Visibility queries in a polygonal region

Rajasekhar Inkulu\textsuperscript{a,*}, Sanjiv Kapoor\textsuperscript{b}

\textsuperscript{a} Institute for Computational Engineering and Sciences, The University of Texas at Austin, USA
\textsuperscript{b} Department of Computer Science, Illinois Institute of Technology, Chicago, USA

\textbf{A R T I C L E I N F O}

\textit{Article history:}
Received 3 August 2007
Received in revised form 27 January 2009
Accepted 25 February 2009
Available online 4 March 2009
Communicated by K. Mehlhorn

\textbf{A B S T R A C T}

In this paper, we consider the problem of computing the region visible to a query point located in a given polygonal domain. The polygonal domain is specified by a simple polygon with \( m \) holes and a total of \( n \) vertices. We provide two bounds on the complexity of this problem. One approach constructs a data structure with space complexity \( O(n^2) \) in time \( O(n^2 \log n) \) and yields a query time of \( O((1 + \min(m, |V(q)|)) \log n + m + |V(q)|) \). Here, \( V(q) \) represents the set of vertices of the visibility polygon of a query point \( q \), and \(|E|\) denotes the number of edges in the visibility graph. The other approach provides a data structure with space complexity \( O(\min(|E|, mn) + n) \) in time \( O(T + |E| + n \log n) \) with the query time of \( O(|V(q)| \log n + m) \). Here, \( T \) is the time to triangulate the given polygonal region (which is \( O(n + m \log^{1+\epsilon} m) \) for a small positive constant \( \epsilon > 0 \)). In both of these approaches, \( O(m) \) additive factor in the query time is eliminated with an additional \( O((\min(|E|, mn))^2) \) space and an additional \( O(m(\min(|E|, mn))^2) \) preprocessing time.

\textcopyright 2009 Elsevier B.V. All rights reserved.

1. Introduction

Consider a polygonal region, \( P \), defined by a simple polygon with \( m \) holes, where each hole is defined by a simple polygon. Two points inside a polygonal region are visible from each other if their connecting line segment remains completely inside the polygon and does not intersect the holes. The \textit{Visibility Polygon} \( V(q) \) of a point \( q \) in \( P \) is defined as the polygonal boundary of the set of points in \( P \) that are visible from \( q \). The \textit{Visibility Polygon Query} problem is to design a data structure for \( P \) that, given \( q \), reports \( V(q) \).

For a simple polygon with no holes, Bose, Lubiw, and Munro [5] compute the visibility polygon \( V(q) \) of a given query point \( q \) in time \( O(\log n + |V(q)|) \) with \( O(n^2 \log n) \) preprocessing time and \( O(n^2) \) space. Also, the same complexities were achieved in Guibas and Raghavan [4]. Later Aronov, Guibas, Teichmann, and Zhang [1] proposed an algorithm which accomplishes the same with the preprocessing time \( O(n^2 \log n) \), space \( O(n^2) \) and query time complexity as \( O(\log^2 n + |V(q)|) \).

For a polygon with holes where there is no query involved, worst-case optimal algorithms for constructing the visibility polygon with total time of \( O(n \log n) \) were presented by Asano [2] and, later by Suri and O’Rourke [12]. This was later improved to \( O(n + m \log m) \) by Heffernan and Mitchell [6]. This problem in the query version was first presented by Pocchiola and Vegter [11]. They have considered the case of a set of convex polygons in the plane and given an algorithm which determines the query polygon \( V(q) \) of any query point \( q \) in time \( O(|V(q)| \log n) \) by \( O(n \log n) \) preprocessing time and \( O(n) \) space. Zarei and Ghodsi [13] considered the case of a polygon (not necessarily convex) with holes and gave an algorithm that finds \( V(q) \) with \( O(n^2 \log n) \) preprocessing time and \( O(n^2) \) space having query complexity \( O((1 + m') \log n + |V(q)|) \), where \( m' = \min(m, |V(q)|) \).

\* Corresponding author.

E-mail addresses: rinkulu@ices.utexas.edu (R. Inkulu), kapoor@iit.edu (S. Kapoor).

0925-7721/$ – see front matter \textcopyright 2009 Elsevier B.V. All rights reserved.

doi:10.1016/j.comgeo.2009.02.004
We provide a method which covers the space with simple polygons and utilizes a sub-procedure for computing the visibility polygon from a point inside a simple polygon. The computation of visibility queries in a simple polygon has been well researched, and we use either of the following two algorithms as a sub-procedure in our algorithm: one given by Aronov, Guibas, Teichmann, Zhang [1], and the other that uses ray-shooting by Hershberger and Suri [7]. Using the approach from [1] as a sub-procedure, we construct a data structure with space complexity \(O(n^2)\) in time \(O(n^2 \log n)\) so that the query complexity is \(O((1 + \min(m, |V(q)|)) lg^2 n + m + |V(q)|)\). Here, \(V(q)\) represents the vertices of the visibility polygon of a query point \(q\), and \(|E|\) is the number of edges in the visibility graph. Using the ray-shooting based approach as a sub-procedure, we construct a data structure with space complexity \(O((\min(|E|, mn) + n) lg n\) in time \(O(T + |E| + n \log n)\) which yields a query time of \(O(|V(q)| \log n + m)\). Here, \(O(T)\) is the time to triangulate the given polygonal region \((O(n + m \log^2 + m)\) for a small positive constant \(\epsilon > 0\) using the algorithm given by Bar-Yehuda and Chazelle [3]. The preprocessing time and space of our algorithm using either of these sub-procedures improves upon [13]. When \(|V(q)| \geq m\), our algorithm with the first approach provides a query complexity of \(O(m \log^2 n + |V(q)|)\) close to the \(O(m \log n + |V(q)|)\) query time achieved by Zarei and Ghodsi [13]. Our algorithm is especially useful when the number of holes is a small constant. Moreover, in both of these approaches, the \(O(m)\) additive factor in the query time is eliminated altogether with an additional \(O((\min(|E|, mn))^2)\) preprocessing time. Table 1 summarizes the results.

