Dynamic algorithms for visibility polygons

R. Inkulu† K. Sowmya

Abstract

We devise dynamic algorithms for the following (weak) visibility polygon computation problems:

- Maintaining visibility polygon of a fixed point located interior to simple polygon amid vertex insertions and deletions to simple polygon.
- Answering visibility polygon query corresponding to any point located exterior to simple polygon amid vertex insertions and deletions to simple polygon.
- Maintaining weak visibility polygon of a fixed line segment located interior to simple polygon amid vertex insertions to simple polygon.
- Answering weak visibility polygon query corresponding to any line segment located interior to simple polygon amid both vertex insertions and deletions to simple polygon.
- Maintaining visibility polygon of a fixed point located in the free space of the polygonal domain amid vertex insertions to simple polygons in that polygonal domain.

The proposed algorithms are output-sensitive, and the time complexities of algorithms for (weak) visibility polygon maintenance are expressed in terms of change in output complexity.

1 Introduction

Let $P$ be a simple polygon with $n$ vertices. Two points $p, q \in P$ are said to be mutually visible to each other whenever the interior of line segment $pq$ does not intersect any edge of $P$. For a point $q \in P$, the visibility polygon $VP(q)$ of $q$ is the maximal set of points $x \in P$ such that $x$ is visible to $q$. The problem of computing the visibility polygon of a point in a simple polygon was first attempted in [9], who presented an $O(n^2)$ time algorithm. Then, ElGindy and Avis [10] and Lee [22] presented $O(n)$ time algorithms for this problem. Joe and Simpson [21] corrected a flaw in [10, 22] and devised an $O(n)$ time algorithm that correctly handles winding in the simple polygon.

For a polygon with holes, Suri et al. devised an $O(n \log n)$ time algorithm in [26]. An optimal $O(n + h \log h)$ time algorithm was given in Heffernan and Mitchell [18]. Algorithms for visibility computation amid convex sets were devised in Ghosh [11]. The preprocess-query paradigm based algorithms were studied in [17, 4, 13, 27, 28, 19, 8, 7]. Algorithms for computing visibility graphs were given in [13]. For a line segment $pq \in P$, the weak visibility polygon $WVP(pq)$ is the maximal set of points $x \in P$ such that $x$ is visible from at least one point belonging to line segment $pq$. Chazelle and Guibas [6], and Lee and Lin [23] gave $O(n \log n)$ time algorithms for computing the weak visibility polygon of a line segment located interior to the given simple polygon. Later, Guibas

*This research is supported in part by NBHM grant 248(17)2014-R&D-II/1049
†Department of Computer Science & Engineering; IIT Guwahati, India, \{rinkulu,k.sowmya\}@iitg.ac.in
et al. [16] gave an $O(n)$ time algorithm for the same. The query algorithms for computing weak visibility polygons were devised in [4, 5, 11]. Ghosh [12] gives a detailed account of visibility related algorithms. Given a simple polygon $P$ and a point $p$ (resp. line segment) interior to $P$, algorithms devised in [20] maintain the visibility polygon of $p$ (resp. weak visibility polygon of $p$) as vertices are added to $P$. To our knowledge, [20] gives the first dynamic (incremental) algorithms in the context of maintaining visibility and weak visibility polygons.

**Our contribution**

In the context of computing visibility/weak visibility polygons, an algorithm is termed *fully-dynamic* when it maintains the visibility/weak visibility polygon as the polygonal domain is updated with vertex insertions and vertex deletions. When a new vertex is added to the current simple polygon or when a vertex of the current simple polygon is deleted, the visibility polygon of a fixed point is updated. An algorithm is *incremental* (resp. *decremental*) if it maintains the visibility/weak visibility polygon amid vertex insertions (resp. vertex deletions). The following dynamic algorithms are devised for maintaining as well as querying for visibility and weak visibility polygons. In each of these algorithms, we first preprocess the given simple polygon or polygonal domain defined with $n'$ vertices in $O(n')$ time to compute $O(n')$ sized data structures. Further, for every vertex insertion/deletion, preprocessed data structures are updated in $O((\log n)^2)$ time where $n$ is the number of vertices that define the current simple polygon or polygonal domain.

* Our first algorithm (refer Subsection 2.1) maintains the visibility polygon of a fixed point $q$ located interior to a simple polygon. When a vertex $v$ is added to the current simple polygon $P$, we update the visibility polygon of $q$ in $O((\log n)^2)$ time; when a vertex $v$ is deleted, the visibility polygon is updated in $O((k+1)(\log n)^2)$ time, where $k$ is the change in representation complexity of the visibility polygon of $q$ in deleting $v$ from $P$ and $n$ is the number of vertices of $P$. (The deletion algorithm takes $O((\log n)^2)$ time to update even when $k$ is zero, hence the stated time complexity.)

Further, in $O(k(\log n)^2)$ time our algorithm outputs visibility polygon of any query point $q$ interior to the current simple polygon, where $k$ is the output complexity.

* Our second algorithm (refer Subsection 2.2) answers visibility polygon queries of a point exterior to simple polygon amid vertex insertions and deletions to simple polygon. The query algorithm takes $O(k(\log n)^2)$ time, where $k$ is the output complexity and $n$ is the number of vertices of the current simple polygon.

* Our third algorithm (refer Subsection 2.3) answers weak visibility polygon queries amid vertex insertions and deletions. The query algorithm takes $O(k(\log n)^2)$ time, where $k$ is the output complexity and $n$ is the number of vertices of the current simple polygon.

* Our fourth algorithm (refer Subsection 2.4) maintains the weak visibility polygon of a fixed edge $pq$ of a simple polygon with vertex insertions. When a vertex $v$ is added to the current simple polygon $P$, we update the weak visibility polygon of $pq$ in $O((k+1)\log n)$ time where $k$ is the sum of changes in combinatorial complexities of $SPT(p)$ and $SPT(q)$. Here, $SPT(p)$ (resp. $SPT(q)$) is the shortest path tree rooted at $p$ (resp. $q$). (Like in the case of first algorithm, deletion algorithm here takes $O((\log n)^2)$ time to update even when $k$ is zero, hence the stated time complexity.)
In this Subsection, we consider the problem of updating vertex insertion/deletion; and, number of vertices of \( P \). 2.1 Maintaining visibility polygon of \( P \) (resp. weak visibility polygon) is updated remains interior to edge \( u \) polygon (resp. weak visibility polygon), the point \( q \) successive vertices along the boundary of the current simple polygon. In maintaining visibility polygon remains simple. Moreover, it is assumed that every new vertex is added between two constructed edges of \( \text{VP}(P) \). Then such an edge \( \text{VP}(P) \) whose visibility polygon (resp. weak visibility polygon) is updated remains interior to \( P \) if it was interior to \( P \) before the vertex insertion/deletion; and, \( q \) (resp. \( l \)) remains exterior if it was exterior to \( P \).

2 Dynamic visibility of simple polygons

We use the following notation and assumptions. The boundary of a simple polygon \( P \) is denoted with \( \text{bd}(P) \). We assume that after adding/deleting any vertex of the current simple polygon, the polygon remains simple. Moreover, it is assumed that every new vertex is added between two successive vertices along the boundary of the current simple polygon. In maintaining visibility polygon (resp. weak visibility polygon), the point \( q \) (resp. line segment \( l \)) whose visibility polygon (resp. weak visibility polygon) is updated remains interior to \( P \) if it was interior to \( P \) before the vertex insertion/deletion; and, \( q \) (resp. \( l \)) remains exterior if it was exterior to \( P \).

