Visibility Maintenance of a Moving Segment Observer inside Polygons with Holes

Hoda Akbari* Mohammad Ghodsi†

Abstract

We analyze how to efficiently maintain and update the visibility polygons for a segment observer moving in a polygonal domain. The space and time requirements for preprocessing are $O(n^2)$ and after preprocessing, visibility change events for weak and strong visibility can be handled in $O(\log |VP|)$ and $O(\log(|VP_1| + |VP_2|))$ respectively, or $O(\log n)$ in which $|VP|$ is the size of the line segment's visibility polygon and $|VP_1|$ and $|VP_2|$ represent the number of vertices in the visibility polygons of the line segment endpoints.

1 Introduction

Visibility problems have broad applications in several areas such as computer graphics, robotics and motion planning, and geographic information systems. Two points inside a polygon are said to be mutually visible iff their connecting segment remains completely inside the polygon (Figure 1). For a collection of point observers — or a segment observer as a special case — a point is weakly visible if it is visible from at least one of the points, and strongly visible if it is visible from all the points. For a line segment in a planar polygonal scene, the collection of all points weakly (strongly) visible to the observer forms a polygon called the weak (strong) visibility polygon(Figure 2).

In this paper, we discuss the problem of efficiently maintaining the weak and strong visibility polygons of a line segment moving in a static planar polygonal domain. The problem can arise in several real world applications, such as finding the regions illuminated by a fluorescent lamp moving among obstacles. As the observer moves, its visibility polygon changes combinatorially at discrete instants. We assume that the observer's coordinates at any instant can be determined by a fixed degree algebraic function of time. There is no other restriction on the motion path (e.g., linear, polyline, etc.). We further assume that the observer's motion equation is allowed to change. To the best of our knowledge, this problem has not been addressed before, and considering the previously solved related problems, has no trivial solution (See section 3.1 for more details).

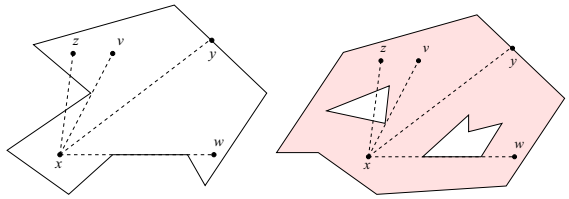

Figure 1: Visibility of points in (left) a simple polygon; and (right) a polygon with holes: v, y and w are visible from x, while z is not.

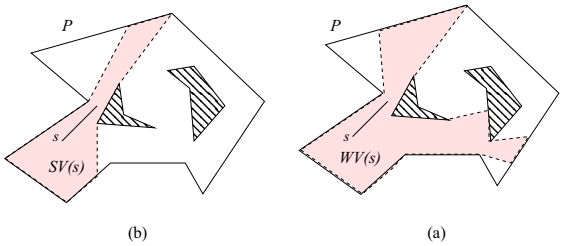

Figure 2: (a) Weak; and (b) strong visibility polygons of a segment observer in a polygon with holes.

This paper is organized as follows: We continue by a review of the related work in section 2. Section 3 explains the proposed solution. The algorithm's space and time complexity is analyzed in section 4.

2 Related Work

Visibility problems have been examined vastly in the computational geometry literature. Related work in this area can be put into three categories: basic problems, query problems and kinetic problems.

In the basic problems, locations of both the observer and the environment objects have been specified before, and the problem asks for finding parts of the environment visible to the observer. All the configurations are static as neither the environment objects nor the observer is considered moving, and once the visible region has been detected, no queries have to be answered anymore.

Among problems of this category, the most basic one is calculating the visibility polygon of a point observer

*Computer Engineering Department, Sharif University of Technology, Tehran, Iran, hodaa@sfu.ca

†Computer Engineering Department, Sharif University of Technology, and IPM School of Computer Science, Tehran, Iran, ghodsi@sharif.edu

in a simple polygon. Among several existing solutions to this problem, those of Elgindy and Avis, and Lee [2] can solve the problem in optimal linear time. The problem of finding the visibility polygon of a point observer located among polygonal obstacles has a worst-case optimal $O(n \log n)$ time solution, proposed by Suri et al. [18], in which they've made use of angular sweep technique and proved optimality by giving a reduction from the problem of sorting n numbers.

