Efficient Visibility Queries in Simple Polygons*

Prosenjit Bose† jit@muff.cs.mcgill.ca
Anna Lubiw‡ alubiw@maytag.waterloo.edu
J. Ian Munro‡ imunro@dragon.waterloo.edu

Abstract

We present a new method of decomposing a simple polygon which allows the preprocessing of the polygon to efficiently answer queries of various forms. Using $O(n^3 \log n)$ preprocessing time and $O(n^3)$ space, we can, given a query point q inside or outside the polygon, recover the number of vertices visible from q in $O(\log n)$ time. Also, we can recover the visibility polygon of q in $O(\log n + k)$ time, where k is the size of the visibility polygon.

The key notion behind the decomposition is the succinct representation of *visibility regions*. We show that there can be $\Theta(n^3)$ distinct visibility regions, each of which can see $\Theta(n)$ points. The main technical theorem is that there can be $\Theta(n^2)$ regions with *minimal visibility sets*, thus permitting preprocessing in $O(n^3 \log n)$ time and $O(n^3)$ space. The notion of visibility regions and their use in visibility polygon queries has been developed independently by Guibas, Motwani and Raghavan[3].

These techniques are extended to handle other types of queries. Given m fixed points in the plane, and a polygon of size n, we can, given a query point q inside or outside the polygon, recover the number of fixed points unobstructed by the polygon boundary in $O(\log mn)$ time and recover the visible set of size k in $O(\log mn + k)$ time, with $O(m^2(m+n)\log n)$ preprocessing time and $O(m^2(m+n))$ space.

We also explore line segment queries. Given a line segment inside or outside a polygon of size n, we can recover its weak visibility set and more generally its weak visibility sequence in $O(k \log n)$ time, using $O(n^3 \log n)$ preprocessing time and $O(n^3)$ space. To answer weak visibility line segment queries on a set of m fixed points in the plane, we need $O(k \log mn)$ query time, given $O(m^2(m+n)\log n)$ preprocessing time and $O(m^2(m+n))$ space.

1 Introduction

The notion of visibility among geometric objects is one of the most fundamental topics in computational geometry. Many different visibility problems have been studied in the literature. O'Rourke's book [6] surveys many visibility problems that have been investigated.

In this paper, we consider several questions concerning visibility, all stemming from the following question: Given a query point q in the interior of a simple polygon, P, find all the *vertices* of P that *see* q. The single- shot version of this problem is a special case of the visibility polygon problem and it can be solved in $O(n)$ time where n is the number of vertices in the polygon. In this paper, we develop a data structure that can be used to answer repeated queries efficiently. After the data structure has been built, the query algorithm can find the number of vertices of P which see q in $O(\log n)$ time, where n is the number of vertices of P. Listing the vertices takes $O(\log n + k)$ time, where k is the number of vertices.

The succinctness of the data structure is based on some properties of the polygon decomposition into *visibility regions*. Several key properties of this polygonal decomposition method allow us to solve extensions of the original problem, as well as some different problems, quite efficiently.

We uncover some properties of the decomposition which allow us to recover the whole visibility polygon of the query point as opposed to just the visible vertices. The visibility polygon represents all the points in the polygon that see the query point. Informally, we can think of the query point q as a light source in a dark art gallery, represented by P, and we want to know what area of the gallery is lit by q. The notion of visibility regions and how to use them to handle visibility polygon queries was discovered independently by Guibas et al. [3]. Although they use the visibility decomposition to solve a problem known as the "Robot Localization Problem", where the reverse question is asked: Given a point and its visibility polygon with respect to polygon P, determine where the point is in P, they also show how to solve the visibility polygon recovery problem.

We generalize from visibility of vertices to a set of fixed points of interest. Given a fixed set of points, S, in the plane, a polygon P, and a query point q, we are interested in the subset of S seen from q, unobstructed by the boundary of P. Informally, we can think of the points of S as light sources in an art gallery, represented by P, and we wish to know which lights are shining on an observer, q, in the gallery.

