Tight Bounds for Edge Guards in Monotone Polygons and Rectilinear Monotone Polygons

Iliana Bjorling-Sachs and Diane L. Souvaine*
Department of Computer Science, Rutgers University, New Brunswick, New Jersey 08903
email: bjorling@paul.rutgers.edu, dls@cs.rutgers.edu

Extended Abstract

1 Introduction

The original art gallery problem asks "What is the minimum number of guards needed to watch every point inside an art gallery". The art gallery here is viewed as a simple polygon and the guards as points inside the polygon. A point x is visible to a guard y as long as the line segment $\overline{xy}$ does not intersect the exterior of the polygon. The polygon is covered if every point inside it is seen by at least one guard. Chvatal showed that no polygon with n vertices requires more than $\lfloor n/3 \rfloor$ guards and that polygons exist for which this number of guards is necessary [2]. Much attention has subsequently been devoted to variations of the original art gallery problem and many results have been obtained [4],[6]. One variation which has remained unsolved is one where guards are allowed the freedom to move along individual edges of the polygon. Guards of this type are called *edge guards*. For edge guards the notion of *weak visibility* is used. A point inside the polygon is considered visible as long as it is seen by some guard at some point during the walk along the edge. Here we show that when the gallery is restricted to the shape of a monotone polygon $\lceil (n-2)/5 \rceil$ is a tight bound on the number of edge guards needed. For a rectilinear monotone polygon the tight bound is $\lceil (n-2)/6 \rceil$ edge guards. The lower bounds are given by means of examples in section 2 and proofs of the upper bounds are given in sections 3 and 4. The proofs are constructive and lead to algorithms for placing the guards in time linear in the number of vertices. Algorithmic aspects are discussed in section 5.

2 The lower bounds.

For a monotone polygon the lower bound is $\lceil (n-2)/5 \rceil$. Starting with the simplest polygon possible, the triangle, exactly one edge guard is clearly needed. Increasing the number of vertices of the polygon, a second guard is needed as soon as the size reaches eight vertices and a third guard when the size reaches thirteen vertices. A polygon with eight vertices requiring two edge guards is shown in figure 1 and a polygon with thirteen vertices requiring three edge guards is shown in figure 2. The polygon in

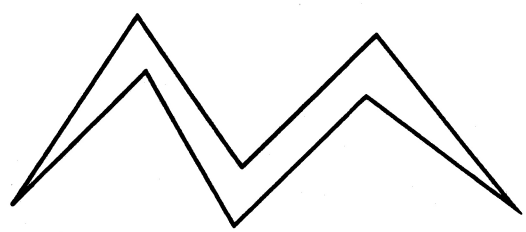

Figure 1: A polygon with 8 vertices requiring 2 edge guards.

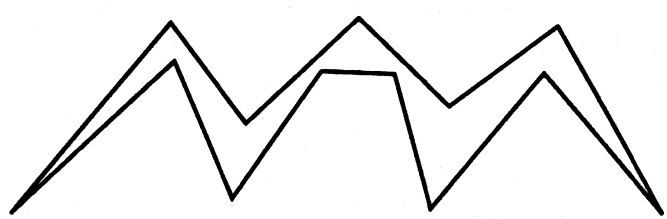

Figure 2: A polygon with 13 vertices requiring 3 edge guards.

figure 2 generalizes to a polygon with k groups of five vertices in addition to the original eight vertices requiring $k+2$ edge guards (see figure 3). This establishes the lower bound for monotone polygons.

For rectilinear monotone polygons the lower bound is $\lceil (n-2)/6 \rceil$. Figure 4 shows a polygon with ten vertices, which requires two edge guards. This polygon

*Supported in part by National Science Foundation Grants #CCR-88-03549 and #CCR-91-04732.

