1
Background

1.1 Notion of Visibility

Visibility is a natural phenomenon in everyday life. We see objects around us and then decide our movement accordingly. Seeing an object means identifying the portions of the object visible from the current position of an observer. The entire object may not be visible as some of its parts may be hidden from the observer. The observer also determines shapes and sizes of visible portions of an object. Visible portions of an object change as the observer moves from one position to another. Moreover, the observer may see several objects in different directions from its current position; the visible portions of these objects form the scene around the observer. Constructing such a scene continuously is very natural for a human observer as the human visual system can execute such tasks effortlessly.

Suppose a robot wants to move from a starting position to a target position without colliding with any object or obstacle around it. The robot constructs the scene around itself from its current position and then guides its motion in the free space lying between itself and the visible portion of the objects around it. The positions of the robot and the objects can be represented in the computer of the robot by their x, y and z co-ordinates and therefore, the scene consisting of visible portions of these objects can be computed for the current position of the robot. The problem of computing visible portions of given objects from a viewpoint has been studied extensively in computer graphics [115]. Since the scene is constructed from thousands of objects of different shapes and sizes lying in different positions, it becomes a complex task from a computational point of view. Even computing visible portions of one object in the presence of several other objects is a non-trivial task. Moreover, computing such a scene for every position on the path of the robot is a time-consuming task even for high-speed computers. Designing efficient algorithms for executing such movements of a robot in the presence of obstacles in a reasonable period of time is one of the objectives in the field of robot path planning [226].

The above problem of robot path planning can be reduced to the corresponding problem in two dimensions. If a mobile robot that maintains contact with the floor is projected on the floor, the two-dimensional footprint of the robot can be modeled as a polygon. Similar projections on the floor can now also be produced for all obstacles. This process yields a map consisting of polygons in two dimensions. The polygon corresponding to the robot can be navigated using this map by avoiding collisions with polygonal obstacles. Thus a collision-free path of the robot can be computed from its starting position to the target position. While navigating, the visible portions of polygonal obstacles are computed to construct the scene around the current position of the robot. Although such a representation in two dimensions has reduced the complexity of the robot path planning problem, designing efficient algorithms for such computations remains a challenging task.

The notion of visibility has also been used extensively in the context of the art gallery problem in computational geometry [271, 310, 333]. The art gallery problem is to determine the number of guards that are sufficient to see every point in the interior of an art gallery room. This means that every interior point of the room must be visible to one of the guards so that all paintings in the gallery remain guarded. There are many theorems and algorithms for the minimization of the number of guards and their placement in the art gallery room.

The study of visibility started way back in 1913 when Brunn [67] proved a theorem regarding the *kernel* of a set. Today visibility is used in many fields of computer science including robotics [66, 226, 249, 283], computer vision [139, 168] and computer graphics [93, 113, 115].

1.2 Polygon

A *polygon* P is defined as a closed region R in the plane bounded by a finite set of line segments (called *edges* of P) such that there exists a path between any two points of R which does not intersect any edge of P. Any endpoint of an edge of P is called a *vertex* of P, which is a point in the plane. Since P is a closed and bounded region, the boundary of P consists of cycles of edges of P, where two consecutive edges in a cycle share a vertex. If the boundary of P consists of two or more cycles, then P is called a *polygon with holes* (see Figure 1.1(a)). Otherwise, P is called a *simple polygon* or a *polygon without holes* (see Figure 1.1(b)). The region R is called the *internal region* or *interior* of P. Similarly, the regions of the plane excluding all points of R are called the *external regions* or *exterior* of P. A vertex of P is called *convex* if the interior angle at the vertex formed by two edges of that vertex is at most π; otherwise it is called *reflex*. Note that the interior angle at a vertex always faces the interior of P.