We are able to extend the structure to deal with query points either inside or outside a simple polygon P, and solve all of the above situations. An alternative approach using persistent data structures and visibility regions is explored.

The visibility decomposition structure reveals much geometric information about the polygon. We study some applications of the visibility decomposition with respect to line segment queries. In particular, we look at weak

*Research supported in part by NSERC and ITRC

†School of Computer Science, McGill University, Montreal, Quebec, Canada

‡Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada

visibility queries with respect to line segments as opposed to strong visibility. A point is weakly visible from a line segment if it sees any part of the line segment, and it is strongly visible from the line segment, if it sees the whole line segment.

2 Overview

We will now give an intuitive notion of the idea used to preprocess the polygon for efficiently answering visibility queries. A *visibility region*, R, in a simple polygon P, is a *maximally connected* subset of P with the property that any two points in R see the same subset of vertices of P. The *visibility set* of a visibility region, R, is the subset of vertices of the polygon P which can be seen from R. Two visibility regions are *neighboring* visibility regions if they share a common edge. A visibility region with *minimal visibility set* is one whose neighboring visibility regions have visibility sets which strictly contain its visibility set. Such regions will be referred to as *sinks* and the analogy to the standard definition of sinks in digraphs will soon become clear.

The notion of visibility regions offers a solution to the initial query problem. Since any two points inside a visibility region sees the same set of vertices, we simply need to decompose the polygon into visibility regions and determine which vertices are visible from each of the visibility regions. Once the polygon has been decomposed, to answer a query, we need to determine in which visibility region the query point lies.

In order to give a compact representation of the decomposition of a polygon into visibility regions, we take advantage of several key properties of the decomposition. First, using a counting argument, we show that a simple polygon P on n vertices has $O(n^3)$ visibility regions. This immediately gives a rough upper bound on the solution to the query problem.

We then consider the dual graph of the planar map of visibility regions. We show that the visibility set of any two neighboring visibility regions differs by only one vertex; we direct the corresponding edge of the dual graph towards the visibility region with the smaller visibility set and label the edge by the lost vertex. We obtain a directed acyclic graph whose sinks correspond exactly to visibility regions with minimal visibility set. From any node representing a visibility region r, we can find a directed path π to a sink s; the visibility set of r is the visibility set of s plus the set of vertices labelling the edges of π.

Through the use of a technical counting argument, we are able to establish that there are $O(n^2)$ sinks or regions with a minimal visibility set. By associating with each such region the subset of vertices it sees, we are able to store the information necessary for answering the queries efficiently, in $O(n^3)$ space. See Figure 1.

During the preprocessing phase, the polygon is decomposed into visibility regions. The regions are then preprocessed for planar point location (Kirkpatrick [4], Lee and Preparata [5], Preparata [7], and Sarnak and Tarjan [9]). The directed dual planar graph of the planar subdivision is built, where only the subset of vertices of P seen by each sink is stored. At query time, a standard $O(\log n)$ planar point location query reveals which region contains the query point. Once the region is located, we enumerate the vertices by following a path from the node to a sink. The time required for preprocessing is $O(n^3 \log n)$ since the cost is dominated by the time required to preprocess for planar point location. The space requirement is $O(n^3)$, which is optimal with respect to the decomposition.

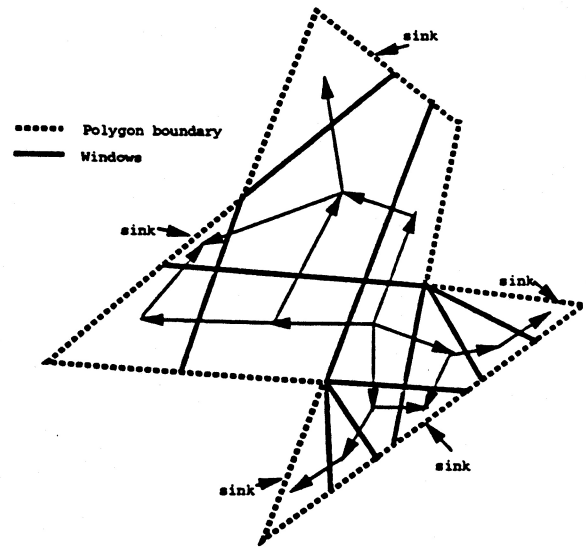

Figure 1: Polygon decomposed into visibility regions with directed dual graph.

