یازدهمین کنفرانس بین المللی کامپیوتر انجمن کامپیوتر ایران، پژوهشگاه دانشهای بنیادی، پژوهشکده علوم کامپیوتر، ۴ تا ۶ بهمن ۱۳۸۴، تهران، ایران.
11th International CSI Computer Conference (CSICC'2006), School of Computer Science, IPM, Jan. 24-26, 2006, Tehran, Iran.

Maintaining Visibility Polygon of a Moving Point Observer in Polygons with Holes*

Alireza Zarei
zarei@mehr.sharif.edu

Amir Ali Khosravi
a_khosravi@ce.sharif.edu

Mohammad Ghodsi
ghodsi@sharif.edu

Computer Engineering Department,
Sharif University of Technology,
P.O. Box 11365-9717, Tehran, Iran

Abstract

Computing the visible region from a moving point in planar environments has many applications in computer graphics and computational geometry. This problem has been considered thoroughly before and several algorithms have been proposed for it. Almost all these solutions use a preprocessing step to build data structures which reflect the visibility coherence of the scene. Then, this data is used to facilitate visibility computation for the moving observer. Since combinatorial structure of the observer visible area is changed in discrete points along its motion path, these algorithms maintain a queue of events which specifies these points. Unfortunately, in these algorithms either some unnecessary events are handled or their handling time is not efficient. In this paper, we present an algorithm for this problem which processes only necessary events as well as the events are handled efficiently. This algorithm uses the method of [1] to preprocess the scene. Although this preprocessing step is expensive, it helps to find and maintain visibility polygon of an arbitrary moving observer more efficiently. The method of [1] computes visibility polygon of a point and here we extend that method for a moving point observer.

Keywords: *Computational geometry, polygon with holes, visibility polygon, moving observer*

1 Introduction

Visibility problems have been defined on different environment and observer types. In this paper, we consider this problem in planar polygonal scenes for moving point observers inside the scene.

A planar polygonal scene which is also called a polygon with holes, is a 2D region with a polygonal border and there are some disjoint polygonal objects inside it. These objects which are also called holes, act as occluders and observers can not see through them.

A point observer q in such an environment can see a point p of this environment if pq does not intersect the environment boundary and the scene objects. The set of these visible points which compose a star-shaped polygon is called the visibility polygon and is denoted by $V(q)$ for an observer q. A moving observer can move in any direction and in a speed which is defined by a constant degree algebraic function over time.

In the rest of this paper, we denote by n the complexity of the scene which is simply the number of the vertices of the scene border and its holes. Also, the number of the scene objects(holes) is denoted by h.

The above problem has many applications in computer graphics, computer games, machine vision, robotics and motion planning. Therefore, it has been considered by many researchers and several algorithms have been proposed for it.

The main solutions for this problem have been presented in [2, 3, 4, 5, 6, 7, 8]. These are based on different notions including visibility decomposition, visibility complex, tangent visibility graph, kinetic data structure, topological map and radial subdivision. All these methods use a preprocessing step to prepare useful information about the visibility coherence of the scene. The prepared information of this step is then used to facilitate finding the initial value of $V(q)$ and updating it as the observer q moves. During the motion, a queue of events is built by which a visibility change is han-

*This work was supported by Sharif University of Technology

dled to update and maintain $V(q)$. Time and space complexities of the preprocessing phase, number of the events and processing time of an event are the major criteria which determine efficiency of these algorithms.

