Chapter: 2

Literature Survey

2.1 Overview

The guard zone computation problem occupies a majority of interest in the field of VLSI physical design automation and design of embedded systems, as resizing is an important problem in VLSI layout design as well as in embedded system design for decades. As consequence, there is a number of existing literature about solving the problem in different ways, i.e. using different tools. In this thesis, our inclination is towards computational and coordinate geometry. In the context of 2D guard zone computation, several different algorithms have been proposed so far using computational geometry; some of those have been discussed in this chapter.

The most discussed tool for guard zone computation is the Minkowski Sum [32]. Apart from Minkowski sum, convolution can also be used as a tool for guard zone computation. Minkowski sum between a line (as a polygonal segment) and a point (perpendicularly at a distance $r$ apart) with the same $x$- and $y$-coordinates gives a line parallel to the given one. However, the question arises is whether the parallel line is inside or outside the polygon. Here the definition of Minkowski sum [32] can be extended as below.

If $A$ and $B$ are subsets of $\mathbb{R}^n$, and $\lambda \in \mathbb{R}$, then $A+B = \{x+y \mid x \in A, y \in B\}$, $A-B = \{x-y \mid x \in A, y \in B\}$, and $\lambda A = \{\lambda x \mid x \in A\}$. Note that $A+A$ does not equal $2A$, and $A-A$ does not equal ‘zero’ in any sense.

The convolution between a polygon and a circle of radius $r$ may give us the desired solution of a guard zone. However, the circles need to be drawn in every possible point of the polygon and thus the time complexity of the algorithm increases. The complexity of computing Minkowski sum of two arbitrary simple polygons $P$ and $Q$ is $O(m^2n^2)$ [27], where $m$ and $n$ are the numbers of vertices of these two polygons. In particular, if one of the two polygons is convex, the complexity of Minkowski sum reduces to $O(mn)$. In [45],
numerous results are proposed on the Minkowski sum problem when one of the polygons is monotone.

Apart from the above mentioned two procedures, a linear time algorithm is developed for finding the boundary of the minimum area guard zone of an arbitrarily shaped simple polygon in [60]. This method uses the idea of Chazelle’s linear time triangulation algorithm [10]. After having the triangulation step, this algorithm uses only dynamic linear and binary tree data structures. Besides the above said literature, there are many more which use triangulation as their tool like Delaunay triangulation [31].

In this chapter, we like to discuss these algorithms for computing guard zone of a simple polygon, and we also cite a relative study of these methods with respect to their performances.

2.2 Convolution

As per one of the dictionary definition, the term convolution means “Something that is complicated, esp. overly complicated”. In mathematics, the convolution is an integral that expresses the amount of overlap of one function $g$ as it is shifted over another function $f$. The convolution is denoted by $f \ast g$. The convolution operation involves ‘mapping’ the function $g$ onto the function $f$, illustrated graphically below.

![Convolution of two functions](image)

**Figure 2.1:** Convolution of two functions $f$ and $g$.

2.2.1 Properties of Convolution

Few important mathematical properties of convolution have been discussed below.
Commutative Property

The order in which two functions are convolved makes no differences; the results are always identical. Therefore, the convolution of two functions $f$ and $g$ is commutative and can be mathematically expressed as follows:

$$f \ast g = g \ast f$$

Associative Property

Convolution is not restricted to two given functions, and it is absolutely possible to convolve three or more functions. As per the associative property, to convolve three functions, first convolve any two functions to produce an intermediate function and then convolve the intermediate function with the remaining function. The same concept can be extended to four or more functions. As an example, the mathematical expression for convolving three functions namely $f$, $g$, and $h$ can be expressed as follows:

$$(f \ast g) \ast h = f \ast (g \ast h)$$

Distributive Property

The operation of convolution is distributive over the operation of addition. That is, for given three functions namely $f$, $g$, and $h$ the following relationship holds:

$$f \ast (g + h) = f \ast g + f \ast h$$

Convolution has a tremendous application in various fields and is quite heavily used in the field of Digital Image Processing and Digital Signal Processing. Convolution also finds its application in the field of computational geometry and can be used as a tool to compute the guard zone of a given simple polygon.

