10th Annual
Conference of Computer Society of Iran
February 15-17, 2005

Some results on computing the visibility of a query point inside polygons with holes

Ali Reza Zarei
zarei@mehr.sharif.edu

Mohammad Ghodsi
ghodsi@sharif.edu

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

Abstract

In this paper, we consider the problem of computing the visibility polygon of a query point inside polygons with holes. The goal is to perform this computation efficiently per query with more cost in the preprocessing phase. Our algorithm is based on solutions in [10] and [11] 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 h^2.

Keywords: *visibility polygon, visibility decomposition, polygon with holes*

1 Introduction

Visibility problems have many applications in motion planning, robotics, computer graphics and geographical information systems. The main problem is to compute that area of a surface that is visible from an object. In this paper we consider the point visibility problem in polygons with holes.

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 query point q has been considered for two decades. For simple polygons, linear time optimal algorithms have been proposed [1, 6, 2, 9]. For polygons with holes, the worst case optimal algorithms with total time of $O(n \log n)$ were presented in [7] and [5]. This was later improved to $O(n + h \log h)$ in [3].

This problem in the query version has been considered by few. The only known solution of this kind 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 are composed of the same sequence of vertices and edges from the underlying polygon. In such a decomposition, the visibility regions are determined in the preprocessing phase and their visibility polygons are computed and maintained in

