Algorithmica (1999) 24: 50–75

Algorithmica

Computing Vision Points in Polygons

S. Carlsson[1] and B. J. Nilsson[2]

Abstract. We consider a restricted version of the art gallery problem within simple polygons in which the guards are required to lie on a given one-dimensional object, a watchman route. We call this problem the *vision point problem*. We prove the following:

- The original art gallery problem is NP-hard for the very restricted class of *street* polygons.
- The vision point problem can be solved efficiently for the class of street polygons.
- A linear time algorithm for the vision point problem exists for the subclass of street polygons called *straight walkable* polygons.

Key Words. Computational geometry, Algorithms, Art gallery problems.

1. Introduction. The problem of placing guards in an art gallery so that every point in the gallery is visible to at least one guard has been considered by several researchers. If the gallery is represented by a polygon (having n vertices) and the guards are points in the polygon, then visibility problems can be equivalently stated as problems of covering the gallery with star-shaped polygons. Chvàtal [8] and Fisk [11] proved that $\lfloor n/3 \rfloor$ guards are always sufficient and sometimes necessary to cover a polygon of n edges.

Consider a convex polygon of n edges. The bound of $\lfloor n/3 \rfloor$ guards is a gross overestimation on the number of guards, since one guard will suffice for any convex polygon. Similarly, depending on the structure of a polygon with n edges, the minimum number of guards may be smaller than the estimate, and, therefore, it becomes interesting to find algorithmic methods to compute the minimum number of guards necessary to cover a given polygon. Alas, Aggarwal [1] and Lee and Lin [22] have proved that this problem is NP-hard. A survey of results on art gallery problems and general visibility problems can be found in O'Rourke's monograph [26] and the articles by Keil and Sack [19] and Shermer [27].

Since the art gallery problem is hard to solve, posing certain restrictions on the problem may yield versions that are efficiently solvable but that still maintain practical use. We consider the following version: given a polygon $\mathbf{P}$ of n edges and a closed polygonal curve W of m edges, what is the minimum number of guards that cover $\mathbf{P}$ and are restricted to lie on W. This, of course, requires that it is possible to make such a placement of guards on W, i.e., we require that the set of points of W form a guard cover for $\mathbf{P}$. A closed curve with this property is called a *watchman route* for $\mathbf{P}$, the minimum number of guards on W are called the *vision points* for W, and this version of the art gallery problem we call the *vision point problem* for a polygon $\mathbf{P}$ and a route W.

[1] Department of Computer Science, Luleå University of Technology, 971 87 Luleå, Sweden.
[2] Department of Computer Science, Lund University, Box 118, 221 00 Lund, Sweden.

Received June 6, 1996; revised September 12, 1997. Communicated by D. T. Lee.

Among the practical applications for the vision point problem, one comes from the area of robotics. A robot moving along a watchman route may be incapable of engaging its vision system continuously, and, therefore, it may have to stop at points on the route to obtain visibility information accurately. Our covering problem can also be stated as a problem of illumination. Consider the polygon as a room with the boundary being walls and the watchman route as an electric cable embedded in the ceiling. We ask for the minimum number of light bulbs and their positions on the cable such that the whole room is illuminated.

Unfortunately, the vision point problem turns out to be NP-hard for simple polygons in general [4], [24]. In the rest of this presentation, we therefore restrict the watchman route and consider the class of simple polygons that allow this restricted type of watchman routes.

The next section is dedicated to defining the type of watchman route that we consider and establishing the class of simple polygons, the class of *streets*.

In Section 3 we prove that the original art gallery problem is NP-hard for streets, and in Section 4 we present an efficient algorithm for the vision point problem in streets.

An important and well studied subclass of the streets are *straight walkable polygons*, and in Section 5 we present a linear time algorithm for the vision point problem in straight walkable polygons when the watchman route obeys certain additional properties.

2. Definitions and Preliminary Results. We begin this section with some definitions. Most of them are standard in computational geometry.

Let $\mathbf{P}$ be a simple polygon and let $\mathcal{S}$ denote a subset of the points in $\mathbf{P}$. The set $\mathcal{S}$ is a *guard set* or a *guard cover*, if, for any point p in $\mathbf{P}$, there is a point q in $\mathcal{S}$ that sees p. Two points p and q in $\mathbf{P}$ are said to *see* each other if the line segment joining p and q lies in $\mathbf{P}$. A subset $\mathcal{S}'$ of $\mathcal{S}$ is a set of *vision points*, if $\mathcal{S}'$ is a guard set for $\mathbf{P}$. A set $\mathcal{OPT}(\mathcal{S})$ is a smallest cardinality set of vision points for a guard set $\mathcal{S}$ in $\mathbf{P}$.

We consider the specific problem where the sets of vision points are subsets of a given polygonal curve in $\mathbf{P}$. Such a curve is known as a *watchman route*. In many cases, watchman routes can be computed efficiently. The boundary of the polygon $\mathbf{P}$ forms a closed watchman route that is easily obtainable. Chin and Ntafos [7] and Carlsson et al. [3], [24] provide polynomial time algorithms for the shortest closed watchman route problem, in rectilinear simple and simple polygons, respectively. In a recent result, Carlsson and Jonsson [2] establish an efficient algorithm for computing the shortest open watchman route, the shortest watchman path, in a simple polygon.

Denote by $\mathbf{VP}(p)$ the visibility polygon of a point p in $\mathbf{P}$. The edges of a visibility polygon in $\mathbf{P}$ that are not edges of $\mathbf{P}$ are called the *windows* of the visibility polygon. Each window separates the visibility polygon from subpolygons of $\mathbf{P}$ that are known as *pockets*; see Figure 1(a).

A *cut* in a simple polygon $\mathbf{P}$ is a directed line segment having its two endpoints on the boundary of $\mathbf{P}$, and at least one interior point in the interior of $\mathbf{P}$. A window of a visibility polygon $\mathbf{VP}(p)$ is a cut if we specify a direction for the window. For windows we choose the direction away from the point p collinear to the window.

Two other specific types of cuts are used extensively in the following. An *essential cut e* is a maximal extension of an edge of a polygon $\mathbf{P}$, i.e., the line segment returned

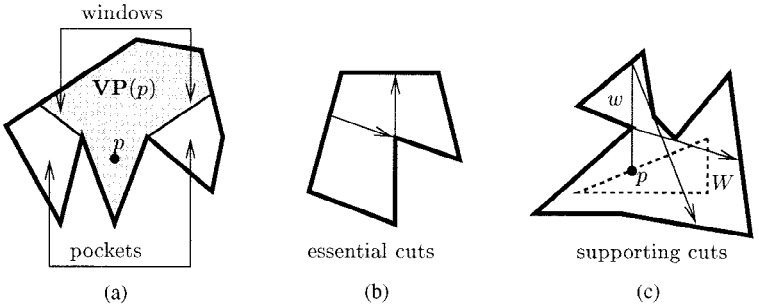

Fig. 1. Illustrating definitions.

by a ray shooting operation from an endpoint of the edge directed away from the edge. The direction associated to e is the same as the direction of the associated edge when the polygon is traversed in counterclockwise order; see Figure 1(b). An essential cut e is a *forward essential cut* with respect to a point p, if p lies to the left of e in **P**.