Tight Bounds for Edge Guards in Monotone Polygons and Rectilinear Monotone Polygons

Iliana Bjorling-Sachs and Diane L. Souvaine*
Department of Computer Science, Rutgers University, New Brunswick, New Jersey 08903
email: bjorling@paul.rutgers.edu, dls@cs.rutgers.edu

Extended Abstract

1 Introduction

The original art gallery problem asks "What is the minimum number of guards needed to watch every point inside an art gallery". The art gallery here is viewed as a simple polygon and the guards as points inside the polygon. A point x is visible to a guard y as long as the line segment $\overline{xy}$ does not intersect the exterior of the polygon. The polygon is covered if every point inside it is seen by at least one guard. Chvatal showed that no polygon with n vertices requires more than $\lfloor n/3 \rfloor$ guards and that polygons exist for which this number of guards is necessary [2]. Much attention has subsequently been devoted to variations of the original art gallery problem and many results have been obtained [4],[6]. One variation which has remained unsolved is one where guards are allowed the freedom to move along individual edges of the polygon. Guards of this type are called *edge guards*. For edge guards the notion of *weak visibility* is used. A point inside the polygon is considered visible as long as it is seen by some guard at some point during the walk along the edge. Here we show that when the gallery is restricted to the shape of a monotone polygon $\lceil (n-2)/5 \rceil$ is a tight bound on the number of edge guards needed. For a rectilinear monotone polygon the tight bound is $\lceil (n-2)/6 \rceil$ edge guards. The lower bounds are given by means of examples in section 2 and proofs of the upper bounds are given in sections 3 and 4. The proofs are constructive and lead to algorithms for placing the guards in time linear in the number of vertices. Algorithmic aspects are discussed in section 5.

2 The lower bounds.

For a monotone polygon the lower bound is $\lceil (n-2)/5 \rceil$. Starting with the simplest polygon possible, the triangle, exactly one edge guard is clearly needed. Increasing the number of vertices of the polygon, a second guard is needed as soon as the size reaches eight vertices and a third guard when the size reaches thirteen vertices. A polygon with eight vertices requiring two edge guards is shown in figure 1 and a polygon with thirteen vertices requiring three edge guards is shown in figure 2. The polygon in figure 2 generalises to a polygon with k groups of five vertices in addition to the original eight vertices requiring $k+2$ edge guards (see figure 3). This establishes the lower bound for monotone polygons.

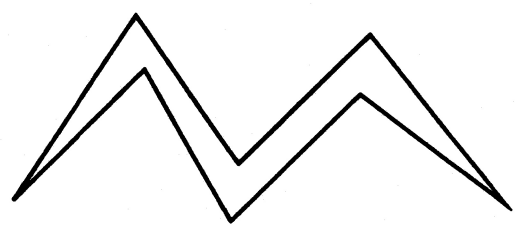

Figure 1: A polygon with 8 vertices requiring 2 edge guards.

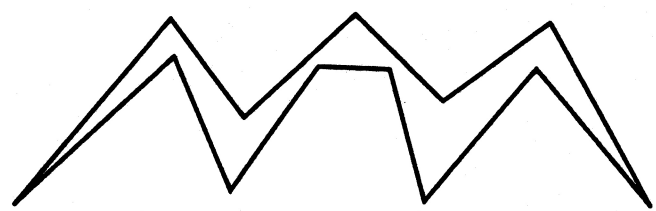

Figure 2: A polygon with 13 vertices requiring 3 edge guards.

For rectilinear monotone polygons the lower bound is $\lceil (n-2)/6 \rceil$. Figure 4 shows a polygon with ten vertices, which requires two edge guards. This polygon

*Supported in part by National Science Foundation Grants #CCR-88-03549 and #CCR-91-04732.

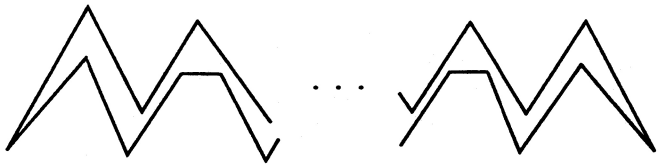

Figure 3: A polygon with $5k + 8$ vertices requiring $k + 2$ edge guards.