![Convolution Diagram]

**Figure 2.2:** Convolution between a straight line and circle of radius $r$.

The convolution between a simple polygon and a circle of radius $r$ (the distance between the given input simple polygon and the computed guard zone) gives the desired
guard zone, but the complexity (or running time) of such operation is tremendous since while computing the guard zone a circle of radius $r$ needs to be drawn at every possible point on the given input polygon. Even though the running time of the convolution operation between a circle and the given input polygon is made such operation highly unlikely to be implemented in practical scenarios but its effectiveness and simplicity remains undoubted. In Figure 2.2, an example of convolution between a circle of radius $r$ and a straight line has been shown.

### 2.3 Minkowski Sum

The *Minkowski sum* or *Minkowski addition* is named after the famous mathematician Hermann Minkowski. Nowadays, the Minkowski sum is an integral part of most commonly used image processing systems, where it is used as a tool in 2D (brush-and-stroke paradigm) and 3D (solid sweep) computer graphics and also plays a central role in mathematical morphology for years. Minkowski sum also finds its application in other areas like motion planning, numerical control machining, computational geometry to name only a few. For example, a popular approach to motion planning for polygonal robots in a room with polygonal obstacles fattens each of the obstacles by taking the Minkowski sum of them with the shape of the robot. This reduces the problem to moving a point from the start to the goal using a standard shortest path algorithm. In order to maintain focused discussion, we like to limit ourselves to the study and application of this tool in the area of computational geometry only, after discussing the definition and the various properties of this important operation.

Given two polygons $P$ and $Q$ in $R^2$, their Minkowski sum 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 $p+q = (p_x+q_x, p_y+q_y)$.

The above definition of Minkowski sum holds good for any dimension and is not restricted to two-dimension only. Before we shift our focus regarding the various mathematical properties of this important operation, we like to explain this phenomenon with the help of an example below. Consider two sets of position vectors denoted by $A$ and $B$ in $R^2$, representing the vertices of two triangles with coordinates, $A = \{(1,0), (0,1), \ldots\}$ and $B = \{(1,1), (0,0), \ldots\}$.
Let us also try to visualize the concept of Minkowski sum with the help of Figure 2.3. The Minkowski sum of a pair of triangles (for the example above) is shown in Figure 2.3(a), and that between a square and a triangle is depicted in Figure 2.3(b).

2.3.1 Properties of Minkowski Sum

The Minkowski sum has a number of mathematical properties, and few such important mathematical properties have been discussed as follows.
Commutative property

The order in which the Minkowski sum is calculated between two polygonal objects is not restricted and hence given two polygonal objects, the Minkowski sum yields the same result irrespective of the order of the objects being considered. Therefore, the Minkowski sum operation is \textit{commutative} and can be mathematically expressed as follows, where \( A \) and \( B \) are the input polygonal objects:

\[ A \oplus B = B \oplus A \]

Associative property

Minkowski sum is not restricted to two inputs, and it is absolutely possible to compute the Minkowski sum of three or more polygonal objects. As per the associative property, to compute the Minkowski sum of three inputs, first obtain the Minkowski sum of any two inputs to produce an intermediate result and then compute the Minkowski sum of the intermediate polygonal object and the remaining input. The same concept can be extended to four or more inputs without loss of generality. As an example, the mathematical expression for computing the Minkowski sum between three polygonal objects as input, namely, \( A \), \( B \), and \( C \) can be expressed as follows:

\[ (A \oplus B) \oplus C = A \oplus (B \oplus C) \]

Minkowski sum of convex sets results in a convex set

This is a very important property of Minkowski sum operation and hence given two polygons \( P \) and \( Q \), the Minkowski sum of \( P \) and \( Q \) yields a convex polygon.

Minkowski sum and convex hull operations are commutative

In real vector space, if we consider two subsets \( S_1 \) and \( S_2 \), the convex hull (Conv) of their Minkowski sum is the Minkowski sum of their convex hull (Conv) [32], which can be mathematically expressed as follows:

\[ \text{Conv}(S_1 \oplus S_2) = \text{Conv}(S_1) \oplus \text{Conv}(S_2) \]

The above observation can also be extended in more generic sense for finite collection of non-empty sets and hence can be expressed as follows:
Conv(∑Sn) = ∑Conv(Sn), where ‘∑’ denotes the Minkowski sum and n denotes the total number of non-empty sets.

The complexity of computing Minkowski sum of two arbitrary simple polygons P and Q is $O(m^2n^2)$ [29], where m and n are the numbers of vertices of the given two polygons, respectively. In particular, if one of the two polygons is convex, the complexity of Minkowski sum reduces to $O(mn)$. However, if both P and Q are convex, then $P \oplus Q$ is a convex polygon with at most $m+n$ vertices, and can be computed in $O(m+n)$ time [5].