As defined above, a simple polygon P is a region of the plane bounded by a cycle of edges such that any pair of non-consecutive edges do not intersect. In this book,

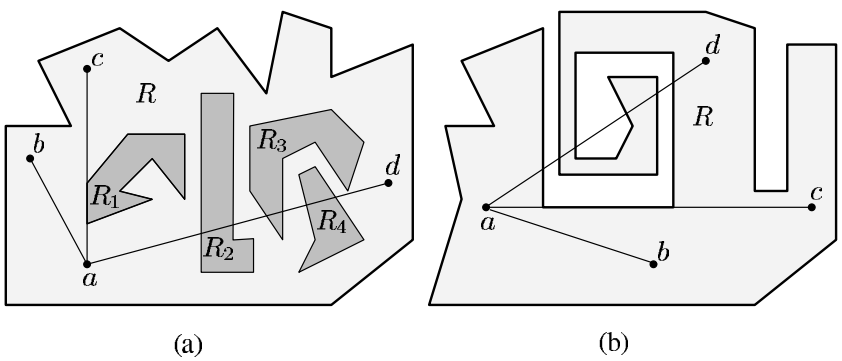

Figure 1.1 (a) In this polygon with holes, b and c are visible from a but not d. (b) In this simple polygon (or a polygon without holes), b and c are visible from a, but not d.

we assume that P is given as a doubly linked list of vertices. Each vertex has two fields containing x and y co-ordinates of the vertex and it has two pointers pointing to the next clockwise and counterclockwise vertices of P. It can be seen that if edges of P are traversed in counterclockwise (or clockwise) order, then the interior of P always lies to the left (respectively, right) of the edges of P.

Let $c_0, c_1, c_2, \ldots, c_h$ be the cycles on the boundary of a polygon P with h holes, where c_0 represents the outer boundary of P. Let R_j denote the region of the plane enclosed by c_j for all $j \leq h$ (see Figure 1.1(a)). Since P is a closed and bounded region, $R_j \subset R_0$ for all $j > 0$. Moreover, $R_j \cap R_k = \emptyset$ where $k \neq j$ and $k > 0$. Therefore, $R = R_0 - (R_1 \cup R_2 \cup \ldots \cup R_h)$. Observe that if P is a simple polygon, then $R = R_0$ as the boundary of P consists of only one cycle c_0 (see Figure 1.1(b)). In this book, we assume that a polygon P with h holes is given in the form of h cycles, where vertices of each cycle are stored in a doubly linked list as stated above and there is an additional pointer to one vertex of each cycle of P to access that cycle.

Two points p and q in P are said to be *visible* if the line segment joining p and q contains no point on the exterior of P. This means that the segment pq lies totally inside P. This definition allows the segment pq to pass through a reflex vertex or graze along a polygonal edge. We also say that p *sees* q if p and q are visible in P. It is obvious that if p sees q, q also sees p. So, we sometime say that p and q are mutually visible. In Figure 1.1, the point a sees two points b and c, but not the point d.

Exercise 1.2.1 *Given two points p and q inside a polygon P, design a method to determine whether p and q are visible in P.*

Suppose a set of line segments in the plane is given such that they do not form a polygon. Let A denote the arrangement of these line segments in the plane (see

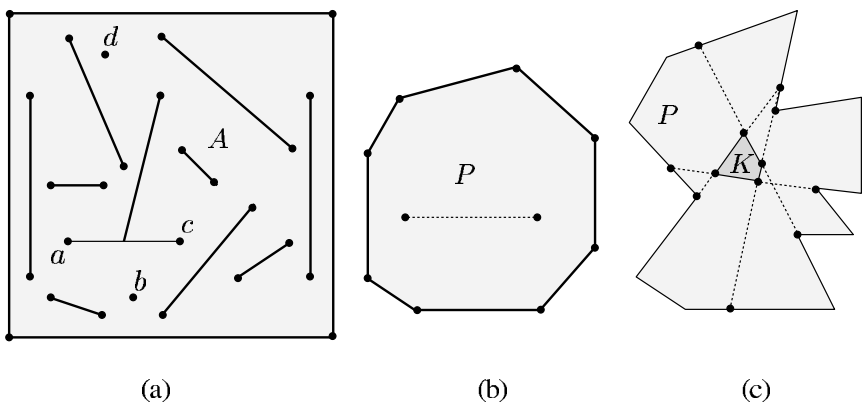

Figure 1.2 (a) In this arrangement of line segments A, points b and c are visible from a but not d. (b) A convex polygon P where any two points are mutually visible. (c) A star-shaped polygon P where the entire polygon is visible from any point of the kernel K.