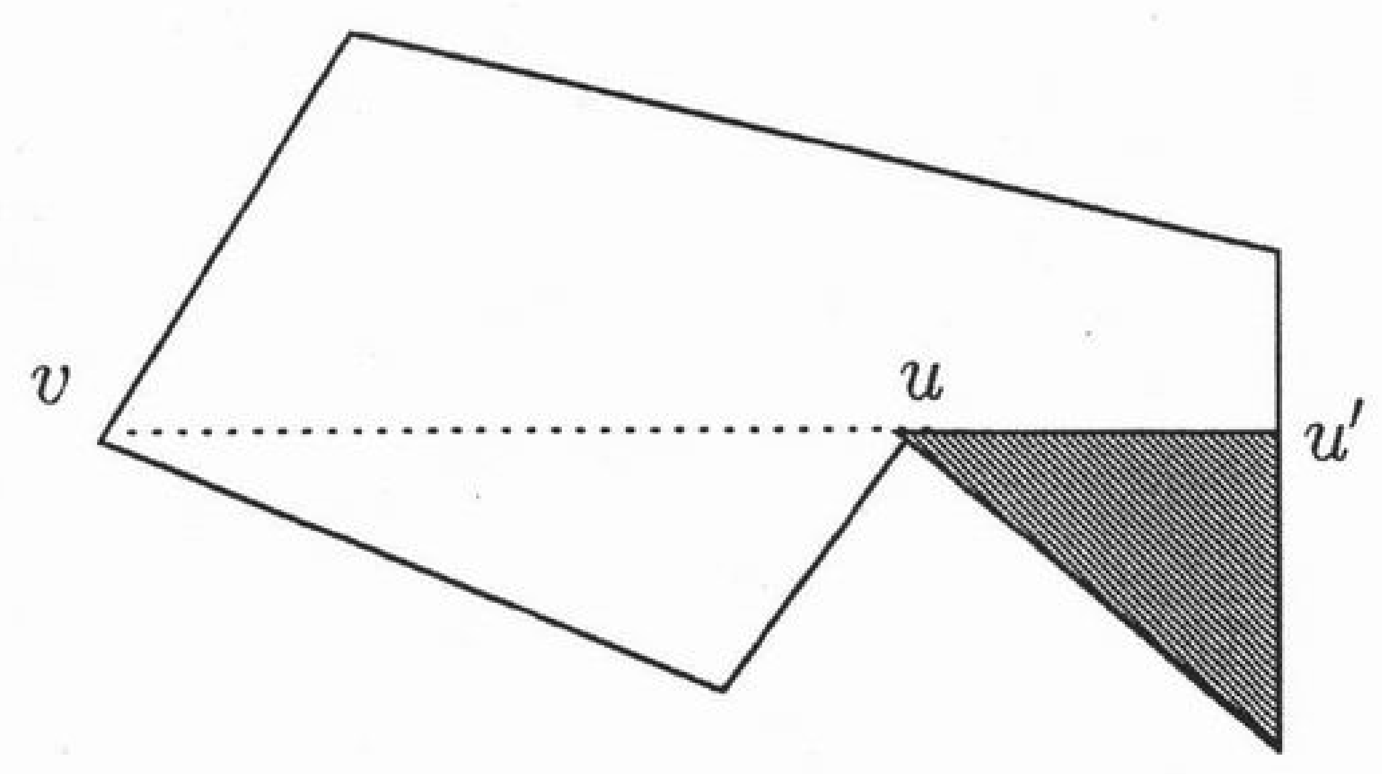

Figure 1: uu' as a window of a polygon.

a proper data structure. For any point q, $V(q)$ can then be obtained by refining the visibility polygon of the region that contains q.

In a simple polygon with n vertices, $V(q)$ can be reported in time $O(\log n + |V(q)|)$ by spending $O(n^3 \log n)$ of preprocessing time and $O(n^3)$ space [8, 10]. Another improvement to this method was done in [11] where the preprocessing time and space were reduced to $O(n^2 \log n)$ and $O(n^2)$ respectively, at expense of more query time of $O(\log^2 n + |V(q)|)$.

However, none of these works are applicable on polygons with holes. In this paper we apply the visibility decomposition method on non-simple polygons and present a method to extend the above algorithms for such polygons. In an overall view, we add some new edges and vertices to the non-simple polygon (called *cut-diagonal*) so that the polygon can be unfolded along these diagonals and converted to a simple polygon. Then, we use an existing algorithm on simple polygons (one of [8] and [10]) to compute a preliminary version of the $V(q)$. With some additional work, we find the final $V(q)$.

For a polygon of total n vertices and h holes, our algorithm needs the preprocessing time of $O(n^3 \log n)$ and memory of size $O(n^3)$. Any query can then be handled in time $O((1+h') \log(n) + |V(q)|)$ in which h' is an output and preprocessing sensitive parameter of at most h^2.

In the rest of this paper and in section two, the visibility decomposition will be applied to non-simple polygons and its time and space complexities will be analyzed. In section three, the new algorithm will be presented. Finally, the materials will be summarized and concluded in section four.

2 Visibility Decomposition

Let $\mathcal{P}$ be a polygon with h holes $H_1, H_2, \cdots, H_h$. Also let q be the query point for which the visibility polygon $V(q)$ is to be computed. A visibility decomposition of $\mathcal{P}$, denoted as 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* and denoted by 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. We then construct a subdivision from these segments. The boundary segments are either the edges of $\mathcal{P}$, or segments that we call *windows* of $\mathcal{P}$. As shown in Figure 1, a window uu' is an extension of the segment between two mutually visible vertices u and v of $\mathcal{P}$. This is because the points below the window uu' are not visible from vertex v, while the upper points are. It is easy to prove that no other kinds of segments are

