Available online at www.sciencedirect.com

ScienceDirect

Computational Geometry 39 (2008) 78–90

Computational Geometry
Theory and Applications

www.elsevier.com/locate/comgeo

Query point visibility computation in polygons with holes [☆]

Alireza Zarei [a,1], Mohammad Ghodsi [a,b,*,2]

[a] *Computer Engineering Department, Sharif University of Technology, Tehran, Iran*
[b] *IPM School of Computer Science, Niavaran, Tehran, Iran*

Received 11 July 2005; received in revised form 1 October 2006; accepted 2 February 2007

Available online 29 April 2007

Communicated by P.G. Spirakis

Abstract

In this paper, we consider the problem of computing the visibility of a query point inside polygons with holes. The goal is to perform this computation efficiently per query considering the cost of the preprocessing phase. Our algorithm is based on solutions in [A.L.P. Bose, J.I. Munro, Efficient visibility queries in simple polygons, Computational Geometry: Theory and Applications 23 (3) (2002) 313–335] and [M.T.B. Aronov, L. Guibas, L. Zhang, Visibility queries and maintenance in simple polygons, Discrete and Computational Geometry 27 (4) (2002) 461–483] proposed for simple polygons. In our solution, the preprocessing is done in time $O(n^3 \log n)$ to construct a data structure of size $O(n^3)$. It is then possible to report the visibility polygon of any query point q in time $O((1+h')\log n + |V(q)|)$, 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 $\min(h, |V(q)|)$. This is claimed to be the best query-time result on this problem so far.
© 2007 Elsevier B.V. All rights reserved.

Keywords: Visibility polygon; Visibility decomposition; Polygon with holes

1. Introduction

Two points inside a polygon are visible from each other if their connecting segment remains completely inside the polygon. The visibility polygon of a point q, called $V(q)$, in a polygon $\mathcal{P}$ is defined as the set of points in $\mathcal{P}$ that are visible from q. The problem of finding $V(q)$ of a point q has been considered for two decades. For simple polygons, linear time optimal algorithms have been proposed [4,5,8,10]. For polygons with holes, the worst case optimal algorithms in $O(n \log n)$ time were presented in [1] and [17]. This was later improved to $O(n + h \log h)$ in [6].

[☆] A preliminary version of this paper appeared in proceedings of the 21st Annual Symposium on Computational Geometry, Pisa, Italy, 2005 [A. Zarei, M. Ghodsi, Efficient computation of query point visibility in polygons with holes, in: Proc. 21st Annual Symposium on Computational Geometry, ACM, 2005, pp. 314–320].

[*] Corresponding author at: Computer Engineering Department, Sharif University of Technology, Tehran, Iran.
E-mail addresses: zarei@mehr.sharif.edu (A. Zarei), ghodsi@sharif.edu (M. Ghodsi).

[1] This author's work has been partly supported by Iran Telecommunication and Research Center (ITRC).

[2] This work has been partly supported by a grant from IPM school of CS (No. CS1385-2-01).

0925-7721/$ – see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.comgeo.2007.02.005

The query version of this problem in polygons with holes has been considered by few. The visibility complex and visibility decomposition are two methods of performing the preprocessing step. The visibility complex was first presented in [14], in which $V(q)$ of any query point q can be reported in time $O(|V(q)|\log n)$ with $O(n \log n)$ preprocessing time and $O(n)$ space. This is the best result so far for polygon with holes. In comparison, our method uses more preprocessing time and space, but the query is performed more efficiently in most cases.

The notion of visibility decomposition is to decompose the polygon into "visibility regions" so that all points in a single such region have equivalent visibility polygons. Two visibility polygons are equivalent if they both are composed of the same sequence of vertices and edges from the underlying polygon. In the visibility decomposition, the visibility regions are determined in the preprocessing phase and their visibility polygons are computed and maintained in a proper data structure. For any query point q, $V(q)$ can then be obtained by refining the visibility polygon of the region that contains q.

Using visibility decomposition in a simple polygon with n vertices, $V(q)$ can be reported in time $O(\log n + |V(q)|)$ by $O(n^3 \log n)$ of preprocessing time and $O(n^3)$ space [9,13]. Another improvement to this result was done in [2] where the preprocessing time and space were reduced to $O(n^2 \log n)$ and $O(n^2)$ respectively, at the expense of more query time of $O(\log^2 n + |V(q)|)$.

In this paper, we apply the visibility decomposition to polygons with holes with some extensions. In an overall view, our algorithm adds some new diagonals to our polygon (called *cut-diagonals)* so that the polygon can be unfolded along these diagonals and converted into a simple polygon. Then, we use an existing algorithm on the simple polygons (one of [9] or [13]) to compute a preliminary version of the $V(q)$, denoted as $V_s(q)$. $V_s(q)$ is then refined to find the final $V(q)$. Performing this refinement step efficiently is the main contribution of this paper.

The preprocessing needed for the refinement step 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 $V_s(q)$.

For a polygon of total n vertices and h holes, our algorithm needs the preprocessing time of $O(n^3 \log n)$ and space of $O(n^3)$. 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 $\min(h, |V(g)|)$.

In the rest of this paper and in Section 2, the visibility decomposition will be applied to polygons with holes and its time and space complexities are analyzed. In Section 3, the new algorithm will be presented. This algorithm is later improved in Section 4.

2. Visibility decomposition

Let $\mathcal{P}$ be a polygon with h holes $H_1, H_2, \ldots, H_h$. Also let q be the query point for which the visibility polygon $V(q)$ is to be computed. A visibility decomposition of $\mathcal{P}$ (or v-decomposition($\mathcal{P}$)), is to partition $\mathcal{P}$ 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 $\mathcal{P}$, called A's *visibility sequence* (or v-sequence(A)), are visible from any point in A.

To construct a v-decomposition($\mathcal{P}$), we first identify the boundary segments of its regions R and then construct a subdivision from these segments. The boundary segments are either the edges of $\mathcal{P}$, or segments that are called *windows*. As shown in Fig. 1, a window uu' is an extension of the segment between two mutually visible vertices u and v. The points below the window uu' are not visible from v, while the upper points are. It is easy to prove that no other kinds of segments are involved in construction of the v-decomposition. More details on the properties of such a decomposition can be found in [13] and [2]. In these papers, the v-decomposition is only defined on simple polygons. However, it is straightforward to apply this notion on polygons with holes too.

Furthermore, some proved properties are also valid for the general polygons. 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.