To define the other type of cut, we must assume that we have a simple polygon **P**, a watchman route W, and a point p on W. The two *supporting cuts* of a point p associated to a window w of $\mathbf{VP}(p)$ are the two maximal line segments in **P**, starting at the two endpoints of w, intersecting W, and having the largest angle to w. The associated directions of the supporting cuts are away from w; see Figure 1(c).

We study the problem of guarding a class of simple polygons, for which we can choose watchman routes that have certain useful properties. Let **P** be a simple polygon such that there exists two distinct points with the property that any simple path between the points forms a guard set for **P**. We show that we can find an optimum set of vision points on the path for this class of polygons efficiently. We define the class formally.

DEFINITION 2.1. Let **P** be a simple polygon. A collapsed watchman route W for **P** is a polygonal curve such that, for all points $p \in \mathbf{P}$, the set $\mathbf{VP}(p) \cap W$ is a connected chain.

If **P** has a collapsed watchman route, we say that **P** is contained in the class of polygons with collapsed watchman routes.

The same class of polygons has been studied before by Klein [20] who called the polygons of this class *streets*. The definition states that a polygon is a street if the boundary can be partitioned into two chains U and D that are mutually visible. In this section we denote the two endpoints of U and D by s and t and assume that the two points are given. This may seem a severe restriction, but the two endpoints can, in fact, be found in linear time [9]. Examples of streets and nonstreets are given in Figure 2.

We prove that the streets are exactly the class of polygons with collapsed watchman routes.

LEMMA 2.1. *A polygon has a collapsed watchman route if and only if it is a street.*

PROOF. We begin by showing that if **P** is a street, then we can use the stronger statement that both U and D are guard sets for **P**. Assume that this is not the case and let p be a

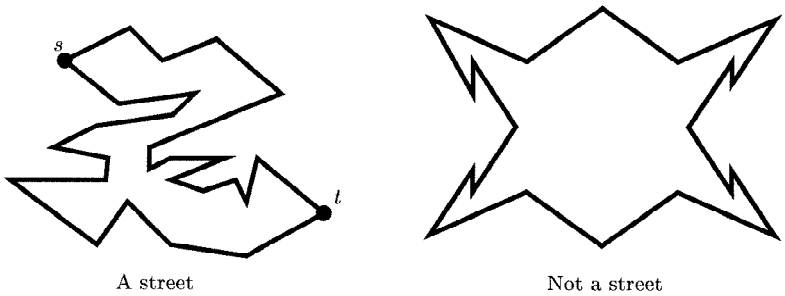

Fig. 2. Examples of polygons.

point seen by only one of the two chains, say U. Let S_p be the shortest path from s to p and similarly let T_p be the shortest path from t to p. The two paths S_p and T_p must have at least the link closest to p in common, otherwise p sees both U and D; see Figure 3(a). Denote this link by l. However, this means that there is an edge e of U adjacent to the turning point of l that is not seen by D, thus contradicting the fact that U and D are mutually visible; see Figure 3(b). Hence, both U and D are guard sets for **P**.

We use this result to show the actual lemma. We begin with the if case. Let **P** be a street and let S be the shortest path from s to t within **P**. Since S is a shortest path, the intersection of S with any visibility polygon in **P** is a chain. Hence, it remains to show that S is a guard set to have shown that **P** has a collapsed watchman route. Let p be a point in **P**. Point p is seen both by a point q on U and by a point q' on D. One of the segments $[p, q]$ or $[p, q']$ intersects S, and hence, a point of S sees p; see Figure 3(c).

The only if case is proved in a similar manner. Let **P** be a polygon with a collapsed watchman route. Let W be the shortest collapsed watchman route and extend the first and last segments until they hit the boundary of **P** at points s' and t'. Let U' be the subchain of the boundary of **P** obtained by walking counterclockwise along the boundary from

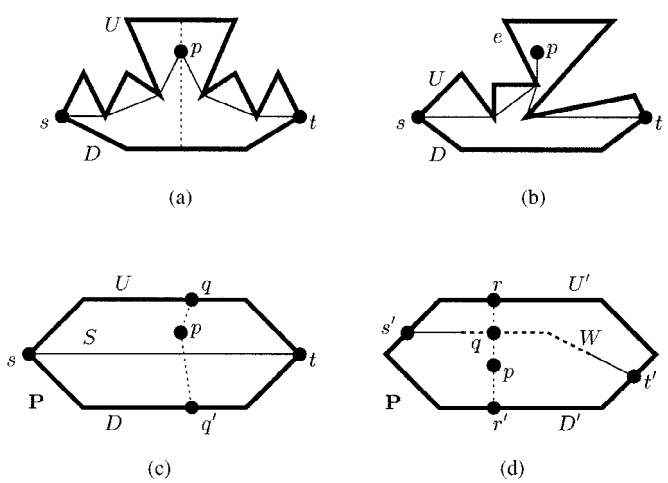

Fig. 3. Illustrating Lemma 2.1

t' to s'. Similarly let D' be the other subchain. Take a point p in **P**. This point is seen from a point q on W. Extend the segment $[p, q]$ until it hits the boundary of **P** on both sides, at points r and r'. By the construction of the chains U' and D' and the fact that W is shortest, the points r and r' must lie on different chains. Hence, both U' and D' are guard sets for **P**; see Figure 3(d). □

We henceforth call this class of polygons streets. The class of streets encompasses many common classes of polygons, e.g., star-shaped polygons, two-guardable polygons, weakly edge visible polygons, and walkable polygons of which two commonly known subclasses are the monotone polygons and the spiral polygons.

In the following we assume that **P** is a street with a boundary having n edges. We divert somewhat and show that computing the optimum guard cover for a street is NP-hard.

3. Complexity of Guarding Streets.

In this section we prove that finding the minimum number of point guards in a street is NP-hard. This is the first result that shows that optimum guarding is NP-hard for a restricted class of simple polygons.

THEOREM 1. *Finding the minimum number of point guards in a street is NP-hard.*

PROOF. We rephrase the problem as a decision problem.

Instance: A street **P** and a positive integer B.
Question: Is there a set of at most B point guards that covers **P**?

The proof is by reduction from 3-SATISFIABILITY, which is NP-complete [12].

Instance: A Boolean formula

$$(l_{1,1} \vee l_{1,2} \vee l_{1,3}) \wedge (l_{2,1} \vee l_{2,2} \vee l_{2,3}) \wedge \cdots \wedge (l_{k,1} \vee l_{k,2} \vee l_{k,3})$$

consisting of k clauses, each containing three literals, where each literal is a Boolean variable $l_{i,j} = u_m$ or the conjugate of a Boolean variable $l_{i,j} = \bar{u}_m$, out of a set of n Boolean variables $u_1, \ldots, u_n$.