2.1 Maintaining visibility polygon of \( q \) when \( q \in P \)

In this Subsection, we consider the problem of updating \( \text{VP}(q) \) when a new vertex is inserted to \( P \) or an existing vertex is deleted from \( P \), resulting in a new simple polygon \( P' \). We let \( n \) be the number of vertices of \( P \). Let \( u_iu_{i+1} \) be an edge on the boundary of \( \text{VP}(q) \) such that (i) no point of \( u_iu_{i+1} \), except the points \( u_i \) and \( u_{i+1} \), belong to the boundary of \( P \), and (ii) one of \( u_i \) or \( u_{i+1} \) is a vertex of \( P \). Then such an edge \( u_iu_{i+1} \) is called a constructed edge. For every constructed edge \( u_iu_{i+1} \), among \( u_i \) and \( u_{i+1} \) the farthest from \( q \) is termed a constructed vertex of \( \text{VP}(q) \). The constructed edges of \( \text{VP}(q) \) partition \( P \) into a set \( R = \{\text{VP}(q), R_1, R_2, \ldots, R_k\} \) of simple polygonal regions such that no point \( p \) interior to region \( R \) is visible from \( q \) for any \( R \in R \) and \( R \neq \text{VP}(q) \). Also, for each \( R \in R \) and \( R \neq \text{VP}(q) \), there exists a constructed vertex associated to \( R_i \). Since each vertex of \( P \) may cause at most one constructed edge, there can be \( O(n) \) constructed edges. Given a simple polygon \( P \), the ray-shooting query of a ray \( \overrightarrow{p} \in \mathbb{R}^2 \) determines the first point of intersection of \( \overrightarrow{p} \) with the \( \text{bd}(P) \). Given two points \( p' \) and \( p'' \) in the interior/exterior of a simple polygon \( P \), the shortest-distance query between \( p' \) and \( p'' \) outputs the geodesic Euclidean distance between \( p' \) and \( p'' \).

The following simple data structures are used in our algorithms: a circular doubly linked list \( L_P \) to store the vertices of the current simple polygon, in the order they occur in the counterclockwise traversal of the current simple polygon; a circular doubly linked list \( L_{vp} \) to store the vertices of the visibility (resp. weak visibility) polygon \( \text{VP}(q) \) (resp. \( \text{WVP}(q) \)) of \( q \), in the order they occur in the counterclockwise traversal of the current \( \text{VP}(q) \) (resp. \( \text{WVP}(q) \)); in the case of maintaining
For every edge $e$ of current simple polygon, an array $L_e$ corresponding to $e$ to store the constructed vertices that lie on $e$; and, data structures needed for dynamic ray shooting and two-point shortest-distance queries from [14]. We maintain pointers between the corresponding nodes of $L_p$ and $L_{op}$ that represent the same vertex.

**Proposition 1 ([14] Theorem 6.3)** Let $T$ be a planar connected subdivision with $n$ vertices. With $O(n)$-time preprocessing a fully dynamic data structure of size $O(n)$-space is computed for $T$ that supports point-location, ray-shooting, and shortest-distance queries in $O((\lg n)^2)$ time, and operations InsertVertex, RemoveVertex, InsertEdge, RemoveEdge, AttachVertex, and DetachVertex in $O((\lg n)^2)$ time, all bounds being worst case.

Further, for two query points $q', q'' \in P$, we note that the fully dynamic data structures in [14] support outputting the first line segment in the geodesic shortest-path from $q'$ to $q''$ in $O((\lg n)^2)$ time whenever there exists a unique path between $q'$ and $q''$. Given a ray $\overrightarrow{r}$ whose origin $q$ is in $P$, the ray-rotating query (defined in [8]) with clockwise (resp. counterclockwise) orientation seeks the first vertex of $P$ visible to $q$ that will be hit by $\overrightarrow{r}$ when we rotate $\overrightarrow{r}$ by a minimum non-negative angle in clockwise (resp. counterclockwise) direction. The first parameter to ray-rotating-query procedure is the ray and the second one determines whether to rotate the input ray by a non-negative angle in clockwise or in counterclockwise direction.

**Proposition 2 ([8] Lemma 1)** A data structure can be built in $O(n)$ time and $O(n)$ space such that each ray-rotating query can be answered in $O(\lg n)$ time.

To support ray-rotating queries, [8] in turn uses ray-shooting data structure and two-point shortest distance query data structures from [15]. But for supporting dynamic insertion (resp. deletion) of vertices to (resp. from) the simple polygon, we use data structures from [14] in place of the ones from [15]. This lead to preprocessing in $O(n)$ time to compute data structures of $O(n)$ space so that to answer ray-shooting, ray-rotating, and two-point distance queries in $O((\lg n)^2)$ time.

First, we devise an algorithm to compute $VP(q)$ using the ray-shooting and ray-rotating queries which is useful in several algorithms devised later. For any two arbitrary rays $r_1$ and $r_2$ with their origin at $q$, the $cone(r_1, r_2)$ comprises of a set $S$ of points in $\mathbb{R}^2$ such that $x \in S$ whenever a ray $r$ with origin at $q$ is rotated with center at $q$ from the direction of $r_1$ to the direction of $r_2$ in counterclockwise direction the ray $\overrightarrow{qx}$ occurs. The $opencone(r_1, r_2)$ is the $cone(r_1, r_2) \setminus \{r_1, r_2\}$. The visvert-inopencone procedure listed underneath outputs all the vertices that are visible in the open cone defined by two arbitrary rays $r_1$ and $r_2$. This is essentially accomplished by sweeping the cone using ray-rotating queries. Refer to Algorithm [1]

Our algorithm to compute $VP(q)$ invokes this procedure twice: once with $r_1$ and $r_2$, and next with $r_2$ and $r_1$. These two invocations together yield all the vertices of $P$ that are visible from $q$ except the ones those that lie along rays $r_1$ and $r_2$. Two invocations of ray-rotating-query, once with ray $r_1$ and next with ray $r_2$, determine these vertices as well. If there are $k$ vertices of $P$ that are visible from $q$, the

---

**Figure 1:** Illustrating three cones, each correspond to a recursive call in visvert-inopencone procedure


algorithm takes \(O(k(\lg n)^2)\) time and it takes \(O(k \lg k)\) time to sort these vertices according to their angular order. For every vertex \(v\) visible from \(q\), our algorithm shoots a ray \(\overrightarrow{qv}\) to determine the possible constructed edge on which \(v\) resides.

\[
\text{visvert-inopencone}(r_1, r_2)
\]

1: \(\theta := \cos^{-1}\left(\frac{r_1 \cdot r_2}{|r_1||r_2|}\right)\)
2: Let \(r_1', r_2'\) be the rays with origin at \(q\) that respectively make \(\theta_1, \theta_2\) counterclockwise angles with ray \(r_1\) such that \(\theta_1 < \theta_2 < \theta\).
3: \(v_1 := \text{ray-rotating-query}(r_1', \text{counterclockwise})\)
4: \(v_2 := \text{ray-rotating-query}(r_2', \text{clockwise})\)
5: output \(v_1, v_2\)
6: \(\text{visvert-inopencone}(r_1, r_1')\)
7: \(\text{visvert-inopencone}(r_2', r_2)\)
8: If \(v_1 \neq v_2\) then \(\text{visvert-inopencone}(qv_1', qv_2')\)

**Algorithm 1**: Given a simple polygon \(P\) and two rays \(r_1\) and \(r_2\) with their origin at a point \(q \in P\), outputs all the vertices in \(P \cap \text{opencone}(r_1, r_2)\) that are visible from \(q\).

