Visibility Polygons and Visibility Graphs among Dynamic Polygonal Obstacles in the Plane

Sanjana Agrawal\textsuperscript{1} and R. Inkulu\textsuperscript{1} *

Department of Computer Science & Engineering
IIT Guwahati, India
\{sanjana18,rinkulu\}@iitg.ac.in

\textbf{Abstract.} We devise an algorithm for maintaining the visibility polygon of any query point in a dynamic polygonal domain, i.e., as the polygonal domain is modified with vertex insertions and deletions to its obstacles, we update the data structures that store the visibility polygon of the query point. After preprocessing the initial input polygonal domain to build a few data structures, our algorithm takes $O(k(\lg |V_{P'}(q)|) + (\lg n')^2 + h)$ (resp. $O(k(\lg n')^2 + (\lg |V_{P'}(q)|) + h)$) worst-case time to update data structures that store visibility polygon $V_{P'}(q)$ of a query point $q$ when any vertex $v$ is inserted to (resp. deleted from) any obstacle of the current polygonal domain $P'$. Here, $n'$ is the number of vertices in $P'$, $h$ is the number of obstacles in $P'$; $V_{P'}(q)$ is the visibility polygon of $q$ in $P'$ ($|V_{P'}(q)|$ is the number of vertices of $V_{P'}(q)$), and $k$ is the number of combinatorial changes in $V_{P'}(q)$ due to the insertion (resp. deletion) of $v$.

As an application of the above algorithm, we also devise an algorithm for maintaining the visibility graph of a dynamic polygonal domain, i.e., as the polygonal domain is modified with vertex insertions and deletions to its obstacles, we update data structures that store the visibility graph of the polygonal domain. After preprocessing the initial input polygonal domain, our dynamic algorithm takes $O(k(\lg n')^2 + h)$ (resp. $O(k(\lg n')^2 + h)$) worst-case time to update data structures that store the visibility graph when any vertex $v$ is inserted to (resp. deleted from) any obstacle of the current polygonal domain $P'$. Here, $n'$ is the number of vertices in $P'$, $h$ is the number of obstacles in $P'$, and $k$ is the number of combinatorial changes in the visibility graph of $P'$ due to the insertion (resp. deletion) of $v$.

1 Introduction

The polygonal domain comprises a set of pairwise-disjoint simple polygons (obstacles) in the plane. We assume the obstacles in the polygonal domain are placed in a large bounding box. For any polygonal domain $P$, the free space $F(P)$ is the closure of bounding box without the union of the interior of all the obstacles in $P$. Any two points $p, q \in F(P)$ are visible to each other if the open line segment joining $p$ and $q$ lies entirely in $F(P)$. A vertex $v$ of the polygonal domain is said to be a visible vertex to a point $q$ whenever $v$ is visible to $q$. For a point $q \in F(P)$, the visibility polygon of $q$, denoted by $V_{P}(q)$, is the maximal set $S$ of points in $F(P)$, such that each point in $S$ is visible to $q$. (When $P$ is clear from the context, the visibility polygon of $q$ is denoted by $V_{P}(q)$.) The visibility polygon query problem seeks to preprocess the given polygonal domain $P$ so that to efficiently compute $V_{P}(q)$ for any query point $q$ located in $F(P)$. Computing visibility polygons is a fundamental problem in computational geometry, and it is studied extensively. The (vertex-vertex) visibility graph of a polygonal domain $P$, denoted by $VG_{P}$, is the directed graph with its vertex set comprising all the vertices of $P$. The edge set of $VG_{P}$ comprises of every line segment with its endpoints $v'$, $v''$ being the vertices of $P$ such that $v'$ and $v''$ are visible to each other among the obstacles of $P$. The visibility graphs have many applications, ex., in computing geodesic shortest paths, minimum link paths, and in computing the weak visibility polygons. In the following, we denote the number of vertices

\* A preliminary version [1] appeared in Proceedings of the 26\textsuperscript{th} International Computing and Combinatorics Conference (COCOON), 2020.
of the simple polygon or the polygonal domain by \( n \), the number of obstacles in the polygonal domain by \( h \), and the set comprising the edges of the visibility graph by \( E \).

The problem of computing the visibility polygon of a point in a simple polygon was first attempted by Davis and Benedikt in [15], and they presented an \( O(n^2) \) time algorithm. Later, both ElGindy and Avis [16], and Lee [32], presented \( O(n) \) time algorithms for the same problem. Joe and Simpson [26] corrected a flaw in [16,32], and devised an \( O(n) \) time algorithm that correctly handles winding in the simple polygon. For a simple polygon with holes, both Suri and O’ Rourke [40], and Asano [4] presented \( O(n \lg n) \) time algorithms, and Heffernan and Mitchell [21] gave a \( O(n + h \lg h) \) time algorithm. The visibility polygon computation among convex sets was considered by Ghosh in [17].

For both the simple polygon as well as the polygonal domain, the previous works considered the visibility polygon query problem. Bose et al. [7] gave an algorithm to preprocess the given simple polygon in \( O(n^3 \lg n) \) time, build data structures of size \( O(n^3) \), and answer any visibility polygon query in \( O(\lg n + |VP(q)|) \) time. Later, Aronov et al. [3] devised an algorithm for the same problem with preprocessing time \( O(n^2 \lg n) \), space \( O(n^2) \), and query time \( O(\lg^2 n + |VP(q)|) \). Zarei and Ghodsi [42] presented an algorithm that preprocesses the given polygonal domain in \( O(n^3 \lg n) \) time to build data structures of size \( O(n^3) \), and answers each visibility polygon query in \( O((1+h') \lg n + |VP(q)|) \) time, where \( h' = \min(h,|VP(q)|) \). The algorithm by Inkulu and Kapoor [23] preprocesses the input polygonal domain in \( O(n^2 \lg n) \) time, builds data structures of size \( O(n^2) \), and answers visibility polygon query in \( O(\min(h,|VP(q)|)(\lg n)^2 + h + |VP(q)|) \) time. This paper also presented another algorithm with preprocessing time \( O(T + |VG| + n \lg n) \), space \( O(\min(|VG|,hn) + n) \), and query time \( O(|VP(q)| \lg n + h) \). Here, \(|VG|\) denotes the number of edges in the visibility graph, and \( T \) is the time to triangulate the free space of the given polygonal domain. Baygi and Ghodsi [6] constructed a data structure of size \( O(n^2) \) in \( O(n^2 \lg n) \) time, and their algorithm answers any visibility polygon query in \( O(|VP(q)| + \lg n) \) time. Lu et al. [33] presented an algorithm to compute a data structure of size \( O(n^2) \) in \( O(n^2 \lg n) \) time, which helps in answering any visibility polygon query in \( O(|VP(q)| + (\lg n)^2 + h \lg (n/h)) \) time. Chen and Wang [12] gave an algorithm that preprocesses the polygonal domain in \( O(n + h^2 \lg h) \) time to construct data structures of size \( O(n + h^2) \), so that to answer any visibility polygon query in \( O(|VP(q)| \lg n) \) time. Pocchiola and Vegter [37] considered the query version of visibility polygon computation in the polygonal domain comprising convex obstacles. Their algorithm computes the visibility polygon of any query point in \( O(|VP(q)| \lg n) \) time after preprocessing the convex polygonal domain in \( O(n \lg n) \) time and building data structures of size \( O(n) \).

To compute the visibility graph of a simple polygon, Lee [31] and Sharir and Schorr [39] gave algorithms that take \( O(n^3 \lg n) \) time. For this problem, algorithms given by Asano et al. [4] and Welzl [41] take \( O(n^2) \) worst-case time, which are worst-case optimal since the number of edges \(|E|\) in the visibility graph is \( \Theta(n^2) \) in the worst-case. For triangulated simple polygons, Hershberger [22] gave an \( O(n + |E|) \) time algorithm. Since any simple polygon can be triangulated in linear time due to an algorithm by Chazelle [8], the algorithm in [22] essentially takes \( O(n + |E|) \) time, which is optimal for this problem.

There are a number of algorithms to compute the visibility graph of a polygonal domain efficiently. Overmars and Welzl [36] gave an output-sensitive algorithm that takes \( O(|E| \lg n) \) time and \( O(n + |E|) \) space. Ghosh and Mount [19] presented an algorithm with \( O(n \lg n + |E|) \) worst-case time with \( O(|E| + n) \) space. Keeping the same time complexity as the algorithm in [19], Pocchiola and Vegter [37] improved the space complexity to \( O(n) \). Kapoor and Maheshwari [28,29] proposed an algorithm with time complexity \( O(T + |E| + h \lg n) \) and \( O(|E|) \) space. Here, \( T \) is the time for triangulating the free space of the polygonal domain, which is \( O(n + h(\lg h)^{1+\delta}) \) due to Bar-Yehuda and Chazelle [5] for a small positive constant \( \delta \).
closely following [19], Chen and Wang [11] devised an algorithm with the same time complexity as the algorithm in [28, 29]. The texts by Ghosh [18] and O’Rourke [34] detail a number of algorithms for computing visibility polygons and visibility graphs in the plane.

In the context of visibility polygons (resp. visibility graphs), having algorithms to maintain the visibility polygon (resp. visibility graph) among dynamic polygonal obstacles helps in updating the visibility polygon (resp. visibility graph) efficiently (that is, with respect to updation time) as compared to computing the entire visibility polygon (resp. visibility graph) from scratch using traditional algorithms. In doing this, the algorithm specifically exploits the recent changes that occurred to the polygonal domain; based on these changes, the update is performed. Both Inkulu and Nitish [24] and Inkulu et al. [25] devised algorithms for maintaining the visibility polygon of a query point in a dynamic simple polygon. After preprocessing to build a few data structures, the first algorithm updates data structures that store the visibility polygon of any query point in $O\left(n + |E|\lg |E|\right)$ time when any vertex is inserted to (resp. deleted from) the simple polygon. Here, $k$ is the number of combinatorial changes in the visibility polygon of $q$ due to the insertion (resp. deletion) of $v$ and $n'$ is the number of vertices in the current simple polygon. An algorithm for maintaining the weak visibility polygon in a dynamic simple polygon is given in [25]. Choudhury and Inkulu [13] devised an algorithm for maintaining the visibility graph of a dynamic simple polygon. The algorithm in [13] preprocesses the initial simple polygon in $O\left(n + |E|\lg |E|\right)$ time to build data structures of size $O\left(n + |E|\right)$, and updates the visibility graph of the current simple polygon $P'$ in $O\left((k + 1)(\lg n')^2\right)$ time when a vertex $v$ is inserted to (resp. deleted from) $P'$. Here, $k$ is the number of combinatorial changes in the visibility graph of $P'$ due to the insertion (resp. deletion) of $v$ to $P'$. The visibility in the context of a moving observer was studied in [9, 2].