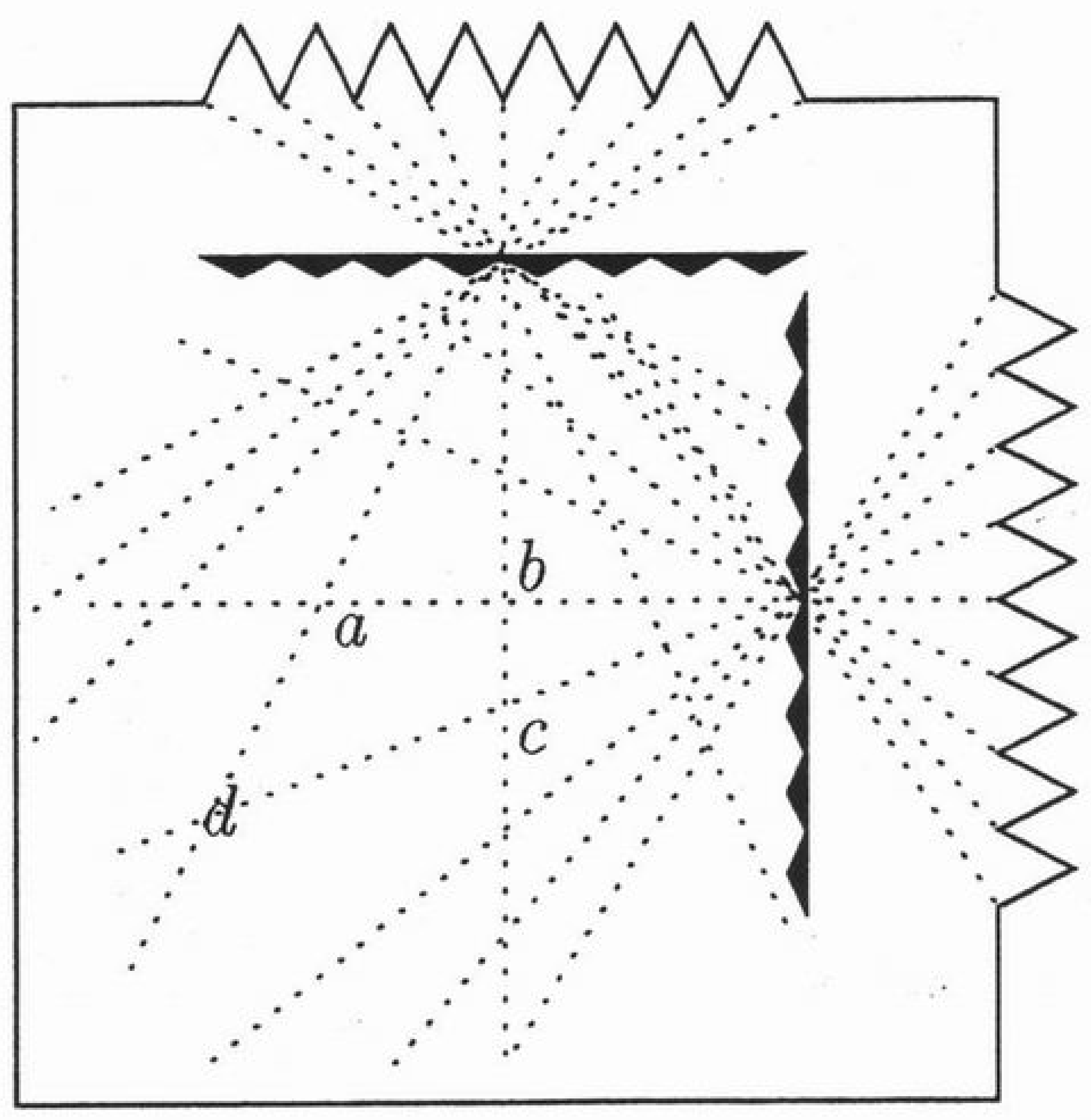

Figure 2: A polygon with $O(n^4)$ v-regions and *sinks*

involved in construction of the v-decomposition. More details on the properties of such a decomposition can be found in [10] and [11].

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

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 Figure 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 compared to all of its adjacent regions. It is sufficient to only maintain the v-sequences of the sinks, from which the v-sequences of all other regions can be computed.

A directed dual graph is built on the v-regions [8, 10]. In a simple polygon, there are $O(n^2)$ sinks. This reduces the space requirement of v-decomposition of simple polygons to $O(n^3)$. Unfortunately, the above property does not hold for non-simple polygons.