Question: Is there a truth assignment to the n variables that satisfies the formula?

Let a 3-SATISFIABILITY instance have k clauses and n variables. We modify the proof of Lee and Lin [22] and Aggarwal [1] that finding a minimum guard cover for simple polygons is NP-hard to hold for streets. The proof gives the construction of a polygon that is guardable with $3k + n + 1$ guards, if and only if the 3-SATISFIABILITY instance is satisfiable.

Each variable corresponds to two *well structures* in the polygon, that represent the variable and its conjugate. The main parts of the wells are seen from a distinguished point x in the upper left corner of the polygon; see Figure 4. Each pair of wells also has a notch u which is seen by the distinguished points F and T that mark whether the variable is set to false or true. Furthermore, we ensure that every notch u is seen by point t, the rightmost vertex of the polygon.

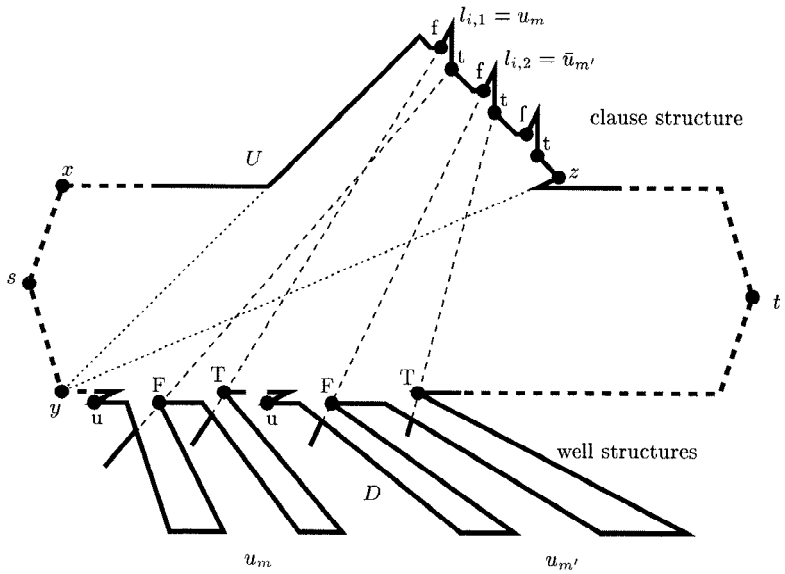

Fig. 4. Illustrating the proof of Theorem 1.

Each clause in the Boolean formula corresponds to a *clause structure* in the upper part of the polygon, having three notches that represent the three literals and two distinguished points f and t in each notch that mark whether the literal is set to false or true; see Figure 4.

A literal notch in a clause structure is connected to the appropriate well structure in the following way: If the literal is a variable, we add two small spikes in the wells in such a way that guards at x, T, and t, or guards at x, F, and f will see the two well structures and the two spikes, as in the example $l_{i,1} = u_m$ of Figure 4. Similarly, if the literal is the conjugate of a variable, we add two small spikes in the wells in such a way that guards at x, T, and f, or guards at x, F, and t will see the two well structures and the two spikes, as in the example $l_{i,2} = \bar{u}_{m'}$ of Figure 4. Furthermore, at least one of the guards in a clause structure must be placed at a point t in order for the clause structure to be completely seen, specifically point z must be seen, thus ensuring that the corresponding clause in the Boolean formula is satisfied by the truth assignment of the variables. Should all the guards in a clause structure be placed at points f, then at least one more guard is needed, e.g., at point y.

The proof thus specifies $6k + 2n + 2$ possible points for guards such that a minimum vertex guard cover uses $3k+n+1$ of them as guard locations and this is a minimum vertex guard cover if and only if the corresponding 3-SATISFIABILITY formula is satisfiable. See Figure 5 for a complete example.

The proof can be generalized to interior point guards in the same way as is done by Aggarwal [1] by altering the polygon in such a way that the distinguished points become interior points of the polygon.

It remains to prove that the polygon we have constructed is a street. We partition the boundary of the polygon into two chains U and D and show that both these chains are mutually visible. Let s and t be the leftmost and rightmost points of the polygon; see

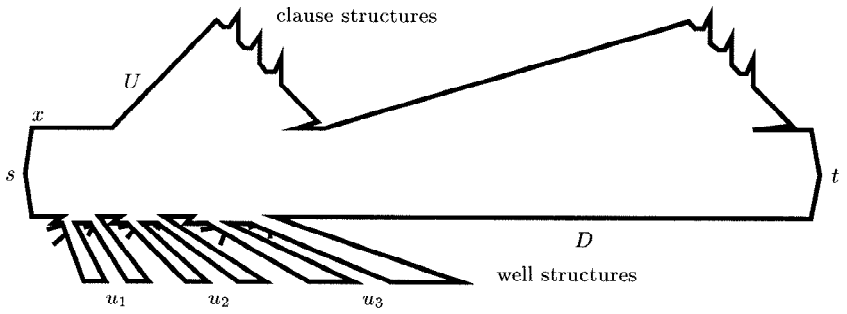

Fig. 5. Example consisting of two clauses and three variables.

Figure 4. Let D be the chain obtained by following the polygon boundary in counterclockwise order from s to t, and let U be the chain obtained by following the boundary in counterclockwise order from t to s.

Let p be a point on U and move p downward until it reaches the polygon boundary at point p'. If p' lies on D, then D sees p, otherwise p lies in a clause structure and is seen by point y. Hence, D sees p.

Now, let p be a point on D and move p upward until it reaches the polygon boundary at point p''. If p'' lies on U, then U sees p, otherwise p lies in a well structure and is seen either by point x, some f or t point in a clause structure, or point t. Hence, U sees p. □

In the following we show that if we restrict the guards to lie on a collapsed watchman route in a street, then we can compute an optimum cover efficiently.

4. Optimum Vision Points in Streets. Assume that **P** is a street with n edges and that W is a collapsed watchman route with m edges inside **P**. The crucial property of the collapsed watchman route W is that the intersection with the visibility polygon of any point in **P** is a connected chain.

The problem we study in this section is to find an optimum set of vision points for **P** on W, i.e., a set $\mathcal{OPT}(W)$ in **P**. Unfortunately, the number of vision points can be arbitrarily large; see Figure 6(a). The number of vision points depends on how narrow the mouth of the notch is. By changing the situation slightly, we can get an even worse situation; see Figure 6(b). If the three vertices are collinear, then we need an infinite number of vision points on W. Hence, a crucial property that we assume in the following is that **P** is in general position, i.e., no three vertices are collinear. Still, since there is no relation between the number of vision points and the size of the polygon or the size of the watchman route, our algorithms will depend on the number of vision points computed. The parameter k will henceforth be used to denote this value.

To solve the problem of computing a smallest set of vision points, we need the concept of a limit point with respect to a given point g on W. To define this formally we introduce some preliminary notation.