Figure 1.2(a)). Two points p and q in the plane are said to be *visible* in the presence of A if the line segment joining p and q does not cross any line segment in A. This definition permits the segment pq to touch a line segment of A. In Figure 1.2, a point a sees two points b and c, but not the point d.

Using the definition of visibility in a polygon, we define two special classes of simple polygons called convex and star-shaped polygons. A simple polygon P is called *convex* if every pair of points in P is mutually visible [176] (see Figure 1.2(b)). It can be seen that the internal angle at every vertex of a convex polygon is at most π [327]. A convex polygon can also be defined as intersections of closed half-planes which is bounded. A simple polygon P is said to be *star-shaped* if there exists a point z inside P such that all points of P are visible from z (see Figure 1.2(c)). The set of all such points z of P is called the *kernel* of P. The kernel of P is always convex [67]. If P is a star-shaped polygon with respect to a point z, it can be seen that the order of vertices on the boundary of P is same as the angular order of vertices around z. We refer to this property by saying that the vertices of P are in *sorted angular order* around z. It follows from the theorem of Krasnosel'skii [223] that a simple polygon P is star-shaped if and only if every triple of convex vertices is visible from some point of P [334]. Note that a convex polygon is also a star-shaped polygon and all points of the convex polygon belong to the kernel.

Exercise 1.2.2 *Prove that a polygon P is star-shaped if and only if every triple of convex vertices is visible from some point of P [223].*

Exercise 1.2.3 *Prove that the kernel of a star-shaped polygon is convex.*

Exercise 1.2.4 *Draw two star-shaped polygons A and B such that each edge of A intersects every edge of B [291].*

1.3 Asymptotic Complexity

The time and space complexity of the sequential algorithms presented in this book are measured using the standard notation $O(f(n))$, where n is the size of the input to the algorithm. The notation $O(f(n))$ denotes the set of all functions $g(n)$ such that there exist positive constants c and n_0 with $|g(n)| \leq c|f(n)|$ for all $n \geq n_0$. We say that an algorithm runs in polynomial time if the running time of the algorithm is $O(n^k)$ for some constant k. The notation $\Omega(f(n))$ denotes the set of all functions $g(n)$ such that there exist positive constants c and n_0 with $g(n) \geq cf(n)$ for all $n \geq n_0$.

The idea of evaluating asymptotic efficiency of an algorithm is to know how the running time (or space) of the algorithm increases with the size of the input. The running time expressed using O notation gives a simple characterization of the efficiency of the algorithm, which in turn allows us to compare the efficiency of one algorithm with another. For more discussion on asymptotic efficiency of algorithms, see the book by Cormen *et al.* [96].

The *real* RAM (*Random Access Machine*) has become the standard model of computation for sequential algorithms in computational geometry. As stated in [291], the real RAM is a random access machine with infinite precision and real number arithmetic. The real RAM can be used to perform addition, subtraction, multiplication, division and comparisons on real numbers in unit time. In addition, various other operations such as indirect addressing of memory (integer address only), computing the intersection of two lines, computing the distance between two points, testing whether a vertex is convex are also available. These operations are assumed to take constant time for execution. For more details of these operations, see O'Rourke [272].

Exercise 1.3.1 *Is $O(2^n) = O(2^{O(n)})$?*

Exercise 1.3.2 *Given a point z and a polygon P, design an $O(n)$ time algorithm to test whether z lies in the interior of P [291].*

All parallel algorithms mentioned in this book (at the end of chapters) are designed for the *Parallel Random Access Machine* (PRAM) model of computations [35, 172, 211]. This can be viewed as the parallel analog of the sequential RAM. A PRAM consists of several independent sequential processors, each with its own private memory, communicating with one another through a global memory. In one unit

of time, each processor can read one global or local memory location. PRAMs can be classified according to restrictions on global memory access. An *Exclusive-Read Exclusive-Write* (or EREW) PRAM is a PRAM for which simultaneous access to any memory location by different processors is forbidden for both reading and writing. In *Concurrent-Read Exclusive-Write* (or CREW) PRAM, simultaneous reads are allowed but not simultaneous writes. A *Concurrent-Read Concurrent-Write* (or CRCW) PRAM allows simultaneous reads and writes. PRAM models of computation allow for infinite precision real arithmetic, with all simple unary and binary operations being computable in $O(1)$ time by a single processor.