1.1 Our results

In this paper, two algorithms are proposed. After preprocessing to build a few data structures, the first algorithm updates data structures that store the visibility polygon of any query point $q$, and the second algorithm updates the data structures that store the visibility graph of the current polygonal domain, both among a set of dynamic polygonal obstacles. These algorithms update relevant data structures whenever any vertex is inserted to any of the obstacles in the current polygonal domain or whenever any vertex is deleted from any of the obstacles.

- Let $VP_{P'}(q)$ be the visibility polygon of a query point $q$ located in the free space of the current polygonal domain $P'$. Also, let $n'$ be the number of vertices of $P'$. When any vertex $v$ is inserted to (resp. deleted from) any of the obstacles of $P'$, our algorithm takes $O\left(k(\lg |VP_{P'}(q)|) + (\lg n')^2 + h\right)$ (resp. $O\left(k(\lg n')^2 + (\lg |VP_{P'}(q)|) + h\right)$) time to update data structures that store $VP_{P'}(q)$. Here, $k$ is the number of combinatorial changes in $VP_{P'}(q)$ due to the insertion (resp. deletion) of $v$. (The combinatorial changes to $VP_{P'}(q)$ include vertices inserted, vertices deleted, edges inserted, and edges deleted from $VP_{P'}(q)$.) Given any query point $q$ located in the free space of the current polygonal domain $P'$, our output-sensitive visibility polygon query algorithm computes $VP_{P'}(q)$ in $O\left(|VP_{P'}(q)|((\lg n')^2 + h\right)$ time. The data structures constructed as part of answering the visibility polygon query algorithm facilitate efficient updations as the polygonal domain changes. We preprocess the initial input polygonal domain $P$ in $O\left(n(\lg n)^2 + h(\lg h)^{1+\epsilon}\right)$ time, and construct data structures of size $O(n)$). Here, $n$ is the number of vertices of $P$, $h$ is the number of polygonal obstacles in $P$, and $\epsilon > 0$ is a small positive constant (resulting from triangulating the free space of $P$ using the algorithm in [5]).
Let $VG_{P'}$ be the visibility graph of the current polygonal domain $P'$. Also, let $n'$ be the number of vertices in $P'$. When any vertex $v$ is inserted to (resp. deleted from) any of the obstacles of $P'$, our algorithm takes $O(k(lg n')^2 + h)$ (resp. $O(k((lg n')^2 + h))$) time to update the visibility graph of $P'$. Here, $k$ is the number of combinatorial changes in $VG_{P'}$ due to the insertion of $v$. (The combinatorial changes in $VG_{P'}$ include vertices inserted, vertices deleted, edges inserted, and edges deleted from $VG_{P'}$.) We preprocess the initial input polygonal domain $P$ in $O(n(lg n)^2 + h(lg h)^{1+\epsilon} + |E| lg |E|)$ time, and construct data structures of size $O(n + |E|)$. Here, $n$ is the number of vertices of $P$, $h$ is the number of polygonal obstacles in $P$, $E$ is the set of edges in the visibility graph of $P$, and $\epsilon > 0$ is a small positive constant (resulting from triangulating the free space of $P$ using the algorithm in [5]).

For convenience, we assume all through the algorithm, the number of polygonal obstacles does not change, and the obstacles are pairwise disjoint. To our knowledge, these are the first algorithms for maintaining the visibility polygon of any given point and the visibility graph of the polygonal domain among the dynamic polygonal obstacles. These algorithms obviate computing the visibility polygon of a given point or the visibility graph from scratch whenever the polygonal domain is modified with a vertex insertion or vertex deletion.

Next, we give an overview of our approach. For any given query point $q$, we compute two binary trees (called visibility trees) to store the visibility polygon of $q$. These visibility tree data structures in our algorithm are a modification to visibility trees defined in [29]. From these trees, we compute the visibility polygon $VP_{P'}(q)$ of $q$ in the current polygonal domain $P'$. Whenever a new vertex $v$ is inserted to an obstacle of the current polygonal domain $P'$, first, we perform a ray-shooting query to determine whether $v$ is visible to $q$. If it is not visible, then our algorithm does nothing further. Otherwise, we traverse the visibility trees in depth-first order to determine all the vertices that are not visible to $q$ due to the insertion of $v$. We update the $VP_{P'}(q)$ by removing these vertices from $VP_{P'}(q)$. As part of this, we also update both the visibility trees. Similarly, in the case of deletion of a vertex $v$, by performing a ray-shooting query, it is determined whether deleted vertex $v$ is visible to $q$. If the deleted vertex $v$ was visible to query point $q$, then the deletion of $v$ may cause the addition of some new vertices to $VP_{P'}(q)$. These new vertices are determined using our output-sensitive visibility polygon query algorithm with $q$ as the query point. Updating $VP_{P'}(q)$ involves merging a set of polygons due to these vertices into data structures that store $VP_{P'}(q)$. Our visibility polygon query algorithm enhances data structures designed in [23], so that they work efficiently among dynamic polygonal obstacles. As in [23], our visibility polygon query algorithm computes visibility trees of a query point $q$ by determining sequences’ of triangles bounding rays of visibility cones initiated at $q$ intersect. When any ray in a visibility cone is found to strike a point on the boundary of any obstacle $O$, we compute tangents from $q$ to $O$. To compute these tangents and, in turn, for updating visibility cones, our algorithm dynamically maintains hull trees of obstacle boundaries using the algorithm by Overmars and van Leeuwen [38].

In updating the visibility graph, we use an important characterization to determine edges of the visibility graph $VG_{P'}$ of the polygonal domain $P'$ that need to be deleted from (resp. included into) $VG_{P'}$ due to the insertion (resp. deletion) of a vertex from $P'$.

### 1.2 Terminology

We denote the initial input polygonal domain by $P$, the number of vertices of $P$ by $n$, and the number of polygonal obstacles in $P$ by $h$. We use $P'$ to denote a polygonal domain just before inserting (resp. deleting) a vertex, and $P''$ to denote a polygonal domain just after inserting (resp. deleting) a vertex. We call $P'$ as the current polygonal domain, and we call $P''$ as the
updated polygonal domain. The number of vertices of \( P' \) is denoted by \( n' \). The number of pairwise disjoint polygonal obstacles in any polygonal domain is denoted by \( h \). Recall the definitions of free space of a polygonal domain, visibility polygon of a point in a polygonal domain, and the visibility graph of a polygonal domain from Section 1. It is assumed that every new vertex is added between two successive vertices of an obstacle. Whenever a new vertex \( v \) is inserted between two adjacent vertices \( v_i \) and \( v_{i+1} \) in polygonal domain \( P' \), it is assumed that two new edges are added to \( P' \): one between vertices \( v \) and \( v_i \), and the other between vertices \( v \) and \( v_{i+1} \). Similarly, in the case of deletion of a vertex \( v \), it is assumed that after deleting \( v \) from \( P' \) which is adjacent to vertices \( v_i \) and \( v_{i+1} \) in \( P' \), a new edge is inserted between vertices \( v_i \) and \( v_{i+1} \). After adding (resp. deleting) any vertex to (resp. from) any obstacle \( O \) in the current polygonal domain, our algorithm assumes \( O \) continues to be a simple polygon. Further, the newly inserted vertex in \( P'' \) is assumed to be contained in the bounding box containing the obstacles of \( P \). The visibility polygon of \( q \) in any polygonal domain \( Q \) is denoted by \( VP_Q(q) \). The visibility graph in any polygonal domain \( Q \) is denoted by \( VG_Q \). For any simple polygon \( P \), the boundary of \( P \) is denoted by \( bd(P) \). Unless specified otherwise, the boundary of any simple polygon is assumed to be traversed in counter-clockwise order. It is assumed that all the angles are measured in the counter-clockwise direction from the positive horizontal axis \( (x \text{-axis}) \). A \textit{constructed edge} \cite{18} is an edge \( u_i u_{i+1} \) on the boundary of \( VP_P(q) \) such that either (i) no point of \( u_i u_{i+1} \), except the points \( u_i \) and \( u_{i+1} \), belongs to the boundary of \( P' \), and at least one of \( u_i, u_{i+1} \) is a vertex of \( P' \), or (ii) every point on the edge \( u_i u_{i+1} \) is lying on the boundary of \( P' \), and neither \( u_i \) and \( u_{i+1} \) is a vertex of \( P' \). For every constructed edge \( u_i u_{i+1} \), among \( u_i \) and \( u_{i+1} \), the farthest one from \( q \) is termed a \textit{constructed vertex} of \( VP(q) \).

![Fig. 1. Illustrating a triangulation of the free space among two obstacles and the corridors (indicated by red solid curves). There are two junction triangles marked by a large dot inside each of them, connected by three solid (red) curves.Removing the two junction triangles results in three corridors. (This illustration is from \cite{10}.)](image)

Next we describe a few structures from \cite{27, 28, 30} which result from decomposing the free space \( F(P) \) of a polygonal domain \( P \). Consider a triangulation \( T_P \) of \( F(P) \). Let \( G_d \) be the dual graph of \( T_P \). First, we prune \( G_d \) by iteratively removing all the vertices of degree one. Let \( G'_d \) be the pruned dual graph of \( T_P \). For every node belonging to \( G'_d \), its degree is either two or three. Every triangle in \( T_P \) that corresponds to a vertex of degree three in \( G'_d \) is known as a \textit{junction}. Removing the junctions leads to connected simple polygonal regions in \( P \). Every such region is known as a \textit{corridor}. (Refer to Fig. 1.) Any corridor \( C \) has at most two poly-lines, each is known as a \textit{corridor side}, and two edges, each is known as a \textit{(corridor) bounding edge} of \( C \). For any corridor \( C \), and for any bounding edge \( v'v'' \) of \( C \), if the angle made by ray \( qv' \) is less than or equal to (resp. greater than) the angle made by ray \( qv'' \) at \( q \), then the corridor side of \( C \) that has \( v' \) is known as the \textit{left side} (resp. \textit{right side}) of \( C \) (with respect to \( q \).)
Let $r'$ and $r''$ be two rays with origin at $p$. Let $uv_1$ and $uv_2$ be the unit vectors along the rays $r'$ and $r''$, respectively. A cone $C_p(r', r'')$ is the set of points defined by rays $r'$ and $r''$ such that a point $p' \in C_p(r', r'')$ if and only if $p'$ can be expressed as a convex combination of the vectors $uv_1$ and $uv_2$ with positive coefficients. When the rays are evident from the context, we denote the cone with $C_p$. A cone $C_p$ is called a visibility cone whenever $C_p$ contains at least one point in $\mathcal{F}(\mathcal{P})$ that is visible to $p$. For any cone $C_p(r_i, r_j)$, among rays $r_i$ and $r_j$, the ray that makes lesser angle with the positive $x$-axis at $p$ is the left ray of $C_p$ and the other ray is the right ray of $C_p$. For any cone, throughout the paper, we assume the counter-clockwise angle between the left ray of $C_p$ and the right ray of $C_p$ is less than $\pi$. Any edge in the visibility graph of the polygonal domain is called a visible edge. We use vertex to denote any endpoint of any edge of the polygonal domain, and we use node to denote any tree node in data structures that we construct.