After every insertion/deletion, we update the data structures relevant to ray-shooting queries. Assuming that the current simple polygon is defined with \(n\) vertices, due to Proposition 1, these updates take \(O((\lg n)^2)\) time.

**Inserting a vertex**

Let \(v\) be the vertex being inserted to the current simple polygon \(P\). And, let edge \(v_iv_{i+1}\) of \(P\) be replaced with edges \(v_iv\) and \(vv_{i+1}\), resulting in a new simple polygon \(P'\). Our algorithm handles the following two cases independently: (i) vertex \(v\) is visible from \(q\) in \(P'\), (ii) \(v\) is not visible from \(q\) in \(P'\). With the ray-shooting query with ray \(\overrightarrow{qv}\), we check whether this ray strikes \(bd(P')\) at \(v\). If it is, then the vertex \(v\) is visible from \(q\) in \(P'\). Otherwise, it is not.

In case (i), we do binary search over the vertices of the current \(VP(q)\) to find two vertices \(v_c, v_{cc}\) of \(VP(q)\) such that the ray \(qv\) lies in the \(\text{cone}(qv_c, qv_{cc})\) and \(v_c, v, v_{cc}\) occur in that order while traversing \(bd(VP(q))\) in counterclockwise direction. Refer to Fig. 2. We determine whether the triangle \(qv_{cc}\) or the triangle \(qvv_c\) or both intersects with the triangle \(v_{i}v_{i+1}\). Suppose the triangle \(qvv_c\) intersects with the triangle \(v_{i}v_{i+1}\). (The other two cases are handled analogously.) We choose a point \(p\) in the \(\text{opencone}(qv_c, qv)\). We invoke ray-rotate query with ray \(qp\) in the counterclockwise direction in \(P'\) to find the vertex \(v'_c\) of \(P'\). All the vertices that occur while traversing \(bd(VP(q))\) from \(v_c\) to \(v'_c\) in clockwise direction are not visible from \(q\). By using the pointers associated with \(v_c\) and \(v'_c\) in \(LP\), we find the corresponding vertices in doubly linked list \(Lvp\); we update the \(Lvp\) by adjusting \(O(1)\) number of pointers of \(Lvp\). Further, we include \(v\) as a counterclockwise neighbor of \(v'_c\) in \(Lvp\) and update the same in \(LP\). Using ray-shooting queries, we compute the constructed edges that could incident to \(v\) and \(v'_c\).
For the case (ii), suppose that the vertex \( v \) is not visible from \( q \). Refer to Fig. We do binary search over the vertices of \( VP(q) \) to find two pairs of vertices \((v'_1, v''_1)\) and \((v'_2, v''_2)\) such that \( v'_1, v'_2 \) are visible from \( q \) and \( v''_1, v''_2 \) are not visible from \( q \). If such pairs does not exist, then nothing is done. (It means that the no section of boundary of the triangle \( v_i v_{i+1} \) is visible from \( q \), and \( v \) is in a region whose interior is occluded from \( q \).

Otherwise, let \( v'_ix \) be the constructed edge that intersects \( vv_i \). (Other cases that could occur are handled similarly.) We replace the constructed edge \( v'_ix \) with \( v'_ix' \), where \( x' \) is the point of intersection of \( v'_ix \) with \( vi \). Then the remaining part of this case is reduced to case (i) in the following way: point \( x' \) is equivalent to \( v \) in case (i). Hence, it is handled in the similar manner to case (i). Further, the constructed edges of \( v_i \) and \( v_{i+1} \) are updated if necessary.

In both the cases, for every constructed vertex \( p \) that incident to edge \( v_i v_{i+1} \) of \( P \), let \( v_j \) be the vertex of \( P \) that incident to line segment \( qp \). We ray-shoot with ray \( \overrightarrow{v_jb} \) in \( P' \) to find the new constructed vertex that incident to either \( v_i v' \) edge or \( v_{i+1} v'' \) edge of \( P' \).

In updating the \( VP(q) \), there are \( O(1) \) ray-shooting queries and ray-rotating queries involved. And, updating \( LP \) and \( Lvp \) take \( O(1) \) time. Hence, the time complexity in updating the visibility polygon is \( O((\log n)^2) \) per vertex insertion, where \( n \) is the number of vertices of \( P \).

**Observation 1** Let \( VP(q) \) be the visibility polygon of a point \( q \) interior to simple polygon \( P \). When a vertex \( v \) is inserted to \( P \), the set of vertices in \( VP(q) \) that are hidden due to this insertion are contiguous along the boundary of \( VP(q) \).

**Lemma 2.1** The time complexity in updating the visibility polygon is \( O((\log n)^2) \) per vertex insertion, where \( n \) is the number of vertices of \( P \).

**Deleting a vertex**

Let \( v_i v \) and \( vv_{i+1} \) be the edges of simple polygon \( P \) that occur in that order while traversing the boundary of \( P \) in counterclockwise direction. Let \( v \) be the vertex to be deleted from \( P \). Also, let \( P' \) be the resultant simple polygon due to the deletion of \( v \) from \( P \). Our algorithm handles the following two cases independently: (i) vertex \( v \) is visible from \( q \) in \( P \), (ii) \( v \) is not visible from \( q \) in \( P \).

A ray-shooting query with ray \( \overrightarrow{qv} \) determines whether \( v \) is visible from \( q \). Then we invoke visvert-inopencone procedure for \( P' \) twice: once with rays \( qv \) and \( qv_{cc} \) as respective first and second parameters; next with rays \( qv_{ce} \) and \( qv \) as respective first and second parameters. The vertices output by these invocations are precisely the ones in \( P \) hidden from \( q \) due to triangle \( v_i v_{i+1} \). If \( k \) vertices become visible from \( q \) after deletion
of v from P, then this sub-procedure requires \(O(k)\) ray-rotations. We compute constructed edges corresponding to each vertex that get visible from q after the removal of v. For every constructed vertex \(p\) that incident to either of the edges \(vv_i\) or \(vv_{i+1}\) of \(P\), let \(v_j\) be the vertex of \(P\) that incident to line segment \(qp\). We ray-shoot with the ray \(v_j\overrightarrow{p}\) in \(P'\) to find the new constructed edge to which \(v_j\) incident. Also, we do the corresponding updates to data structures \(L_{vp}\) and \(L_P\).