For a segment observer in a simple polygon, two problems are computing the weak visibility polygon and the strong visibility polygon. An important fact regarding a segment's strong visibility is that the strong visibility polygon is the visibility polygon of one of the segment's endpoints inside the visibility polygon of the other endpoint. This gives a way to compute the strong visibility polygon both in simple polygons and polygons with holes, which reduces the problem to problems regarding point observers. Based on this fact, there are optimal solutions for computing the strong visibility of a line segment in simple polygons and polygons with holes, which solve the problem in $O(n)$ and $O(n \log n)$ time respectively [2]. Computing the weak visibility is not so straightforward, and has solutions proposed by ElGindy [8], Lee and Lin [14] and Chazzelle and Guibas [7]. There is also an optimal solution of $O(n)$ time, which is a combination of linear time polygon triangulation algorithm and the algorithm of finding the weak visibility polygon of a line segment inside triangulated simple polygons [2].

Query problems are instances of visibility problems in which the observer's location is not determined in advance. Coordinates of the environment objects are fixed and as the observer's location is arbitrarily chosen in query time, suitable preprocessing structure should be built using which the visible region can be computed efficiently in query time. As there is a tradeoff between preprocessing time and space and query time, there is no single optimal solution to any of problems of this kind.

For a point in a simple polygon, there is an algorithm proposed by Bose et al. [6] in which a visibility change graph is constructed on the polygon's visibility decomposition structure, and visibility polygons for sinks of this graph are computed and stored in preprocessing time using $O(n^3 \log n)$ time and $O(n^3)$ space. Traversing this structure and applying proper changes, the visibility polygon $V(q)$ of a query point q can be computed in $O(\log n + |V(q)|)$ time.

Another solution to this problem is that of Aronov et al. [3], in which they build a hierarchical balanced triangulation structure on the simple polygon and preprocess each node of the triangulation tree, such that the visibility polygon of an external query point can be determined efficiently. At query time, the triangulation tree is traversed; external visibility polygons are computed and glued together to shape the final visibility polygon. The preprocessing time and space of this algorithm are $O(n^2 \log n)$ and $O(n^2)$ respectively, and the visibility polygon of a query point q can be reported in $O(\log^2 n + |V(q)|)$ time at query time.

Zarei et al. [19] give an algorithm for computing the visibility of a query point in polygons with holes. This algorithm uses $O(n^3 \log n)$ time and $O(n^3)$ space in preprocessing, after which the visibility polygon of any query point can be reported in $O((1+h') \log n + |V(q)|)$, in which n and h are the number of the vertices and holes of the polygon respectively, $|V(q)|$ is the size of the visibility polygon of q, and h' is an output and preprocessing sensitive parameter of at most $\min(h, |V(q)|)$.

Kinetic problems are the third category of visibility problems. In these problems, the observer or both the observer and environment objects are moving according to predetermined equations, and it is possible to impose restrictions on these movements to obtain a simpler problem or achieve a more efficient algorithm for a special case of the problem. As an example, algorithm of [17] can be considered of this category which is based on visibility complex, a data structure for storing the visibility relations between objects of the scene. The algorithm maintains the view around a point moving from point p to q in total running time of $O(\max(v(p), v(p, q)))$, where $v(p)$ is the size of the view around p and $v(p, q)$ the number of changes of visibility along (p, q).

In [3], the authors consider the problem of maintaining the view of a point moving along a polygonal path. This algorithm uses linear space and handles combinatorial changes in the visibility and changes in the motion in $O(\log^2 n)$ time. Zarei et al. [20] have made use of the method proposed in [19] to maintain the visibility of a point moving in a polygon with holes. The visibility events can be handled in $O(\log n)$ time. In [13], the authors have presented an algorithm based on maintenance of the shortest path tree for maintaining the strong and weak visibility of a moving segment observer in a simple polygon. Their preprocessing data structure is linear sized and can be constructed in linear time. Each change in the visibility can be computed in $O(\log^2(|VP|))$ time when the observer is allowed to change its direction, and in $O(\log(|VP|))$ time when the movement is along a given line.