The preprocessing algorithm and the data structures for maintaining the visibility polygon are detailed in Section 2. Section 3 details the algorithms to maintain the visibility polygon of any query point whenever the current polygonal domain is modified with a vertex insertion or with a vertex deletions. The output-sensitive visibility polygon query algorithm of any given query point is given in Section 4. Section 5 details both the preprocessing algorithm and the algorithms for maintaining the visibility graph. Conclusions are in Section 6.

2 Preprocessing algorithm and data structures

We first preprocess the input polygonal domain $\mathcal{P}$. Using the algorithms in [27, 28, 30], we partition the free space $\mathcal{F}(\mathcal{P})$ of $\mathcal{P}$ into $O(h)$ corridors and junctions. To efficiently compute tangents to dynamic sides of corridors, for each side $S$ of every corridor, we construct a hull tree corresponding to $S$ with the algorithm in [35, 38]. For locating any vertex that is inserted to (resp. deleted from) any obstacle, and for locating query points, we compute a point location data structure for the corridor structure with the algorithm in [20].

As in the algorithm in [23], for every corridor $C$, we construct a set $P_C$ of simple polygons corresponding to corridor $C$. (Refer to Fig. 2.) One of the simple polygons in $P_C$, denoted with $P_1(C)$, is the corridor $C$ itself. This polygon helps in determining vertices of $C$ that are visible to $q$ when $q$ is located in $C$. If $q$ is not located in $C$, the other three simple polygons in $P_C$, denoted by $P_1(S_1), P_2(S_1), P_3(S_2)$, each corresponding to a side of $C$, together help in determining vertices of $C$ that are visible to $q$. In specific, two of these simple polygons $P_1(S_1), P_2(S_1)$ correspond to one side $S_1$ of $C$, and $P_3(S_2)$ correspond to the other side $S_2$ of $C$. For more details, refer to [23].

Whenever an obstacle in the polygonal domain gets modified with a vertex $v$ insertion or deletion, a side of the corridor on which $v$ resides is updated. Whenever a side $S$ of any corridor $C$ changes, we make the changes to $P_4(C)$ as well as to the simple polygons in $P_C$ that correspond to side $S$. In the query phase, using the algorithm in [25], we compute visible vertices and constructed vertices (refer to [18]) in these dynamic simple polygons. As part of preprocessing required for the algorithm in [25], for every corridor $C$, each of the four simple polygons in $P_C$ are further processed in linear time to construct data structures as required by the algorithm in [25]. Analogous to set $P_C$ of simple polygons for $\mathcal{P}$, our algorithm maintains corresponding set $P'_C$ (resp. $P'_C$) of simple polygons for the current polygonal domain $\mathcal{P}'$ (resp. the updated polygonal domain $\mathcal{P}''$).

**Lemma 1.** Given a polygonal domain $\mathcal{P}$ defined with $h$ obstacles and $n$ vertices, our preprocessing algorithm computes data structures of size $O(n)$ in $O(n \log n)^2 + h(\log h)^{1+\epsilon}$ time. Here, $\epsilon$ is a small positive constant resulting from the triangulation of free space $\mathcal{F}(\mathcal{P})$ using the algorithm in [5].
The query algorithm to construct visibility tree data structures $TVIS_B^P(q)$ and $TVIS_U^P(q)$ for the input query point $q$ among the polygonal obstacles of $P'$ is described in Section 4. The visibility polygon of $q$ is determined from the information stored at the nodes of these trees, and our dynamic algorithms update visibility trees as and when the current polygonal domain is modified with vertex insertions and vertex deletions. Our algorithms for updating the visibility polygon of a point $q \in F(P')$ are provided with the visibility trees $TVIS_B^P(q)$ and $TVIS_U^P(q)$.

The visibility trees were first defined in [29]. We modify visibility tree structures from [29] so that they are helpful in our context. Here we describe the structures from [29] together with our modifications. For every corridor $C$ that has at least one point on the boundary of $C$ that is visible to $q$, there exists at least one node in these trees that corresponds to $C$. Any node $t$ in either of these trees corresponds to a corridor $C^t$. With $t$, we store a pointer to $C^t$, a visibility cone $vc^t$ (with its apex at $q$), and two red-black balanced binary search trees (refer [14]), denoted by $RBT_L^t, RBT_R^t$. Refer to Fig. 3. The $RBT_L^t$ (resp. $RBT_R^t$) at node $t$ stores every (constructed) vertex $v^t$ of $VP_{P'}(q)$ that belongs to the left (resp. right) side of corridor $C$ whenever $v^t$ lies in $vc^t$. With each point $p$ in both of these RBTs, we store the angle ray $qp$ makes at $q$. If a point

Fig. 2. Illustrating four simple polygons in $P_C$ corresponding to a corridor $C$. (This illustration is from [23].)
Let \( p \) be any point in \( V C_{C'} \) (resp. \( V C_{C''} \)) then the line segment \( qp \) is guaranteed to intersect \( B \) (resp. \( U \)). For any point \( p \) located on the boundary of an obstacle, and for \( p \) visible to \( q \), the sequence of corridors intersected by the line segment \( qp \) is said to be the corridor sequence of \( qp \). We note that for any two points \( p', p'' \) in a corridor \( C \), with both \( p' \) and \( p'' \) visible to \( q \), the corridor sequence of \( qp' \) is not necessarily same as the corridor sequence of \( qp'' \). Hence, in any visibility tree of \( q \), there could be more than one node that corresponds to any corridor. However, any (constructed) vertex of \( V P_p(q) \) (or, any vertex of \( P' \)) appears at most once in any of the RBTs stored at the nodes of these visibility trees. For any two nodes \( t', t'' \) of any visibility tree, for any point \( p' \) stored in either \( RBT_{L}^{t'} \) or \( RBT_{R}^{t'} \), and for any point \( p'' \) stored in either \( RBT_{L}^{t''} \) or \( RBT_{R}^{t''} \), the corridor sequence of \( qp' \) is not equal to the corridor sequence of \( qp'' \).

For every corridor \( C' \), the list of visibility cones that intersect \( C' \) are stored in a red-black tree, named \( VC_{C'} \). In specific, the visibility cones in \( VC_{C'} \) are stored in sorted order with respect to angle left bounding ray of each cone in \( VC_{C'} \) makes at \( q \). In addition, with each visibility cone \( vc \) in \( VC_{C'} \), we store the pointer to a node in a visibility tree that saved a visible point \( b q(C') \cap vc \). (Refer to Fig. 3.) If no such visible point exists, then the pointer to a node in the visibility tree that represents the corridor nearest to \( p \) along \( qp \), where \( p \in C' \cap vc \), is stored with \( vc \). Whenever a vertex \( v \) is inserted (resp. deleted) to (resp. from) \( C' \), by searching in \( VC_{C'} \), we determine the visibility cone in which we need to update the visibility polygon of \( q \).

![Visibility Tree Diagram](image)

**Fig. 3.** Illustrating visibility tree (right top), \( VC_{C_1} \) (right middle) and \( SC_{t_4 t_5} \) (right bottom) data structures corresponding to a set of corridors (left).

Let \( t' \) be any node in either of the visibility trees of \( q \). And, let \( t'' \) be any child of \( t' \). Also, let \( C' \) and \( C'' \) be the corridors associated to \( t', t'' \) respectively. We note that it is not necessary for corridors \( C' \) and \( C'' \) to be adjacent in the corridor subdivision of \( F(P') \). If \( C' \) and \( C'' \) are not adjacent, then there exists a unique sequence of corridors between \( C' \) and \( C'' \), and this sequence of corridors is stored in a list \( SC_{t't''} \). The list \( SC_{t't''} \) is associated with the edge \( t't'' \) of the visibility tree. (Refer to Fig. 3.) Note that if \( C' \) and \( C'' \) are adjacent in the corridor subdivision of \( F(P') \), then the list \( SC_{t't''} \) would be empty. With every corridor \( C \in SC_{t't''} \), we
store a pointer to the node in the visibility tree that corresponding to visibility cone \(vc'\). In addition, the pointer stored with \(vc'\) in \(VCC\) points to node \(t'\).

3 Maintaining the visibility polygon of any query point

Let \(\mathcal{P}'\) be the polygonal domain just before inserting (resp. deleting) \(v\) to (resp. from) the boundary of an obstacle. Also, let \(v_i\) and \(v_{i+1}\) be the vertices between which \(v\) is located. We assume the triangle \(v_i v_{i+1} v\) is located in one corridor. In this section, we devise algorithms for updating the data structures that save the visibility polygon of any given query point when \(\mathcal{P}'\) is modified with a vertex insertion or a vertex deletion. Mainly, we update the satellite data associated with visibility trees \(TVIS^U_{\mathcal{P}'}(q)\) and \(TVIS^B_{\mathcal{P}'}(q)\). We first describe parts of the algorithm that are common to both the insertion and deletion algorithms. If \(v\) is inserted to an obstacle of \(\mathcal{P}'\), then we insert \(v\) at its corresponding position into at most three simple polygons in \(\mathcal{P}_C\); note that one simple polygon in \(\mathcal{P}_C\) correspond to the side of \(C\) to which \(v\) is incident. If \(v\) is deleted from an obstacle in \(\mathcal{P}'\), then for every simple polygon \(P \in \mathcal{P}_C\), we delete \(v\) from \(P\) if \(v \in P\). Then, for each simple polygon in \(\mathcal{P}_C\) that got modified, we update the preprocessed data structures needed for determining visibility in dynamic simple polygons [25]. Using the algorithm in [20] for ray-shooting in dynamic simple polygons, we determine whether \(v\) is visible to \(q\) among obstacles in \(\mathcal{P}'\). This is accomplished using simple polygons in \(\mathcal{P}_C\): if \(q \in C\), the ray-shooting query with ray \(qv\) is performed in \(P_C(q)\); otherwise, if \(v\) belongs to a side \(S_1\) (resp. \(S_2\)) of a corridor \(C'(\neq C)\), we query with ray \(qv\) in each simple polygon in \(\mathcal{P}_C\) that corresponds to side \(S_1\) (resp. \(S_2\)). From the correctness of characterizations in [23], it is immediate that we correctly determine whether \(v\) is visible to \(q\). If \(v\) is found to be not visible to \(q\), then \(VP_{\mathcal{P}'}(q)\) does not change. In this case, we only update the preprocessed data structures for hull trees of sides of corridor \(C\), as well as the data structures for dynamic point location. We note that all the updations of preprocessed data structures can be accomplished in \(O((\log n)^2)\) time.