Lemma 1. *The number of the v-regions of a polygon with holes $\mathcal{P}$ is $O(n^4)$ and this bound is tight.*

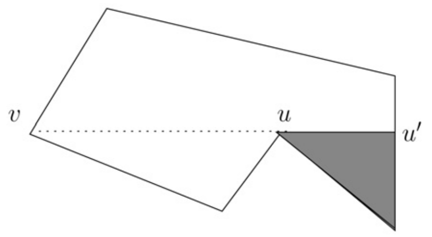


Fig. 1. uu' is a window of the polygon.

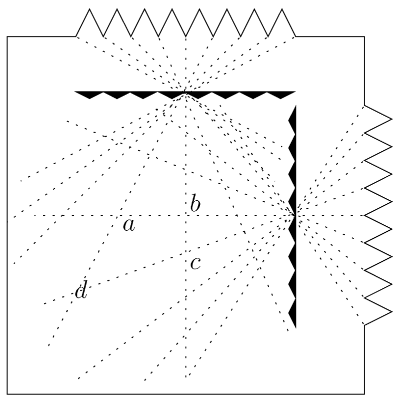


Fig. 2. A polygon with $O(n^4)$ v-regions and sinks.

Proof. Each vertex of $\mathcal{P}$ can be an endpoint of at most n different windows. Hence, the number of all windows is $O(n^2)$. Any two windows can intersect which will lead to at most $O(n^4)$ v-regions. This bound is tight as shown in Fig. 2. □

It is easy to see that 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 other. 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 [9,13].

In a simple polygon, there are $O(n^2)$ sinks. This reduces the space requirement of v-decomposition of a simple polygon to $O(n^3)$. Unfortunately, the above property does not hold for polygons with holes.

Lemma 2. *The space complexity of maintaining the v-sequences of the regions in a polygon with holes is* $O(n^5)$.

Proof. This bound is trivially true, because the number of v-regions is $O(n^4)$ and any one of these regions can have a v-sequence of size $O(n)$. On the other hand, Fig. 2 shows a polygon with $O(n^4)$ sink regions each with v-sequences of size $O(n)$. Along each edge of a region, like *abcd* in this figure, a vertex is visible which is invisible inside that region. Hence, getting out of this region from each side, increases the v-sequence. Therefore, any one of the v-regions, like *abcd*, contains at least one sink. This leads to the space complexity of $O(n^5)$, since there are $O(n^4)$ sink regions. □

By computing the v-regions R of $\mathcal{P}$ and maintaining their v-sequences, $V(q)$ of an arbitrary point q inside $\mathcal{P}$ can be computed as follows: A point location structure is built over the v-regions. From this, the region $r(q)$ containing q can be found in time $O(\log n)$. The v-sequence $(r(q))$ is then traced and refined to exactly compute $V(q)$. This refinement takes a linear time in terms of the size of $V(q)$ [13]. Therefore,

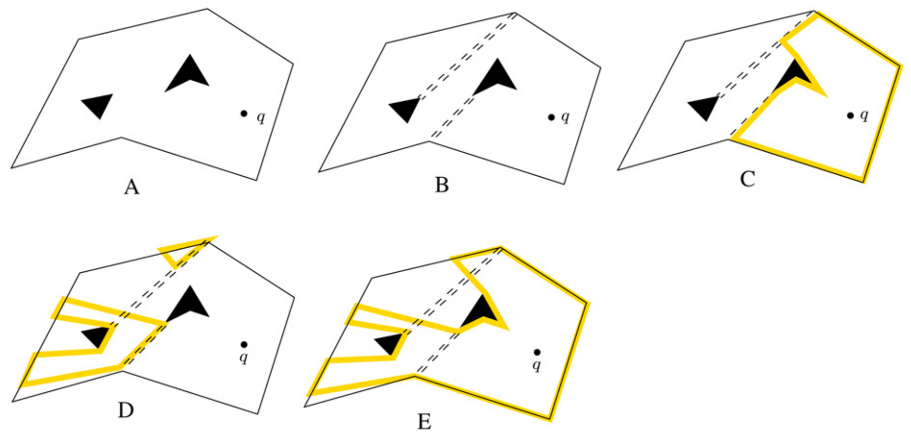


Fig. 3. Computing $V(q)$ inside a polygon with holes: (A) The original polygon $\mathcal{P}$, (B) the cut-diagonals to produce a simple polygon $\mathcal{P}_s$, (C) the visibility polygon $V_s(q)$ targeted at $\mathcal{P}_s$, (D) extra segments of $\mathcal{P}$ viewed from q through the cut-diagonals, and (E) the final $V(q)$ in $\mathcal{P}$.

Theorem 1. *Using* $O(n^5 \log n)$ *time to preprocess a polygon* $\mathcal{P}$ *and maintaining a data structure of size* $O(n^5)$*, it is possible to report* $V(q)$ *in time* $O(\log n + |V(q)|)$.

Proof. We first compute the $V(r)$ for each vertex r of $\mathcal{P}$ [17]. All windows can then be found in time $O(n^2 \log n)$. The v-decomposition and its dual graph can be constructed in time $O(n^4 \log n)$ [3]. The point location structure on the v-decomposition can be constructed in time $O(n^4 \log n)$ [7,11,15,16]. Since, there can be $O(n^4)$ sinks, computing the v-sequences takes $O(n^5 \log n)$ time and the size of any one of these v-sequences can be $O(n)$. Hence, the total preprocessing time is $O(n^5 \log n)$ and the size of the required data structure is $O(n^5)$. $V(q)$ can be found in time $O(\log n + |V(q)|)$ as described above. □

3. The proposed algorithm

Clearly, the time and space complexities of the previous algorithm is too high and it is not acceptable in all applications. In this section we present a new algorithm that needs less preprocessing time and space at expense of more cost of query time.

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. $V_s(q)$ in $\mathcal{P}_s$ is computed using an algorithm in [9,13]. After a refinement process, the final $V(q)$ is computed from $V_s(q)$. A SEE-THROUGH procedure, to be described in Section 3.2, performs this refinement process.

Fig. 3 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).

The details of the algorithm is described in the following subsections.

3.1. Creating a simple polygon from $\mathcal{P}$

We produce a simple polygon $\mathcal{P}_s$ from $\mathcal{P}$ by eliminating its holes. One hole, say H, in $\mathcal{P}$ can be eliminated by adding a cut-diagonals connecting a vertex of H to a vertex of $\mathcal{P}$ in its outer boundary. Cutting $\mathcal{P}$ along these diagonals produces another polygon in which H is no longer a hole. We continue this process on this new polygon to eliminate all holes. A cut-diagonal should lie completely inside $\mathcal{P}$ and should not intersect any other holes. This can be enforced if we eliminate the leftmost hole first; i.e., the hole with the smallest x-coordinate of its leftmost corner.

