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. 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.

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. 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. We further assume that the observer's motion equation is allowed to change. We'll discuss how all the above changes are handled efficiently.

Related Work: Calculating the visibility polygon of a point observer in a simple polygon or a polygon with holes have optimal linear time [1] and $O(n \log n)$ time [8] algorithms respectively. There are optimal solutions for computing the strong visibility polygon of a line segment in simple polygons and polygons with holes, using $O(n)$ space and $O(n \log n)$ time [1]. There are also results for weak visibility in [3] and [1].

Some related query problems have also been solved for computing the visibility of a point in a simple polygon [2] or polygon with holes [9].

There is also related work on kinetic problems for a point moving along a polygonal path [2], a point moving in a polygon with holes [10] and a segment observer moving in a simple polygon [6].

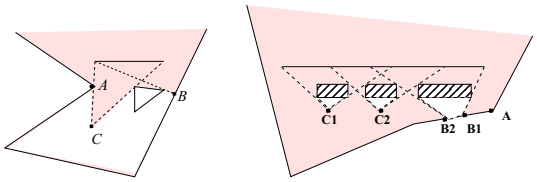

Figure 1: Vertex types in (left) strong and (right) weak visibility polygons. Observer is the horizontal line segment, and the shaded areas constitute its visibility polygon.

2 Problem Solution

2.1 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 1). 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

*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

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*.

2.2 Data Structures

Here, we give an outline of the data structures which will be referred to in subsequent sections. For explanation of technical details see section 3.

(VP_{env}) For each reflex vertex in the environment, we calculate in preprocessing its visibility polygon using 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, and 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 position 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.

2.3 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 equations 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.

2.4 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. 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 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 cause the merging of two holes into one or merging a hole with the outer polygon for split events. The reverse 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.

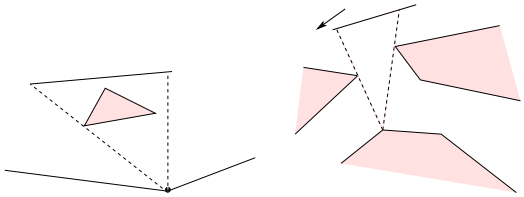

Figure 2: (left) Strong and (right) weak visibility of the vertex cannot be deduced solely based on visibility status of the endpoints of the observer.

2.5 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 2). 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.

2.6 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.

3 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 [7]. 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 [4].*

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 [5].*

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.

4 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] 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., Chapter 19, 829–876, 2000.

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

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

[4] 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 , *Comp. Geom.: Theory and App.*, 29(3):163–175, 2004.

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

[6] A. A. Khosravi, A. Zarei and M. Ghodsi, Efficient Visibility Maintenance of a Moving Segment Observer inside a Simple Polygon, in *CCCG*, 249–252, 2007.

[7] M. H. Overmars and E. Welzl, New Methods for Computing Visibility Graphs, in *SoCG*, 164–171, 1988.

[8] S. Suri and J. O'Rourke, Worst-Case Optimal Algorithms for Constructing Visibility Polygons with Holes, in *SoCG*, 14–23, 1986.

[9] A. Zarei and M. Ghodsi, Efficient Computation of Query Point Visibility in Polygons with Holes, *SoCG*, 314–320, 2005.

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