We say that a parallel algorithm in the PRAM model of computations runs in *polylogarithmic* time if it runs in $O(\log^k n)$ time using $O(n^m)$ processors, where k and m are constants and n is the size of the input to the algorithm. A problem is said to be in the class NC if it can be solved in polylogarithmic time using a polynomial number of processors. A parallel algorithm is called *optimal* if the product of the running time of a parallel algorithm and the number of processors used by the parallel algorithm is within a constant factor of the best sequential algorithm for the same problem.

1.4 Triangulation

In this section, we provide a brief overview of the results on triangulation of a polygon as there are visibility algorithms that depend on a first stage of computing a triangulation of the input polygon. A *triangulation* of a polygon P is a partition of P into triangles by diagonals (see Figure 1.3), where a line segment joining any two mutually visible vertices of P is called a *diagonal* of P [242]. Note that if a line segment joining two vertices u and v of P passes through another vertex w of P and the segment uv lies inside P, then uw and vw are diagonals and not uv.

Exercise 1.4.1 *Prove that every simple polygon admits triangulation [272].*

Exercise 1.4.2 *Using the proof of Exercise 1.4.1, design an $O(n^2)$ time algorithm for triangulating a simple polygon of n vertices [272].*

It can be seen that a triangulation of P is not unique as many subsets of diagonals give triangulations of the same polygon. The dual of a triangulation of P is a graph where every triangle is represented as a node of the graph and two nodes are connected by an arc in the graph if and only if their corresponding triangles share a diagonal (see Figure 1.3). Since there are three sides of a triangle, the degree of every node in the dual graph is at most three. A graph with no cycle is called a *tree*. In the following lemmas, we state some of the properties of triangulations of P.

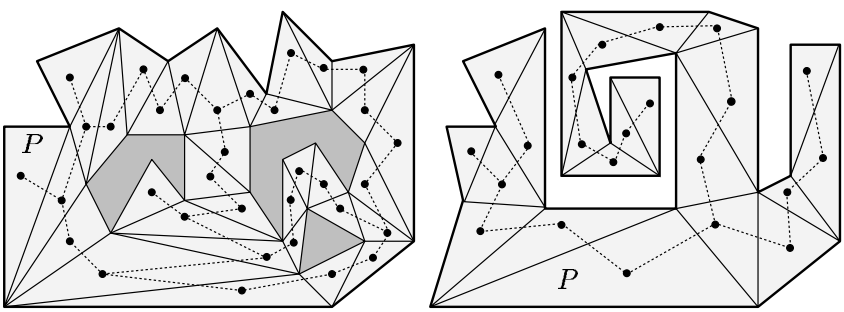

Figure 1.3 (a) A triangulation of a polygon with holes and its dual graph. (b) A triangulation of a simple polygon and its dual tree.

Lemma 1.4.1 *Every triangulation of a simple polygon of n vertices uses $n-3$ diagonals and has $n-2$ triangles.*

Corollary 1.4.2 *The sum of the internal angles of a simple polygon of n vertices is $(n-2)\pi$.*

Lemma 1.4.3 *Every triangulation of a polygon with h holes with a total of n vertices uses $n+3h-3$ diagonals and has $n+2h-2$ triangles.*

Lemma 1.4.4 *The dual graph of a triangulation of a simple polygon is a tree.*

Lemma 1.4.5 *The dual graph of a triangulation of a polygon with holes must have a cycle.*

Exercise 1.4.3 *Prove that there is no cycle in the dual graph of a triangulation of a simple polygon.*

Exercise 1.4.4 *Let a graph G of m vertices denote the dual of a triangulation of a polygon with holes. Design an $O(m)$ time algorithm to locate a cycle in G.*

The first $O(n \log n)$ time algorithm for triangulating a simple polygon P was given by Garey *et al.* [148]. The first step of their algorithm is to partition P into *y-monotone* polygons. A simple polygon is called *y-monotone* if its boundary can be divided into two chains of vertices such that each chain has vertices with increasing y-coordinates. The partition of P into y-monotone polygons can be done in $O(n \log n)$ time by the algorithm of Lee and Preparata [233] for locating a point in a given set of regions. It has been shown by Garey *et al.* that each y-monotone polygon can be triangulated in a time that is proportional to the number of vertices

of the y-monotone polygon. So, the overall time complexity of the algorithm for triangulating P is $O(n \log n)$. This algorithm also works for polygons with holes, as pointed out by Asano *et al.* [29], and it is optimal for this class of polygons.