For $\mathcal{P}$ with total of n vertices and h holes, $\mathcal{P}_s$ will have $n + 2h$ vertices. We know that the upper bound of h is $\lfloor \frac{n-3}{3} \rfloor$. Hence, the number of the vertices of $\mathcal{P}_s$ is also $O(n)$.

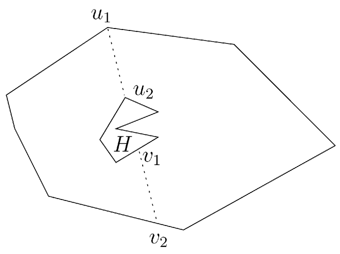


Fig. 4. The imaginary replacement of u_1u_2 by v_1v_2 in SEE-THROUGH(H).

The conversion algorithm described above can be done by first triangulating the polygon and then selecting the proper cut-diagonals, which can be done in $O(n \log n)$ [12].

3.2. *The visibility through cut-diagonals*

As mentioned before, an important step in our approach is the SEE-THROUGH algorithm that updates the $V_s(q)$ on $\mathcal{P}_s$. This step finds new segments of edges in $\mathcal{P}$ which are visible from q through the cut-diagonals. We use the idea presented in [2], which shows that in a simple polygon P of size n and a diagonal that cuts P into two parts, L and R, one can use an $O(n^2 \log n)$ preprocessing time algorithm to construct a data structure of size $O(n^2)$ so that, for any query point $q \in R$, the partial visibility $V_L(q)$ through the diagonal can be reported in $O(\log n + |V_L(q)|)$ time. This is also true when a portion of the diagonal is visible to q and that portion is known.

This result can also be used for our purpose with a simple justification. Assume that $\mathcal{P}$ has only one hole H_i which has been eliminated by one cut-diagonal u_1u_2, as shown in Fig. 4. For any query point q, we intend to find the set of segments of edges in $\mathcal{P}$ that are visible from q through u_1u_2. Continuing u_1u_2 through H_i will lead to another segment v_1v_2 such that v_1 is on H_i and v_2 is the first encounter of this segment with the boundary of $\mathcal{P}$. Now, suppose that the cut-diagonal u_1u_2 is replaced by v_1v_2. Obviously, cutting $\mathcal{P}$ along v_1v_2 (instead of along u_1u_2) will produce another simple polygon, called P'_{H_i}, for which u_1u_2 is an internal diagonal. We can now use the method in [2] to preprocess P'_{H_i} and build an appropriate data structure so that for any query point q, we can find the segments of P'_{H_i} that are visible from q through u_1u_2. These segments are denoted as $V_{H_i}(q)$. Since, no parts of v_1v_2 is visible from q through u_1u_2, $V_{H_i}(q)$ is the set of segments we are looking for. We denote this method by SEE-THROUGH(H_i).

This algorithm can be extended to more holes by performing SEE-THROUGH(H_i) once for each H_i assuming that $\mathcal{P}$ has effectively been cut along the cut-diagonals of other holes, which leads to a polygon with only one hole H_i. Therefore, we have h data structures resulted from these preprocessing steps. Given the query point, the extra segments of $\mathcal{P}$ visible from q through all the cut-diagonals is found by recursively using SEE-THROUGH for each cut-diagonal appearing somewhere in $V_s(q)$.

3.3. *The algorithm*

The preprocessing phase of the algorithm is done as follows:

(a) Add all cut-diagonals to produce the simple polygon $\mathcal{P}_s$, as described in Section 3.1.
(b) Preprocess $\mathcal{P}_s$ and create the data structure so that $V_s(q)$ of any arbitrary query point q in $\mathcal{P}_s$ can efficiently be reported. This step is done as [13]; a simple polygon P can be preprocessed in $O(n^3 \log n)$ time and $O(n^3)$ space such that given an arbitrary query point q inside the polygon, $O(\log n + |V(q)|)$ time is sufficient to recover $V(q)$.
(c) For each hole, H_i perform SEE-THROUGH(H_i) to preprocess the polygon, so that for any query point q, $V_{H_i}(q)$ can be computed efficiently.

Doing the above preprocessing steps, $V(q)$ of any query point q is computed as follows. The data structure built at step (b) of the preprocessing phase is used to find $V_s(q)$, the set of segments viewed by q in $\mathcal{P}_s$. Suppose that a segment $u'v'$ of a cut-diagonal uv associated with a hole H_i is a part of $V_s(q)$. Since the endpoints of $u'v'$ are known, the preprocessing of step (c) is used to find $V_{H_i}(q)$, the extra segments viewed through $u'v'$. The segment $u'v'$ in $V_s(q)$

is then replaced by $V_{H_i}(q)$. This is continued for any such segments in $V_s(q)$, and is finished without encountering any loops, due to the nature of visibility. What remains at the end is $V(q)$, and this is easy to prove. The reason is that any visible segment is either visible directly or through some cut-diagonals which is found in both cases.

Lemma 3. *The preprocessing time and space complexities of the algorithm are* $O(n^3 \log n)$ *and* $O(n^3)$*, respectively.*

Proof. Time complexity of the preprocessing of step (a), as described in Section 3.1, is $O(n \log n)$. Also, the resulting polygon $\mathcal{P}_s$ has $O(n)$ vertices. The time and space complexities of step (b) are $O(n^3 \log n)$ and $O(n^3)$, respectively [13].

As described in Section 3.2, the preprocessing time for any cut-diagonal is of size $O(n^2 \log n)$ and the size of its data structure is $O(n^2)$. There are at most $O(n)$ such diagonals in $\mathcal{P}$, one for each hole. Thus, the total preprocessing time to construct the cut-diagonal data structures is $O(n^3 \log n)$ and they require $O(n^3)$ space. Therefore, the total preprocessing time of the algorithm is $O(n^3 \log n)$ and the prepared data structures require $O(n^3)$ space. □

Lemma 4. *The query time to report* $V(q)$ *is* $O(\log n + h' \log n + h' + |V(q)|)$ *where* h' *is the number of cut-diagonals appearing in* $V_s(q)$ *during the algorithm.*