Let e be an essential cut that crosses W and partitions **P** into two parts, $\mathbf{P}_s$ and $\mathbf{P}_t$,

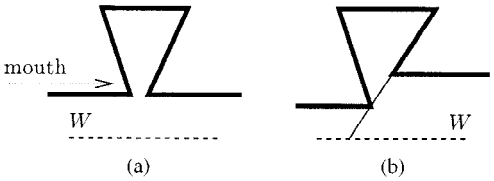

Fig. 6. Examples requiring many vision points.

part $\mathbf{P}_s$ containing s, and $\mathbf{P}_t$ containing t. Denote the endpoint of W that lies in $\mathbf{P}_s$ by g_s, and in the same way, denote the other endpoint of W by g_t; see Figure 7(a).

If we move a point along W from g_s toward g_t, we say that the point moves in the *forward direction*. This makes it possible to talk about points lying before and after other points on W. We say that a point g on W lies before a point g' on W if g is reached before g' when we move along W from g_s in the forward direction. Similarly g' lies after g.

Let p be any point in $\mathbf{P}$. By Definition 2.1 we know that p sees some subchain of W. We denote this subchain by $\mathrm{WC}(p)$, and the two endpoints of this subchain by $left(p)$ and $right(p)$ in such a way that $left(p)$ lies before $right(p)$ on W; see Figure 7(b).

Let g and g' be two points on W, with g lying before g'. Denote by $W(g, g')$ the subchain of W from g to g'. We let the visibility polygon of $W(g, g')$, denoted by $\mathbf{VP}(W(g, g'))$, be the set $\{p \in \mathbf{VP}q \mid q \in W(g, g')\}$.

If $\mathcal{S}$ is a set of points on W, we let $nearest_g(\mathcal{S})$ denote the first point in $\mathcal{S}$ reached from g along W in the forward direction.

Let $\mathbf{Q}$ be some region, possibly consisting of several components. We let $closure(\mathbf{Q})$ denote the closure of the set $\mathbf{Q}$ of points, i.e., the set $\mathbf{Q} \cup bd(\mathbf{Q})$, where $bd(Q)$ denotes the boundary of the region $\mathbf{Q}$.

With these notational conventions we can define the limit point of a point on W.

DEFINITION 4.1. The limit point of a point g on W is the point on W specified by

$$lp(g) = nearest_g(\{right(p) \mid p \in closure(\mathbf{P} \backslash \mathbf{VP}(W(g_s, g)))\}).$$

This definition captures the intuitive notion that the limit point of a guard g is the furthest possible point on W for the next guard. This follows from Lemmas 4.1 and 4.2.

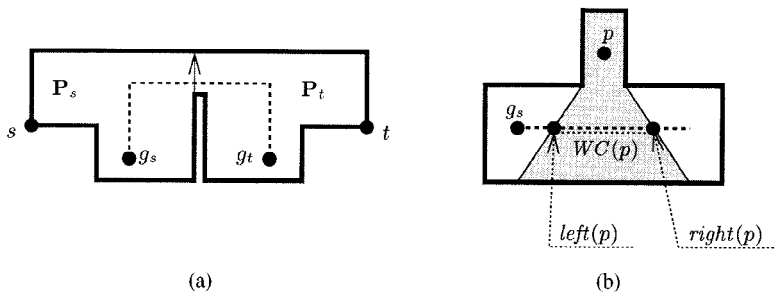

Fig. 7. Illustrating notational conventions.

We can define $lp^i(g)$ by

$$\begin{aligned} lp^0(g) &= g, \\ lp^i(g) &= lp(lp^{i-1}(g)) \qquad \text{for} \quad i > 0. \end{aligned}$$

LEMMA 4.1. *If g is a point on W and $g' = lp(g)$, then*

$$\mathbf{VP}(W(g_s, g)) \cup \mathbf{VP}(g') = \mathbf{VP}(W(g_s, g')).$$

PROOF. Evidently, the region $\mathbf{VP}(W(g_s, g)) \cup \mathbf{VP}(g')$ is a subset of the region $\mathbf{VP}(W(g_s, g'))$, since the set $W(g_s, g) \cup \{g'\}$ is a subset of $W(g_s, g')$. Hence, it only remains to prove the reverse inclusion.

Let p be a point in $VP(W(g_s, g'))$. If p is seen by some point in $W(g_s, g)$, then p lies in $\mathbf{VP}(W(g_s, g)) \cup \mathbf{VP}(g')$, and we are done. If p is not seen by any point in $W(g_s, g)$, then p must be seen by some point g'' that lies between g and g' on W. This means that $left(p)$ lies before g' on W, and since g' is the closest point $right(q)$, for every point q not in $\mathbf{VP}(W(g_s, g))$, it also means that g' lies before $right(p)$, and, hence, g' sees p. □

Furthermore, the subchain between a point and its limit point on W requires a vision point.

LEMMA 4.2. *If g is a point on W, then $W(g, lp(g))$ requires at least one vision point.*

PROOF. By the definition of limit point, there is a point p in a pocket of $\mathbf{VP}(W(g_s, g))$, such that $lp(g) = right(p)$. Evidently, no point on the subchain $W(g_s, g))$ sees p, and, hence, $left(p)$ lies in $W(g, lp(g))$. Thus, we have that $WC(p) \subseteq W(g, lp(g))$, and since $WC(p)$ requires a vision point, the subchain $W(g, lp(g))$ also requires a vision point. □

Define the point g_1 by

$$g_1 = nearest_{g_s}(\{right(p) \mid p \in \mathbf{P}\}).$$

We can prove the following lemma.

LEMMA 4.3. *If $g_1 = nearest_{g_s}(\{right(p) \mid p \in \mathbf{P}\})$, then*

$$\mathbf{VP}(W(g_s, g_1)) = \mathbf{VP}(g_1).$$

PROOF. Evidently, $\mathbf{VP}(g_1) \subseteq \mathbf{VP}(W(g_s, g_1))$, since $g_1 \in W(g_s, g_1)$. Hence, it remains to prove the reverse inclusion.

Let p be a point in $\mathbf{VP}(W(g_s, g_1))$. If $left(p) = g_1$, then p lies in $\mathbf{VP}(g_1)$, and we are done. Otherwise, $left(p)$ lies before g_1 on W, and since g_1 is the closest point $right(q)$, for every point q in $\mathbf{P}$, this means that g_1 lies before $right(p)$, and, hence, g_1 sees p. □

Lemmas 4.1 and 4.3 prove that there is some k for which the set

$$\{lp^i(g_1) \mid 0 \leq i \leq k-1\}$$

is a guard cover for **P**. That k cannot be unbounded follows from the fact that no three vertices of **P** are collinear. This means that there is an $\varepsilon > 0$, such that $length(WC(p)) \geq \varepsilon$, for any point p in **P**. We know that, for any point g on W, $length(W(g, lp(g))) \geq length(WC(p))$, for some point p, and hence, $k \leq length(W)/\varepsilon + 1$ which is bounded since $\varepsilon > 0$.