Consider the case when \(v\) is visible to \(q\). In this case, the insertion of \(v\) (resp. deletion of \(v\)) may cause the deletion of (resp. insertion of) some vertices from (resp. to) the current visibility polygon \(VP_{\mathcal{P}'}(q)\). For every two vertices \(v', v'' \in \{v, v_i, v_{i+1}\}\), we determine the (smaller) angle between rays \(qv'\) and \(qv''\). Among these three possible cones, we find the cone \(vc_m\) with the maximum cone angle. The visible vertices belonging to \(vc_m\) are the potential candidates to be deleted (resp. inserted) from (resp. to) \(VP_{\mathcal{P}'}(q)\) in the insertion (resp. deletion) algorithm. Let \(BC_q\) be the lower bounding edge of corridor \(C_q\) containing \(q\). Without loss of generality, suppose \(vc_m\) intersects \(BC_q\). Noting that \(v \in C\), by searching in \(VCC\), we determine the visibility cone \(vc\) in which \(qv\) lies. We let \(t'\) be the node saved with \(vc\) in \(VCC\), and let \(C'\) be the corridor referred by \(t'\). In the following subsections, we describe the specific details of insertion and deletion algorithms.

3.1 Insertion of a vertex

If corridor \(C'\) is same as corridor \(C\) (to remind, \(v\) is in corridor \(C\)), then \(v\) is inserted to \(RBT_L^{C'}\) (resp. \(RBT_R^{C'}\)) of node \(t'\) if \(v\) is located on the left (resp. right) side of \(C'\). In the other sub-case, \(C'\) is not the same as \(C\). This indicates there is no node present in \(TVIS^B_{\mathcal{P}'}(q)\) that corresponds to \(C\) and \(vc\), i.e., before the insertion of \(v\), there was no point of \(bd(C) \cap vc\) is visible to \(q\). Let \(t'_l, t'_r\) be the left and right children of \(t'\) respectively. A new node \(t''\) is inserted as a left (resp. right) child of \(t'\), if \(v\) is located on the left (resp. right) side of \(C\), and the parent of \(t'_l\) (resp. \(t'_r\)) is changed to \(t''\). The visibility cone \(vc'\) associated with \(t''\) is same as the visibility cone \(vc\) associated with node \(t'\). Without loss of generality, suppose \(t''\) is inserted as the left child of \(t'\).
Let $C''$ be the corridor associated to node $t'_1$. The sequence of corridors $SC_{t'_1}$ associated to edge $t't'_1$ is splitted into two sequences: the corridor sequence between $C'$ to $C$ along visibility cone $vc$ is associated to edge $t't''$, and the corridor sequence between $C$ and $C''$ along the visibility cone $vc$ is saved with edge $t''t'_1$. In addition, for each corridor $C''$ in $SC_{t'_1}$, the pointer stored with visibility cone $vc$ in $VC_{C''}$ is modified so that it points to node $t''$.

![Fig. 4. Illustrating the case in which the rays bounding $vc_m$ are $qv_i$ and $qv_{i+1}$.](image1)

![Fig. 5. Illustrating the case in which rays $qv_{i+1}$ and $qv$ bound $vc_m$.](image2)

Suppose the rays bounding $vc_m$ are $qv_i$ and $qv_{i+1}$. (Refer to Fig. 4.) Then, every vertex of $VP_{p'}(q)$ continues to be visible to $q$. Hence, there is no vertex to be deleted from $VP_{p'}(q)$. However, since $v$ is visible to $q$, we need to insert $v$ into a RBT of $TVIS_{p'}^{B}(q)$. When $v$ is located on the left (resp. right) side of $C$, if $C'$ is same as $C$, then $v$ is inserted to $RBT_L^{t'}$ (resp. $RBT_R^{t'}$); otherwise, if $C'$ is not same as $C$, then vertex $v$ is inserted to $RBT_L^{t''}$ (resp. $RBT_R^{t''}$).

**Observation 1** Let $P'$ be the current polygonal domain. Let $VP_{p'}(q)$ be the visibility polygon of a point $q \in P'$. Whenever a new vertex $v$ is inserted to an obstacle of $P'$, the set of vertices of $VP_{p'}(q)$ that get hidden due to the insertion of $v$ are contiguous along the boundary of $VP_{p'}(q)$. In specific, vertices stored in any red-black tree of any visibility tree hidden due to the insertion of $v$ are contiguous at the leaves.

**Proof:** Let $S = \{v_1, v_2, \ldots, v_j\}$ be the ordered set of vertices stored at a leaf of a red-black tree in $TVIS_{p'}^{B}(q)$ (resp. $TVIS_{p'}^{L}(q)$). The observation is immediate as the vertices in $S$ that fall in $vc_m$ is a subsequence of $S$. \qed

Suppose the rays bounding $vc_m$ are $qv_{i+1}$ and $qv$. (Refer to Fig. 5.) In this case, due to the insertion of $v$, some vertices of $VP_{p'}(q)$ may become not visible to $q$. To determine these vertices, we do the depth-first traversal of $TVIS_{p'}^{B}(q)$, starting from $t'$ if $C'$ is same as $C$; otherwise, we do the depth-first traversal of $TVIS_{p'}^{B}(q)$, starting from node $t''$. Let $\alpha_1$ and $\alpha_2$ be the angles made by rays $qv_{i+1}$ and $qv$, respectively at $q$. Also, let $\alpha_1 < \alpha_2$. For every red-black tree $T$ at every node $t$ encountered in this traversal, we search in $T$ to find the contiguous list of vertices such that each vertex in that list lies in the cone $vc_m$. By Observation 1, all the vertices belonging to this list are the ones that needed to be removed from $T$. Hence, we remove each vertex $v'$ in this list from $T$, as $v'$ is no more visible to $q$. Let $C'$ be the corridor referred by node $t$. During traversal, if visibility cone $vc'$ associated with node $t$ is found to be lying completely inside $vc_m$, we delete the node corresponding to visibility cone $vc'$ in $VC_{C'}$.

The handling of the last case in which the $vc_m$ is bounded by rays $qv_i$ and $qv$ is analogous to the case in which $vc_m$ is bounded by rays $qv_{i+1}$ and $qv$.  

10
**Lemma 2.** Let $\mathcal{P}''$ be the polygonal domain resultant from inserting a vertex $v$ to an obstacle of a polygonal domain $\mathcal{P}'$. Also, let $VP_{\mathcal{P}''}(q)$ be the visibility polygon of $q$ among obstacles in $\mathcal{P}''$ determined by the algorithm. Then, any point $p \in VP_{\mathcal{P}''}(q)$ if and only if $p$ is visible to $q$ among obstacles in $\mathcal{P}''$.

**Proof:** Let a vertex $v$ be inserted on a side of corridor $C \in \mathcal{P}'$. If $v$ is visible to $q$, we search in $VC_C$ to find the visibility cone $vc$ in which $v$ is lying. A new node is inserted in the current visibility tree of $q$ only if there is no node in that visibility tree that corresponds to corridor $C$ and visibility cone $vc$. This ensures that inside a visibility cone, whenever a corridor has at least one visible point on its side, there exists a node corresponding to it in one of the visibility trees. The red-black trees stored at any node of visibility trees that corresponds to $C$ and $vc$ stores all the vertices in $\mathcal{P}'$ and the constructed vertices that belong to $C \cap vc$. Among the vertices stored in these red-black trees, vertices that are residing inside the cone $vc_m$ become not visible to $q$ after the insertion of $v$. After inserting $v$ into the current visibility tree, we traverse the updated visibility tree and determine these vertices. At every red-black tree $T$ encountered during the traversal, we search for the contiguous section of vertices (Observation 1) residing in $\mathcal{P}''$ and remove it from the current visibility polygon. It ensures that any vertex belonging to $VP_{\mathcal{P}''}(q)$ is removed from the current visibility tree only if it has become invisible due to the insertion of $v$. □

**Lemma 3.** Whenever a vertex $v$ is inserted to $\mathcal{P}'$, our algorithm updates the current visibility polygon $VP_{\mathcal{P}'}(q)$ of $q$ in $O((k)(\log |VP_{\mathcal{P}'}(q)|) + (\log n')^2 + h)$ time. Here, $k$ is the number of combinatorial changes in $VP_{\mathcal{P}'}(q)$ due to the insertion of $v$, $n'$ is the number of vertices of $\mathcal{P}'$, and $h$ is the number of obstacles in $\mathcal{P}'$.

**Proof:** Locating the corridor $C$ in which $v$ is inserted takes $O((\log n')^2)$ time. Determining whether $v$ is visible to $q$ with a ray-shooting query in each of the simple polygons in $P_C$ takes $O((\log n)^2)$ time. By searching in $VC_C$, we determine the visibility cone in which $v$ is lying. Every node in $VC_C$ points to a node in either $TVIS^B_{\mathcal{P}'}(q)$ or $TVIS^I_{\mathcal{P}'}(q)$. Since the total number of nodes in either of these trees is $O(h)$, searching $VC_C$ takes $O(\log h)$ time. Updating pointers when a new node is inserted into a visibility tree of $q$ takes $O(h)$ time. Traversing red-black trees at the nodes of visibility trees and removing the vertices not visible to $q$ takes overall $O(k)(\log |VP_{\mathcal{P}'}(q)|)$ time. Updating data structures constructed in the preprocessing phase takes $O((\log n')^2)$ time. □

### 3.2 Deletion of a vertex

Suppose the rays bounding $vc_m$ are $qv_i$ and $qv_{i+1}$. (Refer to Fig. 6.) The deletion of $v$ does not change the visibility of any vertex belonging to $\mathcal{P}'$. Hence, there is no vertex needs to be included in $VP_{\mathcal{P}'}(q)$. However, since $v$ is not going to be visible to $q$, we need to delete $v$ from node $v'$ of $TVIS^B_{\mathcal{P}'}(q)$. The vertex $v$ will be deleted from $RBT^L_{\mathcal{P}'}$ (resp. $RBT^R_{\mathcal{P}'}$) if it is located on the left (resp. right) side of $C$. We determine whether vertices $v_i$ and $v_{i+1}$ are visible to $q$. If any of them is not visible to $q$, then using the algorithm in [25] for dynamic simple polygons and with cone $vc_m$ as input, we determine the endpoints of constructed edges (refer to [18]) residing in $vc_m$ which are incident to edge $v_iv_{i+1}$. In addition, we insert these endpoints into $RBT^L_{\mathcal{P}'}$ (resp. $RBT^R_{\mathcal{P}'}$) if they are located on the left (resp. right) side of $C$.