Another $O(n \log n)$ time algorithm for triangulating a simple polygon was presented by Mehlhorn [257] and uses the *plane sweep* technique. This algorithm was generalized for a polygon with holes by Ghosh and Mount [165] with the same time complexity (see Section 5.3.2). Later, Bar-Yehuda and Chazelle [43] gave an $O(n + h \log^{1+\epsilon} h)$, $\epsilon > 0$ time algorithm for triangulating a polygon with h holes with a total of n vertices.

Many researchers worked for more than a decade on the problem of triangulating a simple polygon P in less than $O(n \log n)$ time. One approach was to consider special classes of simple polygons that could be triangulated in $O(n)$ time [48, 131, 142, 162, 183, 239, 280, 331, 342]. Another approach was to find algorithms whose running time was based on structural properties of simple polygons [78, 193]. Tarjan and Van Wyk [326] were the first to establish an improvement by proposing an $O(n \log \log n)$ time algorithm for this problem. Later, a simpler $O(n \log \log n)$ time algorithm was presented by Kirkpatrick *et al.* [217]. Finally, an $O(n)$ time optimal algorithm for this problem was presented by Chazelle [71] settling this long-standing open problem. We have the following theorem.

Theorem 1.4.6 *A simple polygon P of n vertices can be triangulated in $O(n)$ time.*

The algorithm of Chazelle [71] uses involved tools and notions such as a *planar separator theorem*, *polygon cutting theorem* and *conformality*. Although this algorithm does not use any complex data structure, it is conceptually difficult and too complex to be considered practical. Moreover, although it has been used as a preprocessing step for many of the visibility algorithms presented in this book, the development of a simple $O(n)$ time algorithm for triangulating a simple polygon remains an open problem.

1.5 The Art Gallery Problem

As stated in Section 1.1, the art gallery problem is to determine the number of guards that are sufficient to see every point in the interior of an art gallery room. The art gallery can be viewed as a polygon P of n vertices and the guards are stationary points in P. A point $z \in P$ is visible from a guard g if the line segment zg lies inside P. If guards are placed at vertices of P, they are called *vertex guards*. If guards are placed at any point of P, they are called *point guards*. Since guards placed at points or vertices are stationary, they are referred as *stationary guards*. If guards are mobile along a segment inside P, they are referred as *mobile guards*. If mobile guards move along edges of P, they are referred as *edge guards*.

Exercise 1.5.1 *Draw a simple polygon of $3k$ vertices for $k > 1$ showing that k stationary guards are necessary to see the entire polygon [91].*

In a conference in 1976, V. Klee first posed the art gallery problem (see [198]). Chavátal [91] showed that for a simple polygon P, $\lfloor n/3 \rfloor$ stationary guards are always sufficient and occasionally necessary to see or guard the entire P. Later, Fisk [141] gave a simple proof for this bound. Using this proof, Avis and Toussaint [41] designed an $O(n \log n)$ time algorithm for positioning guards at vertices of P. For mobile guards, O'Rourke [270] showed that $\lfloor n/4 \rfloor$ mobile guards are always sufficient and occasionally necessary. For edge guards, $\lfloor n/4 \rfloor$ edge guards appear to be sufficient, except for some types of polygons (see [333]).