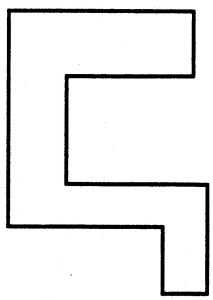

Figure 4: A rectilinear monotone polygon with 10 vertices requiring 2 edge guards.

is easily generalized to a polygon with n vertices requiring $\lceil (n-2)/6 \rceil$ edge guards. This establishes the lower bound for rectilinear monotone polygons.

3 The upper bound for monotone polygons.

The upper bound for monotone polygons is $\lceil (n-2)/5 \rceil$. Let P be a monotone polygon with n vertices. The triangulation of P yields $n-2$ triangles. We show that these triangles can be grouped in such a way that every group is covered by one edge guard and every group except one either contains at least five triangles or contains four triangles with a following group containing at least six triangles. Since there are at most $\lceil (n-2)/5 \rceil$ such groups, this same number of guards is sufficient to cover P.

P is triangulated using the algorithm by Garey, Johnson, Preparata and Tarjan for monotone polygons [3]. Assume without loss of generality that P is monotone in the y-direction. The algorithm processes the vertices in order of descending y-coordinate. Processed vertices are kept on a stack as long as they remain vertices of the still to be triangulated part of P. Triangles are created by internal diagonals drawn from the vertex being processed to vertices on top of the stack, which is then popped. These triangles are of two types. *Cross-triangles* have two vertices on one chain and the third vertex on the opposite chain and thus stretch across the polygon from one side to the other. *Side-triangles* have all vertices on the same chain and can be said to lie either on the right side or the left side of the polygon. We make the following observation regarding the existence of side-triangles:

Observation 1 *The extent in the y-direction of a side-triangle or group of adjacent side-triangles along one chain is spanned by a single edge on the opposite chain.*

This follows from the way in which the algorithm produces the triangles. We group the triangles and assign the edge guards during the triangulation. As the diagonals are generated keep a count on the number of unguarded triangles formed. Each time the generated diagonal is a cross-diagonal check if this count is at least five, is so stop and assign a guard. Since the check is made whenever a cross-diagonal is generated, at least one and at most five cross-triangles have been formed since the last stop. The possible configurations of the unguarded triangles give rise to a number of cases for the choice of edge guard. The different cases are divided into five major categories. Within each category the number of cross-triangles is the same, but the generated cross-diagonals differ as does the position of any existing side-triangles. In each case the guard is chosen in such a way that we have the following invariant:

Invariant: Following the stop at a cross-diagonal any side-triangles left unguarded by the newly assigned guard lie between the two most recently generated cross-diagonals and on the same side as the bottom-most vertex.

All the cases with there associated guards are described in the complete version of our paper. For the most part the choice of guard is straightforward. As an example we show a case in the category where one cross-triangle is present. Assume that the cross-triangle has one vertex l_i on the left chain and two vertices r_j and r_k on the right chain, i.e. the cross-diagonals are $l_i r_j$ and $l_i r_k$. There must be at least 4 side-triangles present or we would not have stopped to assign a guard.

Case 3.0.1 *Cross-diagonals are $l_i r_j$ and $l_i r_k$. Vertex l_i lies below r_j and r_k.*

It follows from the way in which the algorithm generates the diagonals that if $l_i r_j$ is a cross-diagonal, no vertex on the left chain can lie above l_i and below r_j

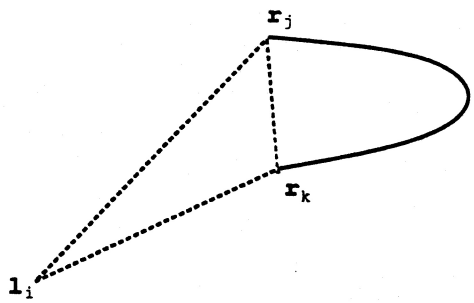

Figure 5: A guard on edge $l_{i-1}l_i$ sees all triangles.