Suppose the rays bounding $vc_m$ are $qv_i$ and $qv_{i+1}$. (Refer to Fig. 7.) In this case, due to the deletion of $v$, some new vertices of $\mathcal{P}'$ may become visible to $q$. As in the above case, $v$ is deleted from $RBT^L_{\mathcal{P}'}$ (resp. $RBT^R_{\mathcal{P}'}$) if it is located on the left (resp. right) side of $C$. Now, to determine
vertices that become visible due to the deletion of \( v \), we invoke the query algorithm described in Section 4 with \( \text{vc}_m \cap \text{vc} \) as the visibility cone and \( q \) as the query point. Let \( \text{vc}' \) be the visibility cone \( \text{vc}_m \cap \text{vc} \). Let \( \text{vc}'_l \) (resp. \( \text{vc}'_r \)) be the left (resp. right) bounding ray of \( \text{vc}' \). Also, let \( C \) be the set comprising of corridors such that the path in \( G_d \) from a node of \( G_d \) that corresponds to \( R \) to the node of \( G_d \) that corresponds to \( C_q \). Given \( C \) and \( \text{vc}' \), our query algorithm determines all the visible vertices on the sides of each corridor in \( C \) to \( q \). (Refer to Subsection 4.1.)

Let \( T' \) be the tree returned by the query algorithm, and \( T'' \) be the subtree rooted at \( t' \) in \( TVIS_{P'}^B(q) \). (Note that \( t' \) is the node from which \( v \) is deleted.) For any corridor \( C_i \), let \( VCC_i \) (resp. \( VC'_C_i \)) be the red-black tree storing the pointers to the nodes belonging to \( T_{t'} \) (resp. \( T'' \)). Without loss of generality, suppose vertex \( v \) is deleted from the left side of the corridor \( C \). Let \( r_{\text{vc}} \) be the right bounding ray of visibility cone \( v_c \). For every node \( t'' \) in \( T'' \), if the right bounding ray of its visibility cone is \( v_c', \) we replace it with the ray \( r_{\text{vc}} \). We also make similar changes in \( VCC'_i \) corresponding to the corridor \( C'' \) referred by \( t'' \).

Further, every vertex belonging to \( t'' \) is added to \( TVIS_{P'}^B(q) \). To accomplish this, we traverse the trees \( T' \) and \( T'' \) in the breadth-first order. For every node \( t'' \) that we encounter in the breadth-first traversal of \( T' \), we search in \( VCC''_i \) to find a visibility cone which is lying entirely inside the visibility cone \( v_c''' \). If such a cone exists, it indicates that node \( t'' \) is present in \( T_{t'} \). In this case, by traversing \( T_{t'} \) in breadth-first order, we locate \( t'' \) in \( T_{t'} \). Significantly, for every \( t'' \), breadth-first traversal of \( T_{t'} \) starts from the node where the traversal in that tree was stopped in the previous search. The red-black trees stored at the node \( t'' \) in \( T'' \) are merged with the red-black trees stored at the node found by the breadth-first traversal in \( T_{t'} \).

In the other sub-case, if node \( t'' \) is not present in \( T_{t'} \), it indicates that before deletion, the corridor represented by this node had no visible vertex on either of its sides in visibility cone \( v_c \). Due to the deletion of \( v \), some portion of \( bd(C_{t''}) \) became visible. Hence, \( t'' \) is inserted in \( T_{t'} \) such that the parent of \( t'' \) in \( T_{t'} \) is same as the parent of \( t'' \) in \( T'' \). At the end, for every corridor \( C_i \), we merge \( VCC'_i \) with \( VCC_i \).

The handling of the last case in which the \( \text{vc}_m \) is bounded by rays \( q_{v_i} \) and \( q \) is analogous to the case in which \( \text{vc}_m \) is bounded by \( q_{v_{i+1}} \) and \( q \).

**Lemma 4.** Let \( P'' \) be the polygonal domain resultant from deleting a vertex \( v \) from a polygonal domain \( P' \). Also, let \( VP_{P''}(q) \) be the visibility polygon of \( q \) among obstacles in \( P'' \) determined by the algorithm. Then, any point \( p \in VP_{P''}(q) \) if and only if \( p \) is visible to \( q \) in \( P'' \).

**Proof:** Let a vertex \( v \) be deleted from a side of corridor \( C \in P' \). If \( v \) is visible to \( q \), we search in \( VCC_v \) to find the visibility cone \( v_c \) in which \( v \) is lying. If the rays bounding \( v_c \) are \( q_{v_{i+1}} \) and \( q_{v_{i+1}} \), we delete \( v \) from a visibility tree and, using the algorithm in [25] for dynamic simple polygons,
Lemma 5. Whenever a vertex \( v \) is deleted from \( P' \), our algorithm updates the current visibility polygon \( VP_{P'}(q) \) in \( O(k(\log n')^2 + \log |VP_{P'}(q)|) + h \) time. Here, \( k \) is the number of combinatorial changes in \( VP_{P'}(q) \) due to the deletion of \( v \), \( n' \) is the number of vertices of \( P' \), and \( h \) is the number of obstacles in \( P' \).

Proof: Locating the corridor \( C \) in which \( v \) is inserted takes \( O((\log n')^2) \) time. Determining whether \( v \) is visible to \( q \) with a ray-shooting query in simple polygons in \( P_C \) takes \( O((\log n')^2) \) time. By searching in \( VC_C \), we determine the visibility cone in which \( v \) is lying. Every node in \( VC_C \) points to a node in either \( TVIS^B_{P'}(q) \) or \( TVIS^L_{P'}(q) \). Since the total number of nodes in either of these trees is \( O(h) \), searching \( VC_C \) takes \( O(\log h) \) time. Determining vertices visible due to the deletion of \( v \) using the query algorithm of Section 4 takes \( O(k(\log n')^2 + h) \). A breadth-first traversal of the tree returned by the query algorithm, as well as the breadth-first traversal of a visibility tree together, takes \( O(h + k) \) time. In addition, the red-black trees stored at the nodes of these trees can be updated in \( O(k\log |VP_{P'}(q)|) \) time. Searching in \( VC \) data structures and updating the visibility cone at the nodes in \( T' \) takes \( O(k\log h) \) time. Updating data structures constructed in the preprocessing phase takes \( O((\log n')^2) \) time. 

4 Determining the visibility polygon of a query point

In this section, we describe an output-sensitive algorithm to determine the visibility polygon of a query point in a dynamic polygonal domain. We modify the algorithm for answering visibility polygon queries in [23], so that the algorithm accommodates dynamic polygonal obstacles.

In the query phase, for any query point \( q \in F(P') \), we compute \( VP_{P'}(q) \). First, we determine all the sides of corridors, each of which has at least one point visible to \( q \). Recall the visibility tree data structure from Section 2. We store the vertices of the visibility polygon in two visibility trees, denoted by \( TVIS^B_{P'}(q) \) and \( TVIS^L_{P'}(q) \). For convenience, in the query algorithm, at every node in both of these visibility trees, we save an additional cone, named auxiliary visibility cone. The auxiliary visibility cone \( vc_{aux} \) defined at a node \( t \) indicates that there is an obstacle \( O \in P' \) such that (i) \( p \in bd(O) \cap vc_{aux}^t \), and (ii) \( p \) is visible to \( q \). (The specific use of auxiliary visibility cones is described in the subsections below.) As in [23], for each such side \( S \), the query algorithm computes all the vertices of \( S \) visible to \( q \). To construct these trees, we use a stack. This stack contains objects which are yet to be processed by the algorithm. Each object \( obj \) in stack is represented by a tuple \([lr_{vc}, rr_{vc}, ptr_l, ptr_r, ptr_t]\). Here, \( lr_{vc} \) (resp. \( rr_{vc} \)) is the left (resp. right) bounding ray of the visibility cone \( vc; ptr_l \) (resp. \( ptr_r \)) is a pointer to the first unexplored corridor in the corridor sequence of line segment \( qp' \) (resp. \( qp'' \)), where \( p' \) (resp. \( p'' \)) is the point...
at which \( lr_{vc} \) (resp. \( rr_{vc} \)) strikes an obstacle \( O \in P' \) or the bounding box; and, \( ptr_t \) is a pointer to the node in a visibility tree that was created at the time of initialization of \( obj \).

First, using the point location data structure, we determine the corridor \( C_q \) containing \( q \). Let \( B_{C_q} \) (resp. \( U_{C_q} \)) be the lower (resp. upper) bounding edge of \( C_q \). Using hull trees, we find the points of tangency on both sides of \( C_q \). Note that there can be at most two points of tangency on each side. For any point of tangency \( p \) that is lying on the left side (resp. right side) of \( C_q \), if the ray \( qp \) intersects \( B_{C_q} \) then point \( p \) is known as \( p^l \) (resp. \( p^r \)), and if the ray \( qp \) intersects \( U_{C_q} \) then point \( p \) is known as \( p'^l \) (resp. \( p'^r \)). (Refer to Fig. 7.)

For the current polygonal domain \( P' \), if the visibility cone \( (p^l, p'^l) \) (resp. \( (p^U, p'^U) \)) exists, then we create a node \( t^B \) (resp. \( t^U \)) as root node of \( TVIS^{B}_{P'}(q) \) (resp. \( TVIS^{U}_{P'}(q) \)). The node \( t^B \) (resp. \( t^U \)) refers to corridor \( C_q \), and the visibility cone \( (p^l, p'^l) \) (resp. \( (p^U, p'^U) \)) is associated with \( t^B \). Also we initialize \( obj^B \) (resp. \( obj^U \)) that corresponds to \( t^B \) (resp. \( t^U \)). And, \( obj^B \), followed by \( obj^U \), are pushed onto the stack.