### Table 1

<table>
<thead>
<tr>
<th></th>
<th>Space</th>
<th>Preprocessing time</th>
<th>Query time</th>
</tr>
</thead>
<tbody>
<tr>
<td>No extra preprocessing</td>
<td>(O(n^2))</td>
<td>(O(n^2 \log n))</td>
<td>(O((1 + \min(m,</td>
</tr>
<tr>
<td>With extra preprocessing</td>
<td>(O(\min(</td>
<td>E</td>
<td>, mn) + n))</td>
</tr>
<tr>
<td>Zarei and Ghodsi [13]</td>
<td>(O(n^2))</td>
<td>(O(n^2 \log n))</td>
<td>(O((1 + \min(m,</td>
</tr>
</tbody>
</table>

1.1. Overview

The algorithm uses a partition of the space into simple polygons, known as corridors, similar to that in [8,10]. The visibility polygon of \(q\) is determined in two phases. In the first phase, we find the sides of corridors that contain at least one visible point. And in the second phase, we find the visible vertices in each such corridor.

When the query point \(q\) is inside the corridor \(C\), we apply the algorithm from either [1] or [7] (both determine the visibility inside a simple polygon) to determine the visible region interior to \(C\). To facilitate in determining the visible region for each corridor \(C'\) when \(q\) is external to \(C\), corridor \(C'\) and the region outside \(C'\) are pre-processed to construct a set of simple polygons (refer to Section 3.4). During the query time, algorithm from either [1] or [7] is applied on these simple polygons.

Determining the sides of corridors that contain at least one visible vertex could be done by scanning the corridors, which would require considering each of \(O(m)\) corridors. A more effective procedure is adopted which traverses relevant corridors and maintains a potentially visible region, that becomes more and more restricted as the query algorithm proceeds. (See Fig. 1.) Initially the region is constructed using the sides of the corridor containing the query point \(q\). This requires locating the corridor containing the query point and constructing supporting lines to the sides of the corridors. To accomplish this, pre-processing is required to set up a planar point location structure. The traversal of relevant corridors is determined by constructing a tree of corridors, each having at least one vertex visible from \(q\) (refer to Section 3.5). Achieving the mentioned query time requires a pre-processed data structure, which contains visible supporting lines from each vertex in the polygonal region to the sides of the corridors (refer to Section 3.2). These supporting lines are constructed during pre-processing using the visibility graph construction from Kapoor and Maheshwari [9].

The rest of the paper is organized as follows: Section 2 lists properties and definitions. The data structures used and the preprocessing phase are described in Section 3. The query algorithm to construct the visibility polygon is detailed in Section 4. Section 5 analyzes algorithm for both the time and space complexities. The proof of correctness is given in Section 6. Section 7 provides the conclusions.

2. Properties and definitions

First, we describe the corridor structures used in this paper. These are a slight variation to the corridors defined in [8,10]. Consider a triangulation of the given polygonal region, \(P\), and the dual graph \(G_D\) formed from the triangulation. The dual graph is first pruned by iteratively removing vertices of degree one and the edge incident on each such vertex. The subset of edges removed forms a subgraph \(H\), which is a forest. The graph \(G_D \setminus H\) has nodes which are of degree two or three. The decomposition into corridors is obtained as follows: identify triangles in \(P\) corresponding to vertices of degree three in...
Consider two vertices \( u_1 \) and \( u_2 \), which are visible from each other in the given polygonal region. We define the **dual distance** from \( u_1 \) to \( u_2 \) as the minimum number of triangle edges (in the triangulation of the polygonal region) intersected by the line segment \( u_1u_2 \).

Throughout the paper, we assume that the angles at a point are measured in the counter-clockwise direction from the positive horizontal direction.

Consider a (convex) cone \( cp \) defined with two rays \( pp_1 \) and \( pp_2 \). For a ray \( pp' = pp_1 \), if \( cp \) is the region obtained by angularly sweeping \( pp' \) in the counter-clockwise direction until \( pp' = pp_2 \), then \( cp \) is denoted with an ordered tuple \([pp_1, pp_2]\). The cone \( cp \) is called a **visibility cone** if it contains at least one point visible from \( p \).

A **corridor sequence** is defined as a sequence of corridors \( SQ = [C_1, C_2, \ldots, C_k] \) such that for any two consecutive corridors \( C_i, C_{i+1} \) in \( SQ \), there exists a junction \( J \) adjacent to both the corridors \( C_i \) and \( C_{i+1} \). For two points \( p, p' \) visible from each other, let the corridor sequence \( SQ \) be defined such that for every corridor \( C', C' \in SQ \) iff \( C' \cap pp' \neq \emptyset \). \( SQ \) is termed as the **corridor sequence of** \( pp' \).

Let \( SQ_1 \) and \( SQ_2 \) be the corridor sequences of two line segments \( l_1 \) and \( l_2 \). Also, let \( SQ \) be the longest common prefix of both \( SQ_1 \) and \( SQ_2 \). Then the last corridor in \( SQ \) is termed as the **last common corridor of** \( l_1 \) and \( l_2 \).

A corridor \( C' \) in a corridor sequence \( SQ \) is called **explored** if and only if the algorithm has already considered \( C' \) in traversing \( SQ \). Alternatively, \( C' \) is called **unexplored**. A sequence of unexplored corridors in a corridor sequence are termed as an **unexplored corridor sequence**.

Let \( p_1 \) and \( p_2 \) be the end points of a bounding edge \( b \) of a corridor \( C' \). Suppose \( b \) has a visible point from \( q \). If the angles of the line segments \( p_1q \) and \( p_2q \), made at \( q \), are not equal, then suppose w.l.o.g. that the angle \( p_1q \) makes is less than the angle \( p_2q \) makes at \( q \). Otherwise, let \( \|ap_1\| \leq \|ap_2\| \). Then the corridor side on which \( p_1 \) resides is known as the **left side of corridor** \( C' \) w.r.t. \( q \) and \( b \), whereas the other side is known as the **right side of corridor** \( C' \) w.r.t. \( q \) and \( b \). Note that when \( q \) and \( b \) are not explicitly mentioned, they are understood from the context. Also, let \( p \) be a visible point from \( q \) located on a corridor side of \( C' \) such that the line segment \( pq \) intersects \( b \). If \( p \) and \( p_1 \) belong to the same side, then we say that \( p \) is **located on the left side of corridor** \( C' \) w.r.t. \( q \) and \( b \). Similarly, if \( p \) and \( p_2 \) belong to the same side, then we say that \( p \) is **located on the right side of corridor** \( C' \) w.r.t. \( q \) and \( b \).

A line \( l \) is said to **support** a set of points \( P \) if and only if all the points in \( P \) belong to the same unique (closed) half-plane defined by \( l \).