Lemma 2 . *The space complexity of maintaining the v-sequences of the regions in a non-simple polygon 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, Figure 2 shows a polygon with $O(n^4)$ sink regions each with v-sequences of size $O(n)$. This is because any one of the v-regions like *abcd* contains at least one sink. □

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 answered as follows: A point location structure will be built over the v-regions. From this, the region $r(q)$ containing q can be found in time $O(\lg n)$. The v-sequence($r(q)$)is then traced and refined to exactly compute $V(q)$. This refinement takes linear time in terms of the size of $V(q)$ [10]. Therefore,

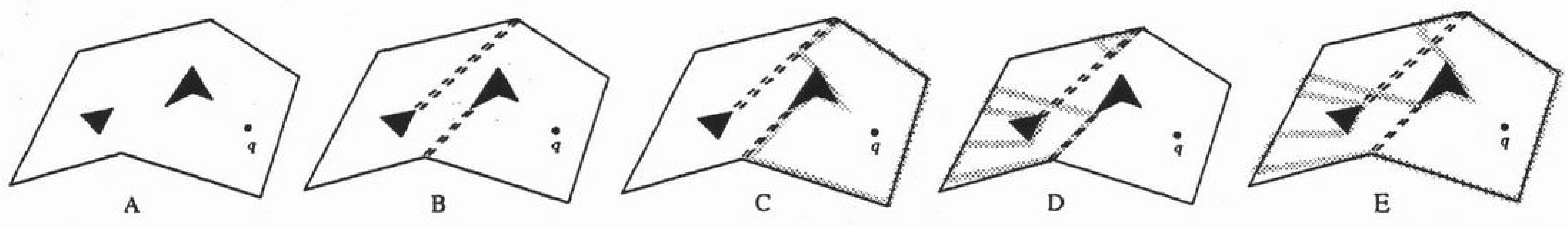

Figure 3: Computing $V(q)$ inside a non-simple polygon: A) The original polygon $\mathcal{P}$, B) The cut-diagonals to produce a simple polygon $\mathcal{P}_s$, C) The visibility polygon $V_s(q)$ targeted to $\mathcal{P}_s$, D) Extra segments of $\mathcal{P}$ viewed from q through the cut-diagonals, 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 $O(\lg n + |V(q)|)$.*

Proof. We first compute the $V(r)$ for each vertex r of $\mathcal{P}$ [5]. All windows can then be found in time $O(n^2 \log n)$. The v-decomposition and its dual graph can then be constructed in time $O(n^4 \log n)$ [12]. The point location structure on the v-decomposition can be constructed in time $O(n^4 \log n)$ [13, 14, 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 anyone 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 $O(n^5)$. $V(q)$ is then found in time $O(\log n + |V(q)|)$ as describe above. $\square$

3 The Proposed Algorithm

Clearly, time and space complexities of the previous algorithm is too high and it is not acceptable in all applications. In this section we propose another algorithm for this problem that needs less preprocessing time and space at expense of higher cost of query time.

The first step of this algorithm is to convert the initial non-simple polygon $\mathcal{P}$ into a simple polygon $\mathcal{P}_s$. This is done by adding some diagonals as "cuts" and an unfold process. $V_s(q)$ in $\mathcal{P}_s$ is then computed using an algorithm in [10, 11], from which the final $V(q)$ in $\mathcal{P}$ is computed. This is done by a SEE-THROUGH procedure to be described below.

Figure 3 depicts an example of the algorithm. The original non-simple $\mathcal{P}$ and its $\mathcal{P}_s$ are shown in parts A and B respectively. $V(q)$ in $\mathcal{P}_s$, denoted by $V_s(q)$ is computed as shown in C. There are segments in $\mathcal{P}$ that are visible from q through the "cut" segments of $V_s(q)$ which are shown in D. These segments are computed (recursively) by the SEE-THROUGH algorithm to replace the cut-segments of $V_s(q)$ which leads to the final $V(q)$, as shown in E.

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 pair of cut-diagonals connecting a vertex of H to a vertex of $\mathcal{P}$ in its outer boundary. Cutting $\mathcal{P}$ along this diagonal 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 hole. This can be enforced if we eliminate leftmost hole first, which is the hole whose leftmost corner has smallest x-coordinate.