Proof. A point location of time $O(\log n)$ is done to find the location of q in $\mathcal{P}_s$ and $V_s(q)$ can be built in $O(|V(g)|)$ time. For any one of the h' cut-diagonals appearing in $V_s(q)$, a point location of size $O(\log n)$ is required to run the SEE-THROUGH algorithm to find $V_{H_i}(q)$. The cut-diagonal is then substituted in $V_s(q)$ by $V_{H_i}(q)$ in $O(|V_{H_i}(q)|)$ time. The number of the edges that appear in the initial $V_s(q)$ and all $V_{H-i}(q)$'s is the size of the final visibility polygon $V(q)$ plus the number of the cut-diagonals appeared in $V_s(q)$ and then removed, which is h'. □

Lemma 5. *The upper bound of* h' *is* $O(h^2)$ *and this bound is tight.*

Proof. The cut-diagonals do not intersect each other except at their end-points. Therefore, if a query point q sees a cut-diagonal l through another cut-diagonal l', then it is impossible for q to see l' through l. Also, only a single segment of another cut-diagonal can *directly* be seen from a query point through another cut-diagonal. By directly we mean that there is no other intermediary cut-diagonals between them. Hence, the upper bound of h' is $O(h^2)$. Fig. 5 shows a sample with tight bound of h'. □

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 in special cases. However, h is $O(n)$ and therefore, the upper bound of h' is $O(n^2)$, which theoretically is too weak. In the next section, we will improve the algorithm and overcome this weakness. This improvement reduces the upper bound of h' to $\min(h, |V(q)|)$ without increasing the preprocessing time and space complexities.

4. Improving the algorithm

As shown in Fig. 5, in some cases, the process of a cut-diagonal e_1 (i.e., SEE-THROUGH(e_1)) does not directly change the final visibility polygon; the algorithm takes useless action on e_1 and moves one step further to another cut-diagonal, like e_2. It would have obtained the same result, if it had skipped e_1 and started from e_2. We will show that

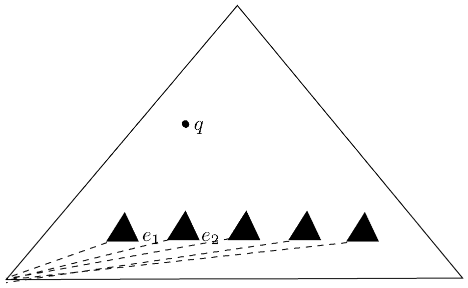


Fig. 5. A polygon with a tight bound of h'.

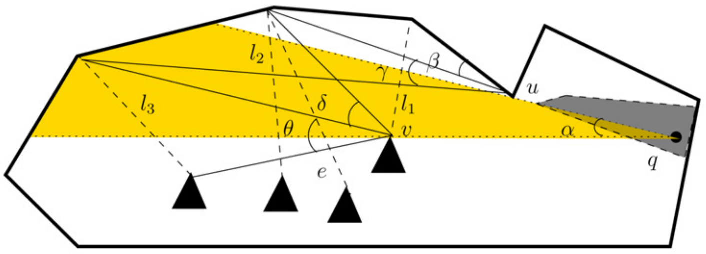


Fig. 6. The cut-diagonal e is q-ineffective for any q in the gray region shown.

the upper bound of h' occurs only when there are many of these cases. If we preprocess to skip these ineffective cases, the upper bound of h' is shown to reduce effectively. For this purpose, we prepare another data structure 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 Fig. 6, processing e is equivalent to substituting it with another cut-diagonal l_2 and therefore it is q-ineffective (for the shown q).

As shown in Fig. 6, a query point q, views a portion of a cut-diagonal l_1 through an angle α, which is constrained 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.

In our algorithm, l_1 is in $V_s(q)$ (actually, a segment of l_1 is) 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 Fig. 6.

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, we first compute the different angular ranges around v through which an observer sees different segments of $\mathcal{P}$. Parameters δ and θ are examples of such ranges for reflex vertex v in Fig. 6. 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. To avoid degenerate cases, we assume that no three vertices of $\mathcal{P}$ are collinear.

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 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, we compute 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 cut-diagonal, which must be processed by the SEE-THROUGH procedure.

Clearly, $\mathcal{P}$ can have $\mathrm{O}(n)$ reflex vertices and the size of RD_v for any reflex vertex v can be $\mathrm{O}(n)$. Hence, there can be $\mathrm{O}(n^2)$ pairs of reflex vertices u and v for which $\mathrm{VR}_{u,v}$ must be maintained. Naively, according to the size of RD, any one of the VR data structures can be of size $\mathrm{O}(n^2)$, and thus, the total size of VR's is $\mathrm{O}(n^4)$.

In the next subsection, we prove that maintaining the $\mathrm{O}(n)$ entries of $\mathrm{VR}_{u,v}$ is sufficient to find $\mathrm{VR}_{u,v}(\alpha,\beta)$ for any values of α and β. So, the total space requirement of these structures is reduced to $\mathrm{O}(n^3)$.

4.1. Space complexity of $\mathrm{VR}_{u,v}$

Assume that a query point q sees a cut-diagonal l whose visibility is limited by reflex vertices u and v and the supporting lines of qu and qv cross through α and β ranges of RD_u and RD_v, respectively. According to the definition of $\mathrm{VR}_{u,v}$, the value of $\mathrm{VR}_{u,v}(\alpha,\beta)$, for some values of α and β is equal to l itself and it is not necessary to store these entries of $\mathrm{VR}_{u,v}$. In addition, qu and qv do not intersect each other and whenever they cross through the α and β ranges, these ranges must be divergent. By divergent we means that the lower bounding line of the right angle does

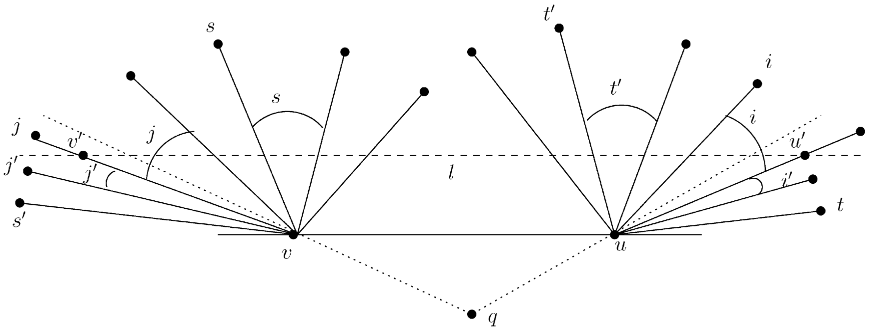


Fig. 7. Only O(n) entries of $\mathrm{VR}_{u,v}$ are not empty.

not intersect the upper bounding line of the other. Two ranges are either divergent or convergent. Therefore, the entries of $\mathrm{VR}_{u,v}(\alpha, \beta)$ for convergent values of α and β are not important and thus are not maintained. Moreover, the values of some entries can be derived from the values of other entries and thus, it is not required to maintain them in $\mathrm{VR}_{u,v}$. These entries are specified in the second option of the proof of the next lemma.