3. **Data structures**

This section describes all the data structures constructed during both the preprocessing and query processing stages of the algorithm.
3.1. Visibility trees

Here we briefly describe the visibility tree data structure from [8–10]. Let \( v \) be a vertex in a corridor \( C \). Also, let \( U = \{u_1, u_2\} \) and \( B = \{b_1, b_2\} \) be the two bounding edges of the corridor \( C \).

The sets \( \text{VIS}_B(v) \) and \( \text{VIS}_U(v) \) are defined herewith. For any vertex \( v' \notin C, v' \in \text{VIS}_B(v) \) if and only if \( v' \) is visible from \( v \) and the line segment \( vv' \) intersects \( B \). For any vertex \( v \in C, v' \in \text{VIS}_B(v) \) if and only if \( v' \) is visible from \( v \) and the dual distance from \( b_1 \) (or \( b_2 \)) to \( v' \) is less than or equal to the dual distance from \( b_1 \) (or \( b_2 \)) to \( v \). A vertex \( v' \in \text{VIS}_U(v) \) if and only if \( v' \) is visible from \( v \) and \( v' \notin \text{VIS}_B(v) \). The method in this paper organizes the set of vertices in \( \text{VIS}_B(v) \) (resp. \( \text{VIS}_U(v) \)) in a tree called visibility tree \( \text{TVIS}_B(v) \) (resp. \( \text{TVIS}_U(v) \)). (See Fig. 2.) Two lists \( \text{LIST}_1(C) \) and \( \text{LIST}_2(C) \) are stored at the root \( r \) of \( \text{TVIS}_B(v) \). The list \( \text{LIST}_1(C) \) stores the vertices in \( \text{VIS}_B(v) \) which belong to left side of corridor \( C \) in order of their increasing dual distance from \( v \). Similarly, \( \text{LIST}_2(C) \) corresponds to the right side of the corridor \( C \). The corridor \( C' \) at an immediate child of root \( r \) is chosen so that the closest vertex visible to \( v \) through \( B \) is a vertex of \( C' \). Since a vertex can belong to at most two corridors, the root can have at most two sons. Among these two corridors, the corridor with a visibility vertex \( v'' \) such that \( vv'' \) makes the least angle at \( v \) is represented by the left son. This relationship of sons of a node in the tree to their parent node is repeated recursively.

The corridor \( C_p \) at node \( p \) may not be adjacent to the corridor \( C' \) associated with the parent of \( p \). In this case there is a unique sequence of corridors connecting \( C' \) to \( C_p \). Suppose each visibility edge is labeled by the sequence of corridors it intersects. This labeling partitions the visibility edges so that the edges that intersect the same sequence of corridors belong to the same partition. At every node \( p \) of every TVIS structure, the vertices stored in all the \( \text{LIST} \) structures at \( p \) correspond to the same partition. For two points \( p' \) and \( p'' \) on the sides of a corridor \( C' \) so that \( p' \) and \( p'' \) are visible from \( v \), the corridor sequence of \( v p' \) may not be same as the corridor sequence of \( v p'' \). Hence, a corridor may be associated with more than one node in a TVIS. However, each vertex in the polygonal region appears only once in the whole tree.

Let \( S' \) be the ordered set consisting of vertices of a partition which belongs to left side of corridor \( C_p \) in order of their increasing dual distance from \( v \). Similarly, \( S'' \) corresponds to visible vertices residing on the other side of the corridor \( C_p \) which belong to the same partition. The list \( \text{LIST}_1(C_p) \) at a non-root node \( p \) of \( \text{TVIS}_B(v) \) corresponding to that partition stores \( S' \), whereas the list \( \text{LIST}_2(C_p) \) at \( p \) stores \( S'' \). In general, a node \( p \) in the tree \( \text{TVIS}_B(v) \) is associated with a corridor \( C_p \), a bounding edge of \( C_p \), and at most two lists, \( \text{LIST}_1(C_p) \) and \( \text{LIST}_2(C_p) \). For a vertex \( v' \in C' \) stored at node \( p \), the bounding edge stored at node \( p \) is the bounding edge of \( C' \) which intersects the line segment \( vv' \). Symmetric description applies to
TVIS_U(v). Since the definition of a corridor in our paper is slightly different from [9], TVIS_U(v), TVIS_B(v), LIST_1(C_p), LIST_2(C_p) data structures are modified by merging LIST structures where appropriate.

3.2. Supporting line lists

We first construct data structures TVIS_U(v) and TVIS_B(v) for each vertex v in the given polygonal region using the algorithm given in Kapoor et al. [9]. As part of the preprocessing, for every vertex v of the given polygonal region, we find a supporting line l from v to the set of vertices in each list L located at each non-root node of TVIS_U(v) and TVIS_B(v). (See Fig. 3.) This we do by a linear scan of such lists L. Suppose vertex v is in corridor C', and C' has B and U as its bounding edges. The supporting lines from v to the left and right sides of the corridors that intersect U are stored in lists LIST_LU_v and LIST_RB_v respectively. Also, the supporting lines to the left and right sides of the corridors that intersect B are stored in lists LIST_LB_v and LIST_RB_v respectively. Each vertex v' in these lists is associated with a node p in either TVIS_U(v) or TVIS_B(v) such that the list at node p contains v'. The supporting lines so found are ordered by non-decreasing counter-clockwise angle at vertex v, and are stored in data structures that facilitate binary search. After building these lists, we delete the lists LIST_1 and LIST_2 stored at each of the nodes of both TVIS_B(v) and TVIS_U(v).

3.3. Hulls of corridor sides

For the detailed descriptions of open and closed corridors, refer to [8,10]. For every open corridor C, we construct a convex hull corresponding to each side of C. For every closed corridor C, we compute the funnels of C where each side of every funnel is a convex hull. These hulls facilitate in finding the supporting lines to sides of C from a point external to C.