(see figure 5). Thus no edge on the right chain spans the y-extent of the area between l_{i-1} and l_i. It follows from the invariant and observation 1, that no side-triangles were left unguarded when a guard was assigned to the group above $l_i r_j$. All unguarded side-triangles must lie on the right chain between r_j and r_k. A guard assigned to edge $l_{i-1}l_i$ sees all triangles and no side-triangles above $l_i r_k$ are left unguarded.◇

A few cases require a more thorough exploration of the relative positions of the vertices. An example of this is the following case in the category where four cross-triangles are present. The four cross-triangles are formed by five cross-diagonals. The endpoints of these may be distributed with one, two or three endpoints on one chain and the remaining five, four or three endpoints on the opposite chain. Consider the case when two endpoints l_i and l_j lie on the left chain and the four endpoints on the right chain are in order r_k, r_l, r_m and r_n. Let three cross-diagonals emanate from l_i and two from l_j, i.e. the cross-diagonals are $l_i r_k$, $l_i r_l$, $l_i r_m$, $l_j r_m$ and $l_j r_n$.

Case 3.0.2 *Cross-diagonals are $l_i r_k$, $l_i r_l$, $l_i r_m$, $l_j r_m$ and $l_j r_n$. Vertex l_i lies below r_m.*

As in the previous case, vertex l_{i-1} cannot lie below r_k and no unguarded side-triangles lie above $l_i r_k$. Furthermore, vertex l_j must lie above r_n since otherwise $l_i r_n$ would be a cross-diagonal instead of $l_j r_m$. Thus no edge on the left chain spans the distance between r_m and r_n and no side-triangles can lie between these two vertices on the right side. If two side-triangles are present above $l_j r_m$, the count would reach five when this diagonal was generated and the stop made there. If no side-triangle is present above $l_j r_m$, the count would still be four at cross-diagonal $l_j r_n$. Therefore exactly one side-triangle is present above $l_j r_m$. This single side-triangle may lie between r_k and r_l, between r_l and r_m or between l_i and l_j.

If the side-triangle lies between r_k and r_l (see figure 6), a guard on edge $r_l r_m$ sees all triangles.

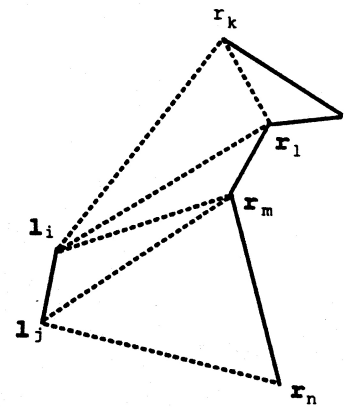

Figure 6: A side-triangle lies between r_k and r_l.

If the side-triangle lies between r_l and r_m (see figure 7), no single edge is guaranteed to see all triangles in every situation. Edge $l_{i-1}l_i$ certainly sees the side-

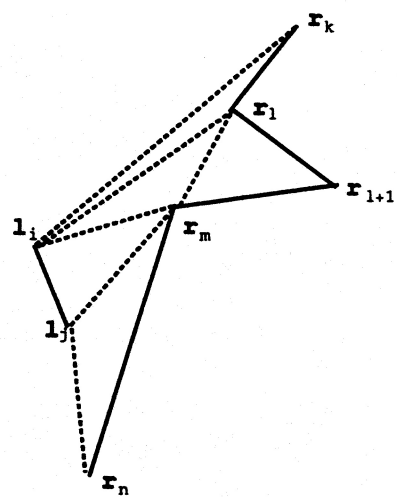

Figure 7: A side-triangle lies between r_l and r_m.