Lemma 6. *It is sufficient to maintain* O(n) *entries of* $\mathrm{VR}_{u,v}$. *Other entries can be obtained from them. Therefore, the size of* $\mathrm{VR}_{u,v}$ *for any pair of reflex vertices is* O(n).

Proof. Consider the reflex vertices u and v for which $\mathrm{VR}_{u,v}$ is to be constructed. If the segment uv intersects $\mathcal{P}$, then it is impossible for u and v to be the bounding vertices for the vision of a query point. Therefore, it is not required to maintain $\mathrm{VR}_{u,v}$ in such situations. So, we assume that this segment is contained completely inside $\mathcal{P}$. The RD ranges of u and v are considered in counter clockwise order and are nominated according to Fig. 7. According to this nomination, the range i of vertex u is bounded from above by segment ui, in which i is a vertex of $\mathcal{P}$. However, RD_u has such ranges only for those vertices of $\mathcal{P}$ which are visible to u.

Assume that the entry $\mathrm{VR}_{u,v}(i, j)$ is not empty and its value is l. This means that for any query point q, that qu and qv extend within the i and j ranges of RD_u and RD_v, respectively, all of the cut-diagonals that lie between l and uv are q-ineffective.

In addition, assume that i and j are the locally farthest apart so that there is no range i' of RD_u before i with nonempty entry of $\mathrm{VR}_{u,v}$, (i, j') and no range j' of RD_v after j with nonempty entry of $\mathrm{VR}_{u,v}(i, j')$. These assumptions lead to the following conclusions:

1. The region $uu'v'v$ does not contain any vertex of $\mathcal{P}$ and has no intersection with edges of $\mathcal{P}$. This is a trivial result of the existence of l as the value of $\mathrm{VR}_{u,v}(i, j)$.
2. For any range s of v lying before j and any range t' of u that exists after i such that s and t' are not convergent, $\mathrm{VR}_{u,v}(t', s)$ is equal to l or another cut-diagonal l' that lies farther than l from uv. If the value of this entry of $\mathrm{VR}_{u,v}$ is equal to l, then there is no need to be maintained and can be induced from the value of $\mathrm{VR}_{u,v}(i, j)$. We do not maintain such unnecessary entries in $\mathrm{VR}_{u,v}$ and leave them empty. When we search the value of $\mathrm{VR}_{u,v}(\alpha, \beta)$ for nonconvergent α and β, the value of $\mathrm{VR}_{u,v}(i, j)$ is returned in which i is the greatest range of u lying before α, j is the smallest range of v lying after β, and $\mathrm{VR}_{u,v}(i, j)$ exists.
3. For any range s before j and ranges t and t' before and after range i respectively, only one of the $\mathrm{VR}_{u,v}(t, s)$ and $\mathrm{VR}_{u,v}(t', s)$ can be nonempty. The reason is that the entry $\mathrm{VR}_{u,v}(t', s)$ can be nonempty only if s exists on the right side of the range i (see Fig. 7). If the entry $\mathrm{VR}_{u,v}(t, s)$ is also nonempty with value l', which is distinct from l, because of the maximality of l, then l and l' must intersect or the right end-point of l, that is a vertex of $\mathcal{P}$, must exist between l' and uv. However, none of these conditions is possible.
4. For any range t' after i and ranges s and s' before and after range j respectively, only one of the $\mathrm{VR}_{u,v}(t', s)$ and $\mathrm{VR}_{u,v}(t', s')$ can be nonempty. The proof is the same as above.
5. For any pair of ranges i and i' of u there is at most one range j of v such that both of $\mathrm{VR}_{u,v}(i, j)$ and $\mathrm{VR}_{u,v}(i', j)$ are nonempty. This is a trivial result of the above two items.

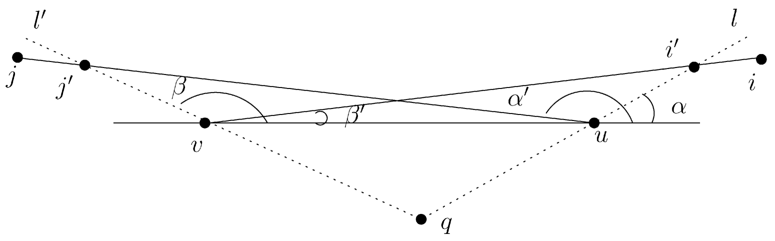


Fig. 8. $\mathrm{VR}_{u,v}(\alpha,\beta)$, $\mathrm{VR}_u(\alpha,\alpha')$ and $\mathrm{VR}_v(\beta',\beta)$ are equal.

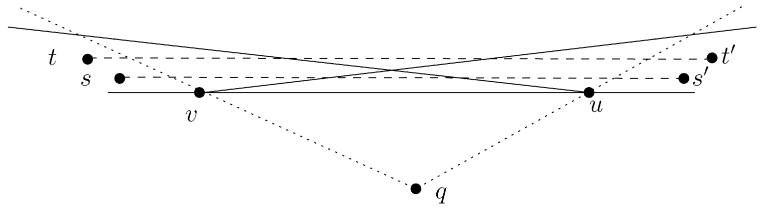


Fig. 9. ss' and tt' segments cannot differ and must be a single cut-diagonal.

The last conclusion implies that there are at most $2n$ nonempty entries in $\mathrm{VR}_{u,v}$ and thus the lemma is correct. □

Therefore, we can maintain the necessary entries of $\mathrm{VR}_{u,v}$ in a data structure of size $O(n)$ from which the values of $\mathrm{VR}_{u,v}(\alpha,\beta)$ for any α and β can be obtained. In the next subsection, we will describe how to construct this structure and how it can affect the efficiency of the proposed algorithm.

4.2. Building $\mathrm{VR}_{u,v}$

According to Fig. 8, assume that a view point lies on vertex u and its vision angle is limited from angle α to α'. We define $\mathrm{VR}_u(\alpha,\alpha')$ to be the first effective cut-diagonal to this view point.