3.4. Four simple polygons per corridor

For each open corridor (or, for each funnel of each closed corridor) C in the polygonal region, we create four simple polygons corresponding to it. Let B_01 and B_02 be the boundaries of obstacles O_1 and O_2 respectively. Let the corridor C have sides S_1 and S_2 such that B_01 \cap S_1 = S_1 and B_02 \cap S_2 = S_2. Also, let the bounding edges of C be U = [u_1, u_2] and B = [b_1, b_2], where u_1, b_1 are incident on S_1 and u_2, b_2 are incident on S_2. Suppose the two lines one which contains the line segments U and the other that contains the line segment B intersect at I such that u_1 and b_1 are closer to I (the case in which u_2 and b_2 are closer to I is symmetric). Let u'_2 and b'_1 be the extreme points on rays u_1I and b_1I respectively, so that both of these points incident on the obstacle O_1. Similarly, let u'_2 and b'_2 be the extreme points on rays u_2I and b_2I respectively, so that both of these points incident on the obstacle O_2. Let BB' be the bounding box enclosing the given polygonal region and I (ignoring the degenerate case in which I is at infinity). Let u be the point of intersection of ray u_1u_2 with the bounding box BB. Let b be the point of intersection of ray b_1b_2 with the bounding box BB. Also, let I' be the point of intersection of ray u_2u_1 with the bounding box BB.

We define below four simple polygons denoted by P_1(S_1), P_2(S_1), P_3(S_2), and P_4(C). For a query point q located external to the corridor C, we find the points on side S_1 which are visible from q using both P_1(S_1) and P_2(S_1), whereas the
points on side $S_2$ which are visible from $q$ are determined using $P_3(S_2)$. And, $P_4(C)$ is used to determine the points on sides of corridor $C$ which are visible from a query point $q$ located in corridor $C$.

- The simple polygon $P_1(S_1)$ is guaranteed to include all the points $p$ external to corridor $C$ such that there exist a point on $S_1$ which is visible from $p$. The simple polygon $P_1(S_1)$ is defined by the sequence of edges (i) $bb_1$, (ii) the edges of side $S_1$, (iii) contiguous bounding edges of obstacle $O_1$ which join $u_1$ with $u'_1$ excluding the edges of $S_1$, (iv) the edge $u'_1l'$, and (v) the edges of the bounding box $BB$ from $l'$ to $b$ chosen to enclose the corridor $C$ (see Fig. 4(a)).

- The simple polygon $P_2(S_2)$ is guaranteed to include all the points $p$ external to corridor $C$ such that there exist a point on $S_2$ which is visible from $p$. The simple polygon $P_2(S_2)$ is defined by the sequence of edges: (i) $bb'_2$, (ii) contiguous bounding edges of obstacle $O_2$ which join $b'_2$ with $b_2$ excluding the edges of $S_2$, (iii) the edges of side $S_2$, (iv) contiguous bounding edges of obstacle $O_2$ which join $u_2$ with $u'_2$ excluding the edges of $S_2$, and (v) the edge $u'_2u$ and the edges of the bounding box $BB$ from $u$ to $b$ chosen to enclose the corridor $C$ (see Fig. 4(c)).

- The simple polygon $P_4(C)$ is guaranteed to include all the points $p$ internal to corridor $C$. Since every $C$ is a simple polygon, $P_4(C)$ is defined as the corridor $C$ itself (see Fig. 4(d)).

3.5. Trees $T_B(q)$ and $T_U(q)$

For a given query point $q$ in a corridor $C$ whose bounding edges are $B$ and $U$, we build at query time two binary trees $T_B(q)$ and $T_U(q)$ using the preprocessed data structures. These auxiliary structures organize the corridor sides which have at least one visible point from $q$ so that the phase of determining the visibility polygon is nicely segregated into an
independent module. A node \(v\) in either of these trees refers to a corridor \(C\) with one or both sides of \(C\) marked for further processing. The root of the tree \(T_B(q)\) always refers to the corridor containing \(q\). Consider a side \(S\) referred from a non-root node \(v\) of \(T_B(q)\) (resp. \(T_U(q)\)) which is marked. Such a side, \(S\), has at least one vertex \(p\) visible from the query point \(q\) such that the visible ray \(qp\) intersects \(B\) (resp. \(U\)). Also, with each marked side in either \(T_B(q)\) or \(T_U(q)\), a visibility cone is associated.

3.6. Stack \(ST\)

During the query processing, in order to construct the trees \(T_B(q)\) and \(T_U(q)\), we maintain yet to be processed objects in a stack, termed \(ST\). Each of these objects is a tuple, \([t_1, t_2, lptr, rptr, vc]\). Here \(t_1\) (resp. \(t_2\)) is a tangent from a vertex \(p'\) in an explored corridor to a point of tangency \(p''\) incident on a left (resp. right) side of a corridor. The variable \(lptr\) (resp. \(rptr\)) refers to the first unexplored corridor in the corridor sequence of \(t_1\) (resp. \(t_2\)). The visibility cone \(vc\) assists in finding the sides of corridors which have vertices visible from \(q\).

4. Query processing

We outline the query processing algorithm first. Given a query point \(q\) located in a corridor \(C\) with bounding sides \(U\) and \(B\), the set of points visible from \(q\) are obtained in two phases. The first phase deals with the corridor \(C\) and is detailed in Section 4.1. The second phase handles the corridors other than \(C\) and is described in Section 4.2.

The root node of \(T_B(q)\) is associated to corridor \(C\) and the appropriate visibility cone. This facilitates in finding the points located on the sides of \(C\) which are visible from \(q\). At most two objects are inserted to the stack \(ST\) initially: one corresponds to the visibility cone \([qp', qp'']\) having the maximum angle, with both the rays \(qp'\) and \(qp''\) of it intersecting \(U\); and another visibility cone intersecting \(B\).

The general outline of our method is as follows: for an object \(obj\) extracted from the stack \(ST\), we use the unexplored corridor sequences of the two tangents associated with \(obj\), to find the first corridors in each of these sequences which differ from one another. When such a pair of corridors \(C_1, C_2\) exists, there are points visible to \(q\) on sides of both \(C_1\) and \(C_2\). We push two new objects corresponding to two new corridor sequences, one consisting of \(C_1\) and the other consisting of \(C_2\) into the stack \(ST\). Suppose the sequences do not differ or one is a prefix of the other. Then for the tangent \(qr\) defined by \(obj\) such that the entire corridor sequence of \(qr\) is explored, we determine a new tangent \(rt'\) whose corridor sequence is to be explored further. We find \(rt'\) using the preprocessed sorted tangents originating from \(r\), so as to find a corridor side which possibly has a visible point from \(q\).