All extensions forged from this structure follow naturally from the initial idea. We omit all proofs in this extended abstract. For full proofs of the theorems and more details on the techniques used, the reader may refer to the technical report[2].

3 Notation and Preliminaries

Most of the geometric and graph theoretic terminology used is standard and for details, we refer the reader to O'Rourke [6], Bondy and Murty [1], Preparata and Shamos [8]. For simplicity of presentation, we assume that no three vertices of the polygon are collinear. We will begin by reviewing some of the terminology used in this paper.

A *simple polygon* P is a simply connected subset of the plane whose boundary is a closed chain of line segments. As we are dealing only with simple polygons, we will refer to them as polygons in the remainder of the paper. We will denote a polygon, P, by a set of vertices $v_1, v_2, \ldots, v_{n-1}, v_n$ such that each pair of consecutive vertices is joined by an edge, including the pair $\{v_n, v_1\}$. We assume that the points are in clockwise order, so that the interior of the polygon lies to the right as the boundary of the polygon is traversed. We will denote the open interior of the polygon

P by $INT(P)$, the boundary by $BND(P)$, and the open exterior by $EXT(P)$. The boundary is considered part of the polygon; that is, $P = INT(P) \cup BND(P)$.

We say a point p is *in* P when $p \in INT(P) \cup BND(P)$. Two points in a polygon *see* each other if the line segment between them does not intersect the exterior of the polygon. There is a parallel notion of *exterior visibility*. When considering exterior visibility, we say a point p is in the *exterior* of P to mean $p \in EXT(P) \cup BND(P)$. Two points in the *exterior* of a polygon *see* each other if the line segment between them does not intersect the open interior of the polygon.

Let x be a point in polygon P. The *visibility polygon* from x, denoted by $VP(x, P)$, is the set of points in P visible from x; it is formally defined by

$$VP(x, P) = \{z \mid z \in P \text{ and } \overline{xz} \cap P = \overline{xz}.\}$$

An edge of $VP(x, P)$ which is not contained in an edge of P is called a *window* of point x. Of the two vertices of a window, the one closest to x is called a *base* and the other is called an *end*. Note that the base of a window is a vertex of the polygon P, but the end may not be. See Figure 2. A window is denoted by $Wind(base, end)$. A pocket is called a *left pocket* if, in the neighborhood of its window, the pocket lies to the left (counterclockwise) of the ray from x containing the window, and the visibility polygon to the right (clockwise). A similar definition holds for *right pockets*. A vertex v of P is called a *pocket vertex* with respect to $VP(x, P)$ if v is on a pocket. We extend this notion to *right pocket vertices* and *left pocket vertices* in the obvious way. A window bounding a left pocket or right pocket is called a *left window* or a *right window*, respectively. See Figure 2.