For the case (ii), if neither \(vv_i\) nor \(vv_{i+1}\) has constructed vertices of VP(q) stored with them, then we do nothing. Otherwise, let \(x', x''\) be the constructed vertices stored with the edge \(vv_i\) such that \(x''\) occurs after \(x'\) in the counterclockwise ordering of the vertices of VP(q). Refer to Fig. 5. (Handling the possible constructed vertices with \(vv_{i+1}\) is analogous.) We invoke visvert-opencone procedure with rays \(qx'\) and \(qx''\) to find a set \(S\) of vertices of \(P'\) that are visible from \(q\). For every vertex \(v_l\in S\), using ray-shooting query, we determine the constructed edge corresponding to \(v_l\). Let \(v'\) (resp. \(v'')\) be the vertex that lie on constructed edge \(qx'\) (resp. \(qx''\)). With two ray-shooting queries in \(P'\), one with ray \(v'x'\) and the other with ray \(v''x''\), we find the new constructed edges on which \(v'\) and \(v''\) respectively lie.

In both the cases, the time complexity is dominated by \(k\) ray-rotating queries, which together take \(O((k + 1)(\log n)^2)\) time, where \(k\) is the change in complexity of visibility polygon due to the removal of vertex \(v\) from \(P\). (Since the deletion algorithm takes \(O((\log n)^2)\) time to update even when \(k\) is zero, we write the update time complexity as \(O((k + 1)(\log n)^2)\) per deletion.)

We note that our data structures accommodate query algorithm from [8], based on ray-rotation queries. Hence, the following Theorem:

**Lemma 2.2** The time taken for maintenance of VP when a vertex is deleted from \(P\) is \(O((k + 1)(\log n)^2)\) per deletion.

**Theorem 2.1** A point \(p\in bd(P)\) is inserted as a vertex of the current visibility polygon VP(q) of a point \(q\) interior to current simple polygon \(P\) whenever \(p\) is visible from \(q\). Similarly, a vertex \(v\in VP(q)\) is removed from VP(q) whenever \(v\) is not visible from \(q\).

**Proof:** In the incremental part, the only vertices that are added to VP are the constructed edges computed using ray shooting queries. Ray shooting queries give the first edge of \(P\) where the ray hits. In the decremental part, we use ray rotating queries in \(P\) to find the range of vertices in which the visibility is affected and we use ray rotating queries in this range in \(P'\) to determine every vertex that is visible.

In the incremental part, as mentioned in Observation the visibility is blocked only for one particular contiguous sequence of vertices which were visible till that point; hence, those vertices are removed from VP. In the decremental part, we do not remove any vertices of \(P\) from VP. Only constructed vertices which are no longer required are removed. \(\square\)

**Theorem 2.2** For a given simple polygon with \(n'\) vertices, we preprocess in \(O(n')\) time to build data structures of size \(O(n')\) to support the following:
(1) Let \( P \) be the current simple polygon. Let \( VP(q) \) be the visibility polygon of a point \( q \in P \). For a vertex \( v \) inserted to \( P \), updating \( VP(q) \) takes \( O((\lg n)^2) \) time. For a vertex \( v \) deleted from \( P \), updating \( VP(q) \) takes \( O((k + 1)(\lg n)^2) \) time, where \( k \) is the change in complexity of visibility polygon due to the deletion of \( v \) from \( P \).

(2) Further, the visibility polygon of any query point \( q \in P \) is computed in \( O(k(\lg n)^2) \) time, where \( k \) is the output complexity.

### 2.2 Maintaining visibility polygon of \( q \) when \( q \notin P \)

There are two cases in considering visibility of a point \( q \notin P \): (i) \( q \) lies outside of \( CH(P) \), and (ii) \( q \) belong to \( CH(P) \) \( P \). We consider these cases independently. The following observations from \[12\] are useful in reducing these problems to computing visibility polygon of a point interior to a simple polygon:

* Let \( q \notin CH(P) \) and let \( t', t'' \) be points of tangencies from \( q \) to \( CH(P) \). Also, let \( \overrightarrow{qt'} \) occur when \( qt'' \) is rotated in counterclockwise direction with center at \( q \). Then the region bounded by line segments \( qt', qt'' \) and the vertices that occur while traversing \( bd(P) \) in counterclockwise direction from \( t' \) to \( t'' \) is a simple polygon \( P'' \).

* Let \( q \in CH(P) \setminus P \) and let \( v' \) be any vertex of \( P \). Let \( q' \) be the closest point of \( q \) among all points of intersections of ray \( \overrightarrow{qv'} \) with \( bd(P) \). Starting from \( q' \), traverse \( bd(P) \) in clockwise (and in counterclockwise) order till a vertex \( t' \) (resp. \( t'' \)) is reached. Note that the vertices \( t' \) and \( t'' \) of \( P \) are consecutive vertices of \( CH(P) \). The region bounded by the edge \( t't'' \) of \( CH(P) \), and the boundary of \( P \) between \( t' \) and \( t'' \) containing \( q' \) is a simple polygon \( P'' \). Moreover, note that the point \( q \in P'' \).

We say the simple polygon \( P'' \) resultant in either of these cases as the simple polygon corresponding to points \( t' \) and \( t'' \).

The changes required in algorithms in the last Section are mentioned here. In \( O((\lg n)^2) \) time, using the dynamic planar point-location query \[14\], we determine whether \( q \in P \). If \( q \notin P \) and points of tangency from \( q \) to \( CH(P) \) does exist (resp. does not exist), then \( q \notin CH(P) \) (resp. \( q \in CH(P) \setminus P \)). The tangents from \( q \) to \( CH(P) \) are computed in \( O(\lg n) \) time \[25\].

We dynamically maintain convex hull \( CH(P) \) of current simple polygon \( P \) using Overmars et al. \[24\]. Whenever a vertex is inserted or deleted, we update the convex hull of the simple polygon. To facilitate this, \[24\] preprocesses \( P \) in \( O(n) \) time to construct data structures of size \( O(n) \) so that to update convex hull of current simple polygon in \( O((\lg n)^2) \) time per vertex insertion or deletion.

Suppose that \( q \notin CH(P) \). Let \( t', t'' \) are the points of tangency from \( q \) to \( CH(P) \). Also, let \( t' \) occurs latter to \( t'' \) in counterclockwise traversal of \( bd(CH(P)) \). We invoke visvert-inopencone procedure with \( \overrightarrow{qt'} \) and \( \overrightarrow{qt''} \) as the first and second parameters respectively. Refer to Fig. 6.

Let \( P'' \) be the simple polygon corresponding to points \( t' \) and \( t'' \). We do not explicitly compute \( P'' \) itself although the ray-shooting is limiting to the interior of \( P'' \). For every vertex \( v' \) that

---

Figure 6: Illustrating the exterior visibility of \( P \) from \( q \) when \( q \notin CH(P) \).
is determined to be visible from \( q \), we ray-shoot with ray \( \overrightarrow{qv'} \) in \( P'' \) to determine the constructed edge that incident to \( v' \).

Suppose that \( q \in CH(P) \setminus P \). First, we note that the ray-shooting algorithms given in [14] work correctly within any one region of the planar subdivision; same is the case with the ray-rotating algorithm from [8]. To account for these constraints, we slightly modify ray-rotating query procedure invoked from visvert-inopencone.