triangle and the three topmost cross-triangles. However part of the fourth cross-triangle $l_j r_m r_n$ may be hidden from view. Similarly edge $l_i l_j$ sees all cross-triangles, but not necessarily the side-triangle. If l_i, l_j, r_n forms a left turn, edge $l_{i-1}l_i$ sees all triangles. If l_i, l_j, r_n forms a right turn but l_i, r_m, r_{l+1} forms a left turn, a guard on edge $l_i l_j$ sees all triangles. If both l_i, l_j, r_n and l_i, r_m, r_{l+1} are right turns, but r_n, r_m, r_l forms a left turn, a guard on edge $r_l r_{l+1}$ sees all triangles. Finally if all the above are right turns, a guard on edge $l_{i-1}l_i$ sees all triangles. This is so because with cross-diagonal $l_i r_k$ present, r_l, r_k

and l_{i-1} must form a right turn.

If the side-triangle lies between l_i and l_j (see figure 8), there is again no single edge guaranteed to see all triangles in every situation. If l_i, l_j, r_n forms a

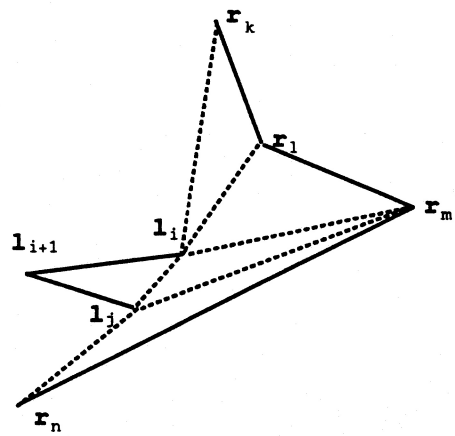

Figure 8: A side-triangle lies between l_i and l_j.

left turn, a guard on edge $l_{i-1}l_i$ sees all triangles. If l_i, l_j, r_n forms a right turn but r_m, l_i, l_{i+1} forms a left turn, a guard on edge $r_l r_m$ sees all triangles. If both l_i, l_j, r_n and r_m, l_i, l_{i+1} are right turns, but r_k, l_i, l_j is a left turn, a guard on edge $l_{i+1}l_j$ sees all triangles. Finally if all the above are right turns, a guard on edge edge $r_m r_n$ sees all triangles.◇

Finally it is possible that the group of triangles for which a guard is sought cannot be covered by any edge above the cross-diagonal where the stop was made. Consider a subcase of the following case, where again exactly four cross-triangles are present and the two vertices on the left chain are l_i and l_j.

Case 3.0.3 *Cross-diagonals are $l_i r_k$, $l_i r_l$, $l_j r_l$, $l_j r_m$ and $l_j r_n$. Vertex l_i lies between r_k and r_l.*

If in addition vertex l_j lies avbove r_l and a side-triangle is present between r_l and r_m, we have a subcase where it is possible that no edge above $l_j r_n$ can see more than four triangles (see figure 9). An edge below $l_j r_n$ will be used. We show that it is always possible to choose either an edge from which all the unguarded triangles above $l_j r_n$ can be seen or an edge from which the bottom cross-triangle l_j, r_m, r_n is seen and which in addition can be used as guard for the next group of triangles. In the latter case this guard is used together with a guard on edge $r_k r_l$ from which the top four triangles are seen. Thus as long as the group below $l_j r_n$ is not the last group in the polygon, the two guards together cover ten triangles. If

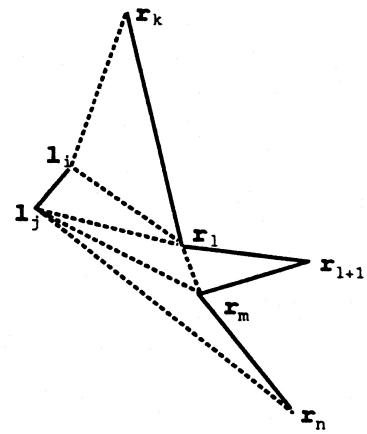

Figure 9: No edge above $l_j r_n$ sees more than four triangles.