3 Problem Solution

3.1 Why the Solution is not Trivial

Although the problem seems similar to the ones addressed in [13] and [20], their methods cannot be trivially extended to this problem. When holes are intro-

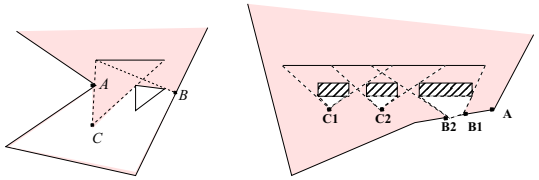

Figure 3: Vertex types in (left) strong and (right) weak visibility polygons. Observer is the horizontal line segment, and the shaded areas constitute its visibility polygon.

duced in a simple polygon, the visibility of a line segment cannot be determined solely based on the visibility of its endpoints — a crucial property needed for the algorithm in [13]. Also, a point visibility maintenance algorithm [20] is not readily extensible to our case, since in a polygon *with holes*, the visibility of a line segment cannot be determined by only tracking the visibility of its endpoints.

3.2 Visibility Polygon Vertices and Events

Vertices of a segment observer's visibility polygon can be considered of three types: type A vertices are the vertices of environment visible to the observer. type B vertices are vertices of the visibility polygon located on the boundary segments but somewhere other than their endpoints. type C vertices are those vertices formed inside the free area of environment, not on the polygon line segments (Figure 3). In strong visibility, the visibility polygon is always a simple polygon in which type A vertices are fixed and vertices of type B and C move as the observer changes place. type B and C vertices in a weak visibility polygon may be either fixed (subcases B2 and C2) or moving (subcases B1 and C1).

During the observer's motion, several events may occur. Vertices of each type can be added to or removed from the visibility polygon. We call the events leading to addition or deletion of type A vertices, as *type A events*. There are two ways for type B vertices to appear: at the same time a type A vertex is added or removed; or when a type C vertex approaches an edge until a collision occurs. The former can be handled at the same time of handling the corresponding type A event, and the latter is called a *split event*. Similarly, disappearance of type B vertices occurs either simultaneous with a type A event, or when during the motion two type B vertices on an edge get closer and closer until they collide. We name the latter a *merge event* (Figures 4 and 5).

3.3 Data Structures

Here, we give an outline of the data structures which will be referred to in subsequent sections. For further explanation and technical details see section 4.

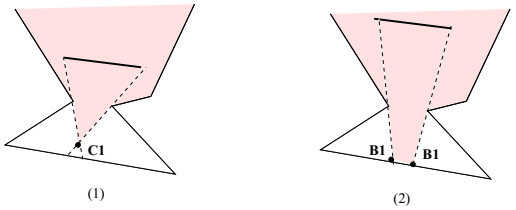

Figure 4: Split ($1 \rightarrow 2$) and merge ($2 \rightarrow 1$) events in strong visibility.

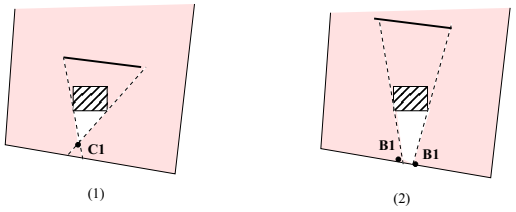

Figure 5: Split ($1 \rightarrow 2$) and merge ($2 \rightarrow 1$) events in weak visibility.

(VP_{env}) For each reflex vertex in the environment, we calculate in preprocessing its visibility polygon by applying angular sweep technique on a circular list of vertices around it sorted by angle.

(VP_{obs}) Visibility polygons of the observer's endpoints can be calculated in preprocessing at the same time as (VP_{env}). Throughout the visibility maintenance process, we update these visibility polygons. We also maintain the following pieces of information associated with each ray r from an observer's endpoint toward a visible vertex v:

(*pos*): the angular position of r in v's circular list constructed as part of (VP_{env}); and

(*hit edge*) : the first polygon edge met by extending r beyond v.