If the ray-shooting procedure determines that a ray \( r \) with origin \( q \) does not strike any point of \( P \), then in \( O(\lg n) \) time we compute the edge \( e \) of \( CH(P) \) that gets struck by \( r \). Let \( v', v'' \) be the endpoints of \( e \). Also, let \( v' \) occurs before \( v'' \) in the clockwise ordering of vertices of \( bd(P) \). The edge \( e \) together with the section of polygon boundary that occurs from \( v' \) to \( v'' \) in the clockwise ordering of edges along \( bd(P) \) is a simple polygon, say \( P'' \). Refer to Fig. 7.

Whenever we encounter edge \( e \) with these characteristics, we re-invoke the visvert-inopencone procedure with \( qv' \) and \( qv'' \) as the first and second parameters respectively. Again, we do not traverse the edge list of \( P'' \) to compute \( P'' \).

**Theorem 2.3** A vertex is added to \( VP \) if and only if it is visible from \( q \).

**Proof:** Starting from \( q \), any ray will hit the boundary of \( P \) on a edge such that the ray is not intersected by any other edge of \( P \). From this point, every vertex that we encounter while performing the ray rotating algorithm is the next visible vertex which is added to \( VP \). Hence, the lemma is true. \( \Box \)

**Theorem 2.4** With \( O(n') \) time preprocessing of simple polygon defined with \( n' \) vertices, data structures of size \( O(n') \) are computed to facilitate vertex insertion, and vertex deletion in \( O((\lg n)^2) \) time, and to output the visibility polygon of any query point located exterior to the current simple polygon \( P \) in \( O(k(\lg n)^2) \) time, where \( k \) is the output complexity and \( n \) is the number of vertices of \( P \).

### 2.3 Querying for weak visibility polygon of \( l \) when \( l \in P \)

Let \( s \) be a query line segment interior to the given simple polygon \( P \). Let \( a \) and \( b \) be the endpoints of \( s \). The weak visibility polygon \( WVP(s) \) of \( s \) is \( \bigcup_{p \in s} VP(p) \). The algorithm needs to capture the combinatorial representation changes of \( VP(p) \) as the point \( p \) moves from \( a \) to \( b \) along \( s \). First, we describe the notation and algorithm for computing weak visibility polygon from [8]. For any two vertices \( v', v'' \in P \), let \( b' \) be the point of intersection of ray \( \overrightarrow{v''v'} \) with the boundary of \( P \). If \( b' \neq v' \), then the line segment \( b'v' \) is termed as a critical constraint of \( P \). Initially, point \( p \) is at \( a \) and we have \( VP(p) \) is same as \( VP(a) \). As \( p \) moves from \( a \) to \( b \) along \( s \), a new vertex of \( P \) could be added to the weak visibility polygon of \( s \) whenever \( p \) crosses a critical constraint of \( P \) [11 [8]. As \( p \) moves along \( s \), the next critical constraint that it encounters is characterized in the following observation from [11].
Observation 2 (from [1]) The next critical constraint of a point \( p \) is defined by two vertices of \( P \) that are either two consecutive children of \( p \) or one, say \( v \), is a child of \( p \) and the other is the principal child of \( v \).

The principal child definition from [8] is mentioned herewith. The shortest path tree rooted at \( p \), \( SPT(p) \), is the union of the shortest paths in \( P \) from \( p \) to all vertices of \( P \). A vertex of \( P \) is in \( VP(p) \) if and only if it is a child of \( p \) in \( SPT(p) \). For any child \( v \) of \( p \) in the tree \( SPT(p) \), the principal child of \( v \) is the child \( w \) of \( v \) in \( SPT(p) \) such that the angle between rays \( \vec{w} \) and \( \vec{w} \) is smallest as compared with the angle between rays \( \vec{w} \) and \( \vec{b} \) for any other child \( w' \neq w \) of \( v \).

We preprocess the same data structures as in the case of dynamic algorithms for updating visibility polygon of a point interior to \( P \). Given a line segment \( s \) with endpoints \( a \) and \( b \), we first compute the visibility polygon when \( p \) is at point \( a \). Let \( r_1 \) and \( r_2 \) be two arbitrary rays whose origin is at \( a \). The \( VP(a) \) is computed in \( O(k(\log n)^2) \) time by invoking the visvert-inopencone procedure twice: once with \( r_1 \) and \( r_2 \); and the next time with \( r_2 \) and \( r_1 \). For any child \( v \) of \( SPT(p) \), as described in [8], with one ray-rotating query, we determine the principal child of \( v \) in \( O((\log n)^2) \) time. The critical constraints that intersect \( s \) are stored in a priority queue \( Q \), with the key value corresponding to a critical constraint \( c \) equal to the distance of the point of intersection of \( c \) and \( s \) from \( a \). The extract minimum on \( Q \) determines the next critical constraint that \( p \) strikes. After crossing a critical constraint, if \( p \) sees an additional vertex \( v' \) of \( P \), then we insert \( v' \) into the appropriate position in \( L_{vp} \). Further, critical constraints that arise due to \( v' \) which intersect with \( s \) are pushed into \( Q \). As and when a vertex of \( P \) is determined to be visible from \( s \), we compute the constructed edge with a ray-shooting query. When \( p \) reaches point \( b \) (endpoint of \( s \)), \( L_{vp} \) is updated so that it represents the \( WVP(s) \).

Theorem 2.5 A vertex is added to \( WVP \) if and only if it is visible from at least one point on line segment \( pq \).

Proof: As mentioned, every point between any two critical segments will have the same vertices of \( P \) are visible to that point. Hence, the union of the visible vertices between every two consecutive critical segments will form the weak visibility polygon. Hence, the lemma is true.

Theorem 2.6 With \( O(n') \) time preprocessing of simple polygon defined with \( n' \) vertices, data structures of size \( O(n') \) are computed to facilitate vertex insertion, and vertex deletion in \( O((\log n)^2) \) time, and to output the weak visibility polygon of a query line segment located interior to the current simple polygon \( P \) in \( O(k(\log n)^2) \) time, where \( k \) is the output complexity and \( n \) is the number of vertices of \( P \).

2.4 Maintaining weak visibility polygon of an edge with vertex insertions

In this Section, we consider the problem of updating weak visibility polygon \( WVP_{pq}(pq) \) of an edge \( pq \) when a new vertex is inserted into \( P \). With slight abuse of notation, we denote \( WVP_{pq}(pq) \) with \( WVP \) when \( pq \) and \( P \) are clear from context. We let \( v_1, \ldots, v_n \) be the vertices of \( P \) in counterclockwise direction. Similarly, let \( u_1, \ldots, u_m \) be the vertices of \( WVP \) in counterclockwise direction. When \( pq \) is a line segment contained within a simple polygon \( Q \), we can partition \( Q \) into two simple polygons \( Q_1 \) and \( Q_2 \) by extending segment \( pq \) on both sides until it hits \( bd(Q) \). Then, the \( WVP \) of \( pq \) in \( Q \) is the union of \( WVP \) of \( pq \) in \( Q_1 \) and \( Q_2 \). Hence, to update \( WVP \) in \( Q \), we
need to update the WVP in $Q_1$ and $Q_2$. Therefore, it suffice to consider the case when $pq$ is an edge of $P$. For any edge $u_iu_{i+1}$ of WVP, if only one of $\{u_i, u_{i+1}\}$ is a vertex of $P$ then that edge is called a constructed edge and the non-vertex among these two points is called a constructed vertex.