Algorithms of [5] and [6] are based on visibility complex which is a data structure for storing the visibility relations between objects of the scene. Visibility complex itself is based on tangent visibility graph(TVG) and can be constructed in time of $O(n \log n + k)$ in which k is the space complexity of the TVG of the scene with minimum and maximum values of $O(n)$ and $O(n^2)$ respectively. Having the visibility complex of the scene, $V(q)$ of an arbitrary point q is computed in $O(|V(q)| \log n)$ [9]. In [5], after preparing the visibility complex of the scene and computing the initial value of $V(q)$ according to [9], a visibility event is handled in $O(\log^2(|V(q)|))$ time. Also for a known motion path, events can be handled in $O(\log(|V(q)|))$ if we spend $O(|V(q)| \log(|V(q)|)$ extra preprocessing time. The algorithm of [6] can be used for an observer which moves along a line segment pq. For such an observer and after computing the initial value of $V(q)$ according to [5], changes of $V(q)$ are handled in $O(\max(|V(q)|, |V(p,q)|))$ in which $|V(p,q)|$ is the number of the visibility changes of q when it moves from p to q along the segment pq . The main deficiency of these algorithms is that the processed events do not necessarily change $V(q)$ and therefore some of the processed events are unnecessary.

Algorithm of [8] is based on visibility graph. In this algorithm, after constructing the visibility graph of the scene which is done in $O(m \log n)$ time (m is the size of the visibility graph which is between $O(n)$ and $O(n^2)$), a visibility event is handled in $O(\log m)$ time. In this method, there are too many events which are processed but do not change $V(q)$. In [4], this method was improved in such a way that only a subset of the visibility graph is constructed named radial subdivision or topological map of the scene with respect to the observer. Size of the radial subdivision is $O(n)$ and it is constructed in $O(n \log n)$ time. Then, the events are handled in $O(\log^2 n)$ time. Although preprocessing cost of this method is less than the one in [8], it still handles some unnecessary events. Also, event handling time in this algorithm is not optimal. Finally, in [3] this method was improved such that by spending $O(n)$ and $O(n \log n)$ preprocessing space and time, visibility events are handled efficiently and the number of the processed events and their handling time are optimal in average case but not in the worst case. The method of [3] is only applicable to convex objects which is extended for concave objects in [7].

In [2] another method was proposed which is based on visibility decomposition and shortest path tree and only can be used in simple polygons. This algorithm uses $O(n \log n)$ and $O(n)$ preprocessing time and space respectively. In this algorithm a visibility event is handled in $O(\log^2(|V(q)|)$ and it processes only necessary events.

As noted above, in none of these proposed solutions the number of processed events and their handling time are optimal concurrently. Also, computing the initial value of $V(q)$ is not efficient and they do not work efficiently for the cases where the observer position is selected randomly.

In this paper we propose an algorithm for this problem which processes the optimum number of events and handles the events efficiently. Also, the initial value of $V(q)$ is computed more efficient in compare with the previous algorithm. Our algorithm is based on the method of [1] which efficiently finds $V(q)$ for an arbitrary query point q inside a polygon with holes.

In the rest of this paper, we will first overview the method of [1] in Section 2. We extend that method for moving observers and analyze it in Section 3. Finally, the materials will be concluded in Section 4.

2 The Base Algorithm

We use the algorithm of [1] as the base algorithm to preprocess the scene and compute the initial value of $V(q)$. It is also used to handle the visibility events efficiently while the observer moves. In other word, we extend the algorithm of [1] for a moving point observer. So, we briefly describe this algorithm in this section.

The first step of this algorithm is to convert the initial polygon $\mathcal{P}$ into a simple polygon $\mathcal{P}_s$. This is done by inserting some diagonals, known as cut-diagonals, with an unfold process over these cuts. Then, $\mathcal{P}_s$ is preprocessed according to the notion of visibility decomposition to build data structures so that $V_s(q)$ of any arbitrary query point q in $\mathcal{P}_s$ can efficiently be reported. Visibility decomposition of a simple polygon $\mathcal{P}_s$ (or v-decomposition($\mathcal{P}_s$)), is to partition $\mathcal{P}_s$ into a set of smaller visibility regions R, called v-regions, such that for each region $A \in R$, the same sequence of vertices and edges of $\mathcal{P}_s$, called A's *visibility sequence* (or v-sequence(A)), are visible from any point in A. Figure 3 shows v-decomposition of a simple polygon.