the group below $l_j r_n$ is the last group of triangles in the polygon, it may contain less than five triangles. However, if there are additional triangles below $l_j r_n$, the two guards chosen as above are still within the $\lceil (n-2)/5 \rceil$ bound. If no triangles lie below $l_j r_n$ then $l_j r_n$ is in fact an edge of the polygon. A guard on this edge sees all the triangles in the group above.◇

The assignment of guards as described by all the cases in the complete version of our paper ensures that no more than $(n-2)/5$ guards are used when the number of triangles in the polygon is a multiple of five. One additional guard is used for any remaining triangles giving the upper bound of $\lceil (n-2)/5 \rceil$ edge guards used.

4 The upper bound for rectilinear monotone polygons

The upper bound for rectilinear monotone polygons is $\lceil (n-2)/6 \rceil$. The proof is similar to that for monotone polygons, but the polygon is partitioned into convex quadrilaterals instead of triangles. These quadrilaterals are grouped and an edge guard is assigned to each group. Taking the place of the triangulation algorithm is Sack's monotone quadrilateralization algorithm [5]. Assume that the polygon is monotone in the y-direction. The algorithm processes the horizontal edges in order of descending y-coordinate. Top edges are pushed onto a stack. Quadrilaterals are created by internal diagonals drawn between bottom edges being processed and top edges popped off the stack. A diagonal can itself be treated as a horizontal edge and be pushed onto the stack or become the

next edge to be processed. A complete description of the algorithm is not given here. We mention only the following two similarities to the triangulation algorithm. Firstly the generated quadrilaterals are of two types. *Cross-quadrilaterals* have vertices on both chains and stretch from one side to the other. *Side-quadrilaterals* have all vertices on the same chain and lie on the right side or the left side of the polygon. Secondly the observation below follows from the way in which the quadrilaterals are produced.

Observation 2 *The extent in the y-direction of a side-quadrilateral or a group of adjacent side-quadrilaterals along one chain is spanned by a single edge on the opposite chain.*

The quadrilaterals can be grouped into groups of three with one edge guard assigned to each group. The guard is always assigned in such a way that we have the following invariant:

Invariant: Following the stop at a cross-diagonal any side-quadrilaterals left unguarded by the newly assigned guard lie on the same side as the bottom-most vertex and immediately above it.

Each time a cross-diagonal has been generated and the number of unguarded quadrilaterals above it is at least three, we stop and assign a guard. The cases arising from the possible different configurations of quadrilaterals are divided into three categories, with category i containing the cases where i cross-quadrilaterals are present. In all the cases the choice of guard is straightforward. An illustration is the following case where two cross-quadrilaterals are present. Assume the three generated cross-diagonals have two vertices l_i and l_j on the left chain and the remaining four vertices r_k, r_l, r_m and r_n on the right chain. At most one side-quadrilateral can lie above the next to last cross-diagonal or we would have stopped when it was generated.

Case 4.0.4 *Cross-diagonals are $l_i r_k$, $l_i r_m$ and $l_j r_n$. Vertex l_i lies between r_l and r_m.*

Side-quadrilaterals are possible between vertices r_k and r_l, between l_i and l_j and between r_m and r_n (see figure 10). As noted above at most one side-quadrilateral lies between r_k and r_l. A guard on edge $r_l r_m$ sees this quadrilateral, any quadrilaterals between l_i and l_j, the two cross-quadrilaterals and at least one of the quadrilaterals that may lie between r_m and r_n. Any additional quadrilaterals between r_m and r_n are left to the next guard assignment.◇

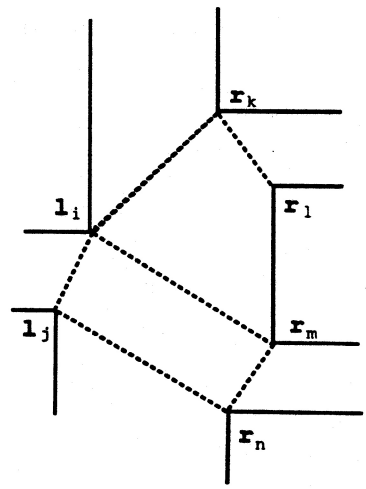

Figure 10: At most one quadrilateral lies between r_k and r_l.