We present the algorithm *Optimum-Vision-Points-in-Street* to guard a street. The algorithm computes the appropriate value k and outputs the set of vision points $\{lp^i(g_1) \mid 0 \leq i \leq k-1\}$. The pseudocode for the algorithm is displayed below.

Next, we show that the presented algorithm actually computes an optimum set of vision points.

LEMMA 4.4. *The algorithm Optimum-Vision-Points-in-Street computes, given a street, an optimum set of vision points on a given collapsed watchman route.*

PROOF. Let $\mathcal{G} = \{g_1, g_2, \ldots, g_k\}$ be the set of vision points computed by our algorithm and let $\mathcal{F} = \{f_1, f_2, \ldots, f_l\}$ be a set of vision points, consisting of fewer points, that most closely matches $\mathcal{G}$, following W from g_s to g_t, i.e., f_1 lies as close to g_1 as possible, subject to this f_2 lies as close to g_2 as possible, and so on. Let g_i be the first vision point that differs from the set $\mathcal{F}$, i.e., $g_{i-1} = f_{i-1}$ but $g_i \neq f_i$.

If f_i lies before g_i, then, by Lemma 4.1, point f_i can be moved to the limit point of f_{i-1}, and we have $lp(f_{i-1}) = lp(g_{i-1}) = g_i$. This contradicts the assumption that $\mathcal{F}$ matches $\mathcal{G}$ the closest.

On the other hand, g_i cannot lie before f_i because this implies that $\mathcal{F}$ is not a guard cover, since $g_i = lp(g_{i-1})$, and, by Lemma 4.2, the portion of W between g_{i-1} and g_i requires at least one vision point. □

Algorithm *Optimum-Vision-Points-in-Street*

Input: A street **P** represented by (U, D) and a collapsed watchman route W
Output: An optimum set of vision points for **P** on W

1 Identify the point g_s
2 Preprocess **P** and W and compute the first vision point g_1
$k := 0$
3 **while P** is not completely guarded **do**
$k := k + 1$
3.1 $g_k := lp^{k-1}(g_1)$
endwhile
return $\{g_i \mid 1 \leq i \leq k\}$
End *Optimum-Vision-Points-in-Street*

It remains to show how to compute the first guard g_1, the limit points, and analyze the complexity of the algorithm. To do this, we need some further lemmas.

LEMMA 4.5. *Let* **Q** *be a polygonal region in* **P** *and let* $\mathcal{V}_{\mathbf{Q}}$ *denote the set of vertices of* **Q**. *For any point* p *in* **Q** *there is a vertex* v *in* $\mathcal{V}_{\mathbf{Q}}$ *such that* $right(v)$ *lies before* $right(p)$.

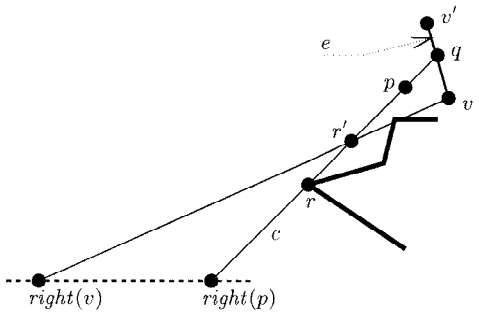

Fig. 8. Illustrating Lemma 4.5.

PROOF. Let p be a point in **Q** and let c be the maximal cut in **P** passing through points p and $right(p)$. The cut c intersects some boundary edge e of **Q** at point q. Denote the endpoints of e by v and v'; see Figure 8. The interior of the cut c intersects the boundary of **P** at some point r, and, hence, the segment $[v, right(v)]$ must intersect c at some point r', with r' lying between r and q on c. This implies that $right(v)$ lies before $right(p)$ on W; see Figure 8. □

The lemma tells us, together with the previous results, that in order to compute the set of vision points, we only have to compute the points $right(v)$ for the vertices of **P**.

We show how the first vision point can be obtained with the use of the forward essential cuts with respect to g_s.

LEMMA 4.6. *The first vision point g_1 is the first intersection point of W and a forward essential cut with respect to g_s, i.e.,*

$$g_1 = nearest_{g_s}(\{W \cap c \mid c \in \mathcal{FEC}\}),$$

where $\mathcal{FEC}$ is the set of forward essential cuts with respect to g_s.

PROOF. By Lemmas 4.3 and 4.5, it is enough to determine the closest point $right(v)$ to g_s of the vertices of **P**.

Next, we can reduce the set of vertices to consider by the following argument. Let v be a vertex of **P** and assume that $right(v)$ is a point on W such that the segment $[v, right(v)]$ is not collinear to a forward essential cut with respect to g_s; see Figure 9. The segment $[v, right(v)]$ intersects a vertex v' of **P**. Consider the boundary edge $[v', v'']$ that lies in a pocket of $\mathbf{VP}(right(v))$. The point $right(v'')$ lies before $right(v)$ and it lies on the forward essential cut with respect to g_s that issues from the edge $[v', v'']$; see Figure 9. □

In the preprocessing of Step 2 of algorithm *Optimum-Vision-Points-in-Street*, all the forward essential cuts with respect to g_s are computed. The computation is done with at most a linear number of $O(\log(n+m))$ time ray shooting operations with a data structure that can be precomputed in $O(n+m)$ time [5], [6], [13], [16]. To do the preprocessing, extend the edge of W containing point g_s until it hits the boundary of **P** at point v_s. The

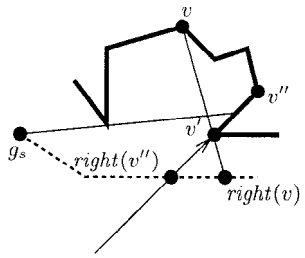

Fig. 9. Illustrating Lemma 4.6.

set $W \cup [g_s, v_s]$ can be viewed as a very thin corridor belonging to the exterior of **P**. The polygon constructed in this manner consists of at most $n + 2m + 1$ edges and we build the ray shooting data structure on this polygon, denoted P'.

The intersection points of the forward essential cuts with respect to g_s and W are computed by performing ray shooting in P' from the vertices in the direction of the adjacent boundary edges to get the intersection point with W, each taking $O(\log(n+m))$ time. From the intersection points thus found, we establish the one closest to g_s. The initial vision point g_1 is then set to be this point on W. Furthermore, we keep the rest of the intersection points, and their associated forward essential cuts with respect to g_s, in a list denoted $\mathcal{FCL}_0$, to be used to compute the rest of the vision points.