(BT) With each edge of the environment, we associate a binary tree. In binary tree of e, we maintain all type B vertices of the observer's visibility polygon which lie on edge e ordered based on their positions along e, such that predecessors/successors in the tree are adjacent type B vertices on e. Whenever a type B vertex is formed (because of a type A or split event), the vertex is inserted in the binary tree corresponding to its edge.

($TrigQuery$) A preprocessing data structure created by considering all the vertices, that can efficiently report the number of points inside any triangular query region.

($PolygIntersect$) Each of the visibility polygons constructed in VP_{env}, is preprocessed as a simple polygon,

such that given a query ray, we can efficiently determine if the ray intersects the polygon.

3.4 Detecting Events

type A Events – Vertex Appearance: Consider a type A event that causes a previously unseen vertex v of the environment to be added to the visibility polygon. We need to calculate the first time at which a ray from an endpoint of the observer toward a reflex vertex acting as an obstacle, reaches v. We use $(VP_{obs}(pos))$ to track extensions of all the rays connecting one endpoint of the observer to a visible reflex vertex which may act as an obstacle. type A vertex appearance events are in one-to-one correspondence with changes in $(VP_{obs}(pos))$. Position change times can be determined considering equations of fixed rays, and motion equation of the moving rays. When an appearance event occurs, we update each $(VP_{obs}(pos))$ to its neighboring position in the obstacle's (VP_{env}) list.

type A Events – Vertex Disappearance: This kind of event happens when a visible reflex vertex begins to act as an obstacle for the ray toward a previously visible vertex. To detect these events, we use (VP_{obs}) and consider the rays from one endpoint of the observer to two of its visibility polygon vertices. We compute instants at which two adjacent rays possibly become collinear. This can simply be done in $O(n)$ at the initial step, after the visibility polygons of the endpoints are found in preprocessing.

$B-C$ Events – Merge Event: When two type B vertices on a polygon edge move, they may gradually become closer until a collision takes place and the vertices become replaced with a type C vertex. When a new type B vertex is formed, (BT) is updated accordingly. Then considering positions and motion equations of predecessor and successor of the new vertex in the tree, we can predict possible collisions.

$B-C$ Events – Split Event: Split events are the reverse of merge events, meaning that a type C vertex approaches an edge, and after a collision takes place, it is replaced by two type B vertices on the edge it's collided with. Detecting this type of event can be performed noting the fact that at the very last instants before the collision, the hit edges associated with the two line segments adjacent to the type C vertex must be the same. Otherwise, we can conclude that the C vertex isn't close enough to any polygon edge to cross it. Changes in the hit edges can only occur when a type A event occurs, and therefore can be handled at the same time as type A events. The new hit edge is the edge adjacent to the previous hit edge in the visibility polygon of the vertex playing the obstacle's role in the type A event. Having maintained updated information about the hit edges, we can use it to predict split events: According to the motion equation of the observer and moving coordinates of the type C vertex, we can insert the possible split event (the instant at which the moving vertex crosses the hit edge) into the event queue if the extensions hit the same edge.

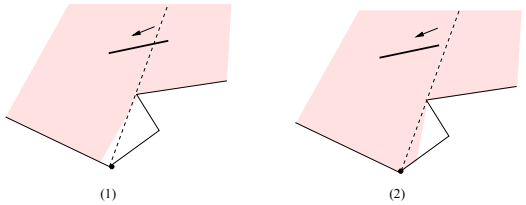

Figure 6: type A vertex appearance event in strong visibility (Case 1).

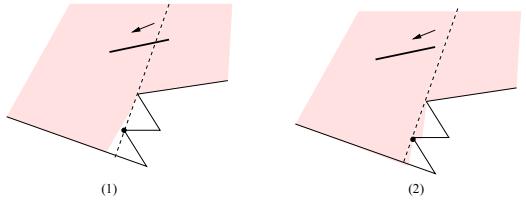

Figure 7: type A vertex appearance event in strong visibility (Case 2).