4.1 Constructing \( TVIS^{B}_{P'}(q) \) and \( TVIS^{U}_{P'}(q) \)

The visible vertices in corridors other than corridor \( C_q \) are determined by processing objects in the stack. Let \( obj = [lr_{vc}, rr_{vc}, ptr_l, ptr_r, ptr_t] \) be the object popped from the stack. Let \( CS_l \) and \( CS_r \) be the corridor sequences of line segments \( lr_{vc} \) and \( rr_{vc} \) respectively. When both \( ptr_l \) and \( ptr_r \) refer to the same corridor \( C' \), starting from \( C' \) in \( CS_l \) and \( CS_r \), we find the last common corridor \( C''' \) that occurs in both \( CS_l \) and \( CS_r \). For every corridor \( C_i \) between \( C' \) and \( C''' \) in \( CS_l \) (or, \( CS_r \)), a node \( t' \) associated with \( ptr_l \) and \( vc \) is inserted to \( VC_{C_i} \), and \( C_i \) saves a pointer to \( t' \). (This denotes the visibility of corridor \( C_i \) is hindered by the corridor corresponding to the node pointed by \( ptr_l \).) Let \( C_l \) (resp. \( C_r \)) be the corridor after \( C''' \) in \( CS_l \) (resp. \( CS_r \)).

When the corridor \( C_l \) referred by \( ptr_l \) is different from the corridor \( C_r \) referred by \( ptr_r \), it is immediate that there is an obstacle \( O \) that separates \( C_l \) from \( C_r \). We find a tangent \( qp_t \) to the right side of \( C_l \) from \( q \) and a tangent \( qp_t \) to left side of \( C_r \) from \( q \). We insert one node \( t_l \) as the left child of \( t \) which refers to \( C_l \), and another node \( t_r \) as the right child of \( t \) which refers to \( C_r \). For every corridor \( C''' \) in the sequence of corridors from \( C' \) to \( C''' \), a node \( t'' \) with \( ptr_l \) and \( vc \) is inserted to \( VC_{C_i} \) (resp. \( VC_{C_i} \)). In addition, an auxiliary visibility cone \( (qp_l, qp_r) \) is stored at node \( t \). We initialize \( obj \) (resp. \( obj_r \)) that corresponds to \( t_l \) (resp. \( t_r \)). And, \( obj_r \) is pushed onto the stack, followed by \( obj_l \). (Refer to Fig. 9.)

If no corridor exists after \( C''' \) in \( CS_l \), then we determine the point \( p \) at which ray \( lr_{vc} \) strikes an obstacle. Let \( C_p \) be the corridor in which \( p \) is located. We find the point of tangency \( p_t \) from \( q \) to the left side of \( C_p \), using the hull tree corresponding to that side. (Refer to Fig. 10.) One new node is inserted as left child \( t_l \) of \( t \) that correspond to \( C_p \) and \( vc \). A node \( t' \) with \( ptr_l \) and \( vc \) is inserted to \( VC_{C_p} \). We initialize an object \( obj \) that corresponds to \( t_l \) and push that object onto the stack. For every corridor \( C''' \) in the sequence of corridors from \( C' \) to \( C''' \), a node \( t'' \) with \( ptr_l \) and \( vc \) is inserted to \( VC_{C_i} \). When tangent to the left side of \( C_p \) does not exist, no object is pushed onto the stack. (Refer to Fig. 11.) The algorithm for handling when there is no corridor after \( C''' \) in \( CS_r \) is analogous.
4.2 Computing visible vertices at the nodes of $TVIS^B_P(q)$ and $TVIS^U_P(q)$

As in [23], we traverse each of the visibility trees in depth-first order. At every node $t$, for each side $S$ of $C_t$, we determine vertices of $VP_{p'}(q)$ that belong to $S \cap vc^t$, by applying the algorithm in [25] to each simple polygon in $P_{C_t}$ that corresponds to $S$, with $q$ and $vc^t$ as the additional two parameters. In addition, we store visible vertices on the left side (resp. right side) of $P$ in $[25]$ to each simple polygon in $As in [23], we traverse each of the visibility trees in depth-first order. At every node $t$, we determine all the vertices that are visible in $vc^t_{aux}$. Let $t_l$ and $t_r$ be the left child and right child of $t$, respectively. Let $C_l$ be the corridor referred by node $t_l$. Let $O$ be the obstacle that lies in $vc^t_{aux}$. Also, let $C_l$ and $C_r$ respectively be the corridors corresponding to $t_l$ and $t_r$. In other words, $bd(O)$ is the union of right side $S_r$ of $C_l$ and the left side $S_l$ of $C_r$. We determine all the vertices of $S_r$ (resp. $S_l$) that are visible to $q$ and located in cone $vc^t_{aux}$, by applying the algorithm in [25] to each simple polygon in $P_{C_t}$ that corresponds to $S_r$ (resp. $S_l$), with $q$ and $vc^t_{aux}$ as the additional two parameters. In addition, we store these visible vertices in $RBT^l_R$ (resp. $RBT^l_R$).

4.3 Constructing $VP_{p'}(q)$ using $TVIS^B_P(q)$ and $TVIS^U_P(q)$

To construct $VP_{p'}(q)$, we traverse both the visibility trees in depth-first order. First we traverse $TVIS^B_P(q)$ followed by $TVIS^U_P(q)$. At every node $t$ encountered during the traversal, we traverse the leaf nodes of $RBT^l_R$ in left to right order and output the respective points stored at them. Then, we recursively traverse the left subtree of $t$, followed by the right subtree of $t$. After that, we traverse the leaf nodes of $RBT^l_R$ in the right to left order and output the respective points stored at them. Let $S_{df}$ be the ordered set of points obtained from the depth-first traversal of $TVIS^B_P(q)$ followed by $TVIS^U_P(q)$ as described above. The sequence of edges obtained by joining the consecutive points in the ordered set $S_{df}$, followed by the edge joining the first and last point in $S_{df}$, is the boundary of visibility polygon of query point $q$.

**Lemma 6.** Let $P'$ be a polygonal domain. Also, let $VP_{p'}(q)$ be the visibility polygon of $q$ among obstacles in $P'$ determined by the algorithm. Then, any point $p \in VP_{p'}(q)$ if and only if $p$ is visible to $q$ in $P'$.

**Proof:** Consider a point $p$ that is visible to $q$. The point $p$ may be a vertex of $P'$, or it may be a visible point on an edge of $P'$ that will appear as an endpoint of a constructed edge of $VP_{p'}(q)$. As the algorithm starts, we determine the corridor $C_q$ containing $q$. Let $BC_q$ (resp.
Given a polygonal domain $\mathcal{P}$ defined with $h$ obstacles and $n$ vertices, we preprocess $\mathcal{P}$ in $O(n(\lg n)^2 + h(\lg h)^{1+\epsilon})$ time to construct data structures of size $O(n)$ so that (i) whenever a vertex $v$ is inserted to the current polygonal domain $\mathcal{P}'$, the algorithm updates data structures that store visibility polygon $V_{P'}(q)$ of a query point $q$ in $O(k(\lg |V_{P'}(q)|) + (\lg n')^2 + h)$ time, (ii) whenever a vertex $v$ is deleted from the current polygonal domain $\mathcal{P}'$, the algorithm updates data structures that store visibility polygon $V_{P'}(q)$ of a query point $q$ in $O((\lg n')^2 + (\lg |V_{P'}(q)|) + h)$ time, and (iii) whenever a query point $q$ is given, the algorithm outputs the visibility polygon in the current polygonal domain in $O(|V_{P'}(q)|((\lg n')^2 + h)$ time. Here,
\( \epsilon \) is a small positive constant resulting from the triangulation of the free space \( \mathcal{F}(\mathcal{P}) \) using the algorithm in [5], \( k \) is the number of combinatorial changes in \( V\mathcal{P}_{P'}(q) \) due to the insertion or deletion of \( v \), and \( n' \) is the number of vertices of \( \mathcal{P}' \).

5 Maintaining the visibility graph

In this section, we describe an algorithm to maintain the visibility graph among dynamic polygonal obstacles in the plane. We first detail the preprocessing algorithm with the input polygonal domain \( \mathcal{P} \) with \( n \) vertices and \( h \) polygonal obstacles. Our algorithm relies on the algorithm for maintaining the visibility polygon among dynamic polygonal obstacles in Section 2. Hence, we compute all the data structures as required in Lemma 1. In addition, we need the following data structures. Using [29], we construct the visibility graph of \( \mathcal{P} \) in \( O(|E| + h \log n + (\log h)^{1+\epsilon}) \) time. Here, \( E \) is the number of edges in the visibility graph of \( \mathcal{P} \). For every vertex \( v \in \mathcal{P} \), we construct a red-black tree (detailed in [14]), denoted by \( RBT_v \), that contains all the visible edges that are incident to \( v \). Every leaf of \( RBT_v \) represents a unique visible edge that is incident to \( v \). For every visible edge that is determined to be incident to \( v \), we insert it to \( RBT_v \) in \( O(\log |E|) \) time. With every visible edge, \( RBT_v \) stores the angle it makes with the positive \( x \)-axis. The left to right order of visible edges stored at the leaves of \( RBT_v \) is the sorted order of visible edges incident to \( v \) with respect to angle each makes with the positive \( x \)-axis. Whenever a vertex is inserted to or deleted from any of the obstacles of the current polygonal domain \( \mathcal{P}' \), as part of updating the visibility graph of \( \mathcal{P}' \), we update these red-black tree data structures as well as the data structures for visibility polygon maintenance (refer to Theorem 1) and return the updated visibility graph.

Lemma 8. Given a polygonal domain \( \mathcal{P} \) defined with \( h \) holes and \( n \) vertices, as part of preprocessing \( \mathcal{P} \), in \( O(n(\log n)^2 + h(\log h)^{1+\epsilon} + |E|\log |E|) \) time, our algorithm constructs data structures of size \( O(n + |E|) \). Here, \( |E| \) is the number of edges in the visibility graph of \( \mathcal{P} \).

Let \( \mathcal{P}' \) be the polygonal domain before inserting \( v \) to (resp. deleting \( v \) from) the boundary of an obstacle. Also, let \( v_i \) and \( v_{i+1} \) be the vertices between which \( v \) is located. We first describe parts that are common to both the insertion and deletion algorithms. Using point location data structure, we determine the corridor \( C \) in which \( v \) is located. If \( v \) is inserted to an obstacle of \( \mathcal{P}' \), then we insert \( v \) at its corresponding position into at most three simple polygons in \( P_C' \) wherein each of these simple polygons has both \( v_i \) and \( v_{i+1} \). If \( v \) is deleted from an obstacle of \( \mathcal{P}' \), then for every simple polygon \( P \in P_C' \), we delete \( v \) from \( P \) if \( v \in P \). In addition, for each simple polygon in \( P_C' \) that got modified, we update the preprocessed data structures needed to determine the visibility in dynamic simple polygons using the algorithm in [25].