The preprocessing of Step 2 also consists of the computation of the shortest path tree rooted at g_t [13], [23]. The tree is extended to an *augmented shortest path tree* by traversing the boundary of **P**, and for each edge e perform the following steps: The two endpoints of e correspond to nodes v and v' in the shortest path tree. We can, in linear time, construct a data structure [14] enabling us to find the nearest common ancestor u to v and v'. It is clear, by the fact that the path from u to v is a shortest path and similarly that the path from u to v' is a shortest path, that the two paths are reflex chains. Hence, we can extend each edge of the two paths and introduce an intersection point on e. In the tree we let this intersection point correspond to a node connected to the node of the extended edge; see Figure 10. The points of the polygon corresponding to nodes of the augmented shortest path tree will henceforth be denoted *asp-vertices* to distinguish them from the vertices of **P**. Note that the set of vertices is a subset of the set of asp-vertices, and that the number of asp-vertices is $O(n)$ since we add at most one new asp-vertex for each vertex of **P**. We use the augmented shortest path tree to guide ray shooting operations to compute limit points.

To compute the other vision points we also need the concept of supporting cut with which we can prove the following two lemmas.

LEMMA 4.7. *A forward essential cut with respect to g_s intersects W in at most one point and a supporting cut with respect to a window of* $\mathbf{VP}(W(g_s, g))$ *intersects W in at most two points, with g being some point on W.*

PROOF. Assume that e is a forward essential cut with respect to g_s and that e intersects W in two or more points. This means that there is a point p on the edge associated to e

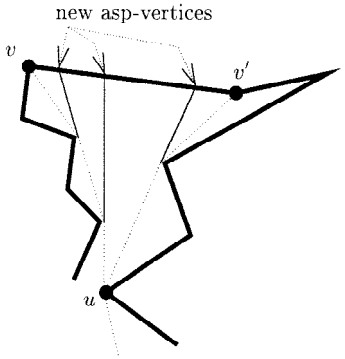

Fig. 10. Constructing the augmented shortest path tree.

such that the set $W \cap \mathbf{VP}(p)$ is not connected, contradicting the fact that W is a collapsed watchman route.

Similarly, if we assume that c is a supporting cut with respect to a window of $\mathbf{VP}(W(g_s, g))$ that intersects W in three or more points, then the set $W \cap \mathbf{VP}(p)$, where p is the starting point of c, is not connected, once again contradicting the fact that W is a collapsed watchman route. That c can have two intersection points can be seen in Figure 11(b). □

The intersection point of a supporting cut and the route W that lies furthest from g_s is called the *true* intersection point.

LEMMA 4.8. *Each vision point g_{i+1}, with $i \geq 1$, is either the first intersection point between W and a forward essential cut with respect to g_s issuing from an edge of* **P** *that is not completely in* $\mathbf{VP}(W(g_s, g_i))$*, or it is the first true intersection point between W and a supporting cut associated to some window of* $\mathbf{VP}(W(g_s, g_i))$*, i.e.,*

$$g_{i+1} = nearest_{g_s}(\{W \cap c \mid c \in \mathcal{FEC}_i \cup \mathcal{SUC}_i\}),$$

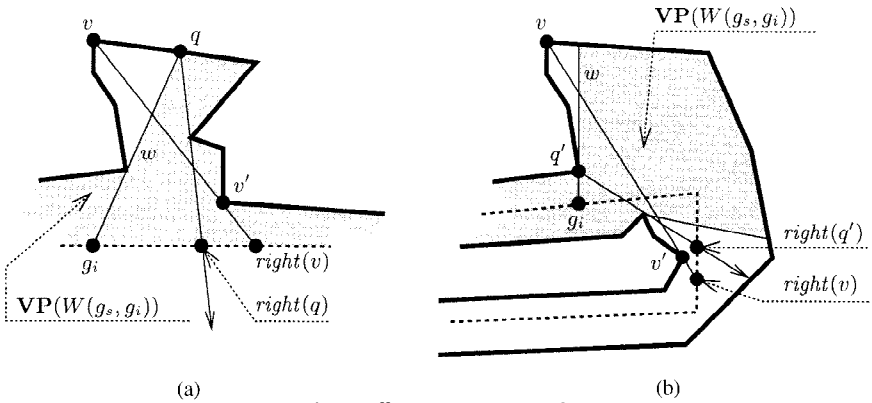

Fig. 11. Illustrating Lemma 4.8.

where $\mathcal{FEC}_i$ is the set of forward essential cuts with respect to point g_s in the region closure($\mathbf{P}\backslash\mathbf{VP}(W(g_s, g_i))$) possibly consisting of several components, and $\mathcal{SUC}_i$ is the set of supporting cuts with respect to the windows of $\mathbf{VP}(W(g_s, g_i))$.

PROOF. By Definition 4.1 and Lemma 4.5 it is enough to determine the closest point *right*(v) to g_s of the vertices in the region *closure*($\mathbf{P}\backslash\mathbf{VP}(W(g_s, g_i))$).

Let v be a vertex in the region *closure*($\mathbf{P}\backslash\mathbf{VP}(W(g_s, g_i))$) and assume that the segment [v, *right*(v)] is not collinear to any forward essential cut in $\mathcal{FEC}_i$ or any supporting cut in $\mathcal{SUC}_i$. The segment [v, *right*(v)] intersects a vertex v' of **P**; see Figure 11. To complete the proof we have to distinguish between three different cases:

1. If the segment [v, v'] intersects $\mathbf{VP}(W(g_s, g_i))$ and v' lies on the same boundary chain as v, then v lies in a pocket of $\mathbf{VP}(W(g_s, g_i))$ having the window w. Let q be the convex vertex of $\mathbf{VP}(W(g_s, g_i))$ on w. The segment [q, *right*(q)] must intersect the segment [v, *right*(v)] in the interval between v and v', and, hence, *right*(q) lies before *right*(v) and the segment [q, *right*(q)] is collinear to one of the supporting cuts associated to w; see Figure 11(a).
2. If the segment [v, v'] intersects $\mathbf{VP}(W(g_s, g_i))$ but v' lies on the other boundary chain, then v once again lies in a pocket of $\mathbf{VP}(W(g_s, g_i))$, having window w. We let q' be the reflex vertex of $\mathbf{VP}(W(g_s, g_i))$ on w. The segment [q', *right*(q')] must intersect the segment [v, *right*(v)] in the interval between v and v', and, hence, *right*(q') lies before *right*(v) and the segment [q', *right*(q')] is collinear to one of the supporting cuts associated to w; see Figure 11(b).
3. If the segment [v, v'] lies completely in *closure*($\mathbf{P}\backslash\mathbf{VP}(W(g_s, g_i))$), we apply the exact same argument as in the proof of Lemma 4.6 to show that there is a vertex v'' such that *right*(v'') lies before *right*(v) and the segment [v'', *right*(v'')] is collinear to some forward essential cut in $\mathcal{FEC}_i$.

This concludes the proof. □

The loop of Step 3 is performed k times. Each time the limit point of the previous vision point is computed in $O(n\log(n+m))$ time as follows: Let g_i be the previously computed vision point. Compute $\mathbf{VP}g_i$ in $O(n)$ time [10], [18], [21]. For each pocket in $\mathbf{VP}(W(g_s, g_i))$ identify the two endpoints q and q' of the associated window. Let q be the convex vertex of $\mathbf{VP}(W(g_s, g_i))$ on the window and let q' be the other endpoint; see Figure 11.