In [19], an algorithm of time complexity $O(mn \log (mn))$ is proposed in the context of the polygon containment problem. The studied problem is to decide whether a convex polygon Q can be translated to fit within an arbitrary polygon P. In [27], numerous results are proposed on the Minkowski sum problem when one of the polygons is monotone.

For two arbitrary polygons P and Q, where P is monotone and Q is convex, the size of the Minkowski sum as well as the time and space complexities of the algorithm are $O(mn)$. In case both the given input polygons, P and Q are monotone, the size of the Minkowski sum is $O(mn a(\min(m, n)))$, where $a(\cdot)$ is the inverse of the Ackermann function. The time complexity of the proposed algorithm is $O(mn \log (mn))$. In a situation where P is an arbitrary simple polygon and Q is monotone, the time complexity of the proposed algorithm is $O((k + mn) \log (mn))$, where k is the size of the Minkowski sum. The worst case value of k may be $O(m^2n)$.

An algorithm for finding the outer face of the Minkowski sum of two simple polygons is presented in [14]. It uses the concept of convolution, and the running time of the algorithm is $O((k + (m + n)\sqrt{l}) \log_2 (m + n))$. Here 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.

2.3.2 Relationship between Minkowski Sum and Convolution

Given two planar, curved objects $O_1$ and $O_2$, if both $O_1$ and $O_2$ are convex objects, the convolution curve $O_1*O_2$ is exactly the same as the Minkowski sum boundary $\partial(O_1 \oplus O_2)$. However, $\partial(O_1 \oplus O_2)$ is a subset of $O_1*O_2$ in general.
To construct $\partial(O_1 \oplus O_2)$, we need to execute the following two steps:

- Compute the convolution curve $O_1 \ast O_2$, and
- Eliminate the redundant parts of $O_1 \ast O_2$, which do not contribute to $\partial(O_1 \oplus O_2)$.

The Minkowski sum computation of given two polygons using the method of convolution is being discussed in [25, 26], whereas Ramkumar [66] used this approach to devise an efficient algorithm for computing the outer boundary of the Minkowski sum of two polygons. An efficient implementation details of computing Minkowski sum using the method of convolution for given two polygons is being discussed in [75], along with the experimental results, where the proposed algorithm computes the Minkowski sum in three steps as stated below:

**Step 1:** Computation of the cycles of the convolution $P \ast Q$, where $P$ and $Q$ are the given inputs.

**Step 2:** Construction of the planar arrangement induced by the segments that comprise the convolution cycles.

**Step 3:** Extraction of the Minkowski sum from the arrangements obtained in previous steps.

Clearly, the Minkowski sum can also be used to compute the guard zone of a given simple polygon. Minkowski sum between a line and a point with same x- and y-coordinate creates a line parallel to the given line and hence can be used to draw lines parallel to the input polygonal edges. More specifically, the Minkowski sum between a circle of radius $r$ (the guard zonal distance) and the given input polygon yields the desired solution but one has to deal with the removal of redundant portion(s) from the computation, and in fact, this could be a bit complex while obtaining the final desired guard zone. Moreover, to the best of our knowledge, no algorithm exists which can compute the boundary defined by the Minkowski sum of an arbitrary simple polygon and a circle or a regular convex polygon in time linear in the worst case size of output (combinatorial complexity) of the problem. As discussed earlier, the complexity of computing Minkowski sum depends on the shape of the polygons. The situation gets worse when both the input polygons are non-convex, and
the size of the sum can be $O(m^2n^2)$. It is worthwhile also to highlight the flip side of Minkowski sum, where holes can be either created or destroyed [38].

### 2.4 Triangulation

As per the dictionary definition, triangulation has been described as “A trigonometric method of determining the position of a fixed point from the angles to it from two fixed points a known distance apart; useful in navigation”. Triangulation can also be considered as a method of surveying, where a given area is divided into triangles, and the length of one side and its angles with the other two are measured, and then the lengths of the other sides can be calculated. The word “triangulation” has various definitions based on the context of the word has been used. For example, the word triangulation has its own definitions in the field of psychology, social science, and political science. However, to continue our discussion we prefix the word “polygon” with the word “triangulation” and hence in subsequent paragraphs we limit ourselves in discussions regarding *polygon triangulation* only.

In the field of computational geometry, polygon triangulation is the decomposition of a simple polygonal area into a set of triangles and is being considered as one of the most primitive operation in the field.