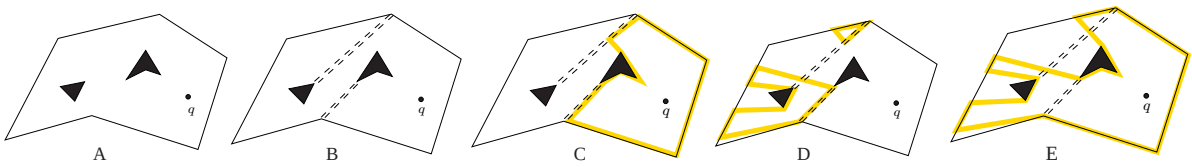

Figure 1: Computing $V(q)$ inside a polygon with holes: (A) The original polygon $\mathcal{P}$, (B) The cut-diagonals to produce a simple polygon $\mathcal{P}_s$, (C) The visibility polygon $V_s(q)$ targeted at $\mathcal{P}_s$, (D) Extra segments of $\mathcal{P}$ viewed from q through the cut-diagonals, and (E) The final $V(q)$ in $\mathcal{P}$.

The main properties of this decomposition are as follows: the v-regions are convex, v-sequences of two adjacent v-regions differ in only one vertex, and all points of a v-region have equivalent visibility polygons.

The v-sequences of two adjacent v-regions in a simple polygon differ only in a single vertex which is visible from the points of one region and is invisible from the others. This fact helps reduce the space complexity of maintaining the v-sequences of the v-regions in simple polygons. This is done by defining the *sink regions*. A region is sink if the size of its v-sequence is smaller than that for any of its adjacent regions. It is therefore sufficient to only maintain the v-sequences of the sinks, from which the v-sequences of all other regions can be computed. This is done by constructing a directed dual graph over the v-regions and maintaining the difference between v-sequences of adjacent v-regions as the label of the edges of this graph [11, 10].

The data structures of v-decomposition is used to find $V_s(q)$, the set of segments viewed by q in $\mathcal{P}_s$ which is a preliminary version of the final $V(q)$. After a refinement process, the final $V(q)$ is computed from $V_s(q)$. A procedure named See-Through performs this refinement process. This is done by constructing a data structure for each cut-diagonal so that the visible portions of the polygon, from an arbitrary query point, through that cut-diagonal can be obtained efficiently. Having these structures, any segment of the cut-diagonals which appears in $V_s(q)$ is replaced by the portions of the polygon that are visible through that segment. This process continues until no cut-diagonal is remained unrefined in $V_s(q)$.

Figure 1 depicts an example of the algorithm. The original polygon $\mathcal{P}$ and its simple version $\mathcal{P}_s$ are shown in parts (A) and (B) respectively. $V(q)$ in $\mathcal{P}_s$, denoted by $V_s(q)$ is computed as shown in part (C). There are portions in $\mathcal{P}$ that are visible from q through the cut-diagonals of $V_s(q)$ which are shown in part (D). These portions are computed (recursively) by the See-Through algorithm to replace the cut-diagonals of $V_s(q)$ which leads to the final $V(q)$, as shown in part (E).

As shown in Figure 2, a query point q, views a portion of a cut-diagonal l_1 through an angle α, which is bounded by two reflex vertices. These vertices may be the endpoints of l_1 or reflex vertices of $\mathcal{P}$ that lie between q and l_1. In this figure, u and v are the reflex vertices associated with q and l_1.

When this algorithm is performed for q in Figure 2, l_1 exists in $V_s(q)$ (actually, a segment of l_1) which must be replaced by applying See-Through. This will replace l_1 by another cut-diagonal e. Applying See-Through on e will further replace e by another cut-diagonal l_2. Finally, See-Through on l_2 replaces l_2 by l_3 and an edge of $\mathcal{P}$. Therefore, we could have started with l_2 as if l_1 and e never existed. This is true for all points q in the polygon whose visibility angle is bounded by u and v and the lines qu and qv extend within angles γ and θ, respectively. The region containing these points is shown in gray in Figure 2. For this purpose, another data structure is prepared by which these *ineffective* intermediate cut-diagonals can be skipped. A cut-diagonal e is called *q-ineffective* if processing e only replaces a segment of another cut-diagonal with that in e. Otherwise, e is *q-effective*. In Figure 2, processing e is equivalent to substituting it with another cut-diagonal l_2 and therefore it is q-ineffective (for the shown q).