We handle each of the three cases of Lemma 4.8 separately.

1. Given the point q, we determine the two asp-vertices v and v' that lie on either side of q. Furthermore, we determine the nearest common ancestor u of v and v' in the augmented shortest path tree using the data structure computed previously. Now, perform ray shooting from q in the direction of u in P', and if the shot reaches W, remember the intersection point as *right*(q).
2. We know that point q' is a vertex of **P**, and, hence, that it is an asp-vertex of the augmented shortest path tree. Let u be the father of q' in the tree. We perform ray shooting from q' in the direction of u in P', and if the shot reaches W, we perform a second ray shooting operation from the intersection with W in the same direction

to get the possible second intersection point with W; as in Figure 11(b). The last of these two is remembered as $right(q')$.

3. Finally, we have to compute the set of current intersection points between W and the forward essential cuts with respect to g_s. We can assume that we have the list $\mathcal{FCL}_{i-1}$ of intersection points, and we show how to compute the updated list $\mathcal{FCL}_i$. Note that we have shown how to compute the list $\mathcal{FCL}_0$ previously. Identify the polygon edges that are completely seen by the points $g_1, \ldots, g_i$, i.e., the edges of $\mathbf{VP}(W(g_s, g_i))$, and remove the intersection points in $\mathcal{FCL}_{i-1}$ of the forward essential cuts associated to these edges to get the current list of intersection points $\mathcal{FCL}_i$.

To get the limit point, take the true intersection point of the essential cuts and the supporting cuts with W closest to g_i on W.

To analyze the complexity, we note that we compute the visibility polygon for g_i once, taking linear time, at most $O(n)$ ray shooting operations are performed, i.e, three per window, each taking $O(\log(n+m))$ time. Hence, the total time to compute each vision point is bounded by $O(n\log(n+m))$. The total time complexity of the algorithm is $O((n+m)+kn\log(n+m)) = O(kn\log(n+m)+m)$, and the storage use is $O(n+m+k)$, since we only use linear-sized data structures.

Thus, we have the following theorem.

THEOREM 2. *The Optimum-Vision-Points-in-Street algorithm computes an optimum set of vision points on a collapsed watchman route W in a street* $\mathbf{P}$. *The algorithm uses $O(kn\log(n+m)+m)$ time and $O(n+m+k)$ storage, where k is the size of the optimum solution, n is the size of* $\mathbf{P}$, *and m is the size of W.*

5. Optimum Vision Points in Straight Walkable Polygons. An interesting subclass of street polygons is the class of straight walkable polygons. Informally, a polygon is straight walkable if it is possible to move two distinct points along U and D, continuously from s to t, in such a way that the two points always see each other, and neither of the points needs to backtrack its path along the boundary chain. This class of polygons encompasses spiral and monotone polygons.

DEFINITION 5.1 [17]. Let U and D be a partitioning of the boundary of a polygon having the two endpoints s and t. A *straight walk* of the polygon is a pair of continuous monotone functions $(\mathcal{U}, \mathcal{D})$ such that

1. $\mathcal{U} : [0,1] \to U$ and $\mathcal{D} : [0,1] \to D$,
2. $\mathcal{U}(0) = \mathcal{D}(0) = s$ and $\mathcal{U}(1) = \mathcal{D}(1) = t$, and
3. $\mathcal{U}(x)$ sees $\mathcal{D}(x)$, for all $0 \le x \le 1$.

A polygon is *straight walkable* if it admits a straight walk.

The domain of the two functions $\mathcal{U}$ and $\mathcal{D}$ can be chosen arbitrarily. We select the set $[0,1]$ to conform with Icking's and Klein's definition [17].

We construct an optimal $O(n+m)$ time algorithm to compute an optimum set of vision points on a watchman route inside a straight walkable polygon. The algorithm we

present is a version of the *Optimum-Vision-Points-in-Street* algorithm and the reduced complexity comes from the fact that in each step of the algorithm we only need local information, i.e., given the position of the ith guard, we only need to look in a small neighborhood of this guard to be able to compute the positioning of the $i+1$st guard.

To achieve the linear time bound, we have to ensure that the supporting cuts we compute as the algorithm proceeds, intersect W at most once. From Lemma 4.7, we know that a collapsed watchman route can intersect a forward essential cut with respect to g_s in at most one point, but a supporting cut can intersect the route in two points. To disallow this, we place a further restriction on the structure of a collapsed watchman route W, and require that the intersection of W and any segment $[p, q]$ in **P**, with p on U, q on D, and no other point intersecting the boundary of **P**, is a connected set. A collapsed watchman route that obeys this additional requirement is said to be a *straight watchman route*. One instance of a straight watchman route is the shortest path in **P** that connects the points s and t. Other examples are the two boundary chains U and D. For the rest of this section, we assume that W is a straight watchman route.

Icking and Klein [17] present an $O(n \log n)$ time algorithm that computes a straight walk given the points s and t of a straight walkable polygon with n edges. Heffernan [15] improves the time bound to linear, which is optimal.

The straight walk $(\mathcal{U}, \mathcal{D})$ computed in the algorithm by Heffernan [15] consists of $l = O(n)$ pairs of piecewise linear functions, hence, the straight walk can be represented by a list of pairs of linear functions as

$$(\mathcal{U}, \mathcal{D}) = \begin{cases} (\mathcal{U}_1, \mathcal{D}_1) & \text{for} \quad x_0 \le x < x_1, \\ (\mathcal{U}_2, \mathcal{D}_2) & \text{for} \quad x_1 \le x < x_2, \\ \vdots & \\ (\mathcal{U}_{l-1}, \mathcal{D}_{l-1}) & \text{for} \quad x_{l-2} \le x < x_{l-1}, \\ (\mathcal{U}_l, \mathcal{D}_l) & \text{for} \quad x_{l-1} \le x \le x_l, \end{cases}$$

with $x_0 = 0$ and $x_l = 1$.

We present a version of the algorithm examined in the previous section that we call *Optimum-Vision-Points-in-Straight-Walkable-Polygon*, and analyze its complexity. The pseudocode of the algorithm is displayed below.

Algorithm *Optimum-Vision-Points-in-Straight-Walkable-Polygon*

Input: A straight walkable polygon **P** represented by (U, D),
a straight walk $(\mathcal{U}, \mathcal{D})$, and a straight watchman route W
Output: An optimum set of vision points for **P** on W

1 Preprocess **P**, W, and $(\mathcal{U}, \mathcal{D})$ and position g_1 on W
$k := 1$
2 **while P** is not completely guarded **do**
$k := k + 1$
2.1 $g_k := lp(g_{k-1})$
endwhile
return $\{g_i \mid 1 \le i \le k\}$
End *Optimum-Vision-Points-in-Straight-Walkable-Polygon*

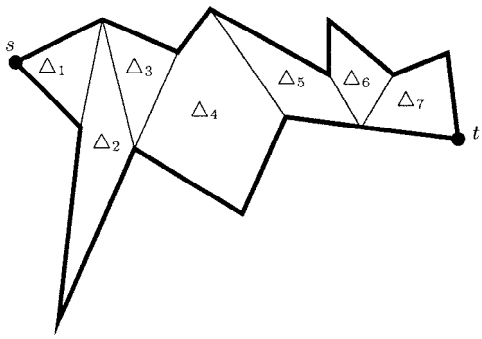

Fig. 12. A straight walkable polygon with its pyramids.