Whenever a side \(S\) of corridor \(C\) is found to contain at least one visible point from \(q\), the corridor \(C\) is referred from the appropriate node in \(T_B(v)\) (or, \(T_U(v)\)) with the side \(S\) being marked. We continue processing objects from the stack \(ST\) in this way until \(ST\) is empty.

At the end, we traverse both the trees \(T_B(q)\) and \(T_U(q)\) in depth first order. For each marked side encountered during the traversal of either \(T_B(q)\) or \(T_U(q)\), we determine the vertices visible on that side from \(q\) using the procedure given in Section 4.4. By determining the additional vertices (if at all there are any), Section 4.5 computes the visible polygon.

4.1. Processing the corridor containing \(q\)

First, we find the corridor \(C\) in which the given query point \(q\) resides using the triangulation refinement method. Let \(S_1, S_2\) be the left and right sides of corridor \(C\) w.r.t. \(q\). Also, let \(B\) and \(U\) be the bounding edges of \(C\). Then we invoke the procedure based on either [1] or [7] with the simple polygon \(P_4(C)\), query point \(q\), and the visibility cone \([qp, qp']\) (with cone angle \(2\pi\)) for any point \(p \in R^2\), so as to find the region \(R\) in \(C\) which is visible from \(q\). By traversing the vertices of \(R\), we determine two points \(t', t''\) located on \(S_1\) such that for any point \(p'\) located on \(S_1\) and visible from \(q\), \(qt'\) (resp. \(qt'')\) makes an angle less (resp. greater) than \(qp'\) at \(q\). For any point \(r\) on \(U\), if the dual distance from \(r\) to \(t'\) (resp. \(t''\)) is less than or equal to the dual distance from \(r\) to \(t''\) (resp. \(t'\)) then \(t'\) (resp. \(t''\)) is known as \(t_{1u}\) and \(t''\) (resp. \(t'\)) is known as \(t_{1b}\). A symmetric definition applies for \(t_{2u}\) and \(t_{2b}\). (See Fig. 5.)

We also initiate the trees \(T_B(q)\) and \(T_U(q)\) with the root nodes associated with the corridor \(C\) and a flag indicating that we have determined the vertices of visible polygon located on sides \(S_1\) and \(S_2\).

Below, w.l.o.g. we suppose the counter-clockwise angle from \(qt_{1u}\) (resp. \(qt_{1b}\)) to \(qt_{2u}\) (resp. \(qt_{2b}\)) is less than \(\pi\). If the cone \([qt_{1u}, qt_{2u}]\) has a finite angle, we initiate the object with the tuple \([qt_{1u}, qt_{2u}, null, null, [qt_{1u}, qt_{2u}]\]). Similarly, if the cone \([qt_{1b}, qt_{2b}]\) has a finite angle, we initiate the object with the tuple \([qt_{1b}, qt_{2b}, null, null, [qt_{1b}, qt_{2b}]\]). Here, the null flags indicate that there is no unexplored corridor along either of the tangents \(qt_{1u}, qt_{2u}, qt_{1b}\) and \(qt_{2b}\). Among these two objects, the one with the tangent that makes the least angle at \(q\) is pushed onto the stack \(ST\) after the other one.

4.2. Processing the corridors not containing \(q\)

We process the corridors other than the one containing \(q\) by extracting objects from the stack \(ST\) and inserting new nodes into the trees \(T_B(q)\) and \(T_U(q)\). The procedure terminates whenever the stack \(ST\) is empty. For each object extracted from the stack \(ST\), the following procedure is invoked (see Fig. 6).
Let the object popped from the stack $ST$ be $obj = [l_i, r_i, lptr, rptr, [q_{r1}, q_{r2}]]$. Suppose the point $l_i$ (resp. $r_i$) is on $S_l$ (resp. $S_r$). Note that either of $l_i$ or $r_i$ may possibly be $q$. Let the corridor sequences intersecting with the line segments $l_i$ and $r_i$ be $SQ_l$ and $SQ_r$ respectively. Also, let $SQ_l$ be $C_1, C_2, \ldots, C_j, \ldots, C_k$, and let $SQ_r$ be $C_1', C_2', \ldots, C_j', \ldots, C_k'$. Suppose $lptr$ and $rptr$ refer to $C_1$ and $C_1'$ respectively. By moving $lptr$ and $rptr$ pointers respectively forward along the unexplored corridor sequences of $SQ_l$ and $SQ_r$ (i.e., by determining the next unexplored corridors successively), we find the first corridors in each of these sequences which differ from one another. Suppose these corridors are $C_{j+1}$ and $C_{j+1}'$ respectively, implying that $C_1 = C_1' = C_{j+1}' = \ldots, C_j = C_j'$. Note that we may or may not find such a pair. The procedure switches to the appropriate case described below.

- **Case 1:** This is the case in which we found the pair of corridors $C_{j+1}$ and $C_{j+1}'$ (see Fig. 6(a)). Let the right and left sides of $C_{j+1}$ and $C_{j+1}'$ be $S_r$ and $S_l$ respectively. Then we find the tangents $qt_r$ and $qt_l$ to sides $S_r$ and $S_l$ respectively, using the visibility cone $[q_{r1}, q_{r2}]$ and the hulls defined in Section 3.3. Note that both the points of tangencies $t_l$ and $t_r$ are visible from $q$.

We initiate two objects $obj_1$ and $obj_2$ with tuples $[l_i, q_{r'}, lptr, null, [q_{r1}, q_{r2}]]$ and $[q_{t}, r_i, null, rptr, [qt_r, q_{t'}]]$ respectively. The object $obj_1$ is pushed to the stack $ST$ after the object $obj_2$ so that the corridor sequences are processed in non-decreasing angular order.

Let $v_l$ (resp. $v_r$) be the node in either $T_B(q)$ or $T_U(q)$ such that $v_l$ (resp. $v_r$) refers to the corridor containing $l_i$ (resp. $r_i$).

We insert two nodes in the corresponding tree: one node as a child of $v_l$ which refers to the corridor $C_{j+1}$ with right side marked; and, the other node as a child of $v_r$, which refers to the corridor $C_{j+1}'$ with left side marked. While the first node is associated with the visibility cone $[q_{r1}, q_{t'}]$, the latter is associated with the visibility cone $[qt_r, q_{t'}]$.