To ignore ineffective cut-diagonals, for any pair of reflex vertices, a data structure is maintained so that, for any query point q, it efficiently determines the first q-effective cut-diagonal. For each reflex vertex v, the different angular ranges around v are computed through which an observer sees different segments of $\mathcal{P}$. Parameters δ and θ are examples of such ranges for reflex vertex v in Figure 2. These ranges are produced by connecting v to the vertices of $\mathcal{P}$ that are visible from it. These ranges produce a radial decomposition of $\mathcal{P}$ around v which is referred to as RD_v.

For each pair of reflex vertices u and v, another data structure, denoted as $\mathrm{VR}_{u,v}$ (for *Visibility Ranges*), is built over these vertices' radial decompositions. In this data structure, $\mathrm{VR}_{u,v}(\alpha, \beta)$ is

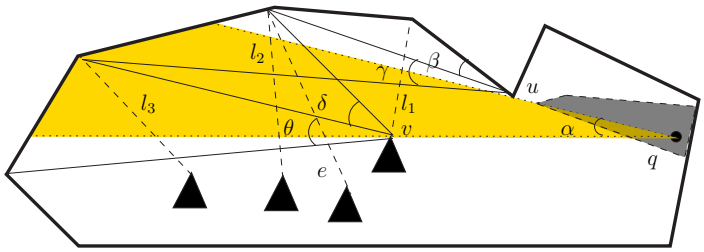

Figure 2: The cut-diagonal e is q-ineffective for any q in the gray region shown.

the first effective cut-diagonal to all points whose lines of sight lie within the ranges α of u and β of v. For each (α, β), so that α and β are two angular ranges of RD_u and RD_v respectively, $\mathrm{VR}_{u,v}(\alpha, \beta)$ is computed and maintained.

These data structures are prepared in the preprocessing phase. For any query point q that sees a cut-diagonal through the reflex vertices u and v, the algorithm finds the ranges α and β which respectively are the ranges in which the extensions qu and qv lie. Having (α, β), its associated q-effective cut-diagonal, $\mathrm{VR}_{u,v}(\alpha, \beta)$, is found from $\mathrm{VR}_{u,v}$ and reported as the first q-effective diagonal which must be processed by the See-Through procedure.

For a polygon of total n vertices and h holes, this algorithm needs the preprocessing time and space of $O(n^3 \log n)$ and $O(n^3)$, respectively. Any query can be handled in time $O((1 + h') \log(n) + |V(q)|)$ in which h' is an output and preprocessing sensitive parameter of at most $\min(h, |V(q)|)$. The value of h' for a query point depends on the position of the point and the cut-diagonals, and the upper bound of h' is reached only in special cases.

3 Moving Observer

The initial visibility polygon of a point observer q is computed using the base algorithm. But, when the observer moves, $V(q)$ is changed continuously. For a while, these changes do not alter the combinatorial structure of $V(q)$, meaning that the sequence of visible vertices and edges of $V(q)$ is not changed. However, as the observer moves further, some new vertices and edges will be visible to q and some of its previously visible vertices or edges may not be visible any more.

We know that $V(q)$ only contains the sequence of visible vertices and edges, and the exact visible area is computed from $V(q)$ by a linear trace of it. Therefore, in order to correctly maintaining $V(q)$ as the observer moves, it is enough to detect the times at which $V(q)$ is changed and as soon as any one of such changes occurs $V(q)$ is updated accordingly.