The constructed edges of WVP partition $P$ into regions $R = \{WVP, R_1, R_2, \ldots, R_s\}$ such that no point $q$ interior to $R_i$ is visible from any point $r \in pq$ for any $i \in \{1, 2, \ldots, s\}$. Also, for each $R_i$, there exists a constructed vertex associated to $R_i$. Since each vertex of $P$ can cause at most one constructed edge, there can be $O(n)$ constructed edges.

We review the algorithm by Guibas et al. \[16\] which is used in computing the initial WVP($pq$) as well as to update the same. First, the shortest path tree $SPT(p)$ is computed: $SPT(p)$ is the union of $SP(p,v_i)$ for every $v_i \in P$. Then, the depth first traversal is performed over $SPT(p)$. If the shortest path to a child $v_j$ of $v_i$ makes a right turn at $v_i$, then a segment is constructed by extending $v_kv_i$ to intersect $bd(P)$ ($v_k$ is parent of $v_i$ in $SPT(p)$). The portion of $P$ lying on the right side of such constructed segments does not belong to WVP($pq$) and hence it is removed from $P$. Let $P''$ be the resulting polygon. A similar procedure is performed with respect to $q$ on $P''$, resulting in WVP($pq$). From the algorithm of Guibas et al. \[16\], as new vertex is added to $P$, it is apparent that to update the WVP($pq$), we need to update $SPT(p)$ and $SPT(q)$ and remove the vertices which are not part of WVP($pq$).

For updating $SPT(p)$ and $SPT(q)$, we use the algorithm by Kapoor and Singh \[29\]. Their algorithm is divided into three phases: first phase computes segment $e_i$ of $SPT(p)$ intersecting $\Delta v_i v_k v_i$; in the second phase, $SPT(p)$ is updated to include the endpoints $z_i$ of $e_i$ that are not visible; and in the final phase, the algorithm updates $SPT(p)$ to include every children of all such $z_i$. As a result, \[29\] updates $SPT(p)$ in $O(k \log (n/k))$ time in the worst-case; here, $k$ is the number of changes made to $SPT(p)$.

A funnel consists of a vertex $r$, called the root, and a segment $xy$, called the base of the funnel. The sides of the funnel are $SP(r,x)$ and $SP(r,y)$ (Fig. 8). In \[29\], $SPT(p)$ is stored as a set of funnels. For a funnel $F$ with root $r$ and base $xy$, the segments from $r$ to $x$ and from $r$ to $y$ are stored in a balanced binary search trees $T_1$ and $T_2$ respectively. The root nodes of $T_1$ and $T_2$ contain pointers to each other. It can be seen that every edge $e$ that is not a boundary edge lies in exactly two funnels say $F_i$ and $F_j$. Then the nodes in $F_i$ and $F_j$ corresponding to $e$ contains pointers to each other. The whole of this data structure require $O(n)$ space.

The vertices of $P$ and WVP are stored in doubly linked lists $L_P$ and $L_{WVP}$ in the order in which they appear on the bd($P$) and bd(WVP). Additionally, nodes in $L_{WVP}$ and $L_P$ which correspond to the same vertex $u_i \in WVP$ contain pointers to each other. For each region $R_i$, vertices are stored in a balanced binary tree $T_{R_i}$ such that a node in $T_{R_i}$ and a node in $L_P$ corresponding to the same vertex $v_j \in R_i$ contain pointers to each other. We call the pointer from a node in $L_P$ to a node in $T_{R_i}$ as a tree pointer. For vertices $v_i \notin WVP$, the tree pointer is set to null. All of these data structures, together with the ones to store funnels, take $O(n)$ space.

In our algorithm, we first update $SPT(p)$ and $SPT(q)$. Then, with the depth first traversal starting from $v_a$ in the updated $SPT(p)$ and $SPT(q)$, we remove the regions that are not entirely visible from $pq$. While removing vertices in these regions, we also remove the corresponding nodes.
in $SPT(p)$ and $SPT(q)$ so that only the vertices visible from $pq$ are left. Like in previous Section, the tree pointers are used to identify which of the four cases are applicable in a given situation while the algorithm splits/joins regions as it proceeds.

We assume that after adding $v_a$, simple polygon $P$ continues to be a simple polygon and that $pq$ is still an edge of the new simple polygon $P'$.

Preprocessing

As mentioned above, we compute $SPT(p)$ and $SPT(q)$ using [16] and store them as funnels using the algorithm from [29]. We also compute $WVP$ and the balanced binary trees for each region. The preprocessing phase takes $O(n)$ time in the worst-case.

Updating $WVP(pq)$

We have the following four cases depending on the locations of $v_b$ and $v_c$ and the disjoint regions $R_i, R_j \in \mathcal{R}$.

(a) $v_b, v_c \in WVP$ (Fig. 9(a))

(b) $v_b \notin WVP$ and $v_c \in WVP$ or vice versa (Fig. 9(b))

(c) $v_b, v_c \notin WVP$ and $v_b \in R_i, v_c \in R_j$ such that $i \neq j$ (Fig. 9(c))

(d) $v_b, v_c \notin WVP$ and $v_b, v_c \in R_i$ (Fig. 9(d))

Further, depending on the location of the newly inserted vertex $v_a$, there are two sub-cases in each of these cases: (i) $v_a \notin P$, (ii) $v_a \in P$. We first consider the cases (a)-(d) when $v_a \in P$. We make use of join, split and insert operations on the balanced binary trees corresponding to regions, each of which take $O(\log n)$ time, as given in Tarjan [30]. Identifying these cases is done in a manner similar to the earlier Section. We use $k_1$ and $k_2$ to denote the number of changes required to update $SPT(p)$ and $SPT(q)$ respectively.

Case (a) When $SP(p, v_b)$ and $SP(p, v_c)$ do not intersect $\triangle v_a v_b v_c$, (Fig. 10(i)), add $v_a$ to $SPT(p)$ by finding a tangent from $v_a$ to funnel with base $v_b v_c$ in $SPT(p)$. Analogously, if $SP(q, v_b)$ and $SP(q, v_c)$ do not intersect $\triangle v_a v_b v_c$, then $v_a$ is added to $SPT(q)$ as well. Further, $v_a$ is added to $WVP(pq)$ in place of $v_b$ and remove $v_b$ and $v_c$. No further changes are required.

Figure 9: Illustrating cases based upon the locations of $v_b$ and $v_c$.

Figure 10: (i) $SP(p, v_b)$ and $SP(p, v_c)$ do not intersect $\triangle v_a v_b v_c$ and (ii) $SP(p, v_c)$ intersects $\triangle v_a v_b v_c$.
We now consider the case when \( SP(p,v_c) \) intersects \( \triangle v_a v_b v_c \) (Fig. 10(ii)). It is possible that \( v_a \) lies in a region \( R_i \) with constructed edge \( v_i x_i \) (Fig. 11). While updating \( SPT(p) \), we check if one of the edges in \( SPT(p) \) intersecting \( \triangle v_a v_b v_c \) is \( v_i x_i \). If it is, all vertices on \( bd(P) \) from \( v_c \) to \( x_i \) need to be joined into a single region and the vertices on \( bd(WVP) \) from \( v_c \) to \( x_i \) need to be deleted from \( WVP \). This is because \( \triangle v_a v_b v_c \) blocks the visibility of these vertices from \( pq \). And, this is accomplished with the union of all the regions \( R_j \) that are encountered while traversing along \( bd(WVP) \) from \( v_c \). For \( O(k_1) \) constructed vertices, joining takes \( O(k_1 \lg n) \) time. The resultant region is in turn joined with \( R_i \).