Exercise 1.5.2 *Let P be a triangulated simple polygon of n vertices. Design an $O(n)$ time algorithm for positioning at most $\lfloor n/3 \rfloor$ stationary guards at vertices of P such the entire P is visible for these guards [41, 141].*

A polygon is said to be *rectilinear* if its edges are aligned with a pair of orthogonal coordinate axes. For a simple rectilinear polygon P where edges of P are horizontal or vertical, Kahn *et al.* [207] showed that $\lfloor n/4 \rfloor$ stationary guards are always sufficient and occasionally necessary to guard P. An alternative proof for this bound was given later by O'Rourke [269]. These proofs first partition P into convex quadrilaterals and then $\lfloor n/4 \rfloor$ guards are placed in P. A convex quadrilaterization of P can be obtained by using the algorithms of Edelsbrunner *et al.* [121], Lubiw [250], Sack [299] and Sack and Toussaint [301]. For mobile guards in rectilinear polygons P, Aggarwal [11] proved that $\lfloor (3n+4)/16 \rfloor$ mobile guards are always sufficient and occasionally necessary to guard P. Bjorling-Sachs [55] showed later that this bound also holds for edge guards in rectilinear polygons.

Exercise 1.5.3 *Let P be a triangulated simple polygon of n vertices. Design an $O(n)$ time algorithm for partitioning P into convex quadrilaterals [250, 301].*

For a polygon P with h holes, O'Rourke [271] showed that P can always be guarded by at most $\lfloor (n+2h)/3 \rfloor$ vertex guards. For point guards, Hoffmann *et al.* [194] and Bjorling-Sachs and Souvaine [56] proved independently that $\lceil (n+h)/3 \rceil$ point guards are always sufficient and occasionally necessary to guard P. Bjorling-Sachs and Souvaine also presented an $O(n^2)$ time algorithm for positioning guards in P. No tight bound is known on the number of mobile guards sufficient

for guarding P. However, since $\lceil (n+h)/3 \rceil$ point guards are sufficient for guarding P, the bound obviously holds for mobile guards. For an rectilinear polygon P with h holes, Györi *et al.* [182] showed that $\lfloor (3n+4h+4)/16 \rfloor$ mobile guards are always sufficient and occasionally necessary for guarding P. For more details on art gallery theorems and algorithms, see O'Rourke [271], Shermer [310] and Urrutia [333]. We do not cover this subfield of visibility in this book.

The minimum guard problem is to locate the minimum number of guards for guarding a polygon with or without holes. O'Rourke and Supowit [276] proved that the minimum point, vertex and edge guard problems are NP-hard in polygons with holes. Even for simple polygons, these problems are NP-hard as shown by Lee and Lin [231].

There are approximation algorithms for these NP-hard problems. Ghosh [152] presented approximation algorithms for minimum vertex and edge guard problems for polygons P with or without holes. The approximation algorithms run in $O(n^5 \log n)$ time and yield solutions that can be at most $O(\log n)$ times the optimal solution. This means that the approximation ratio of these algorithms is $O(\log n)$. These algorithms partition the polygonal region into convex pieces and construct sets consisting of these convex pieces. Then the algorithms use an approximation algorithm for the minimum set-covering problem on these constructed sets to compute the solution for the minimum vertex and edge guard problems in P. Recently, Ghosh [158] has improved the running time of these approximation algorithms by improving the upper bound on the number of convex pieces in P. After improvement, the approximation algorithms run in $O(n^4)$ time for simple polygons and $O(n^5)$ time for polygons with holes.

Efrat and Har-Peled [122] also gave approximation algorithms for the minimum vertex guard problem in polygons with or without holes. Let c_{opt} denote the number of vertices in the optimal solution. Their approximation algorithm for simple polygons runs in $O(nc_{opt}^2 \log^4 n)$ time and the approximation ratio is $O(\log c_{opt})$. Their other approximation algorithm is for polygons with holes, which runs in $O(nhc_{opt}^3 polylog\ n)$ time, where h is the number of holes in the polygon. The approximation ratio is $O(\log n \log(c_{opt} \log n))$. For the minimum point guard problem in simple polygons, they gave an exact algorithm which runs in $O((nc_{opt})^{3(c_{opt}+1)})$ time.

Observe that in the worst case, c_{opt} can be a fraction of n. So, the approximation ratio of approximation algorithms of Ghosh [152, 158] and Efrat and Har-Peled [122] is $O(\log n)$ in the worst case. On the other hand, Eidenbenz [123, 124] showed that the problems of minimum vertex, point and edge guards in simple polygons are APX-hard. This implies that there exists a constant $\epsilon > 0$ such that no polynomial time approximation algorithm for these problems can guarantee an approximation ratio of $1+\epsilon$ unless $P = NP$.