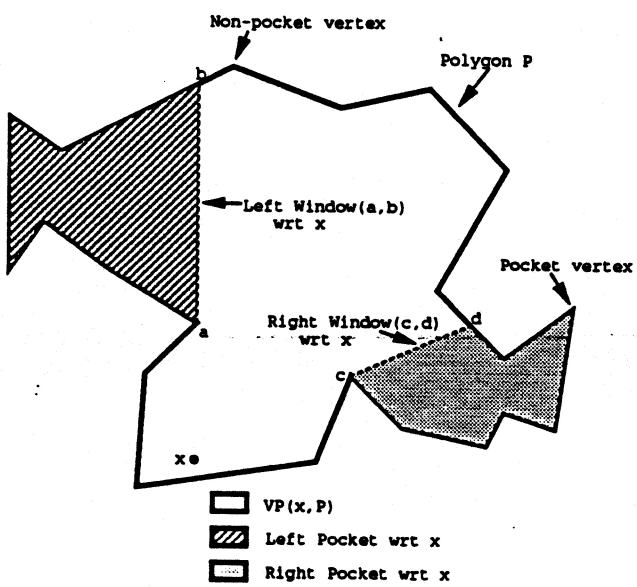

Figure 2: Windows and Pockets

4 Visibility Decomposition and its Properties

We begin by studying the properties of windows in polygons. We first note that the base of any window is a reflex vertex of the polygon, and that each reflex vertex can be the base of at most one window with respect to a given point x.

We now describe how to obtain a decomposition of a polygon into visibility regions. Given a polygon P, let W_i be the set of all windows in P with respect to vertex v_i. Let

$$W = BND(P) \cup \bigcup_{i=1}^{n} W_i$$

Consider the planar subdivision induced by all of the line segments in W. We can show that the set of non-external faces in a subdivision induced by all of the line segments in the set W is precisely the set of visibility regions. This implies that two visibility regions sharing a common edge have visibility sets differing by 1 vertex, and that a visibility region in a simple polygon is a convex region.

A polygon can have $O(n)$ reflex vertices; thus, the set W contains $O(n^2)$ windows, but a planar subdivision induced by n^2 arbitrary line segments in the plane can have $O(n^4)$ faces. The following two lemmas reveal the restrictions imposed by the notion of visibility in polygons. These restrictions are crucial in showing that there are only $O(n^3)$ faces in the planar subdivision of visibility regions induced by n^2 line segments (the windows).

Lemma 4.1. *Given a line segment $\overline{ab}$ inside polygon P, if a point z in P sees a and b then z sees any point on the line segment $\overline{ab}$.*

Lemma 4.2. *Given an arbitrary line segment inside a polygon P, and a point x inside P, at most two windows of x intersect the line segment.*

This leads to the following theorem:

Theorem 4.1. *There are $O(n^3)$ visibility regions in a polygon P on n vertices and there are polygons on n vertices that have $\Theta(n^3)$ visibility regions.*

The succinct representation of the information contained in the planar subdivision of visibility regions relies on a bound of $O(n^2)$ regions of minimal visibility or sinks. We are able to achieve this by demonstrating more crucial restrictions imposed on the regions by the notion of visibility.

The following lemma summarizes the essential properties that allow us to show that there are only $O(n^2)$ sinks.

Lemma 4.3. *Let R_x be the set of right windows of point x and R_y be the right windows of point y in polygon P. Then, there is at most 1 intersection between the line segments of R_x and the line segments of R_y.*

The above lemma lies at the heart of the following theorem:

Theorem 4.2. *A decomposition of a polygon P on n vertices into visibility regions contains $O(n^2)$ sinks and there exist polygons on n vertices which have $\Theta(n^2)$ sinks.*

5 The Visibility Query Algorithm

In this section, we give a high level description of the preprocessing step as described in the overview of Section 1.

1. Construct the set W (as defined in section 4).
2. Construct the planar subdivision induced by this set.
3. Preprocess the subdivision for planar point location.
4. Construct the dual planar graph of the subdivision.
5. Construct the special dual directed planar graph from the dual planar graph.
6. Associate, with each sink in the special dual directed planar graph, the subset of vertices of P seen.
7. Associate, with each node, the number of vertices seen.

The time complexity is dominated by step 2,3. The total time complexity of the preprocessing step is $O(n^3 \log n)$. The space required is $O(n^3)$.

Given an arbitrary query point, the algorithm does as follows:

1. locates the region containing the point;
2. starting at the node representing the given region, follows any path leading to a sink.