The point of intersection \( x_i' \) of \( v_a v_b \) and \( v_i x_i \) is added as a constructed vertex to both \( WVP \) and \( SPT(p) \), and \( R_i \) is associated with \( x_i' \). To update \( SPT(q) \), we find all edges \( e_i \) in \( SPT(q) \) that intersect \( \triangle v_a v_b v_c \) and delete all the endpoints \( z_i \) that are not visible from \( q \) along with their children from \( SPT(q) \). This completes the update in this sub-case.

If \( v_a \in WVP \), then no constructed edge intersects \( \triangle v_a v_b v_c \) and we update \( SPT(p) \) (Fig. 12). Let \( F_i \) be the funnel containing \( v_a \) with root \( r_i \) and base \( v_x v_y \). We then perform the depth first traversal in \( SPT(p) \), starting from vertex \( v_a \). If for any vertex \( v_j \) with parent \( v_i \) in \( SPT(p) \), \( SP(r_i,v_j) \) takes a right turn at vertex \( v_i \) we need to add a constructed edge. This is done by finding the leftmost child \( v_{j'} \) of \( v_i \) for which \( SP(v_i,v_{j'}) \) still takes a right turn at \( v_i \). Let \( v_k \) be the parent of \( v_i \) in \( SPT(p) \), then the constructed vertex \( x_i \) is found by extending \( v_k v_i \) on to \( v_{j'} v_{j'+1} \). This results in a new region \( R_i \).

To add vertices to this region and delete them from \( WVP \), we traverse \( bd(WVP) \) in counterclockwise direction and add every vertex from \( v_{j'} \) to \( v_i \). Every region \( R_j \) that is encountered in this traversal is joined with \( R_i \). As there can be \( O(k_1) \) constructed vertices in this traversal along \( bd(WVP) \), it requires \( O(k_1 \lg n) \) time. Finally, the constructed vertex \( x_i \) is added to \( WVP \) and \( SPT(p) \). To update \( SPT(q) \), we find all edges \( e_i \) in \( SPT(q) \) that intersect \( \triangle v_a v_b v_c \). If the hidden endpoint \( z_i \) of an edge \( e_i \) has a non-null tree pointer (it lies in some region \( R_j \)), then \( z_i \) is deleted along with its children. Then, similar to the depth first traversal in \( SPT(p) \) above, we perform the depth first traversal in the leftover \( SPT(q) \) to update \( WVP \). This completes the update. Note that whole of this algorithm takes \( O(k_2 \lg n) \) time in the worst-case. When \( SP(p,v_b) \) intersects \( \triangle v_a v_b v_c \), we first update \( SPT(q) \) and steps analogous to above are performed. Further, we update \( SPT(p) \), and finally we have the updated \( WVP \). Since both \( SPT(p) \) and \( SPT(q) \) are updated, our algorithm takes \( O(k_1 \lg n + k_2 \lg n) \) time in the worst-case.

**Case (b)** We assume that \( v_b \notin WVP \) and \( v_c \in WVP \). Let \( v_b \in R_i \) with constructed vertex \( x_i \) and
constructed edge \(v_a x_i\). If \(v_a v_c\) intersects \(v_i x_i\) at \(x_i'\) (Fig. 13(i)), then \(v_a \in R_i\).

We update \(WVP\) by deleting \(x_i\) and adding \(x_i'\). Similarly, we update \(SPT(p)\) and \(SPT(q)\) by deleting \(x_i\) and adding \(x_i'\). Finally, we add \(v_a\) to \(R_i\). Overall, handling this sub-case takes \(O(\lg n)\) time.

When \(v_a \notin R_i\) (Figs. 13(ii), 13(iii)), we find the intersection of \(v_a v_b\) with \(v_i x_i\) at \(x_i'\). We update \(WVP\) by deleting \(x_i\) and adding \(x_i'\). Similarly, we update \(SPT(p)\) and \(SPT(q)\) by deleting \(x_i\) and adding \(x_i'\). This case is then similar to case (a) where \(v_b\) is replaced by \(x_i'\); hence, takes \(O(k_1 \lg n + k_2 \lg n)\) time.

**Case (c)**

Let \(v_b \in R_i\) and let \(v_c \in R_j\). Also, let \(x_i\) and \(x_j\) be the respective constructed vertices on \(v_b v_c\) (Fig. 14). Then we have three sub-cases which are similar to case (b) when \(v_a \in R_i\). Here, we additionally update \(SPT(p)\) and \(SPT(q)\) by adding the new constructed vertices (intersection points \(x_i'\) and \(x_j'\)) of \(\triangle v_a v_b v_c\) with the constructed edges). This additional step takes \(O(\lg n)\) time.

**Case (d)**

Let \(v_b, v_c \in R_i\) and \(x_i\) be the corresponding constructed vertex and \(v_i x_i\) be the constructed edge. If \(v_a \in R_i\), then no change is made to \(WVP\) but we need to add \(v_a\) to \(T_{R_i}\) (Fig. 15(i)) which takes \(O(\lg n)\) time. If \(v_a \notin R_i\) (Figs. 15(ii), 15(iii)), we assume a portion of \(v_a v_b\) is visible from \(pq\). (When a portion of \(v_a v_c\) is visible from \(pq\), it is handled analogously.) Then the polygonal region \(R_i\) is split into two regions. Hence, we split the tree \(T_{R_i}\) as well. This results in two trees: \(T_{R_{i1}}\) with vertices from \(v_i\) to \(v_b\) along the counterclockwise traversal of \(bd(P)\); and \(T_{R_{i2}}\) with vertices from \(v_c\) to \(x_i\) along the counterclockwise traversal of \(bd(P)\). Let \(v_a v_b\) and \(v_a v_c\) respectively intersect \(v_i x_i\) at points \(x_1\) and \(x_2\) respectively. We add \(x_1\) and \(x_2\) to \(SPT(p)\) and \(SPT(q)\). Also, we associate \(x_1\) as a constructed vertex with \(T_{R_{i1}}\). This case is then similar to case (a) with \(\triangle v_a x_1 x_2\) except that in the
last step, we join tree $T_{R_a}$ with a newly formed tree. This completes the case (d). And, this case require $O(k_1 \log n + k_2 \log n)$ time.

When $v_a \notin P$, each of the four cases (a) to (d) can be handled similar to case (c). Each of these cases again take $O(\log n)$ time in the worst-case.

**Distinguishing cases (a)-(d)** To determine the appropriate case among cases (a)-(d), we follow the tree pointers associated with $v_b$ and $v_c$ in $L_P$. For $v_b$, if the tree pointer is not null, then it points to a node in a tree say $T_{R_b}$. We follow the parent pointers from this node to reach the root of $T_{R_b}$, say $r_1$, which takes $O(\log n)$ time. Similarly, $r_2$ is found if the tree pointer for $v_c$ is not null. The appropriate case can thus be found using these tree pointers. To identify whether $v_a \in P$ or $v_a \notin P$, we can check if $v_b, v_c, v_a$ turns left or right respectively.

