjee, S., Pal, R.K., A time-optimal algorithm for guard zone problem. In: Proceedings of 22nd IEEE Region 10 International Conference on Intelligent Information Communication Technologies for Better Human Life (IEEE TENCON 2007), CD: Session: ThCP-P.2 (Computing) (Four pages). Taipei (2007)

7. Mehera, R., Chatterjee, S., Pal, R.K.: Yet another linear time algorithm for guard zone problem. Icfai J. Comput. Sci. **II**(3), 14–23 (2008)
8. Mehera, R., Chakraborty, A., Datta, P., Pal, R.K.: An innovative approach towards detection and exclusion of overlapped regions in guard zone computation. In: Proceedings of 3rd International Conference on Computer, Communication, Control and Information Technology (C3IT 2015), pp. 1–6. Academy of Technology, West Bengal (2015)
9. Hwang, K., Briggs, F.A.: Computer Architecture and Parallel Processing. McGraw-Hill, New York (1984)
10. Grewal, B.S.: Higher Engineering Mathematics, 39th edn. Khanna Publishers, Delhi India (2005). ISBN 81-7409-195-5
11. Chakravorty, J.G., Ghosh, P.R.: Analytical Geometry and Vector Analysis. U.N. Dhur and Sons Private Ltd., Kolkata (2009)