According to the definition of the visibility decomposition, all points of a region of the visibility decomposition of a simple polygon have the same sequence of visible vertices and edges. Therefore, while the containing region of the observer q has not changed, $V(q)$ is valid and whenever it leaves its containing region and enters another one, $V(q)$ is also changed. We denote by CR_q the containing region of q in the visibility decomposition. This is shown in Figure 3. While q lies in region A, $V(q)$ is constant. But when it enters region B or C, one of the previously visible vertices and edges will be hidden to it and must be removed from $V(q)$. Based on the the direction and speed of the observer motion, the time at which this event occurs is computed and predicted. This type of events are called *Decomposition Event* or *D-event* for simplicity.

As discussed in Section 2, we add some cut-diagonals to convert the initial polygon with holes into a simple polygon. D-events only reflect the visibility changes with respect to this simple polygon. There is another type of events which occurs because of visibility changes through cut-diagonals. Such an event is called a *Cut-Diagonal Event* or simply a *C-event*.

A C-event occurs whenever the sequence of visible vertices and edges through a visible cut-diagonal from the observer is changed. In Figure 4, the initial polygon is converted into a simple polygon by adding uu' cut-diagonal. The observer q sees some portions of the polygon through this cut-diagonal. Vertices u and v bound the vision angle of the observer through this cut-diagonal and α and β are visibility ranges of v and u which contain the supporting lines of qv and qu. These ranges are denoted by $CR_{q\to v}$ and $CR_{q\to u}$, respectively. As observer q moves, while these containing ranges are fixed, the sequence of the visible vertices and edges to q are not changed. Therefore, a C-event oc-

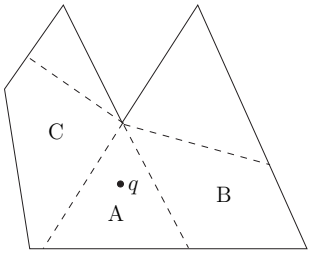

Figure 3: While q lies inside region A, $V(q)$ is constant and it is changed as q leaves A and enters one of the regions B and C.

curs whenever one of the supporting lines of qu or qv leaves its containing range and enters another range of its associated vertex. In other word, a C-event occurs whenever $CR_{q\to v}$ or $CR_{q\to u}$ is changed. For example, whenever the supporting line of qu leaves the range β and enters θ or γ as shown in Figure 4. Knowing the direction and speed of the observer motion is sufficient to compute and predict the time at which such an event occurs.

Theorem 1 . *D-events and C-events reflect all visibility changes of a moving observer in polygons with holes.*

Proof. Clearly, in simple polygons only D-events happen and according to the definition of the visibility decomposition, any one of the changes of $V(q)$ for a moving observer q corresponds to a D-event.

On the other hand, according to the definition of the visibility ranges of reflex vertices, vision of an observer q through a cut-diagonal will be changed only if $CR_{q\to u}$ or $CR_{q\to v}$ changes in which u and v are the bounding vertices of the vision angle of q through the cut-diagonal. Any one of these changes corresponds to a C-event and therefore is detected properly.

Consequently, all types of visibility changes of a moving observer in polygon with holes can be predicted and identified by D-events and C-events. □

3.1 Event queue initialization

The initial value of $V(q)$ is computed using the base algorithm. In order to detect visibility change events, a queue of future events is prepared. These events are composed of a single D-event and $2h'$ number of C-events and their occurrence times depend on the observer motion. Assume that the observer moves along a straight line with a known

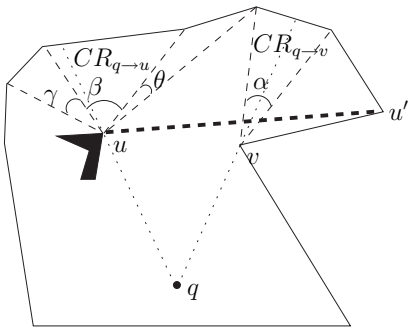

Figure 4: A C-event occurs whenever the containing range of one of the supporting lines of the segments qu and qv which connect the observer to the bounding vertices of its vision through a cut-diagonal uu' is changed or simply whenever $CR_{q\to v}$ or $CR_{q\to u}$ is changed.

speed. Before starting the motion, the events of this queue are identified as follows.