**Lemma 2.3** The cases (a)-(d) are exhaustive. Further, the corresponding sub-cases are exhaustive as well.

**Proof:** In case (a), first consider the case when $v_bv_c$ is the base of a funnel $F$. Then $\triangle v_av_bv_c$ can intersect at most one side of $F$. Now, consider that $v_bv_c$ is a edge on the side of a funnel $F'$. Then, the shortest path to at most one of $v_b$ and $v_c$ will be affected. In cases (b), (c) and (d), similar arguments can be given.

**Lemma 2.4** In updating, a vertex $u_i \in WVP$ is deleted if and only if it is not weakly visible from $pq$ and a (constructed) vertex is added whenever it is weakly visible from $pq$.

**Proof:** If a vertex $u_i \in WVP$ is not visible from $p$, then either $SP(p, u_i)$ takes a right turn at some vertex $v_j$ or $SP(q, u_i)$ takes a left turn at some vertex $v_j'$. But such vertices are removed from $WVP$ during the depth-first traversals. If a vertex $u_i \in WVP$ is deleted from $WVP$, then either $SP(p, u_i)$ turns to the right or $SP(q, u_i)$ turns to the left. This implies $u_i$ is not visible from $pq$.

**Lemma 2.5** The worst-case time in updating the weak visibility polygon is $O((k + 1) \log n)$.

Identifying the appropriate case requires $O(\log n)$ time. Each case takes $O(k_1 \log n + k_2 \log n) = O(k \log n)$ time. Note that in case (d) sub-case (i), $k = 0$. Since $k$ equals to zero in sub-case (i) of case (d), overall time complexity for updating the weak visibility polygon is $O((k + 1) \log n)$.

**Theorem 2.7** Let $WVP_p(pq)$ be the weak visibility polygon of an edge $pq$ of a simple polygon $P$. Let $v_a$ be the new vertex inserted to $P$ such that an existing edge $v_bv_c$ is replaced by edges $v_av_b$ and $v_bv_c$, resulting in a new simple polygon $P'$. After preprocessing $P$ and $WVP_p(pq)$, the weak visibility polygon $WVP_{P'}(pq)$ is updated in $O((k + 1) \log n)$ time, wherein $k$ is the total number of changes required to update $SPT(p)$ and $SPT(q)$ in adding new vertex $v_a$ to $P$.

### 3 Incremental visibility of polygonal domain

We extend the algorithm for maintaining visibility polygon in a simple polygon to polygonal domain. For any simple polygonal domain $P$, the free space defined by the closure of $R^2$ sans the union of
We also devised dynamic algorithms to query for the weak visibility polygon of a line segment $s$ of a point $q$. We have presented algorithms to dynamically maintain as well as to query for the visibility polygon of a fixed point $q$. Given a polygonal domain $P$, our algorithm updates the visibility polygon of $q$ in $O(k + \lg n)$ time per vertex insertion, where $k$ is the number of changes in visibility polygon and $n$ is the number of vertices in the current polygonal domain.

4 Conclusions

We have presented algorithms to dynamically maintain as well as to query for the visibility polygon of a point $q$ interior or exterior to simple polygon $P$, as the vertices are inserted or deleted from $P$. We also devised dynamic algorithms to query for the weak visibility polygon of a line segment $s$ when $s$ is located interior to the simple polygon. Further, we extended the visibility polygon maintenance
in simple polygon to polygonal domain. The time complexities of all of these algorithms are output-sensitive. To our knowledge, this is first result to give fully-dynamic algorithms' for maintaining visibility polygons.

References

[1] B. Aronov, L. J. Guibas, M. Teichmann, and L. Zhang. Visibility queries and maintenance in simple polygons. *Discrete and Computational Geometry*, 27(4):461–483, 2002.

[2] T. Asano. Efficient algorithms for finding the visibility polygons for a polygonal region with holes. *Transactions of IEICE*, E68(9):557–559, 1985.

[3] T. Asano, T. Asano, L. J. Guibas, J. Hershberger, and H. Imai. Visibility of disjoint polygons. *Algorithmica*, 1(1):49–63, 1986.

[4] P. Bose, A. Lubiw, and J. I. Munro. Efficient visibility queries in simple polygons. *Computational Geometry*, 23(3):313–335, 2002.

[5] M. N. Bygi and M. Ghodsi. Weak visibility queries in simple polygons. In *Proceedings of the Canadian Conference on Computational Geometry*, 2011.

[6] B. Chazelle and L. J. Guibas. Visibility and intersection problems in plane geometry. *Discrete and Computational Geometry*, 4:551–581, 1989.

[7] D. Z. Chen and H. Wang. Visibility and ray shooting queries in polygonal domains. *Computational Geometry*, 48(2):31–41, 2015.

[8] D. Z. Chen and H. Wang. Weak visibility queries of line segments in simple polygons. *Computational Geometry*, 48(6):443–452, 2015.

[9] 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.

[10] 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.

[11] S. K. Ghosh. Computing the visibility polygon from a convex set and related problems. *Journal of Algorithms*, 12(1):75–95, 1991.

[12] S. K. Ghosh. *Visibility algorithms in the plane*. Cambridge University Press, New York, NY, USA, 2007.

[13] S. K. Ghosh and D. M. Mount. An output-sensitive algorithm for computing visibility graphs. *SIAM Journal on Computing*, 20(5):888–910, 1991.

[14] Michael T. Goodrich and Roberto Tamassia. Dynamic ray shooting and shortest paths in planar subdivisions via balanced geodesic triangulations. *Journal of Algorithms*, 23(1):51–73, 1997.

[15] L. J. Guibas and J. Hershberger. Optimal shortest path queries in a simple polygon. *Journal of Computer and System Sciences*, 39(2):126–152, 1989.
[16] L. J. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2:209–233, 1987.

[17] L. J. Guibas, R. Motwani, and P. Raghavan. The robot localization problem. *SIAM Journal on Computing*, 26(4):1120–1138, 1997.

[18] 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.

[19] R. Inkulu and S. Kapoor. Visibility queries in a polygonal region. *Computational Geometry*, 42(9):852–864, 2009.

[20] 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.

[21] B. Joe and R. Simpson. Corrections to Lee’s visibility polygon algorithm. *BIT Numerical Mathematics*, 27(4):458–473, 1987.

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

[23] D. T. Lee and A. K. Lin. Computing the visibility polygon from an edge. *Computer Vision, Graphics, and Image Processing*, 34(1):1–19, 1986.

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

[25] F. P. Preparata and M. I. Shamos. *Computational Geometry: an introduction*. Springer-Verlag, New York, NY, USA, 1st ed. edition, 1985.

[26] 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.

[27] G. Vegter. The visibility diagram: a data structure for visibility problems and motion planning. In *Proceedings of Scandinavian Workshop on Algorithm Theory*, pages 97–110, 1990.

[28] 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.

[29] S. Kapoor and T. Singh. Dynamic Maintenance of Shortest Path Trees in Simple Polygons. In *Proceedings of Foundations of Software Technology and Theoretical Computer Science*, pages 123–134, 1996.

[30] R. E. Tarjan. *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1983.