3.5 Event Handling

type A Events – vertex appearance: To process a vertex appearance event, we examine the event based on Table 1, and determine if it is an internal or external event. If external, the type A vertex should be inserted in the visibility polygon, and type B vertices must be properly updated. If the new vertex appears as a result of a type B vertex approaching a corner, the type B vertex is removed. Otherwise the type B vertex is updated such that its corresponding obstacle changes from the event's reflex vertex to the newly appeared vertex. Depending on whether or not the hit edge on which the type B vertices must be placed is adjacent to the newly appeared vertex in the original polygon, one or two new type B vertices should be inserted beside the newly appeared vertex respectively (Figures 6, 7, 8 and 9). In strong visibility, these two vertices may be merged at the same time to form a type C vertex. These vertices should reside on the edge of the visibility polygon of the reflex vertex, which is adjacent to the newly appeared point.

After applying changes to the visibility polygon, the event queue should be updated by recalculating the vertex disappearance events corresponding to the vertices adjacent to newly added vertex. Also, if a reflex vertex, the newly appeared vertex may act as an obstacle, and

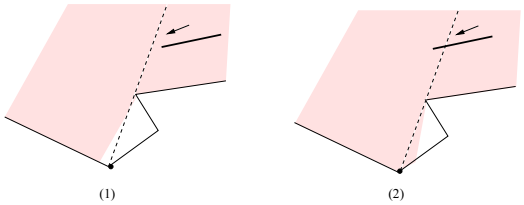

Figure 8: type A vertex appearance event in weak visibility (Case 1).

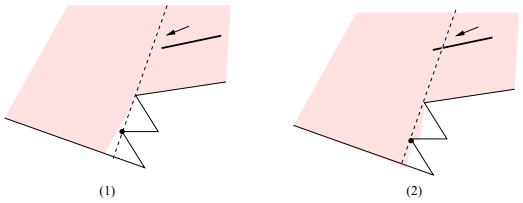

Figure 9: type A vertex appearance event in weak visibility (Case 2).

therefore according to the angular ordering of vertices in its visibility polygon and the observer's movement, the first vertex that may appear from behind this obstacle can be found by a binary search in the visibility polygon of this vertex.

Table 1: Different cases of a type A event.

	Weak Visibility	Strong Visibility
Vertex Appearance	Appears.	*Appears if seen by all the points on the observer. (Case 1)*
Vertex Disappearance	*Disappears if not seen by any point on the observer. (Case 2)*	Disappears.

type A Events – vertex disappearance: After verifying that the event is external based on table 1, we update the visibility polygon to reflect the changes. Visibility polygon updates in vertex disappearance events are the reverse of those of vertex appearance events. Vertex appearance and disappearance events whose obstacles are the newly disappeared vertex should be removed from the event queue.

$B-C$ Events Updating visibility polygon when merge or split events occur is as simple as the definitions of these events themselves. Either two type B vertices must be replaced by a type C vertex or the reverse of this change happens. In weak visibility, this change may result in the merging of two holes into one or merging a hole with the outer polygon for split events. The reverse

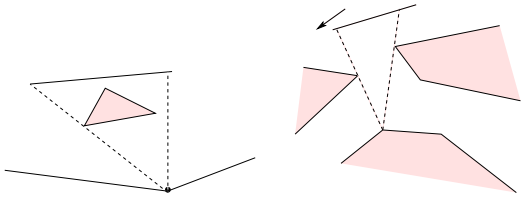

Figure 10: (left) Strong and (right) weak visibility of the vertex cannot be deduced solely based on visibility status of the endpoints of the observer.

of this scenario happens for merge events, resulting in forming a hole. When a split event happens, possible merge events of the new type B vertices and their adjacent type B vertices must be recalculated and inserted in the event queue. For a merge event, possible split event of the newly created type C vertex is inserted in the event queue.