Polygons are the most effective way so far, to represent the boundary of two-dimensional objects in computer systems. However, undoubtedly a polygon can also be complex in terms of both increased number of vertices and presence of holes. To process such a complex polygon, it is primitive to decompose such polygons into comparatively simple polygons and more specifically into primitive polygons like triangles.

Before we proceed with the discussion of various algorithms that exists for the triangulation of a simple polygon, let us now discuss some key properties of triangulation.

- **For simple polygons a triangulation always exists**

A simple polygon with $n$ vertices can always be triangulated, and the proof is discussed in [5].
- **Triangulation of a simple \( n \)-vertex polygon has exactly \( n - 2 \) triangles**

The triangulation process of a simple \( n \)-vertex polygon always results in \( n - 2 \) triangles and the proof is also discussed in [5].

- **A triangulated simple polygon can always be 3-coloured**

To understand this property of triangulation, we like to introduce the discussion regarding the art gallery problem; the question that arises: How many cameras are required to guard a given gallery and the decision of where to place them?

![Triangulated polygon with 3-colouring](image)

**Figure 2.4:** A triangulated simple polygon with 3-colouring.

In general, the gallery is represented by its floor-plan and not by actual 3-dimensional representation for the ease of computer modeling and this representation is sufficient to decide regarding the placement of the cameras. It is being studied that placing cameras at each vertex yield the optimum result since once the given polygon (represented as the floor-plan for the gallery) is triangulated, a vertex can be incident to many triangles and a camera at that vertex guards all of them. Now this approach is being mapped to graph colouring problem as discussed below.

Now consider for a given simple polygon \( P \), its triangulation is represented as \( T_P \). Therefore, to place a camera one need to select a subset of the vertices of \( P \) such that any
triangle in $T_p$ has one selected vertex and need to place the cameras at the selected vertices. One can find such a subset of $P$ by assigning a unique colour to each of the vertices of $P$, governed by the rule saying no two vertices will have the same colour that is connected by a direct edge or a diagonal. Towards this goal, if someone assigns a unique colour Red, Blue, and Green to each vertex of $P$ by following the rule described above, is called a 3-colouring of a triangulated polygon. Therefore, in a 3-colouring triangulated polygon; each triangle has a Red, Blue, and a Green vertex. Now if someone chooses the smallest colour class to place the cameras the gallery represented by $P$ can be guarded using at most $\lfloor n/3 \rfloor$ cameras. The proof has been discussed in [5]. The same concept has been illustrated with the help of Figure 2.4.

In this figure, a triangulated simple polygon with 3-colouring has been shown. It is worth to highlight that the triangulation shown in this figure is one of many possibilities of the way the diagonals can be considered among the vertices of the given polygon. Even though different arrangements of the diagonals result in different triangulated polygons for the same input polygon but all such triangulated polygons satisfy the properties of triangulation. For example, in Figure 2.4, the input polygon has 12 vertices namely, $A$ through $L$, and the triangulated polygon contains $12-2 = 10$ triangles. Once we have performed the 3-colouring of the triangulated polygon it results in five Blue vertices, four Red vertices, and three Green vertices, and hence for the art gallery problem the cameras need to be placed on the Green vertices, which also satisfies the count for cameras, $\lceil 12/3 \rceil \geq 3$ (the number of Green coloured vertices).

A number of algorithms have been devised for the triangulation of a simple polygon. On a high-level the triangulation algorithm can be broadly classified as follows [42]:

(a) Algorithms based on insertion of diagonals.

(b) Algorithms based on Delaunay triangulation.

(c) Algorithms using Steiner points.

Selection of any particular triangulation algorithm for a given input simple polygon is not only driven by the running time of the algorithm but also depends on the quality of
triangle the algorithm produces. The quality of triangle is determined by the interior angle of the triangles produced by the triangulation process and the triangles with larger interior angle considered to be of better quality compared to the ones having smaller interior angles. Figure 2.5 demonstrates the quality of triangles produced by various classes of triangulation algorithm.

![Figure 2.5](image)

**Figure 2.5:** Outputs of various classes of triangulation algorithm. (a) Low-quality triangulation. (b) High-quality triangulation. (c) Triangulation with Steiner points.

It has been studied in [43] that the Delaunay triangulation produces better quality triangles (see Figure 2.5(b)) and the use of Steiner points (in Figure 2.5(c)) improves the quality of triangulation significantly.