If we wish to know only the number of vertices seen then only step 1 is necessary as this information is stored in each region. Thus, the time requirement is $O(\log n)$. Now suppose that the length of the path to the sink is s and that there are t vertices in the list associated with the sink. Each arc represents one vertex seen by the query point and the list of vertices of the sink is seen by the query point. Let $k = s + t$. The time required for a query that enumerates the vertices seen is $O(\log n + k)$. The correctness of this approach follows from the discussion in the previous section. Hence, we have

Theorem 5.1. *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 inside the polygon, it takes $O(\log n + k)$ time to list the k visible vertices and $O(\log n)$ time to give the number of visible vertices.*

6 Recovering the Visibility Polygon

Until this point, we have only concentrated on the set of vertices seen by a query point. When answering a query, we have ignored the order of the vertices as they appear in the polygon. The cyclic order of the original polygon vertices is the same as the clockwise ordering as seen from a query point, thus, recovering the order of the vertices as seen from the query point is an essential step towards recovering the visibility polygon of the query point. To capture this notion of ordering, we need to examine closely some properties of a path in the dual graph from a region that is a sink to a region that is not. These properties will allow us to recover the order of the vertices seen by the query point.

By following a path from the sink to the region containing the query point, we can recover the ordering by following these rules:

1. If a left window is crossed, insert the vertex, in the label of the window, before the base.
2. If a right window is crossed, insert the vertex, in the label of the window, after the base.

The order of the vertices is preserved in the sink, so these insertions return the order of the vertices as seen by the query point. Although the algorithm is simple, some care must be taken in its implementation in order to remain within the desired query time complexity.

To recover the visibility polygon of a given query point, we must extend this notion of ordering to include certain key edges. We define the *visibility sequence* of a point x in a polygon P, written as $VS(x, P)$, to be the clockwise ordering of the edges and vertices of $BND(VP(x, P)) \cap BND(P)$. Note that an edge in this context refers to the label of the edge in P and not the exact piece of it that is seen from the point x. By generalizing the above algorithm to account for the key edges encountered in the visibility sequence, we have:

Theorem 6.1. *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 inside the polygon, $O(\log n + k)$ time is used to recover the visibility sequence of size k.*

Given the visibility sequence of a query point, we can construct its visibility polygon in linear time in the size of the sequence. Thus, the recovery of the visibility polygon of a query point is within the desired time and space complexity.

Theorem 6.2. *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 inside the polygon, $O(\log n + k)$ time is required to recover the visibility polygon of size k.*

7 Query points outside the polygon

In this section, we refine the solution to handle query points in the *exterior* as well as the interior of the polyon. To avoid the ambiguity between exterior and interior visibility, a point on the boundary of the polygon will be considered to be in the interior. If a given query point is inside the polygon, we answer the query using the solution already developed. We now turn our attention to query points which are strictly outside a polygon and satisfy the queries with respect to exterior visibility.

We define the *convex hull* of a polygon P to be the smallest convex polygon, say $CH(P)$, containing P. A vertex on $CH(P)$ is known as a *convex hull vertex*. If an edge of $CH(P)$ is not an edge of P, we say it is a *bay edge*. If $\overline{v_a v_b}$ is a bay edge, then the corresponding *bay* is the polygon formed of the vertices $(v_a, v_b, v_{b-1}, v_{b-2}, \ldots, v_{a+2}, v_{a+1}, v_a)$. A vertex on a bay, but not on the convex hull, is known as a *bay vertex*.

The general idea is to divide the exterior points into

two classes: exterior points inside bays and exterior points outside the convex hull of the polygon. Since a bay is a polygon, it can be preprocessed using the techniques developed; this takes care of query points inside bays. The following results show that the time and space complexities necessary to preprocess the bays are $O(n^3 \log n)$ and $O(n^3)$, respectively.