For rectilinear monotone polygons there is no counterpart to the case where for monotone polygons we need to assign two guards that together cover ten triangles. Instead one guard is always assigned for each group of at least three quadrilaterals. Since the number of quadrilaterals in a polygon with n vertices is $(n-2)/2$, this ensures that no more than $(n-2)/6$ guards are used when the number of quadrilaterals is a multiple of three. With one additional guard assigned to any remaining quadrilaterals the number of guards is at most $\lceil (n-2)/6 \rceil$.

5 Algorithmic aspects

The proofs in sections 3 and 4 use existing algorithms for partitioning the polygon. These algorithms are modified to enable them to assign edge guards.

The main modification is the addition of a case statement, where each case results in the output of an edge guard. We also include a count, which keeps track of the number of unguarded triangles or quadrilaterals above the most recent cross-diagonal. The case statement is run through each time the count reaches five or three, respectively.

To distinguish between the cases we need to know what the configuration of the currently unguarded triangles or quadrilaterals looks like. Specifically we need to know the vertices of the cross-triangles or cross-quadrilaterals, the number of side-triangles or side-quadrilaterals below each such vertex and the generated cross-diagonals. Therefore where the original algorithm outputs a cross-diagonal the modified algorithm instead stores this diagonal in an array CD.

The endpoints are also stored in two separate arrays L and R. In the algorithm for rectilinear monotone polygons L and R also contain vertices that are not endpoints of cross-diagonals, but which lie on a cross-quadrilateral. Where the original triangulation algorithm outputs a diagonal with both endpoints on the same chain, the modified algorithm increases by one a count of the number of side-triangles and remembers the top endpoint of the diagonal. In a collection of adjacent side-triangles the last diagonal to be generated has as top endpoint a vertex on a cross-triangle. Once a cross-diagonal has been generated below this vertex, the count is stored with the remembered vertex and added to the count of unguarded triangles before being reset to 0. A similar process works for side-quadrilaterals. Since the information for a particular group is no longer needed after a guard for the group has been assigned, all the arrays are of constant size.

Checking the cross-diagonals in the array CD against those present in a particular case takes constant time. So does checking the number of vertices in L and R, making any needed comparisons between these vertices and looking up the number of side-triangles or side-quadrilaterals below a vertex. Since there are a constant number of cases, it takes constant time to output one edge guard once the diagonals for the group have been generated. Both the triangulation algorithm and the quadrilateralization algorithm take $O(n)$ time to generate all the diagonals, where n is the number of vertices. The number of edge guards assigned is also $O(n)$. With constant time taken by the assignment of each guard, total time taken is $O(n)$.

References

[1] Aggarwal, A. "The Art Gallery Theorem: Its Variations, Applications and Algorithmic Aspects," Ph.D. thesis, Johns Hopkins University, 1984.

[2] Chvatal, V. "A Combinatorial Theorem in Plane Geometry," *J. Combinatorial Theory Series* B 18 (1975), 39-41.

[3] Garey, Johnson, Preparata and Tarjan, "Triangulating a Simple Polygon," *Info. Proc. Letters* 7 (1978), 175-180.

[4] O'Rourke, J. "Art gallery problems and algorithms," Oxford University Press (1987).

[5] Sack, J.R. and Toussaint, G. "Guard Placement in Rectilinear polygons," *Computational Morphology* (1988), 153-175.

[6] Shermer, T. "Recent Results in Art Galleries," Simon Fraser University Technical Report, October 1990.