3.6 Detecting internal events

Table 1 summarizes different cases we may face while handling a type A event, in which the italicized cells represent cases that are candidates of being internal. When these internal events occur, considering only states of the two endpoints of the observer is not sufficient to deduce visibility status of the vertex subject to the event (Figure 10). To identify internal events, using ($TrigQuery$) we check if any obstacle resides in the triangle bounded by the observer's endpoints and the point subject to the event. If this is the case, there are still some points on the observer behind obstacles with respect to the vertex and the event is an internal event of case 1. If an event of case 2 occurs, using ($PolygIntersect$) we can check if the observer still intersects the point's visibility polygon and therefore the point remains visible.

3.7 Handling Changes in the Motion Equation

Suppose the equation of motion of the observer changes. For appearance or disappearance events of type A, the vertex which is going to appear or disappear may change, but as this change is limited to one position change in the angular ordered lists, both detecting new events and updating event times can be done in time linear to the number of events. A similar discussion is valid for split and merge events. In all cases, the event queue must be reordered, which can be performed in $O(k \log k)$ time if k is the number of events in the queue.

4 Analysis of Time and Space Requirements

Preprocessing time and space: For each vertex(polygon vertices or observer's endpoints) we can

construct a circular list of vertices around it sorted by angle using $O(n^2)$ preprocessing time and space [15]. Using angular sweep technique on these lists both (VP_{env}) and (VP_{obs}) can be initialized. To initialize (pos) values, we apply a binary search technique on (VP_{env}) of each reflex vertex visible from one of the observer's endpoints. To set up $(TrigQuery)$, we use the following lemma:

Lemma 1 *We can preprocess a set of n points using $O(n^2)$ time and space, to create a data structure such that given a triangular query region Δ, the number of points inside Δ can be reported in $O(\log n)$ time [9].*

We apply the following lemma to each visibility polygon in (VP_{env}), to obtain the $(PolygIntersect)$ preprocessing data structure:

Lemma 2 *In a simple polygon with n vertices, using $O(n)$ time and space in preprocessing, the first intersection of an arbitrary ray with the polygon can be reported in $O(\log n)$ time [12].*

Taking all the above into account, preprocessing time and space are $O(n^2)$.

Size of the event queue: For each vertex in visibility polygons of the observer's endpoints, there may be at most one vertex appearance and one vertex disappearance at each time instant during the observer's movement; thus making a total number of $O(|VP_1|+|VP_2|)$. For each type B or type C vertex in the observer's visibility polygon, there may be at most one scheduled split or merge event. Therefore, total size of the event queue will be $O(|VP_1|+|VP_2|+|VP|)$.

Initializing the event queue: For vertex appearance events, as we have the ordered list of vertices around any vertex in the environment, appearance event considering each of the visible vertices as obstacle can be calculated in $O(\log n)$ time. Thus all events of this type can be computed in $O((|VP_1|+|VP_2|)\log n)$ time. All vertex disappearance events can be calculated by a linear scan of vertices of the observer endpoints' visibility polygons and creating possible disappearance events for each two adjacent vertices. This can be done in $O(|VP_1|+|VP_2|)$ time. The number of different split and merge events is $O(|VP|)$ and having the prepared preprocessing structures, each of these events can be computed in $O(1)$ time. The initial visibility polygon can be obtained using the existing static algorithms, requiring $O(n\log n)$ and $O(n^4)$ time for strong and weak visibility respectively.

Event handling time: For type A events, detecting whether the event is internal can be done in $O(\log n)$ time using the preprocessing structures. Computing new events and inserting them in the event queue can also be performed in total time of $O(\log n)$. Handling a split event includes inserting new type B events in their edge's binary tree of type B vertices. Updating the visibility polygon and calculating and updating merge events can be done in constant time. Excluding the $O(\log n)$ time needed for inserting events in the queue, merge events can be handled in $O(1)$ time as the necessary processing consists of updating visibility polygon and calculating possible split event of the new type C vertex.

Query time: Query processing requires no processing other than calculating the exact coordinates based on the combinatorial structure, in time linear to the output size.

5 Conclusion

We presented an algorithm for maintaining the visibility polygon of a line segment observer moving in a polygon with holes. Time and space requirements for preprocessing are both $O(n^2)$, which is a good result compared to worst-case optimal $O(n^4)$ time and space requirements of computing initial weak visibility polygon in the same environment. Efficient logarithmic time event handling and linear output sensitive query time have been achieved.