When the base algorithm is used to find initial value of $V(q)$, CR_q is identified as well. Therefore, the D-event is identified by determining the edge e of CR_q which is crossed by q and the time t at which this intersection takes place is attached to this event as its occurrence time. Since we have the sequence of the edges of CR_q, e is found by a binary search on CR_q edges.

C-events belong to the cut-diagonals which are visible to q. During the initial computation of $V(q)$ and whenever See-Through algorithm is applied to a visible cut-diagonal, $CR_{q\to u}$ and $CR_{q\to v}$ ranges are also computed and maintained. A C-event occurs whenever $CR_{q\to u}$ or $CR_{q\to v}$ changes. Knowing the direction and speed of the observer motion, the time at which such a change happens is computed and its associated event is built.

These C-events and the single D-event are maintained in a priority queue and in each step the nearest event is removed from this queue and is handled to apply the occurred change to $V(q)$.

Trivially, this queue depends on the direction of the observer motion and whenever the observer changes its motion direction all of these events must be computed again and their priorities must be computed with respect to the new direction.

3.2 Handling events

When a D-event happens, it means that the observer q has left its old CR_q and enters a new one. Absolutely, the new CR_q is adjacent to the old one and their common edge has been identified when the D-event is computed. In order to handle this

event, the old one is removed from the event queue and the new D-event which belongs to the new CR_q is inserted into the queue.

Updating $V(q)$ is the main task which must be done when an event occurs. Whenever CR_q is changed, it means that a vertex and an edge which were not visible to q before the event, are now visible or a previously visible vertex and an edge are not further visible to q. The direction and label of the underlying graph of the visibility decomposition determines which of these two cases has happened. For the case of losing visibility, the previously visible vertex and edge are removed from $V(q)$. For the case of acquiring visibility, the newly visible vertex and edge are inserted into their proper position in $V(q)$.

However, the newly appeared or disappeared edge may be a cut-diagonal. In such cases, if a new cut-diagonal has been added to $V(q)$, the SEE-THROUGH algorithm is applied to it and $V(q)$ is updated according to this algorithm. Also the new C-events are also added to the event queue. We remind that applying SEE-THROUGH algorithm may lead to appearance of another cut-diagonal in $V(q)$. Therefore, this update must be done recursively on all cut-diagonals which appear in $V(q)$.

A C-event happens if $CR_{q\to u}$ is changed for a visible vertex u which bounds the vision angle of q through a visible cut-diagonal. In such events, $CR_{q\to u}$ is transferred to one of the adjacent ranges of the old one. In order to handle this event, the old one is removed from the event queue and the new C-event which belongs to the new $CR_{q\to u}$ is inserted into the queue. Moreover, $V(q)$ must also be updated here. This update is exactly the same as what we described above for D-events.

We assume that the observer does not intersect the boundary of the polygonal scene and the objects inside it. However, it is possible that the observer crosses a cut-diagonal. In such cases the initial value of $V(q)$ and the event queue are built from scratch as if the observer has been selected now at its new position after crossing the diagonal.

3.3 Analysis of the algorithm

In this subsection we analyze efficiency of the proposed method. We first compute size of the event queue and time complexity of computing an event and initializing and maintaining the event queue. Then, we compute the time required to handle each type of events and update $V(q)$ when an event happens. Finally, the effect of changing the motion direction is considered.