Let \( v' \) be any vertex distinct from \( v, v_i, \) and \( v_{i+1} \). Using the algorithm in [20] for ray-shooting in dynamic simple polygons, we determine whether \( v \) is visible to \( v' \) among obstacles in \( \mathcal{P}' \). This is accomplished using simple polygons in \( P_C' \): if \( v' \in C \), the ray-shooting query with ray \( v'v \) is performed in \( P_C'(C) \); otherwise, if \( v \) belongs to a side \( S_1 \) (resp. \( S_2 \)) of a corridor \( C'(\neq C) \), we query with ray \( v'v \) in each simple polygon in \( P_C' \) that corresponds to side \( S_1 \) (resp. \( S_2 \)). From the correctness of characterizations in [23], it is immediate that we correctly determine whether \( v \) is visible to \( v' \). If \( v \) is found to be not visible to \( v' \), then \( V\mathcal{P}_{P'}(v') \) does not change. In this case, we only update the preprocessed data structures for hull trees of sides of corridor \( C \) and the data structures for dynamic point location. We note that all the updatings of the preprocessed data structures can be accomplished in \( O((\log n)^2) \) worst-case time. Consider the case when \( v \) is visible to \( v' \). In this case, the insertion of \( v \) (resp. deletion of \( v \)) may cause the deletion of (resp. insertion of) edges from (resp. to) \( V\mathcal{G}_{P'} \) that are incident to \( v' \) and intersect the triangle \( vv_i v_{i+1} \). In addition, in the case of insertion of \( v \), we introduce edge \( v'v \) into \( V\mathcal{G}_{P'} \).
The following lemma statement is simplified from the one in Choudhury and Inkulu [13], and for completeness and to show the proof extends to polygonal domains, we include a proof. This lemma helps in efficiently determining the set of edges that intersect the triangle $v_iv_{i+1}$.

**Lemma 9.** Let $v_i, v$, and $v_{i+1}$ be consecutive vertices along an obstacle of $P'$. For every visible edge $tu$ of $VG_{P'}$ that intersects the triangle $vv_iv_{i+1}$, either (a) at least one of $t$ or $u$ is visible to $v$, or (b) there exists a path $R$ in $VG_{P'}$ from a vertex $v'$ to $t$ (resp. $u$) such that each edge in $R$ intersects the triangle $v_iv_{i+1}$ and $v'$ is visible to $v$.

**Proof:** If either $t$ or $u$ is visible to $v$, there is nothing to prove. Otherwise, among all the edges of $VG_{P'}$ that intersect both the triangles $vv_iv_{i+1}$ and $tuv$, let $t'u'$ be a visible edge in $P'$ that makes the least angle with the line segment $tu$ at either $t' = t$ or $u' = u$. Without loss of generality, we suppose $u' = u$. (Refer to Fig. 12.) Note that both $t$ and $u$ are reachable from $t'$ in $VG_{P'}$. Besides, both the edges $tu$ and $ut$ intersect the triangle $vv_iv_{i+1}$. Further, $t'$ is closer to $v$ as compared to $t$, with respect to Euclidean distance. Inductively, $t'$ is reachable from $v$ in $VG_{P'}$ as described in the lemma statement. Since there are finite vertices, there exists some edge $t''u''$ such that $t''u''$ intersects $vv_iv_{i+1}$ and either $t''$ or $u''$ is visible to $v$. $\Box$

![Fig. 12. Illustrating the notation in Lemma 9 statement. The path $R$ is shown in blue color. The dotted line between $v$ and $v'$ illustrates an endpoint $v'$ of $R$ is visible to $v$. (This illustration is from [13]).](image)

In the following subsections, we describe more details of insertion and deletion algorithms.

### 5.1 Insertion of a vertex

Let $v_i$ and $v_{i+1}$ be the vertices between which the vertex $v$ is inserted. Also, let $S$ be the set of visible edges that intersect the triangle $vv_iv_{i+1}$. We observe that any visible edge $e \in VG_{P'}$ appears in the updated visibility graph $VG_{P''}$ if and only if $e$ does not belong to set $S$. First, we determine vertices that are visible to $v$. To accomplish this, we invoke the visibility polygon query algorithm among dynamic obstacles detailed in Section 3, with $v$ as the query point. A new red-black tree, denoted by $RBT_v$ is initiated and, for every vertex $v' \in P'$ determined by the visibility query algorithm, we insert the visibility edge $vv'$ into $RBT_v$ as well into $RBT_{vc}$. This completes the insertion of new visible edges to the visibility graph $VG_{P'}$. Next, we determine the set $S$ of visible edges of $VG_{P'}$ whose endpoints have become not visible to each other after the insertion of $v$. To efficiently determine edges in $S$, we use Lemma 9 in iteratively finding all the vertices to which visible edges to be removed from $VG_{P''}$ are incident. Let $S_1$ be the set comprising of all the vertices of $P'$ visible to $v$. The points in $S_1$ are determined from the visibility polygon of $v$. For each vertex $v' \in S_1$, we determine all the vertices $v''$ such that $v'v''$ is a visible edge in $VG_{P'}$ and it intersects the triangle $vv_iv_{i+1}$. Without loss of generality, suppose the ray $v'v$ makes larger angle with $v''v$ as compared to $v'v_{i+1}$. Let $vc_m$ be the cone with rays $v'v$ and $v'v_i$ bounding it. The visible edges lying in cone $vc_m$ are the potential candidates to be deleted from $VG_{P'}$. We search in $RBT_v$ to find every vertex $v''$ such that the edge $v'v''$ belongs to cone $vc_m$, and delete the edge $v'v''$ from both $RBT_v$ and $RBT_{vc}$. In specific, for every $j \geq 1$, let $S_{j+1}$ be equal to $\bigcup_{v' \in S_j} S_{v'}$. Following the same procedure, for every $j > 1$, for every vertex $v' \in S_j$ and $v' \notin S_j'$ with $j' < j$, we find the set $S_{j+1}$ comprising of all the visible edges in $P'$.
that are incident to $v'$ and intersect triangle $vv_iv_{i+1}$. If at any point, $S_j \subseteq S_{j'}$ for some $j' < j$, we terminate the algorithm. (Refer to Fig. 13.)

![Fig. 13. Illustrating the insertion of vertex $v$ (left), the data structure $RBT_{v_1}$ before insertion of $v$ (middle), and the data structure $RBT_{v_1}$ after inserting $v$ (right).](image)

**Lemma 10.** In the updated polygonal domain $P''$, any edge $pq \in VG_{P''}$ if and only if $p$ and $q$ are mutually visible among obstacles in $P''$.

**Proof:** All the visible edges incident to $v$ are computed correctly using the algorithm from [29]. For every vertex $v'$ that is an endpoint of a visible edge incident to $v$, we do binary search in $RBT_{v'}$ to prune visible edges incident to $v'$ that intersect with triangle $vv_iv_{i+1}$ using the cone $vc_m$. Following Lemma 9, these edges are computed iteratively. It ensures that an edge is removed from $VG_{P'}$ only if it intersects the triangle $vv_iv_{i+1}$. Further, by the correctness of Lemma 9, it is guaranteed that every visible edge that is intersecting the triangle $vv_iv_{i+1}$ is removed from $VG_{P'}$.

**Lemma 11.** Whenever a vertex $v$ is inserted to $P'$, our algorithm updates the visibility graph $VG_{P''}$ in $O(k(lg n')^2 + h)$ time. Here, $k$ is the number of combinatorial changes in $VG_{P'}$ due to the insertion of $v$, $n'$ is the number of vertices of $P'$, and $h$ is the number of obstacles in $P'$.

**Proof:** From Theorem 1, finding all the vertices of $P'$ that are visible to $v$ takes $O(k_1(lg n')^2 + h)$ time. Here, $k_1$ is the number of vertices in $P'$ that are visible to $v$. Inserting visible edges corresponding to these vertices in $VG_{P''}$ takes $O(k_1(lg n'))$ time. To recursively determining edges in $S$, search in red-black trees stored at the endpoints of edges in $S$, and to delete edges in $S$ together takes $O(k_2(lg n'))$ time, where $k_2 = k - k_1$.

### 5.2 Deletion of a vertex

Let $v$ be a vertex in $P'$. Also, let $v_i$ and $v_{i+1}$ be the vertices adjacent to $v$ along the boundary of an obstacle in $P'$. Further, let $VG_{P'}$ be the visibility graph of $P'$. Suppose the vertex $v$ is deleted. Then, the deletion algorithm updates $VG_{P''}$. For line segment joining any two vertices $v', v''$ in $P''$, if $v'v''$ intersects the triangle $vv_iv_{i+1}$, edge $v'v''$ needs to be introduced into $VG_{P''}$. In this algorithm, like in the insertion algorithm, we use Lemma 9 to efficiently determine the visible edges in $VG_{P''}$ that does not belong to $VG_{P'}$.
Let $S_1$ be the set of vertices of $P'$ visible to $v$. For each vertex $v' \in S_1$, we determine all the vertices $v''$ such that edge $v'v''$ intersects triangle $v_{i+1}v_i$. Without loss of generality, suppose ray $v'v_i$ makes a larger angle with ray $v'v$ as compared to ray $v'v_{i+1}$. Let $vc_m$ be the cone with rays $v'v$ and $v'v_i$ bounding it. The visible edges lying in cone $vc_m$ are the potential candidates to be inserted to $VG_{P'}$. We invoke the query algorithm in Section 3 with query point $v'$ and the visibility cone $vc_m$. As a result, our algorithm finds every vertex $v'' \in P''$ that is visible to $v'$ such that the edge $v'v''$ resides in cone $vc_m$. The edge $v'v''$ is inserted as a new visible edge into both $RBT_v$ and $RBT_{v'}$. In specific, for every $j \geq 1$, let $S_{j+1}$ be equal to $\bigcup_{v' \in S_j} S_{v'}$. Following the same procedure, for every $j > 1$, for every vertex $v' \in S_j$ and $v' \notin S_j'$ with $j' < j$, we find the set $S_{j+1}$ comprising of all the visible edges in $P''$ that are incident to $v'$ and intersect the triangle $v_{i+1}v_i$. If at any point, $S_j \subseteq S_j'$ for some $j' < j$, we terminate the algorithm. In the end, we delete $RBT_v$ as part of removing all the visible edges of $VG_{P'}$ that are incident to $v$. (Refer to Fig. 14.)

![Fig. 14](image-url)

Fig. 14. Illustrating the vertex $v$ to be deleted (left), the data structure $RBT_v$ before deleting $v$ (middle), and the data structure $RBT_v$ after deleting $v$ (right).

**Lemma 12.** In any updated polygonal domain $P''$, any edge $pq \in VG_{P''}$ if and only if $p$ and $q$ are mutually visible among obstacles in $P''$.