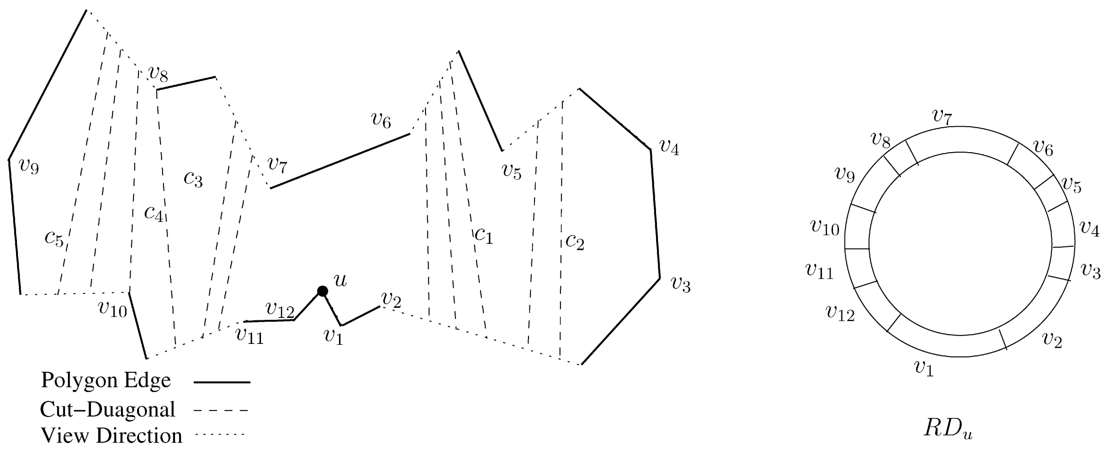
Consider a query point q (see Fig. 8) which sees within the reflex vertices u and v and therefore, its vision angle is limited from angle α to angle β. Assume that β' is the greatest visibility range of vertex v with an endpoint i that lies before the line l, so that there is no vertex in the vui' region. Moreover, assume that α' is the smallest visibility range of u with an endpoint j that lies after the line l', such that there is no vertex in the vuj' region (see Fig. 8).

Lemma 7. *If none of* $\mathrm{VR}_u(\alpha,\alpha')$ *and* $\mathrm{VR}_v(\beta',\beta)$ *intersect* vu*,* $\mathrm{VR}_{u,v}(\alpha,\beta)$ *is the same as* $\mathrm{VR}_u(\alpha,\alpha')$ *and* $\mathrm{VR}_v(\beta',\beta)$*.*

Proof. We first prove the equality of $\mathrm{VR}_u(\alpha,\alpha')$ and $\mathrm{VR}_v(\beta',\beta)$. Definition of i and j in Section 4.1, forces the equality of $\mathrm{VR}_u(\alpha,\alpha')$ and $\mathrm{VR}_v(\beta',\beta)$. If one is empty, then the other one must also be empty. Assume that $\mathrm{VR}_u(\alpha,\alpha')$ is empty. This means that there is a vertex t which is directly (not through a cut-diagonal) visible to u within its defined vision angle. This vertex is contained inside the region with $luj'l'$ boundary or the region with $l'j'j$ boundary. The latter is impossible due to the definition of j which is the smallest (in polar coordinate around u) visible vertex to u existing after the line l'. So, the former case must be true.

On the other hand, according to the definition of i, this vertex cannot exist inside the $i'uv$ region. Therefore, it must be contained inside the region with $li'vl'$ boundary. This implies that $\mathrm{VR}_v(\beta',\beta)$ must also be empty. If $\mathrm{VR}_v(\beta',\beta)$ is not empty, then the vertex t would not be further visible to u directly, but through this cut-diagonal. This contradicts our assumption. So, either both $\mathrm{VR}_u(\alpha,\alpha')$ and $\mathrm{VR}_v(\beta',\beta)$ are empty or none of them are.

Now, assume that both are nonempty and they are not equal. According to the definition of i and j as discussed above, the endpoints of these cut-diagonals must lie under the chain $jj'vui'i$ in Fig. 8. While these cut-diagonals do not intersect each other and the uv segment, they must be similar to the segments tt' and ss' of Fig. 9. Trivially, ss', the closer one to vu, is either effective to both u and v or ineffective to both of them, which means that these cut-diagonals must be unique.



L_1	(v_3, v_5, c_2)	(v_3, v_6, c_1)	(v_8, v_{11}, c_3)	(v_9, v_{10}, c_5)	(v_9, v_{11}, c_4)
L_2	(v_3, v_5, c_2)	(v_3, v_6, c_1)	(v_9, v_{10}, c_5)	(v_9, v_{11}, c_4)	(v_8, v_{11}, c_3)

VR_u

Fig. 10. Building VR_u and RD_u.

Since, $\mathrm{VR}_{u,v}(\alpha, \beta)$, if it is not empty, must be existed above the line uv, a similar argument implies equality of $\mathrm{VR}_{u,v}(\alpha, \beta)$ with $\mathrm{VR}_u(\alpha, \alpha')$, and $\mathrm{VR}_{u,v}(\alpha, \beta)$ with $\mathrm{VR}_v(\beta', \beta)$. $\square$

If both $\mathrm{VR}_u(\alpha, \alpha')$ and $\mathrm{VR}_v(\beta', \beta)$ intersect vu, they must intersect each other or at least one of them has an end point in vuq region. The cut-diagonals cannot intersect; so the latter option must happen. For this case, $\mathrm{VR}_{u,v}(\alpha, \beta)$ is trivially empty. Without loss of generality, assume that only $\mathrm{VR}_u(\alpha, \alpha')$ intersects vu. This implies that $\mathrm{VR}_v(\beta', \beta)$ is empty and thus $\mathrm{VR}_{u,v}(\alpha, \beta)$ is also empty.

Knowing this relation between $\mathrm{VR}_{u,v}$ and VR_u, we can think of $\mathrm{VR}_{u,v}$ as the composition of VR_u and the α' ranges. These α' ranges which are computed with respect to v, are maintained in a data structure called $\mathrm{VR}'_{u,v}$. Then, for any query point q that sees within the reflex vertices u and v with visibility ranges of α and β respectively, to find $\mathrm{VR}_{u,v}(\alpha, \beta)$, we first find α' from $\mathrm{VR}'_{u,v}$ associated with the range β. If $\mathrm{VR}_u(\alpha, \alpha')$ is not empty and it does not intersect vu, it will be reported as the value of $\mathrm{VR}_{u,v}(\alpha, \beta)$ (first q-effective cut-diagonal) and empty is reported otherwise. We can also use $\mathrm{VR}_{u,v}$ and VR'_v instead of $\mathrm{VR}'_{u,v}$ and VR_u, in the same manner.