- **Case 2:** This is the case in which there are no corridors in $SQ_l$ and $SQ_r$, which differ from one another, and every corridor in the corridor sequence of $l_i$ is explored. We find the supporting line from $q$ to the side on which $l_i$ resides. Let the supporting line be incident at $l_i$ and let $q_l$ make an angle $\alpha$ at $q$. Also, let the angles made by the tangents in the list $LIST_{1,8i}$ be $\alpha_1, \alpha_2, \ldots, \alpha_j$, with the corresponding points of tangencies as $t_1, t_2, \ldots, t_j$. Using binary search over these angles, (ignoring degeneracies) we determine a $t_i$ such that $\alpha_i < \alpha < \alpha_{i+1}$.

- **Sub-case 1:** This sub-procedure is invoked whenever there exist such a point of tangency $t_{i+1}$. Since $l_i$ is located on the left side of a corridor, the points of tangencies $t_1, t_2, \ldots, t_j$ are not visible from $l$ (see Fig. 6(b)). We initiate and push the tuple $[l_{i+1}, r_i, lptr, rptr, [q_{r1}, q_{r2}]]$ to the stack $ST$. Here $lptr$ points to the first corridor in the corridor sequence of $l_{i+1}$.

- **Sub-case 2:** This sub-procedure is invoked whenever we cannot find a new corridor sequence to explore further. It indicates that there is no tangent to a side $S$ from $q$, which belongs to the visibility cone associated with the corridor sequence (see Fig. 6(c)). However, $S$ may comprise visible vertices from $q$. Hence, we insert $S$ to the appropriate tree $T_B(q)$ or $T_U(q)$ for further processing.

Suppose $S$ is the left (resp. right) side of a corridor. Let $p$ be a point on $S$ such that $qp$ lies inside the cone $[q_{r1}, q_{r2}]$. Also, let $v$ be the node consisting of the last corridor $C'$ in the corridor sequence of $qp$ such that $C'$ has a visible vertex from $q$. Then we insert a new node in tree $T_B(q)$ or $T_U(q)$ as a child of $v$. The new node refers to the corridor having side $S$ with the left (resp. right) side marked. Also, we associate the visibility cone $[q_{r1}, q_{r2}]$ with side $S$ at the new node.

- **Case 3:** This case is symmetric to Case 2 except that this is invoked whenever every corridor in the corridor sequence of $r_i, r_e$ (rather than $l_i, l_e$) is explored.

4.3. Determining visible vertices from the tree $T_B(q)$ or $T_U(q)$

We traverse both the trees $T_B(q)$ and $T_U(q)$ in depth first order to determine visible vertices. For a node $v$ encountered for the first time, we check whether the left side, $LS$, of the corridor represented by $v$ is marked. If it is, we invoke
Fig. 6. Processing corridors not containing $q$. 

(a) Case 1

(b) Sub-case 1 of Case 2

(c) Sub-case 2 of Case 2
the procedure listed in Section 4.4 with \( q, LS \), and the visibility cone associated with side \( LS \) at node \( v \). This procedure determines the vertices of \( LS \) which are visible from \( q \) using the visibility cone \( vc \). After exploring all the child nodes of \( v \), we check whether the right side, \( RS \), of corridor represented by \( v \) is marked. If it is, we invoke the procedure listed in Section 4.4 with \( q, RS \), visibility cone associated with \( RS \). However, for the (sides of) corridor stored at the root node, we use the visible vertices already determined in Section 4.1.

### 4.4. Visible vertices of a corridor side

Given a corridor side \( S' \), a point \( q \), and the visibility cone \( vc \), this procedure finds the vertices of the side \( S' \) which are visible from \( q \). The simple polygon \( P' \) which is processed later is defined herewith. Let \( S_1 \) and \( S_2 \) for the corridor \( C \) be as defined in Section 3.4. If \( S' \) corresponds to \( S_2 \) of \( C \) and \( q \) is external to \( C \), then we use the simple polygon \( P_2(S_2) \) associated with the corridor \( C \) as \( P' \) to find the visible vertices from \( q \) on \( S_2 \). If \( S' \) corresponds to \( S_1 \) of \( C \) and \( q \) is external to \( C \), then we use once \( P_1(S_1) \) as \( P' \), and next \( P_2(S_1) \) as \( P'' \). Note that all these simple polygons are computed during the preprocessing time as described in Section 3.

We use either of the following two approaches to find the region in the simple polygon \( P' \) which is visible from \( q \) within the visibility cone \( vc \). In the first approach, we invoke the procedure given in [1]. In the second approach, we invoke the procedure which uses ray-shooting and is defined in [7]. These approaches report the vertices of the visible region in the simple polygon \( P' \). Note that the reported polygon may include either points on the sides of the bounding box \( BB \) or the points on the bounding edges of a corridor. However, these points are deleted from the output visibility polygon.

Consider any two vertices \( p', p'' \) of \( P' \), for \( p', p'' \) belonging to a corridor side \( S' \) where \( S' \) is the left (resp. right) side w.r.t. \( q \). Then this sub-procedure outputs \( p' \) before \( p'' \) if and only if the angle made by \( qp' \) at \( q \) is less (resp. greater) than the angle made by \( qp'' \) at \( q \). This ordering is compatible with the ordering of the vertices in the visibility polygon for \( q \).

### 4.5. Visible polygon determination

Suppose the list \( L' \) comprises the sequence of all the vertices visible from \( q \), which is obtained with the procedure listed in Section 4.3. We claim that the edges joining the consecutive vertices in list \( L' \) together with the edge joining the last vertex in list \( L' \) to the first one, together yield the visibility polygon of the query point \( q \).

### 5. Analysis

**Theorem 5.1.** When the approach suggested in [1] is used as the sub-procedure, the algorithm spends \( O(n^2 \log n) \) time during the preprocessing. When the ray-shooting [7] based approach is used as the sub-procedure, the algorithm spends \( O(T + |E| + n \log n) \) time during the preprocessing.