Observation 7.1. *A bay vertex can be in only one bay.*

Observation 7.2. *A convex hull vertex can be in at most two bays.*

Lemma 7.1. *The time required to preprocess all of the bays is $O(n^3 \log n)$ and the space required is $O(n^3)$.*

Therefore, by preprocessing each bay using the techniques developed so far, we are still within the desired overall time and space complexity. Turning our attention to the structure outside the convex hull, we obtain the following results:

Lemma 7.2. *A convex-hull vertex of a polygon P on n vertices can have at most two of its windows intersecting the outside of the convex hull.*

Lemma 7.3. *A bay vertex of a polygon P on n vertices can have at most two of its windows intersecting the outside of the convex hull.*

Since a polygon vertex can only be a bay vertex or a convex hull vertex, these two lemmas imply that there are at most $2n$ windows outside the convex hull. This means that the planar subdivision induced by these windows has only $O(n^2)$ regions. We do not even need to worry about sinks. By simply placing the necessary information in each visibility region outside the convex hull, we remain within the desired time complexity. Thus, we have:

Theorem 7.1. *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 inside or outside the polygon, $O(\log n + k)$ time is required to list the k visible vertices or recover the visibility polygon of size k and $O(\log n)$ time to give the number of visible vertices.*

8 Visibility of Fixed Points

In this section, we develop a technique to deal with visibility queries concerning a fixed set of points. The techniques used previously decompose the polygon with respect to polygon vertex visibility. We will show how to generalize this to decompose a polygon with respect to a set of fixed points in the plane rather than the polygon vertices. We will concentrate only on a fixed set of points contained within the polygon, since the extension to fixed points both inside and outside the polygon follows from the discussion in section 7. Intuitively, we can think of the set of fixed points as a set of point light sources. In the query, we are simply asking for the light sources which are shining light on the query point. For simplicity of presentation, we will assume that no three points are collinear among the fixed points and polygon vertices.

We must generalize the notion of a visibility region to the following: Given a set S of fixed points contained in a simple polygon P, a *visibility region* R, in P is a *maximally connected* subset of P with the property that any two points in R see the same subset of fixed points from S. All other terms are defined similarly.

Let $\|S\| = m$. Let the points in S be labelled $s_1 \ldots s_m$. Analogous to the definition of the set W in section 4, we define a set W'. Let W'_i be the set of all windows in P with respect to point s_i. Let

$$W' = BND(P) \cup \bigcup_{i=1}^{n} W'_i$$

Consider the planar subdivision induced by all of the line segments in W'. We are able to generalize the lemmas presented in Section 4; thus we have:

Theorem 8.1. *With respect to a set of fixed points $\|S\| = m$, a decomposition of a polygon P on n vertices into visibility regions contains $O(m^2 n)$ regions, and there exist some polygons which have $\Theta(m^2 n)$ visibility regions.*

Theorem 8.2. *With respect to a set of fixed points $\|S\| = m$, a decomposition of a polygon P on n vertices into visibility regions contains $O(m(m+n))$ sinks.*

Theorem 8.3. *Let $\|S\| = m$ be a set fixed points in the plane and P be a simple polygon with n vertices. Given an arbitrary query point inside or outside the polygon, we can recover the number of visible fixed points in $O(\log mn)$ time and recover the visible set of k fixed points in $O(\log n + k)$ time, with $O(m^2(m+n)\log n)$ preprocessing time and $O(m^2(m+n))$ space.*

9 Line Segment Queries

Now, we revert to visibility queries concerning polygon vertices for simplicity of presentation, but instead of query points, we have line segment queries. When dealing with line segments, there exist two notions of visibility. We might want to know the set of polygon vertices which see the whole line segment, known as *strong visibility*, or we might want to know the set of polygon vertices which see any part of the line segment, known as *weak visibility*. We consider the problem of weak visibility; strong visibility of line segments seems to be a more difficult problem.

To recover the weak visibility set, we determine the set of vertices visible to one end point. Let us call this set WVS. The line segment will intersect a set of regions in the decomposition. We follow the path formed by regions which are intersected by the line segment. If a window is crossed and visibility is gained, then the vertex is added to the WVS. If a window is crossed and visibility is lost, then nothing is done. When the region containing the other end point is reached, the set WVS represents the set of vertices weakly visible from the line segment.