### 2.4.1 Study of Triangulation Algorithms based on Insertion of Diagonals

This is probably the simplest approach for obtaining triangulation of a simple polygon. This approach is based on the fact that every simple polygon can be divided into $n-2$ triangles by introducing $n-3$ diagonals. One of the simplest algorithms proposed by Lennes in the year 1911 [46], which works recursively by inserting diagonals between a pair of vertices of the given input polygon with an asymptotic running time of $O(n^2)$, where $n$ is the total number of vertices of the given input polygon. In the year 1975, Meisters [57] has proposed a triangulation algorithm based on ‘ear’ searching method and then cutting them off, with an asymptotic running time of $O(n^3)$. An ‘ear’ is defined as a triangle whose two
sides belong to the edges of the input polygon $P$ and the third one completely resides within $P$. In 1990, ElGindy, Everett, and Toussaint [17] have discovered that prune and search technique finds an ‘ear’ in linear time and hence using the technique the asymptotic running time of Meisters’ algorithm reduces to $O(n^2)$.

Garey, Johnson, Preparata, and Tarjan [22] proposed a divide-and-conquer based algorithm that solves the triangulation problem in $O(n \log n)$ time. Their proposed algorithm follows a two-step approach where the first step decomposes the simple polygon into monotone sub-polygons in $O(n \log n)$ time and during the second step the algorithm triangulates the sub-polygons in linear time.

Chazelle has also proposed a different divide-and-conquer based approach for the triangulation process, and the algorithm also takes $O(n \log n)$ time [9]. Next, an improvement in speed was achieved by a variety of randomized algorithms with a running time of $O(n \log^* n)$. The most well-known randomized version for triangulation was suggested by Seidel [70], in the year 1991. On a high-level, the Seidel’s algorithm can be described as follows:

**Step 1:** Trapezoidal decomposition of input polygon.

**Step 2:** Determination of monotone polygon’s chains.

**Step 3:** Triangulation of monotone polygon’s chain found in Step2.

Hertel and Mehlhorn [28] proposed a sweep-line based triangulation algorithm that runs in $O(n + r \log r)$ time, where $r$ is the number of concave vertices in the input polygon. Therefore, the algorithm performs well when there is less number of concave vertices present in the input polygon.

The shape of the input polygon really plays a vital role in an overall running time of the triangulation process. Chazelle and Incerpí also proposed an algorithm that runs in $O(n \log s)$ time where $s < n$ [9], and $s$ denotes the sinuosity of the polygon representing how many times the polygon’s boundary alternates between complete spirals of opposite orientation.

Toussaint proposed an adaptive version of Triangulation algorithm [74], whose worst case running time is $O(n^2)$. However, also proposed a numerous enhancements for
special classes of polygons, for which the algorithm runs in \( O(n(1 + t_0)) \), \( t_0 < n \), where \( t_0 \) is the number of triangles contained in the triangulation obtained that share zero edges with the input polygon.

Another new triangulation approach based on Graham’s scan has been introduced by Kong, Everett, and Toussaint in [41]. Although the worst case running time of their algorithm is still \( O(n^2) \), the average case running time has been expressed as \( O(kn) \), where \( (k - 1) \) is the number of concave vertices present in the input polygon.

In the year 1991, Chazelle has done a tremendous breakthrough in the history of triangulation computation by introducing an algorithm whose worst case running time is \( O(n) \) [10]. To achieve the triangulation process for a given input polygon in linear time, Chazelle has introduced the concept of visibility map, a structure that can be considered as the generalization of trapezoidation. The proposed algorithm uses the concept of divide-and-conquer, where the input polygon of \( n \) vertices is partitioned into chains with \( n/2 \) vertices, and these again into chains of \( n/4 \) vertices, and so on. Finally, the visibility map is formed by combining the maps of its sub-chains. Conceptually speaking, this approach generally takes \( O(n \log n) \) time, but Chazelle has smartly improvised it by dividing the step into two phases.

The first phase includes computing coarse approximations of the visibility maps in such a way that the merging process can be accomplished in linear time. The second phase also gets executed in linear time where the coarse map is fine tuned to obtain a complete visibility map. Finally, the algorithm produces the desired triangulation from the trapezoidation defined by the visibility map. The algorithm described in his paper is far more detailed and really complex to understand. However, this does not dilute his achievement in solving the triangulation problem in linear time. However, it is also important to highlight that, as per best of our knowledge there exists no implementation of Chazelle’s linear time triangulation algorithm as of today [33].