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 $O(n)$.

The conversion algorithm described above can be done by first triangulating the polygon and then selection the proper cut-diagonals, which can be done in $O(n\log^* n)$ [17].

3.2 Computing visibility through cut-diagonals

As mentioned before, an important step of our approach is the SEE-THROUGH algorithm to update the $V_s(q)$ on $\mathcal{P}_s$. This step finds new segments of edges of $\mathcal{P}$ that are visible from q through the cut-diagonals. We use the ideas presented in [11].

Theorem 2 . [11] *Given a simple polygon P with size n and a cut-diagonal which cuts P into two parts, L and R, by using $O(n^2\log n)$ time, we could 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 theorem does not hold for non-simple polygons. We however, use its idea for polygons with holes. Assume that $\mathcal{P}$ has only one hole H_i which has been eliminated by one cut-diagonal u_1u_2 as shown in Figure 8. For any query point q, we intend to find the set of segments on edges of $\mathcal{P}$ 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 $\mathcal{P}$. Now suppose that the cut-diagonal u_1u_2 is replaced by v_1v_2. Obviously, cutting $\mathcal{P}$ along v_1v_2 will produce another simple polygon, called P'_{H_i} for which u_1u_2 is a diagonal. We can now use Theorem 2 to preprocess P'_{H_i} and build the appropriate data structures so that for any query point q, we can find the segments of P'_{H_i} that are visible from q through its diagonal u_1u_2. These segments are denoted as $V_{H_i}(q)$. Since, no part of v_1v_2 is visible from q through u_1u_2, $V_{H_i}(q)$ is the same as set of segments we are looking for. Let us denote the above algorithm by SEE-THROUGH(H_i).

This algorithm can be extended to more holes. This is done 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, we can find the extra segments of P visible from q through all the cut-diagonals, to be described in next subsection.

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 subsection 3.1.

b Preprocess $\mathcal{P}_s$ and create the data structure so that $V(q)$ of any arbitrary query q in $\mathcal{P}_s$ can efficiently be reported. Theorem 3 is used for this step.

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.

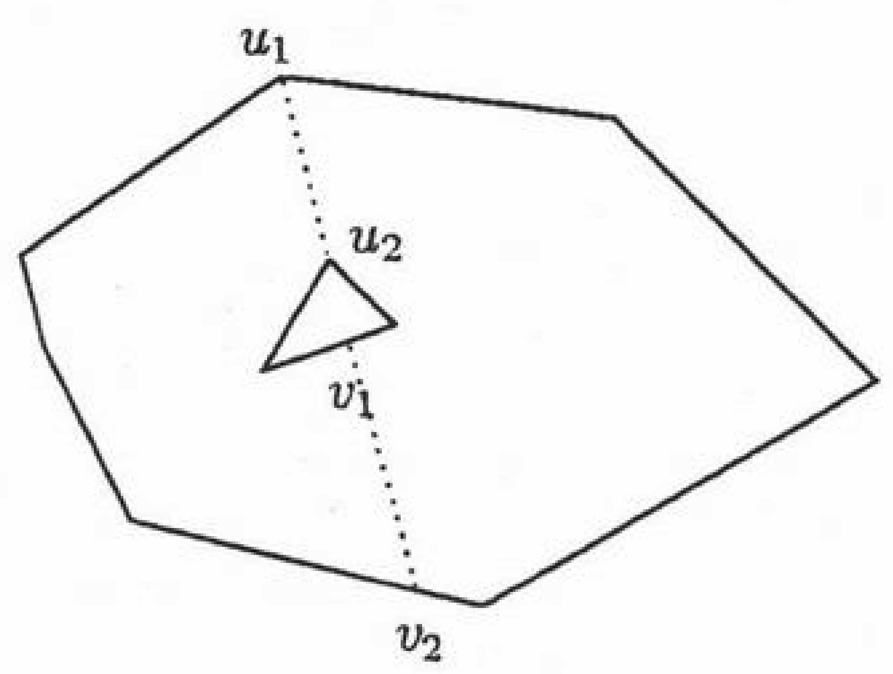

Figure 4: Replacing u_1u_2 with v_1v_2.