The only difficulty is finding the path formed by the line segment. Since each visibility region is convex, the

problem reduces to the following: Given a point inside a convex region and a direction, find which edge of the region is intersected. This intersection is easily found by performing a binary search on the ordered set of edges of the convex region. Thus, we have

Theorem 9.1. *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 line segment inside the polygon, $O(k \log n)$ time is required to recover k weakly visible vertices.*

The extension to visibility of a fixed set of points follows from discussion in Section 8. The extension to line segment queries outside the polygon follows from the discussion in Section 7.

10 An Alternative Approach Using Persistence

We present a method using persistent data structures that gives an alternative solution to all of the problems we have been considering so far. The solutions are similar except that the persistent data structure replaces the special directed dual planar graph. This removes some of the geometric intuition gained from the knowledge of sink regions.

By Lemma 4.2, we know that each window in the planar subdivision has at most $O(n)$ intersections with other windows, where n is the number of vertices in the polygon P. Let us focus our attention on a single window, say W, in this planar subdivision. Between the two endpoints of a window are the intersection points with other windows. Let $p_1 \ldots p_r$ be the consecutive ordering of these points of interest with p_1 being one endpoint and p_r the other. Aside from the endpoints, all other points represent an intersection between the window W and some other window. A window forms a boundary between a region seen by a vertex and a region not seen by a vertex. From this, we deduce that each $\overline{p_i p_{i+1}}$ where $1 < i < r-1$ forms the boundary of a different visibility region, each adjacent to the next.

Let L be the set of vertices seen from p_1 on window W. Since the point p_2 represents an intersection with another window, the set of vertices seen from any point between p_2 and p_3 is L with the addition or deletion of one vertex. Thus, each intersection point identifies an update to the list L. We will sketch the use of a persistent data structure.

If we consider updates to a data structure as steps in time, then a persistent data structure allows access to any version of the data structure at any time. A structure is partially persistent if one can only update the structure in the present (queries are allowed in the past and present). A structure is fully persistent if we can update the structure both in the past and the present. In this situation, we need only a partially persistent data structure to handle queries and updates of insertions, deletions. In particular, Sarnak and Tarjan [9] develop persistent search trees, which have the property that $O(n)$ updates to an empty persistent search tree can be recorded in $O(n \log n)$ time and $O(n)$ space. Therefore, the information for each window can be stored in $O(n \log n)$ time and $O(n)$ space. There are $O(n^2)$ windows which implies $O(n^3)$ total storage and $O(n^3 \log n)$ time.

In order to recover the vertices seen by the query point, a planar point location query will identify the region containing the point. Given the region, choose one of the windows forming the region boundary. To recover the visible vertices, we perform an inorder traversal of the tree associated with that window for that time represented by a point of the region on the window. The extension to visibility sequence queries, visibility polygon queries, fixed point queries, and visibility outside a polygon follows naturally from the discussions developing the solutions using the dual planar graph.

11 Conclusion

There are several questions which are generated from this investigation:

1. Can a non-trivial lower bound be proved for this problem?
2. Is there a smooth trade-off between query time and preprocessing time?
3. Can we recover the strong visibility set of a line segment query in a $O(k \log n)$ time using the same amount of preprocessing as in the weak visibility case?

References

[1] J.A. Bondy and U.S.R. Murty. *Graph Theory with Applications.* Elsevier Science, New York, New York, 1976.

[2] P. Bose. *Visibility in Simple Polygons.* University of Waterloo technical report, 1991.

[3] L. Guibas, R. Motwani, and P. Raghavan. *The Robot Localization Problem in Two Dimensions.* Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 259-268, 1992.

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

[5] 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.

[6] J. O'Rourke. *Art Gallery Theorems and Algorithms.* Oxford University Press, New York, New York, 1987.

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

[8] F. Preparata and M. Shamos . *Computational Geometry, An Introduction* . Springer-Verlag, New York, New York, 1985.

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