2.4.2 Study of Triangulation Algorithms based on Delaunay Triangulation

Delaunay Triangulation (DT) named after famous Russian scientist Boris Nikolayevich Delaunay. He proposed his idea during 1934 in [16]. The Delaunay triangulation for a set
of points $P$, on a plane is a triangulation $DT(P)$ such that no point in $P$ is inside the circumcircle of any triangle in $DT(P)$ [31], where the circumcircle of a polygon is a circle that passes through all the vertices of the polygon. There exists a more generalized version of Delaunay triangulation known as Constrained Delaunay Triangulation (CDT) and is being considered for situations where the output demands certain segments to be part of the triangulation process. For example, while triangulating a simple polygon it is absolutely necessary to include the polygonal edges in the output and hence while triangulating a simple polygon we need to use CDT instead of DT. The most important aspect of DT is that it maximizes the minimum angle of all the angles of the triangles obtained after triangulation by avoiding the generation of any skinny triangles. The DT has its own set of properties (a brief on various properties supported by DT can be found in [43]) but to maintain the focused discussion we skipped such micro-level details and concentrated on various algorithms proposed so far to obtain CDT for simple polygons only.

The CDT of a simple polygon can be obtained by first calculating the CDT by including all the edges of the given simple polygon and then by removing the triangles that are in the exterior of the given simple polygon. In [47], Lewis and Robinson have proposed a divide-and-conquer based approach to obtain the CDT of a given simple polygon. The proposed algorithm first decomposes the given simple polygon $P$ into near equal sized sub-polygons by recursively subdividing the boundary of $P$, and then each such sub-polygon is separately triangulated in parallel. Finally, the results are combined to produce the required CDT. The proposed algorithm runs in $O(n^2)$ time, where $n$ is the number of polygonal vertices.

De Floriani, Falcidieno, and Pienovi introduced a recursive algorithm [15] to compute CDT for a given simple polygon. The algorithm first computes the visibility graph $G_{vis}(P)$ of the vertices of the given simple polygon $P$ and also computes the Voronoi diagram $Vor(P)$ of the vertices of $P$ and then obtain the resulting CDT by joining each vertex $v$ of $P$ to the vertices which are both visible (in $G_{vis}(P)$) from $v$ and also Voronoi neighbours of $v$. To compute the visibility graph the algorithm takes $O(n^2)$ time and to compute the Voronoi diagram the algorithm takes $O(n \log n)$ time, and hence the overall worst case running time of the algorithm is $O(n^2)$. 
In [43], Lee and Lin have proposed an $O(n \log n)$ time algorithm, where to obtain the CDT for a given simple polygon $P$, the algorithm first subdivides $P$ into two sub-polygons $P_l$ and $P_r$ and then recursively computes CDT $T_l$ and $T_r$. Finally, the resulting triangulation is obtained by merging $T_l$ and $T_r$.

### 2.4.3 Study of Triangulation Algorithms with Steiner Points

The introduction of Steiner points in the triangulation process guarantees the quality of triangles produced. The quality of triangles is measured in terms of the minimum interior angle of the output triangles. The first such algorithm was proposed by Chew in [12], where he extended the idea of Delaunay triangulation to obtain a mesh from the given simple polygon. In this article, he also claimed that the triangles produced by his algorithm guarantees to have angles between $30^\circ$ and $120^\circ$ and all edge lengths are between $h$ and $2h$, where $h$ is the input parameter chosen by the user. The worst case running time of Chew’s algorithm is $O(n^2)$, where $n$ is the number of triangles in the final triangulation.

In [67], Ruppert has extended the idea proposed by Chew, and developed an algorithm that produces the output triangles with angles within $\Pi - 2\alpha$, where $\alpha$ is the input parameter for which selection range can vary anywhere between $0^\circ$ and $20^\circ$. The worst case running of the algorithm proposed by Ruppert is $O(n^2)$, where $n$ is the size of the output.

There also exists a family of algorithms which guarantees shape of the output triangles without using the concept of Delaunay triangulation directly or indirectly. Such algorithms use special data structures like grids or quad trees and especially more difficult from an implementation point of view. Such algorithms have been elucidated by Baker, Grosse, and Rafferty in [4] and by Bern, Eppstein, and Gilbert in [6].

There exists several other triangulation algorithms and researchers are constantly refining those, but all such discussions are beyond the scope of this thesis.