**Proof.**

- The triangulation of the polygonal region takes \( O(n + m \log^{1+\varepsilon} m) \) time, represented as \( O(T) \). Determining the corridors given the triangulation takes \( O(n + m \log n) \) time with the procedure suggested in [8,10] and using the variation in Section 3.1. Since there are \( m \) obstacles, there can be at most \( O(m) \) corridors.
- Computing the visibility tree data structures using the approach listed in [9] takes \( O(T + |E| + m \log n) \) time. In finding supporting lines from each vertex \( v \) to sides which have at least one point visible from \( v \), we traverse all the lists in \( TVIS \) data structures once. Hence it takes \( O(|E|) \) time to compute all the supporting lines. We find a supporting line from \( v \) to a side \( S \) whenever a vertex of \( S \) appears at a node of either of the \( TVIS \) data structures. There may be more than one supporting line from \( v \) to a side. As there can be \( O(|E|) \) nodes in all \( TVIS \) structures together and the supporting lines are a subset of \( E \), the total number of supporting lines from all the vertices together is upper bounded by \( O(|E|) \). Hence \( O(|E|) \) time is used in building the sorted lists \( \text{LIST}_{LBV}, \text{LIST}_{RBV}, \text{LIST}_{LB}, \text{LIST}_{RB} \).
- Preprocessing each side for the triangulation refinement method takes \( O(n \log n) \) time.
- Constructing hulls corresponding to sides of open corridors and the sides of funnels takes \( O(n \log n) \) time (see Section 3.3).
- The overall time to compute simple polygons \( P_1(S_1), P_2(S_1), P_3(S_2), P_4(C) \) for each corridor \( C \) as described in Section 3, together takes \( O(n) \) time.
- The overall preprocessing time involved with the sub-procedure [1] is \( O(n^2 \log n) \). Including this with the above, the overall preprocessing time complexity of our algorithm with this sub-procedure is \( O(n^2 \log n) \). The overall preprocessing time involved with the ray-shooting based approach [7] is \( O(n) \). Including this with the above, the overall preprocessing time complexity with this approach is \( O(T + |E| + n \log n) \).

**Theorem 5.2.** When the approach suggested in [1] is used as the sub-procedure, the size of preprocessed data structures is \( O(n^2) \). When the ray-shooting [7] based approach is used as the sub-procedure, the size of preprocessed data structures is \( O(\min(|E|, mn) + n) \).
Proof.

- The visible tree data structures from [9] are of space $O(|E|)$. But we are deleting LISTS stored at each node of these data structures, once we build the lists $LIST_{LU}$, $LIST_{RU}$, $LIST_{LB}$, $LIST_{RB}$ for each vertex $v$. Since there can be at most $O(m)$ nodes in the TVIS$_B(v)$ and TVIS$_U(v)$ together, there can be $O(m)$ supporting lines from $v$. Considering supporting lines from all the vertices, the size of these lists together takes $O(mn)$ space. But if we consider the space $O(|E|)$ of all the visibility edges, the space complexity of all the lists together is $O(\min(|E|, mn))$.
- The data structures required for the triangulation refinement method are of $O(n)$ size.
- The space complexity of hulls of all the corridor sides and the sides of funnels is $O(n)$.
- The space complexity of simple polygons built for all corridors together is $O(n^2)$.

Theorem 5.3. When the sub-procedure from [1] is used, the query complexity is $O((1 + \min(m, |V(q)|)) \log^2 n + m + |V(q)|)$. When the ray-shooting based sub-procedure [7] is used, the query complexity is $O(|V(q)| \log n + m)$.

Proof.

- First, consider the computation in Section 4.1. Finding the corridor $C$ in which the query point $q$ resides takes $O(\log n)$ time with the triangulation refinement method. Let the number of vertices visible from $q$ on the sides of corridor $C$ be $O(k)$. Then computing the visible region of $C$ and the points $t_1$, $t_2$, $t_3$ with the procedure based on [1] takes $O(\log^2 n + k)$ time, whereas computing the same using the ray-shooting based approach from [7] takes $O(k \log n)$ time.
- Consider the computation in Section 4.2. Once a corridor $C'$ is considered while moving pointers $lptr$, $rptr$ along a corridor sequence, the same corridor $C'$ is not considered again. Since there are $O(m)$ corridors, the overall computing cost of the last common corridors is $O(m)$. Since there are $O(m)$ nodes in TVIS, at most $O(m)$ objects are pushed/popped from the stack $ST$. For each object popped from the stack $ST$, the object is processed in at most two cases.
  - In Case 1, for each common corridor we are finding a tangent to two sides while these sides have at least one vertex visible from $q$. Finding these tangents takes $O(\log n)$. We are inserting a new node in either $T_B(q)$ or $T_U(q)$ whenever we find a new vertex which is visible from $q$.
  - The binary search in Case 2 (or Case 3) takes $O(\log m)$ time as there are $O(m)$ supporting lines originating from each vertex $v$. This cost is charged to the vertex $v$ which is visible from $q$.
    * When we find a tangent in Sub-case 1 of Case 2 or Case 3, we insert a new node to either $T_B(q)$ or $T_U(q)$ which takes $O(1)$ time.
    * When we cannot find a tangent, we may add at most one node to either $T_B(q)$ or $T_U(q)$ in Sub-case 2 of Case 2 or Case 3. But this can be charged to the last corridor in the corridor sequence which has a visible vertex.

Computing the information associated with an object to be pushed to the stack $ST$ takes $O(1)$ time. Inserting a new node to either of the trees takes $O(1)$ time. In all the above cases, we are doing $O(\log n)$ computation whenever we need to find a corridor sequence of a tangent and determine a new vertex which is visible from $q$. Therefore, the total computation involved in processing all the objects pushed/popped from the stack $ST$ together takes $O(\min(m, |V(q)|) \log n + m)$ amortized time.
- As the total number of nodes in $T_B(q)$ (or, $T_U(q)$) are $O(\min(m, |V(q)|))$, depth first search takes $O(\min(m, |V(q)|))$ time.

When we use the algorithm by Aronov et al. in [1] to find the visible vertices from a corridor side having $O(k)$ vertices visible from $q$, the query complexity is $O(\log^2 n + k)$. Since we invoke this procedure at each node of $T_B(q)$ (or, $T_U(q)$), when there are $O(|V(q)|)$ vertices visible from $q$, the overall query time spent in this procedure is $O(\min(m, |V(q)|) \log^2 n + |V(q)|)$.

When we use the ray-shooting [7] based approach to find the visible vertices from a corridor side having $O(k)$ vertices visible from $q$, the query complexity is $O(k \log n)$. Since we invoke this procedure at each node of $T_B(q)$ (or, $T_U(q)$), when there are $O(|V(q)|)$ vertices visible from $q$, the overall query time spent in this procedure is $O(|V(q)| \log n)$.