Therefore, for our purpose, building $\mathrm{VR}_{u,v}$ is equivalent to building $\mathrm{VR}_{u,v}$, and $\mathrm{VR}'_{u,v}$.

We use the original version of our algorithm (before the improvement) to compute VR_u. For a vertex u, $V(u)$ is computed and during this computation, when an effective cut-diagonal appears in $V(u)$, its VR_u, is updated accordingly. Moreover, RD_u is also constructed as a byproduct of this process. As shown in Fig. 10, RD_u entries are maintained in a circular sorted list. Entries of VR_u which are of the form (α, β, cut-diagonal) are maintained in two sorted lists L_1 and L_2. The entries in L_1 are sorted in ascending order of their start ranges (α) and if two entries have equal start ranges, they will be sorted by their end ranges (β) in ascending order. The entries in list L_2 are sorted in ascending order of their end ranges (β) and if two entries have equal end ranges they will be sorted by their start ranges (α) in descending order. These lists are shown in Fig. 10 for vertex u.

The value of $\mathrm{VR}_u(\alpha, \beta)$ is found as follow. The entries (i, j, c) and (i', j', c') of VR_u are found where (i, j, c) is the greatest item in L_1 so that range i contains α and (i', j', c') is the smallest item in L_2 so that range j' contains β. There are four possibilities:

- β lies after the range j and α lies before the range i'. This case happens for $\mathrm{VR}_u(\alpha, \beta)$ in Fig. 11. In such cases, the value of $\mathrm{VR}_u(\alpha, \beta)$ is empty, which means that there is no cut-diagonal (effective or ineffective) for such vision ranges.

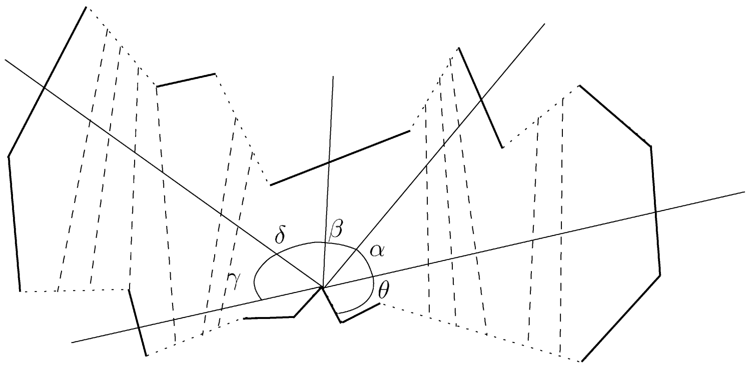


Fig. 11. Computing the values of VR_u from L_1 and L_2.

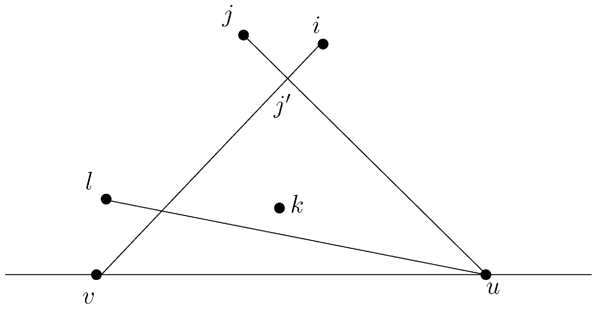


Fig. 12. Finding the entries of $\mathrm{VR}'_{u,v}$.

- β and α lie after the ranges j and i', respectively. This case happens for $\mathrm{VR}_u(\theta, \alpha)$ in Fig. 11. In such cases, c' is the value of $\mathrm{VR}_u(\alpha, \beta)$.
- β and α lie before the ranges j and i', respectively. This case happens for $\mathrm{VR}_u(\delta, \gamma)$ in Fig. 11. In such cases, c is the value of $\mathrm{VR}_u(\alpha, \beta)$.
- β lies before the range j and α lies after the range i'. This case happens for $\mathrm{VR}_u(\beta, \gamma)$ in Fig. 11. In such cases, c is equal to c' and it is the value of $\mathrm{VR}_u(\alpha, \beta)$.

Now we describe how to build $\mathrm{VR}'_{u,v}$ for any pair of reflex vertices. Be reminded that $\mathrm{VR}'_{u,v}$ associates a range α' of RD_u for any range β of RD_v. This association must satisfy a constraint: α' is the smallest range of u whose endpoint lies after the supporting line of the range β and no vertex exists in the vuj' region in which j' is the intersection of the supporting lines of the α' and β ranges. As shown in Fig. 12, for range i of v its associated range of u is j if the region vuj' does not contain any vertex. If a vertex like k exists, the associated range will be l which satisfies the above constraint.

To build $\mathrm{VR}'_{u,v}$, the entries of RD_v are considered according to their counter-clockwise order starting from the line vu. According to Fig. 12, when a range i is considered, the smallest range j in RD_u which contains the vertex i is found. Note that the endpoints of these ranges can be the same (the vertex i is the same as the vertex j). If the region vuj' does not contain any vertex, the range j is associated with the range i and is maintained in $\mathrm{VR}'_{u,v}$. Otherwise, assume that k is a vertex in region vuj' with the smallest $\angle vuk$ angle. In this state, the range l is associated with the range i which satisfies the mentioned constraint. Notice that the vertex l may be the same as the vertex v in some situations. Since we are considering the ranges of RD_v according to their order, we do not need to search for the vertex k and we have already considered it before. Therefore, it is enough to maintain this vertex and update it as we consider the ranges of RD_v accordingly.

4.3. *Efficiency of the improved algorithm*

Cost and benefit of this improvement are as follows:

Lemma 8. *Maintaining a data structure of size* $O(n^3)$ *which can be constructed in* $O(n^3 \log n)$ *time makes it possible to skip ineffective cut-diagonals and find the first effective one in* $O(\log n)$ *time.*

Proof. The size of VR_u for any reflex vertex u is $O(n)$ by the same argument presented for $\mathrm{VR}_{u,v}$. This structure can pessimistically be constructed by using our visibility algorithm in time $O(n^2 \log n)$. Also, RD_u, whose size is $O(n)$, can be built as a byproduct of this process. Computing $\mathrm{VR}'_{u,v}$ for any pair of reflex vertices u and v can be done in $O(n \log n)$ time by a radial sweep on RD_v and a binary search on RD_u ranges. The size of $\mathrm{VR}'_{u,v}$ is also $O(n)$. There are $O(n^2)$ pairs of reflex vertices. Hence, the total time and space required to build and maintain $\mathrm{VR}'_{u,v}$ structures are $O(n^3 \log n)$ and $O(n^3)$ respectively.