### 2.4.4 Relationship between Triangulation and Minkowski Sum

Minkowski sum for given two simple polygons can be obtained by first triangulating both the simple polygons and afterwards performing the union operation among the triangles
obtained in the first step. However, it is also being studied by Flato in [18] that the use of triangulation to decompose the given simple polygon to obtain the Minkowski sum slows down the overall performance. Similar to the triangulation, union of polygons is another widely researched area under computational geometry, but details of such algorithms are beyond the scope of this thesis.

The original problem statement addressed in this thesis is based on the publication by Nandy, Bhattacharya, and Hernández-Barrera [60], where the problem name was coined as safety zone problem, that has been described as follows: “Given a simple polygon $P$ and a fixed parameter $\delta$, the safety zone (of width $\delta$) of the polygon $P$ is a closed region $S$ of minimum area such that $P$ is completely inside $S$ and there exists no pair of points $p$ and $q$, where $p$ is on the boundary of $P$ and $q$ is on the boundary of $S$, such that $d(p,q)$, the Euclidean distance between $p$ and $q$, is less than $\delta$”.

In this very publication [60], the trio has also proposed a linear time algorithm to compute the safety zone (SZ) of the given simple polygon. They also have highlighted that the SZ of a given simple polygon $P$ can be obtained from the medial axis of $P$ in linear time, where they have given the reference to the algorithm proposed by Chin, Snoeyink, and Wang [14] and the algorithm proposed by Klein and Lingas [40]. Now as per their conclusion both the algorithms to obtain the medial axis of a given simple polygon is quite complex, and hence the complexity increases while computing the safety zone of the simple polygon.

The important claims highlighted in their publication related to safety zone computation are briefly summarized as follows:

(a) The use of medial axis computation as part of safety zone computation is difficult to understand and hence more difficult to implement in software programs, but even though their algorithm uses Chazelle’s linear time triangulation algorithm as the first step, and after that it uses linear linked list and binary tree data structures to compute the SZ, and hence comparatively much easier to tackle in software programs compared to that of medial axis approach.
(b) Due to the use of Chazelle’s algorithm the overall worst case time and space complexity of SZ computation reduces to $O(n)$, where $n$ is the number of vertices of the given input polygon.

Now to understand the algorithm proposed in [60], it is important to appreciate the concept of a notch, which is most important in terms of the overall computation of SZ and the entire algorithm focuses on how to handle the scenario in presence of a notch, since the SZ of a simple polygon can be easily obtained in absence of notch(s). As per their definition, “a notch is a polygonal region outside a polygon $P$, which is formed with a chain of edges of $P$ initiating and terminating at two vertices of a false hull edge (if there exists any). Clearly, the area $\text{CH}(P) - P$ may consists of one or more number of disjoint notches outside the polygon $P$, where CH($P$) denotes the convex hull of $P$ and false hull edge refers to the hull edge introduced while computing the CH($P$), which does not coincide with any polygonal edges”. On the other hand, the solid hull edge refers to the hull edge that coincides with a polygonal edge.

The overall algorithm is divided into two major steps:

(1) The processing of polygonal regions other than the notch(s).

(2) Processing of notch(s).

As the first step, the algorithm computes the convex hull of the given simple polygon and then draws the line parallel to the polygonal edges at a distance $\delta$ outside the polygon and draws circular arcs of radius $\delta$ for each convex vertex, which are considered as the safe boundaries of the given polygon and then the algorithm connect those safe boundaries.

For the second step, we focus our discussion in processing a single notch, and the same procedure is applicable for all the notches present as part of the input polygon. The algorithm first triangulate the notch using Chazelle’s linear time triangulation algorithm and constructs a directed tree structure termed as triangulation tree (TT), where each node of the graph represents the triangles created using Chazelle’s triangulation algorithm and an edge between any two nodes indicates that the triangles corresponding to these nodes share a common triangulation edge. The root node of the tree corresponds to the triangle
adjacent to a *false hull edge*. The direction of the tree structure is assigned by traversing the edges in a depth-first manner.

In order to draw the safe boundary of a notch, the algorithm traverses the nodes of the TT in post-order (*Left-Right-Root*) fashion. Since, the subtrees of a node are traversed before the processing of a node; hence, once the node processing is over, the safe boundaries corresponding to a triangle is drawn.

The most crucial step during the notch processing is to check for intersections with various safe boundaries that are drawn. To facilitate such intersection detection, the algorithm uses a separate doubly connected linked list named “*visibility list*” for each edge of the TT. The algorithm uses this information smartly, by checking only selected set of already drawn safe boundaries inside any triangle for possible intersections. After processing each node within the tree, the algorithm propagates the information higher up the ladder so that the cumulative information from subtrees can be used for intersection detection while processing the current node in the tree.