Also for each invocation of either of these sub-procedures, we spend $O(1)$ time in deleting the vertices of the visible region which reside on the bounding box $BB$.
- Computing all the visible polygon vertices in forming the list $L'$ takes $O(|V(q)|)$ time.

Query time improvement

We reduce the query time by precomputing the last common corridor between every two tangents. Since all the LIST structures are of size $O(\min(|E|, mn))$, there are $O((\min(|E|, mn))^2)$ two-combinations. We take $O(m)$ time in computing the last common corridor between each such two-combination. Hence the preprocessing time complexity is...
we are inserting these sides to tree the common corridor between two tangents during the query time, Hence saving in Section 4.4 finds vertices belonging to output visibility polygon, which are not vertices of the given polygonal domain.

Proof. Consider the following cases:

- Suppose \( v \in C \), for the corridor \( C \) containing \( q \). In determining \( v \), the procedure in Section 4.4 uses \( C \) as \( P_4(C) \) and a visibility cone with cone angle \( 2\pi \). Since \( v \in C \), it must be the case that \( v \) incidents on either of the sides of \( C \). The correctness of the procedure in Section 4.4 relies on the correctness of the sub-procedure based on either \([1]\) or \([7]\). Therefore, \( v \) is guaranteed to be visible from \( q \). Since \( v \) is a vertex in the given polygonal region, it is immediate that it is a vertex of \( V(q) \).

- Suppose \( v \in C' \), for a corridor \( C' \) not containing \( q \). Since a point \( q \) external to a corridor \( C' \) can view a point located inside \( C' \) only through either of the bounding edges of \( C' \), the simple polygons \( P_1(S_1), P_2(S_1), P_3(S_2) \) are defined to guarantee that no point in \( C' \) visible from \( q \) is excluded from \( P_1(S_1) \cup P_2(S_1) \cup P_3(S_2) \). More specifically, every point \( p \) on side \( S_1 \) visible from \( q \) is such that \( p \in P_1(S_1) \cup P_2(S_1) \). Similarly, for every point \( p \) on side \( S_2 \) visible from \( q \), \( p \in P_3(S_2) \). The procedures in Sections 4.3 and 4.4 together choose the appropriate polygons correctly. Therefore, the correctness of the output from sub-procedure based on either \([1]\) or \([7]\) relies on another input parameter, visibility cone \( \text{vc}(q) \). The following exhaustive cases consider the \( \text{vc}(q) \) refinement.

\[ \text{vc}(q) \] refinement due to sides of corridor \( C \), for the corridor \( C \) containing \( q \) (refer to Section 4.1). The correctness of determining the visible region \( R \) relies on the procedure based on either \([1]\) or \([7]\). By visiting the vertices of \( R \), the points \( t_{1u}, t_{1b}, t_{2u}, t_{2b} \) are correctly determined. The visibility cone \( [qt_{1u}, qt_{2u}] \) is defined if and only if both \( qt_{1u} \) and \( qt_{2u} \) intersect \( U \). If it is defined, it represents the visible region \( \text{w.r.t.} \ q \) intersecting \( U \) correctly. Same is true with the visibility cone \( [qt_{1b}, qt_{2b}] \).

\[ \text{vc}(q) \] refinement due to sides of corridor \( C' \), for any corridor \( C' \) not containing \( q \) (refer to Section 4.2). In Case 1, we compare corridor sequences corresponding to two tangents to find the first two corridors in these sequences which differ from one another. At least one vertex on the sides of each of these two corridors is guaranteed to be visible from \( q \) as the vertex belonging to a side from each corridor falls within the visibility cone. For Sub-case 1 of Case 2, since the \( \{ \text{parts of} \} \) sides having supporting lines to the left of line \( ql \) do not have a visible point, the algorithm considers only sides other than these. A symmetric argument is true for Sub-case 1 of Case 3. If we cannot find a tangent, there may be a side which has a visible point. This is considered in Sub-case 2 of Case 2 and Case 3, which inserts the side into the tree. Hence the cones confined are computed correctly. \( \Box \)

Lemma 6.2. Each vertex \( v \) of the visibility polygon of \( q \), \( v \) is guaranteed to be determined in the algorithm.

Proof. Suppose there exist a vertex \( v \) in the visibility polygon of \( q \) which is not found by the algorithm.

- The vertex \( v \) is on a corridor side \( S \), whereas \( S \) is not inserted to either of the trees \( T_B(q) \) or \( T_U(q) \). It is trivial to note that this is not the case for the sides of the corridor in which \( q \) resides. For any other corridor, both the correctness of TVIS data structures from \([9]\), and the correctness of Case 2 (or, Case 3) argued in the above lemma contradicts that a side \( S \) is not inserted to these trees.

- The vertex \( v \) of visibility polygon is not a vertex of any corridor side i.e., \( v \) is a point on a corridor side and it appears as a vertex in the visibility polygon. In other words, \( v \) is a point on an edge whose one end or both end points are not visible from \( q \). However, \( v \) and all such vertices are readily determined by sub-procedure listed in Section 4.4. \( \Box \)

Theorem 6.1. The visibility polygon of \( q \) is computed correctly.

Proof. From the above two lemmas, we know that a vertex \( v \) is determined as visible in the algorithm if and only if \( v \) is a vertex of the visibility polygon of \( q \).

After finding a point of tangency to a side from \( q \), rather than immediately exploring a side for the visible vertices, we are inserting these sides to tree \( T_B(q) \) (or, \( T_U(q) \)). This facilitates in ordering vertices along the visibility polygon of \( q \). Exploring the left and right sides while traversing the tree \( T_B(q) \) (or, \( T_U(q) \)) in depth first order as explained in Section 4.5, yields the required ordering. Hence, the vertices in list \( U' \) (refer to Section 4.4) are ordered angularly. Sub-procedure listed in Section 4.4 finds vertices belonging to output visibility polygon, which are not vertices of the given polygonal domain. \( \Box \)

7. Conclusions

This paper presented an output-sensitive algorithm for determining the visibility polygon of a query point \( q \) inside a polygonal region by preprocessing the given region to build data structures with improved space and preprocessing com-
plexities. The query time is competitive with the previous methods when there are either a small constant number of holes or \( |V(q)| \geq m \). With the additional \( O((\min(|E|, mn))^2) \) space with extra preprocessing, both the space and query complexities are superior and competitive respectively to previous methods whenever \( n > m^2 \). Also, it would be interesting to find whether this approach is useful in computing the visibility complex.

References