Lemma 1 . *Size of the event queue at a given time is equal to $1+2h'$ in which h' is the number of the effective cut-diagonals to the observer at that time.*

Proof. Always, there is a single D-event. Also, there are two C-events for any one of the visible and effective cut-diagonals. Since at each time stamp we have h' of such diagonals, the total number of the events is $1+2h'$. □

Lemma 2 . *Knowing CR_q, the D-event is computed in $O(\log(|V(q)|)$ time.*

Proof. Knowing CR_q, its associated D-event is computed by finding the edge of CR_q which is crossed by q and the time at which this intersection takes place. This edge is found using a binary search on CR_q edges. The time of occurrence of this event is obtained from the direction and speed of the observer motion and the distance between the observer and this edge. This computation is assumed to be done in constant time. So, it is enough to show that CR_q has $O(|V(q)|)$ number of edges.

Any one of the edges of CR_q belongs to two vertices of $\mathcal{P}$ where at least the closer one is visible to q. In addition, this closer vertex can be in common between at most two edges of CR_q. The reason is that the supporting lines of all visibility decomposition edges which belong to a singe vertex intersect each other in that vertex. Thus, a visibility region always lies between two adjacent lines of them. □

Lemma 3 . *Knowing $CR_{q\to u}$, its associated C-event is computed in constant time.*

Proof. For these events, it is enough to compute the time at which the supporting line of qu overlaps one of the edges of $CR_{q\to u}$. If the direction and speed of the observer motion are known, this will be computed in $O(1)$ time. □

Theorem 2 . *Events of the initial placement of the observer can be found and maintained in a priority queue in $O(h'\log(h')+\log(|V(q)|))$ time.*

Proof. When we use the base algorithm to find the initial value of $V(q)$, CR_q and $CR_{q\to u}$ of all events are also determined. This fact along with the above three lemmas result in the theorem. □

Whenever the direction of the observer motion is changed, the new event queue can also be constructed using the same amount of time as what

was described in Theorem 2. The reason is that we already know CR_q and $CR_{q\to u}$ for all visible cut-diagonals at the time at which the motion direction has changed. So,

Lemma 4 . *Whenever the direction of the observer motion is changed, new event queue can be constructed in $O(h' \log(h') + \log(|V(q)|))$ time.*

On an event occurrence, other than just updating the event queue whose cost was computed in lemmas 2 and 3, $V(q)$ must also be updated with respect to the occurred event. The following theorem specifies the upper bound of the total time required to handle an event.

Theorem 3 . *Any one of the D-events and C-events is handled in $O(\log n)$ time.*

Proof. Whereas CR_q or $CR_{q\to u}$ of the occurred event is known, the new event will be computed in $O(\log(|V(q)|)$ or $O(1)$ if the event is a D-event or C-event, respectively. Moreover, since the size of the event queue is $O(h')$, removing the old event and inserting the new one is done in $O(\log(h'))$ time. The event is applied to $V(q)$ by removing a vertex and an edge from it or by inserting a vertex and an edge into it. Both of these works can be done in $O(\log(|V(q)|))$ time. However, the removed edge may be a cut-diagonal. In such cases, the associated events of that cut-diagonal must be removed from the event queue. This is done in $O(1)$ time if we maintain pointers from visible cut-diagonals to their C-events. On the other hand, the added edge may be a cut-diagonal to which the See-Through algorithm must be applied. This is done in $O(\log(n))$ time. The reason is that it is enough to find the initial ranges of the C-events of this cut-diagonal and add to $V(q)$ the portions of $\mathcal{P}$ which are visible to q through this cut-diagonal. Since at the time of the event occurrence only a single vertex or edge is visible through this cut-diagonal, its computation time is constant and $O(\log n)$ time is sufficient to find the initial ranges of the C-events. Recursive calling of the See-Through algorithm is also done in $O(\log n)$ time because ineffective cut-diagonals are ignored.

Consequently, the total time required to handle a visibility change of a moving observer is equal to $O(\log n)$. □

It is considerable that the number of the events which are processed in this algorithm is efficient and all of the processed events are necessary while the previous algorithms sometimes process unnecessary or ineffective events. Moreover, the events are handled efficiently which is a result of the prepared data structures of the preprocessing phase.