The other data structure maintained by the algorithm is as follows:

- An array of vertices and edges of a notch stored in clockwise order.

To maintain the continuity and for the completeness of this discussion, the high-level algorithm proposed in [60] to compute the safe boundary of a given simple polygon has been included below.

**The Algorithm:** Computation of the safe boundary of a given simple polygon.

**Input:** A simple polygon \( P \).

**Output:** The safe boundary of the input polygon \( P \).

**Begin**

**Step 1:** Compute the convex hull of \( P \).

**Step 2:** For each notch do

**Step 2.1:** Construct the triangulation tree \( T \), by triangulating the notch using Chazelle’s linear time triangulation algorithm.
**Step 2.2:** Recursively traverse $T$ in postorder fashion and process each triangle in $T$, which includes the creation of the visibility list $L$ for each triangulation edge and is being used for intersection detection during the processing of each triangle in $T$.

**Step 3:** Draw the safe boundaries for the *solid hull edges* and the *hull vertices* of the convex hull.

// This essentially constructs the safe boundary for the polygonal edges and vertices other than those belong to a notch. //</br>

**Step 4:** Connect the safe boundary of each *solid hull edge* to the safe boundary of its attached hull vertices, obtained in Step 3.

**Step 5:** For each notch do

**Step 5.1:** Merge each end of the visibility list $L$ of that notch to the safe boundary of the hull vertex attached to its *false hull edge*.

**Step 6:** Traverse the visibility list $L$ from any element to output the safe boundary of the polygon.

**End**

**Computational complexity**

The overall time and space complexity of the proposed algorithm is $O(n)$, where $n$ is the total number of vertices in the input polygon $P$. This has been concluded based on the following artifacts:

1. As described in [44], the convex hull for an $n$-vertex simple polygon can be drawn in $O(n)$ time.

2. The triangulation step requires $O(n)$ time due to the use of Chazelle’s linear time triangulation algorithm [10].

3. It has been established and proved in [60] that the time required to draw the safe boundary of a notch having $n$ vertices is $O(n)$ using the triangulation tree and visibility list data structure.
(4) In the worst case, the time required to draw the safe boundaries of all solid hull edges and the hull vertices is $O(n)$.

(5) Finally, based on the input polygon it may be necessary to concatenate the safe boundary of a notch to the safe boundaries of some other notch or some solid hull edge that are attached to the two vertices of its corresponding false hull edge. This requires $O(K)$ time, where $K$ is the number of notches present in the input polygon [60].

Therefore, the overall running time of the algorithm proposed in [60] is linear in terms of the total number of input vertices in the given simple polygon.

- **Couple of important observations regarding the algorithm proposed in [60]**

1) The proposed algorithm uses Chazelle’s linear time triangulation algorithm, which significantly contributes to the overall time complexity achieved by the algorithm. Without disregarding the novelty of the algorithm, it is also very important to highlight that to the best of our knowledge the Chazelle’s linear time triangulation algorithm is not only difficult to understand but also extremely complex to implement in software programs, and that is the fundamental reason that till date there is no software program in the market that has implemented Chazelle’s triangulation algorithm [33]. Moreover, Chazelle’s algorithm also does not guarantee the quality of triangulation. The complication regarding the implementation of Chazelle’s algorithm is also highlighted in [60].

2) For the practical implementation purpose, the use of any other triangulation algorithm other than the Chazelle’s one increases the overall time complexity of the proposed algorithm from $O(n)$ to $O(n \log n)$.

**2.5 Summary**

The overall study discussed in this chapter has provided a basis for the development of our algorithms. The algorithms proposed in this thesis are unique in the sense that none of them uses the triangulation mechanism as a tool to compute the safety zone (or guard zone) of a
given simple polygon. Moreover, the approach discussed in subsequent sections is comparatively easy to understand and implement in software modules. Again, we have studied the real-life requirement of the safety zone in case 3D objects and accordingly developed algorithms for the construction of guard zone for a 3D object. In Chapter 3, a number of algorithms have been devised to compute the guard zone of a given simple polygon in 2D space. Chapter 4 introduces the parallel version of one of the algorithms proposed in Chapter 3. Chapter 5 has extended the idea of guard zone computation in 3D space, and Chapter 6 discusses the parallel version of the algorithm developed in Chapter 5.