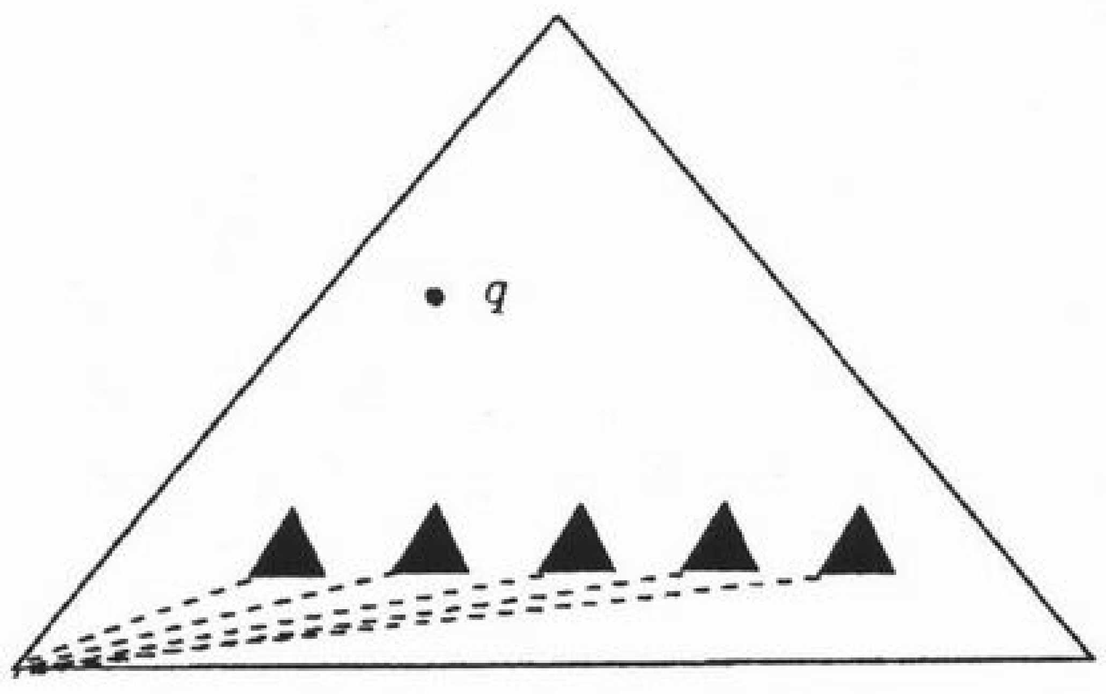

Figure 5: A polygon with tight bound of h'.

Theorem 3 . [10] *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 $V(q)$.*

The $V(q)$ of any q is computed as follows. The data structure build 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 of the cut-diagonal uv of a hole H_i is a part of $V_s(q)$. The preprocessing of step (c) is then used to find $V_{H_i}(q)$, the extra segments viewed through uv. The segment uv in $V_s(q)$ is then replaced by $V_{H_i}(q)$. This is continued for any such segments in $V_s(q)$. This process will finish without any loop, due to the nature of visibility. What remains at the end is $V(q)$, and this is easy to prove.

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 resulted polygon $\mathcal{P}_s$, like $\mathcal{P}$, has $O(n)$ vertices. The time and space complexities of step (b) are as in Theorem 3.