References

[1] A. Zarei, Efficient Visibility Computation and Simplification in Different Environments, Ph.D. Thesis, Sharif University of Technology, Tehran, Iran, Jun. 2008.

[2] T. Asano, S. K. Ghosh, and T. Shermer: Visibility in the Plane, *Handbook of Computational Geometry*, J.R. Sack and J. Urrutia Eds., Elsevier Science Publishers B.W.(2000), Chapter 19, 829-876.

[3] B. Aronov, L. Guibas, M. Teichmann and L. Zhang: Visibility Queries and Maintenance in Simple Polygons, *Discrete and Computational Geometry 27*(4) (2002), 461-483.

[4] B. Aronov, L. Guibas, M. Teichmann and L. Zhang: Visibility Queries in Simple Polygons and Applications, in *Proc. of Ninth Annual International Symposium on Algorithms and Computation (ISAA'98)* (1998), 357-366.

[5] J. Basch, L. Guibas, and J. Hershberger: Data Structures for Mobile Data, in *Proc. of the 8th Annual ACM-SLAM Symposium on Discrete Algorithms* (1997), 747-756.

[6] P. Bose, A. Lubiw and J. I. Munro: Efficient Visibility Queries in Simple Polygons, in *Proc. 4th Canadian Conference on Computational Geometry* (1992), 23-28.

[7] B. Chazelle and L. Guibas: Visibility and Intersection Problems in Plane Geometry, *ACM Trans. of Graphics 4* (1989), 551-581.

[8] H. A. ElGindy : Efficient Algorithms for Computing the Weak Visibility Polygon from an Edge, Technical Report MS-CIS-86-04, University of Pennsylvania (1986).

[9] P. P. Goswami, S. Das and S. C. Nandy : Triangular Range Counting Query in 2D and its Application in Finding k Nearest Neighbors of a Line Segment , *Computational Geometry: Theory and Applications 29*(3) (2004), 163-175.

[10] L. Guibas and J. Hershberger: Optimal Shortest Path Queries in a Simple Polygon, in *Proc. 3rd ACM Symposium on Computational Geometry* (1989) 50-63.

[11] O. A. Hall-Holt: Kinetic Visibility, Ph.D. Dissertation, Stanford University, California, USA, Aug. 2002.

[12] J. Hershberger and S. Suri: A Pedestrian Approach to Ray Shooting: Shoot a Ray, Take a Walk, *Journal of Algorithms 18* (1995), 403-431.

[13] A. A. Khosravi, A. Zarei and M. Ghodsi: Efficient Visibility Maintenance of a Moving Segment Observer inside a Simple Polygon, in *Proc. Canadian Conference on Computational Geometry* (2007), 249-252.

[14] D.T. Lee and A.K. Lin: Computing the Visibility Polygon from and Edge, *Comput. Vision, Graphics, and Image Process.* 34 (1986), 1-19.

[15] M. H. Overmars and E. Welzl : New Methods for Computing Visibility Graphs, in *Proc. the 4th annual Symposium on Computational Geometry (1988)*, 164-171.

[16] M. Pocchila and G. Vegter : The Visibility Complex, *International Journal of Computational Geometry Applications* 6(3) (1996) 279-308

[17] S. Riviere : Walking in the Visibility Complex with Applications to Visibility Polygons and Dynamic Visibility, in *Proc. 4th Canadian Conference on Computational Geometry* (1997).

[18] S. Suri and J. O'Rourke: Worst-Case Optimal Algorithms for Constructing Visibility Polygons with Holes, in *Proc. of the second annual Symposium on Computational Geometry* (1986), 14-23.

[19] A. Zarei and M. Ghodsi: Efficient Computation of Query Point Visibility in Polygons with Holes, *Symposium on Computational Geometry 2005*, 314-320.

[20] A. Zarei, A. A. Khosravi and M. Ghodsi: Maintaining Visibility Polygon of a Moving Point Observer in Polygons with Holes, *11th CSI Computer Conference (CSICC'2006)*, IPM School of Computer Science, Tehran, 2006.