**Proof:** Let a vertex $v$ is deleted from the boundary of an obstacle in $P'$. For every vertex $v' \in P'$ that is visible to $v$, the edge $vv'$ is deleted from $VG_{P'}$. No other edge needs to be deleted from the current visibility graph. For any two vertices $v'v''$, if the edge intersect the triangle $v_{i+1}v_i$, it should be added as a visible edge in $VG_{P'}$. Following Lemma 9, these edges are computed iteratively. In each iteration, for every vertex $v'$, we invoke the query algorithm from Section 4 with $v'$ as the query point and $vc_m$ as the visibility cone. By the correctness of Lemma 9 and Lemma 7, it is guaranteed that every visible edge that is intersecting the triangle $v_{i+1}v_i$ is determined correctly and added to $VG_{P'}$. □

**Lemma 13.** Whenever a vertex $v$ is deleted from $P'$, our algorithm updates the visibility graph $VG_{P''}$ in $O(k(\lg n')^2 + h)$ time. Here, $k$ is the number of combinatorial changes in $VG_{P''}$ due to the deletion of $v$, $n'$ is the number of vertices of $P''$, and $h$ is the number of obstacles in $P'$.

**Proof:** All the visible edges in $VG_{P'}$ that are incident to $v$ are deleted in $O(k_1 \lg n')$ time. Here, $k_1$ is the number of visible edges incident to $v$. From Theorem 1, all the invocations of the visibility polygon query algorithm together takes $O(k_2((\lg n')^2 + h))$ time, where $k_2 = k - k_1$ is the number of visible edges in $VG_{P''}$ that are determined with this algorithm. Inserting the
newly found visible edges into corresponding red-black trees takes $O(k_2 \lg n')$ time. \hfill \Box

**Theorem 2.** Given a polygonal domain $\mathcal{P}$ defined with $h$ obstacles and $n$ vertices, we preprocess $\mathcal{P}$ in $O(n(\lg n)^2 + h(\lg h)^{1+\epsilon} + |E| \lg |E|)$ time to construct data structures of size $O(n + |E|)$ so that whenever a vertex $v$ is inserted to (resp. deleted from) the current polygonal domain $\mathcal{P}'$, the algorithm updates the data structures that save visibility graph of $\mathcal{P}'$ in $O(k(\lg n')^2 + h)$ (resp. $O(k((\lg n')^2 + h))$) time. Here, $\epsilon$ is a small positive constant resulting from the triangulation of the free space $\mathcal{F}(\mathcal{P})$ using the algorithm in [5], $|E|$ is the number of edges present in the visibility graph of $\mathcal{P}$, $k$ is the number of combinatorial changes in updating the visibility graph of $\mathcal{P}'$ due to the insertion (resp. deletion) of $v$, and $n'$ is the number of vertices of $\mathcal{P}'$.

6 Conclusions

We proposed algorithms for the following problems among dynamic polygonal obstacles in the plane: (i) dynamically updating the visibility polygon of any given query point, (ii) answering visibility polygon queries, and (iii) dynamically maintaining the visibility graph of the polygonal domain. An immediate extension of this algorithm would handle new obstacles being introduced into the polygonal domain as well as allowing the newly inserted vertex to be in a corridor different from the corridor in which its neighbors reside. Further, it would be interesting to devise efficient dynamic algorithms for these problems so that the update time complexities mainly depend on the number the combinatorial changes to update the visibility polygon. In specific, as other parameters are under logarithm, this involves removing the dependency on the number of obstacles in the polygonal domain from the update time complexities.

Acknowledgement

This research of R. Inkulu is supported in part by SERB MATRICS grant MTR/2017/000474. The authors also like to acknowledge reviewers of the preliminary version of this paper for their valuable input.

References

[1] S. Agrawal and R. Inkulu. Visibility polygon queries among dynamic polygonal obstacles in plane. In Proceedings of International Computing and Combinatorics Conference, pages 136–148, 2020.
[2] K. Akbari and M. Ghodsi. Visibility maintenance of a moving segment observer inside polygons with holes. In Proceedings of Canadian Conference on Computational Geometry, pages 117–120, 2010.
[3] B. Aronov, L. J. Guibas, M. Teichmann, and L. Zhang. Visibility queries and maintenance in simple polygons. Discrete & Computational Geometry, 27(4):461–483, 2002.
[4] T. Asano, T. Asano, L. J. Guibas, J. Hershberger, and H. Imai. Visibility of disjoint polygons. Algorithmica, 1(1):49–63, 1986.
[5] R. Bar-Yehuda and B. Chazelle. Triangulating disjoint jordan chains. International Journal of Computational Geometry & Applications, 4(4):475–481, 1994.
[6] M. N. Baygri and M. Ghodsi. Space/query-time tradeoff for computing the visibility polygon. Computational Geometry, 46:371–381, 2013.
[7] P. Bose, A. Lubiw, and J. I. Munro. Efficient visibility queries in simple polygons. Computational Geometry, 23(3):313–335, 2002.
[8] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete & Computational Geometry*, 6:485–524, 1991.
[9] D. Z. Chen and O. Daescu. Maintaining visibility of a polygon with a moving point of view. *Information Processing Letters*, 65(5):269–275, 1998.
[10] D. Z. Chen, R. Inkulu, and H. Wang. Two-point $L_1$ shortest path queries in the plane. *Journal of Computational Geometry*, 7(1):473–519, 2016.
[11] D. Z. Chen and H. Wang. A new algorithm for computing visibility graphs of polygonal obstacles in the plane. *Journal of Computational Geometry*, 6(1):316–345, 2015.
[12] D. Z. Chen and H. Wang. Visibility and ray shooting queries in polygonal domains. *Computational Geometry*, 48(2):31–41, 2015.
[13] T. Choudhury and R. Inkulu. Maintaining the visibility graph of a dynamic simple polygon. In *Proceedings of Conference on Algorithms and Discrete Applied Mathematics*, pages 42–52, 2019.
[14] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2009.
[15] L. S. Davis and M. L. Benedikt. Computational models of space: isovists and isovist fields. *Computer Graphics and Image Processing*, 11(1):49–72, 1979.
[16] H. A. ElGindy and D. Avis. A linear algorithm for computing the visibility polygon from a point. *Journal of Algorithms*, 2(2):186–197, 1981.
[17] S. K. Ghosh. Computing the visibility polygon from a convex set and related problems. *Journal of Algorithms*, 12(1):75–95, 1991.
[18] S. K. Ghosh. *Visibility algorithms in the plane*. Cambridge University Press, New York, USA, 2007.
[19] S. K. Ghosh and D. M. Mount. An output-sensitive algorithm for computing visibility graphs. *SIAM Journal on Computing*, 20(5):888–910, 1991.
[20] M. T. Goodrich and R. Tamassia. Dynamic ray shooting and shortest paths in planar subdivisions via balanced geodesic triangulations. *Journal of Algorithms*, 23(1):51–73, 1997.
[21] P. J. Heffernan and J. S. B. Mitchell. An optimal algorithm for computing visibility in the plane. *SIAM Journal on Computing*, 24(1):184–201, 1995.
[22] J. Hershberger. An optimal visibility graph algorithm for triangulated simple polygons. *Algorithmica*, 4:141–155, 1989.
[23] R. Inkulu and S. Kapoor. Visibility queries in a polygonal region. *Computational Geometry*, 42(9):852–864, 2009.
[24] R. Inkulu and T. Nitish. Incremental algorithms to update visibility polygons. In *Proceedings of Conference on Algorithms and Discrete Applied Mathematics*, pages 205–218, 2017.
[25] R. Inkulu, K. Sowmya, and N. P. Thakur. Dynamic algorithms for visibility polygons in simple polygons. *International Journal of Computational Geometry & Applications*, 30(1):51–78, 2020.
[26] B. Joe and R. Simpson. Corrections to Lee’s visibility polygon algorithm. *BIT Numerical Mathematics*, 27(4):458–473, 1987.
[27] S. Kapoor. Efficient computation of geodesic shortest paths. In *Proceedings of Symposium on Theory of Computing*, pages 770–779, 1999.
[28] S. Kapoor and S. N. Maheshwari. Efficient algorithms for Euclidean shortest path and visibility problems with polygonal obstacles. In *Proceedings of Symposium on Computational Geometry*, pages 172–182, 1988.
[29] S. Kapoor and S. N. Maheshwari. Efficiently constructing the visibility graph of a simple polygon with obstacles. *SIAM Journal on Computing*, 30(3):847–871, 2000.
[30] S. Kapoor, S. N. Maheshwari, and J. S. B. Mitchell. An efficient algorithm for Euclidean shortest paths among polygonal obstacles in the plane. Discrete & Computational Geometry, 18(4):377–383, 1997.

[31] D. T. Lee. Proximity and reachability in the Plane. PhD thesis, University of Illinois at Urbana-Champaign, 1978. Ph.D. thesis and Technical Report ACT-12.

[32] D. T. Lee. Visibility of a simple polygon. Computer Vision, Graphics, and Image Processing, 22(2):207–221, 1983.

[33] L. Lu, C. Yang, and J. Wang. Point visibility computing in polygons with holes. Journal of Information and Computational Science, 16(7):4165–4173, 2011.

[34] J. O’Rourke. Art Gallery Theorems and Algorithms. Oxford University Press, Inc., New York, USA, 1987.

[35] M. H. Overmars and J. van Leewen. Maintenance of configurations in the plane. Journal of Computer and System Sciences, 23(2):166–204, 1981.

[36] M. H. Overmars and E. Welzl. New methods for computing visibility graphs. In Proceedings of the Fourth Annual Symposium on Computational Geometry, pages 164–171, 1988.

[37] M. Pocchiola and G. Vegter. Topologically sweeping visibility complexes via pseudotriangulations. Discrete & Computational Geometry, 16(4):419–453, 1996.

[38] F. P. Preparata and M. I. Shamos. Computational Geometry: an introduction. Springer-Verlag, New York, USA, 1985.

[39] M. Sharir and A. Schorr. On shortest paths in polyhedral spaces. SIAM Journal on Computing, 15(1):193–215, 1986.

[40] S. Suri and J. O’Rourke. Worst-case optimal algorithms for constructing visibility polygons with holes. In Proceedings of the Symposium on Computational Geometry, pages 14–23, 1986.

[41] E. Welzl. Constructing the visibility graph for $n$-line segments in $O(n^2)$ time. Information Processing Letters, 20(4):167–171, 1985.

[42] A. Zarei and M. Ghodsi. Efficient computation of query point visibility in polygons with holes. In Proceedings of the Symposium on Computational Geometry, pages 314–320, 2005.