4 Conclusion

In this paper, we considered the problem of maintaining visibility polygon of a moving point in a polygon with holes. This problem has been considered before and several algorithms have been proposed for it. In all of these solutions, the underlying polygon is processed to prepare data structures by which $V(q)$ of an observer q is computed. In order to efficiently handle changes of $V(q)$ as q moves, a queue of events is produced in which the future changes of $V(q)$ are maintained and scheduled in order of their occurrence time. Efficiency of these algorithms is identified by the size of the preprocessing data structures and their computation time, the size of the event queue, the time required to handle an event and optimality of the number of the handled events. These criteria are not consistent and the solutions make a trade-off among them.

In comparison, our algorithm spends more preprocessing time and maintains larger data structures. But, the initial value of $V(q)$ for an arbitrary observer q is computed more efficiently than the previous algorithms and the size of the event queue and the event handling time are also more efficient.

Another considerable achievement of this algorithm is that it only processes the necessary events while in the previous algorithms some unnecessary events are also processed.

Our algorithm uses [1] to preprocess the initial polygon and to compute the initial value of $V(q)$ efficiently. It also uses a refined version of the data structures of [1] to produce and handle the future events efficiently.

Like [1], our algorithm requires $O(n^3 \log n)$ time to preprocess the initial polygon of n vertices and h holes and build data structures of total size of $O(n^3)$. Having these data structures, the initial value of $V(q)$ is computed in $O(|V(q)| + h' \log n)$ time in which $h' = O(\min(h, |V(q)|))$. The size of the event queue in this algorithm is $1 + 2h'$ at each moment and it can be initialized in $O(\log(|V(q)|) + h' \log h')$ time. An event is handled in $O(\log n)$ time in this algorithm, which we think that it can be reduced to $O(\log(|V(q)|))$.

There are several extension points and future works related to this method:

1. How can this method be used to compute weak or strong visibility polygon of a line segment?
2. How can this method be used to maintain the visibility polygon of a moving segment?

3. Is it possible to use this method in dynamic scenes in which the objects also move?

4. Is it possible to reduce the preprocessing time and space of this method without increasing the size of its event queue or the event handling time?

References

[1] A. Zarei and M. Ghodsi, *Efficient Computation of Query Point Visibility in Polygons with Holes*, In Proc. 21st Annual Symposium on Computational Geometry, June 6-8, 2005, Pisa, Italy.

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

[3] Olaf H. Holt, *Kinetic Visibility*, PhD Thesis, 2002.

[4] K. Nechvile and P. Tobola, *Local approach to dynamic visibility in the plane*, In Seventh Int. Conf. in Central Europe on Computer Graphics and Visualization, WSCG '99, February 1999.

[5] S. Rivi'ere, *Dynamic visibility in polygonal scenes with the visibility complex*, In Proc. 13th ACM Sympos. Comput. Geom., pp. 421-423, 1997.

[6] S. Rivi'ere, *Walking in the Visibility Complex with Applications to Visibility Polygons and Dynamic Visibility*, In Proc. Canadian Conf. on Comp. Geom., 1997.

[7] S. Hornus and C. Puech, *A Simple Kinetic Visibility Polygon*, In Proc. 18th EWCG'02, pp. 27-30 - 2002.

[8] S. Ghali and A. J. Stewart, *Incremental update of the visibility map as seen by a moving viewpoint in two dimensions*, In Seventh International Eurographics Workshop on Computer Animation and Simulation, pp. 1-11, August 1996.

[9] M. Pocchiola and G. Vegter, *The visibility complex*, Internat. J. Comput. Geom. Appl., 6(3):279308, 1996.

[10] P. Bose, A. Lubiw and J. I. Munro, *Efficient Visibility Queries in Simple Polygons*, Computational Geometry: Theory and Applications, 23(3), pp. 313-335, 2002.

[11] L. Guibas, R. Motwani and P. Raghavan, *The robot localization problem in two dimensions*, SIAM Journal of Computing, 26(4):1120–1138, 1997.