As described in subsection 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 $\mathcal{P}$. Thus, the total preprocessing time to construct cut-diagonal data structures is $O(n^3 \log n)$ and they 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 appeared 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 $\mathcal{P}_s$. For any one of the h' cut-diagonals, appeared in $V_s(q)$, a point location of size $O(\log n)$ is required to run the SEE-THROUGH algorithm. The number of the edges that appear in $V_s(q)$ is the size of the final visibility polygon $V(q)$ plus the number of the cut-diagonals appeared in $V_s(q)$. □

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 $O(h^2)$. Figure 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 happened rarely. However, h is $O(n)$ and therefore, the upper bound of h' is $O(n^2)$. We have improved this algorithm so that the size of h' is reduced to at most $\min(h, |V(q)|)$. This will be presented in our next paper.

4 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 non-query version. Here, we proposed an efficient algorithm for the query version of the problem where we preprocess the polygon to build a data structure by which the $V(q)$ of any query point q could be reported rapidly.

We first applied and analyzed the notion of visibility decomposition to this type of polygons. We then presented an algorithm to report $V(q)$ of any q in time $O((1 + h')\log n + |V(q)|)$ by spending $O(n^3 \log n)$ time to preprocess the polygon and maintaining a data structure of size $O(n^3)$. The factor h' is an output and preprocess sensitive parameter of size at most h^2.

We have improved this algorithm so that the size of h' is reduced to at most $\min(h, |V(q)|)$. This will be presented in our next paper.

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 $O(\lg h')$.

References

[1] H.El Gindy and D. Avis, *A Linear Algorithm for Computing the Visibility Polygon from a Point*, Journal of Algorithms, 2, pp. 186-197, 1981.

[2] L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. Tarjan, *Linear Time Algorithms for Visibility and Shortest Path Problems inside Simple Polygons*, Proc. Second Annual ACM Symp. on Computational Geometry, 1986, pp. 1-13.

[3] F. Dehne, J.-R. Sack, N. Santoro, *An Optimal Algorithm for Computing Visibility in the Plane*, SIAM Journal on Computing, Vol. 24, No. 1, pp. 184–201, February, 1995.

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

[5] S. Suri and J. O'Rourke, *Worst-Case Optimal Algorithms for Constructing Visibility Polygons with Holes*, In Proc. of the second annual symposium on Computational geometry, 1986, pp. 14-23.

[6] D.T. Lee, *Visibility of a Simple Polygon*, Computer vision, Graphics, and Image Processing 22, 1983, pp. 207-221.

[7] 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.

[8] L. Guibas, R. Motwani, and P. Raghavan, *The Robot Localization Problem in two Dimensions*, SIAM J. of Computing 26, 4, 1997, pp. 1120-1138.

[9] B. Chazelle and L. Guibas, *Visibility and Intersection Problems in Plane Geometry*, In Proc. 1th Annu. ACM Sympos. Comput. Geom., 1985, pp. 135-156.

[10] P. Bose, A. Lubiw, and J.I. Munro,, *Efficient Visibility Queries in Simple Polygons*, In Proc. 4th Canad. Conf. Comput. Geom., 1992, pp. 23-28.

[11] 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.

[12] J.L. Bentley and Th. Ottmann, *Algorithms for Reporting and Counting Geometric Intersections*, IEEE Transactions on Computers, 28, pp. 643-647, 1979.

[13] D. Kirkpatrick, *Optimal Search in Planar Subdivisions*, SIAM Journal of Computing, 12, 1, pp. 28-35, 1983.

[14] D.T. Lee and F.P. Preparata, *Location of a Point in a Planar Subdivision and its Applications*, SIAM Journal of Computing, 6, 3, pp. 594-606, 1977.

[15] F.P. Preparata, *A New Approach to Planar Point Location*, SIAM Journal of Computing, 10, 3, pp. 473-482, 1981.

[16] N. Sarnak and R. Tarjan, *Planar Point Location Using Persistent Search Trees*, Communications of the ACM, 29, 7, pp. 669-679, 1986.

[17] R. Seidel, *A Simple and Fast Incremental Randomized Algorithm for Computing Trapezoidal Decomposition and for Triangulating Polygons*, Comput. Geom. Theory Appl. 1:51-64, 1991.