To find the first effective cut-diagonal of a query point, $\mathrm{VR}'_{u,v}$ and VR_u data structures are searched each requires $O(\log n)$ time. Therefore, the query time of reporting the first effective cut-diagonal is $O(\log n)$. □

Lemma 9. *The number of effective cut-diagonals that will be processed by the algorithm for a query point* q *is* $O(|V(q)|)$. *Thus,* $h' = O(|V(q)|)$.

Proof. Any effective cut-diagonal adds at least one edge or vertex to the visibility polygon of the query point and no one of these changes are repeated twice. Therefore, an upper bound of the number of these cut-diagonals is the size of the final visibility polygon. □

Lemma 10. *The number of effective cut-diagonals that will be processed by the algorithm for a query point* q *is* $O(h)$. *Thus,* $h' = O(h)$.

Proof. A cut-diagonal can be q-effective from its both ends. If a diagonal like l in Fig. 13, has another q-effective $u'v'$ portion, there must exist two holes, H_1 and H_2, between q and l. Since $u'v'$ is q-effective, there must be some vertices and edges of $\mathcal{P}$ which are visible from q through $u'v'$. Hence, at least one of the L or R chains of Fig. 13 must exist. Without loss of generality, assume that L exists. Assume that a middle portion of another cut-diagonal l' is also q-effective and it is visible from q through $u'v'$. Then, the chain L cannot be a part of the outer boundary of $\mathcal{P}$ and must be a portion of a hole like H_3. If H_3 does not completely contained inside $\angle\alpha$ then H_1 cannot further do its role for another q-effective segment as what it did for $u'v'$ segment from its right side. Therefore, each hole can produce at most two such q-effective segments each from one side. For the case in which H_3 lies completely inside $\angle\alpha$, the new effective cut-diagonals of l' are assigned to H_3 instead of H_1 and H_2.

Thus, an upper bound for the number of the q-effective cut-diagonals which must be processed for a query point q is $O(h)$. □

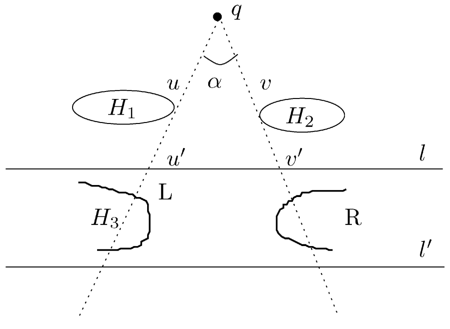


Fig. 13. A hole can produce only two q-effective cut-diagonals.

So, the efficiency of our algorithm can be rephrased as:

Theorem 2. *A polygon $\mathcal{P}$ with a total of n vertices and h holes inside $\mathcal{P}$, can be preprocessed in time $\mathrm{O}(n^3 \log n)$ to build data structures of size $\mathrm{O}(n^3)$, so that the visibility polygon of an arbitrary query point q within $\mathcal{P}$ can be reported in time $\mathrm{O}((1+h')\log n + |V(q)|)$ in which $|V(q)|$ is the size of the output and h' is an output and preprocessing sensitive parameter of size at most* $\min(h, |V(g)|)$.

5. Conclusion

In this paper, we have considered the problem of computing the visibility polygon $V(q)$ of a point q inside a polygon with holes. This problem has been solved efficiently before, but in the nonquery version. Here, we proposed an efficient algorithm for the query version of the problem where we preprocess the polygon and build data structures by which the $V(q)$ of any query point q can be reported rapidly.

We first applied and analyzed the notion of visibility decomposition on this type of polygons. We then presented an algorithm to report $V(q)$ of any q in time $\mathrm{O}((1+h')\log n + |V(q)|)$ by spending $\mathrm{O}(n^3 \log n)$ time to preprocess the polygon and maintaining a data structure of size $\mathrm{O}(n^3)$. The factor h' is an output and preprocess sensitive parameter of size at most $\min(h, |V(g)|)$.

It is interesting to know if this method can be used to solve other similar problems such as finding the visibility polygon of a moving point and a line segment, or the visibility graph of a point set especially in dynamic environments. Another question is whether we can omit the factor h' or reduce it to $\mathrm{O}(\log n)$.

References

[1] T. Asano, Efficient algorithms for finding the visibility polygons for a polygonal region with holes, Manuscript, Department of Electrical Engineering and Computer Science, University of California at Berkeley, 1984.

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

[3] J. Bentley, T. Ottmann, Algorithms for reporting and counting geometric intersections, IEEE Transactions on Computers 28 (1979) 643–647.

[4] B. Chazelle, L. Guibas, Visibility and intersection problems in plane geometry, in: Proc. 1st Annu. ACM Sympos. Comput. Geom., 1985, pp. 135–156.

[5] H. Gindy, D. Avis, A linear algorithm for computing the visibility polygon from a point, Journal of Algorithms 2 (1981) 186–197.

[6] P.J. Heffernan, J.S.B. Mitchell, An optimal algorithm for computing visibility in the plane, SIAM Journal of Computing 24 (1) (1995) 184–201.

[7] D. Kirkpatrick, Optimal search in planar subdivisions, SIAM Journal of Computing 12 (1) (1983) 28–35.

[8] D.L.M.S.L. Guibas, J. Hershberger, R. Tarjan, Linear time algorithms for visibility and shortest path problems inside simple polygons, in: Proc. Second Annual ACM Symp. on Computational Geometry, 1986, pp. 1–13.

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

[10] D. Lee, Visibility of a simple polygon, Computer Vision, Graphics, and Image Processing 22 (1) (1983) 207–221.

[11] D. Lee, F. Preparata, Location of a point in a planar subdivision and its applications, SIAM Journal of Computing 6 (3) (1977) 594–606.

[12] M.O. Mark de Berg, M. van Kreveld, O. Schwarzkopf, Computational Geometry: Algorithms and Applications, Springer, 2000.

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

[14] M. Pocchiola, G. Vegter, The visibility complex, Internal. J. Comput. Geom. Appl. 6 (3) (1996) 279–308.

[15] F. Preparata, A new approach to planar point location, SIAM Journal of Computing 10 (3) (1981) 473–482.

[16] N. Sarnak, R. Tarjan, Planar point location using persistent search trees, Communications of the ACM 29 (7) (1986) 669–679.

[17] S. Suri, J. O'Rourke, Worst-case optimal algorithms for constructing visibility polygons with holes, in: Proc. of the Second Annual Symposium on Computational Geometry, 1986, pp. 14–23.