We use the straight walk given as input to guide the incremental computation of the vision points.

It remains to show how to compute the first vision point g_1, the limit point of a point on W, how to establish that the polygon is completely guarded, and to show the time complexity of the algorithm.

The preprocessing of Step 1 in the algorithm consists of the computation of the shortest path tree rooted at g_t [13], [23], exactly as in the algorithm of the previous section. In the same way, the tree is extended to an *augmented shortest path tree* that we use to guide implicit ray shooting operations to compute the vision points.

We also make a subdivision of **P** into so-called *pyramids*.[3] A pyramid is a polygonal region in **P** bounded by a subchain of U, a subchain of D, and two walk segments $[\mathcal{U}(x), \mathcal{D}(x)]$ and $[\mathcal{U}(x'), \mathcal{D}(x')]$. The construction of the pyramids is done by inserting cuts that are collinear to the walk segments, into **P**, separating the different pyramids. Consider a reflex vertex v of **P** other than s and t. Let $c = [\mathcal{U}(x), \mathcal{D}(x)]$ be a segment of the straight walk such that $v = \mathcal{U}(x)$ or $v = \mathcal{D}(x)$. We let c be a separating cut between two pyramids. From the monotonicity of the straight walk, it follows that no two separating cuts intersect and therefore that the pyramidal regions are consecutive and nonintersecting. Hence, we can order the pyramids from the "leftmost" pyramid containing s to the "rightmost" one containing t and we can enumerate the pyramids in this order; see Figure 12.

Consider some pyramid of **P**. It is bounded by two cuts from the straight walk of **P** and two chains of edges of **P** where the vertices are convex. Icking and Klein [17] show that a straight walk has certain nice properties. One of them is that the angle between a walk segment and the polygon boundary is never more than 180°. It therefore follows that the pyramids are all convex regions.

To place the initial vision point g_1, we walk along W from g_s until we find the first intersection with a forward essential cut with respect to g_s. Let e be the last upper boundary edge of the first pyramid Δ_1, extend the edge e, and compute the intersection

[3] The name pyramid is somewhat misleading, but is used due to the lack of a better name.

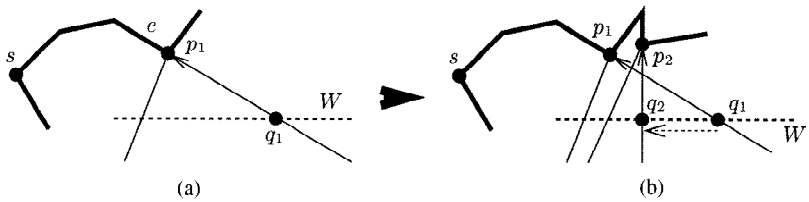

Fig. 13. Computing the first vision point g_1.

with W, if it exists. Otherwise, continue with the next pyramid. This process gives us the intersection of W and the first forward essential cut with respect to g_s. Let p_1 be a point on e and let q_1 be the intersection of the extended edge with W; see Figure 13(a). The point q_1 lies in some pyramid Δ_{j_1} of $\mathbf{P}$. Now, find the last upper boundary edge of the next pyramid Δ_2, in order, and construct the line extending from the last edge. If this line intersects the segment $[p_1, q_1]$, then the intersection point of this essential cut and W will lie closer to g_s, and, hence, we backtrack along W to get the currently closest intersection point, i.e., the new point q_2, which together with a point p_2 of the last upper edge of the current pyramid gives us a new segment $[p_2, q_2]$; see Figure 13(b). The point q_2 lies in some pyramid Δ_{j_2}, with $j_2 \leq j_1$. We continue the process of repeatedly computing new intersection points $q_3, q_4, \ldots$ with W, that each lie in $\Delta_{j_3}, \Delta_{j_4}, \ldots$, until the last computed q_i lies in the current pyramid Δ_{j_i}.

We repeat the steps for the lower boundary D, choose the intersection point closest to g_s, and position the first vision point g_1 at this point.

Next, we show how to compute the limit point $lp(g_i)$ incrementally, given the vision point g_i. Assume that g_i lies in the pyramid Δ_{j_i} bounded by the cuts $[\mathcal{U}(x_{j_i-1}), \mathcal{D}(x_{j_i-1})]$ and $[\mathcal{U}(x_{j_i}), \mathcal{D}(x_{j_i})]$ of the straight walk, with $x_{j_i-1} < x_{j_i}$. By Lemma 4.8, our aim is to compute the closest intersection point between W and the forward essential cuts with respect to g_s that start in the pockets of $\mathbf{VP}(W(g_s, g_i))$ and the closest intersection point between W and the supporting cuts with respect to the windows of $\mathbf{VP}(W(g_s, g_i))$.

In Lemma 4.8 we introduced the two sets $\mathcal{FEC}_i$, of forward essential cuts with respect to g_s that have issuing edges in $closure(\mathbf{P}\backslash\mathbf{VP}(W(g_s, g_i)))$, and $\mathcal{SUC}_i$, of supporting cuts of windows of $\mathbf{VP}(W(g_s, g_i))$. We partition each of these sets into two new sets $\mathcal{FEC}_i^U$ and $\mathcal{FEC}_i^D$, together with $\mathcal{SUC}_i^U$ and $\mathcal{SUC}_i^D$. The set $\mathcal{FEC}_i^U$ consists of those forward essential cuts in $\mathcal{FEC}_i$ that issue from the chain U. Similarly the set $\mathcal{FEC}_i^D$ consists of those issuing from D. We partition the set $\mathcal{SUC}_i$ in the same way.

If c is a cut in $\mathbf{P}$, we let l_c denote the directed line collinear to c having the same direction as c. Let $\mathbf{lHP}(l)$ and $\mathbf{rHP}(l)$ denote the left half-plane and the right half-plane respectively of the directed line l. If $\mathbf{Q}$ is some planar region, we let $bd(\mathbf{Q})$ denote the boundary of $\mathbf{Q}$. With these notational conventions we can define two curves that are important to compute limit points efficiently. Define the curves C_i^U and C_i^D by

$$C_i^U = bd\left(\bigcap_{c\in\mathcal{FEC}_i^U} \mathbf{lHP}(l_c) \cap \bigcap_{c\in\mathcal{SUC}_i^U} \mathbf{rHP}(l_c)\right),$$

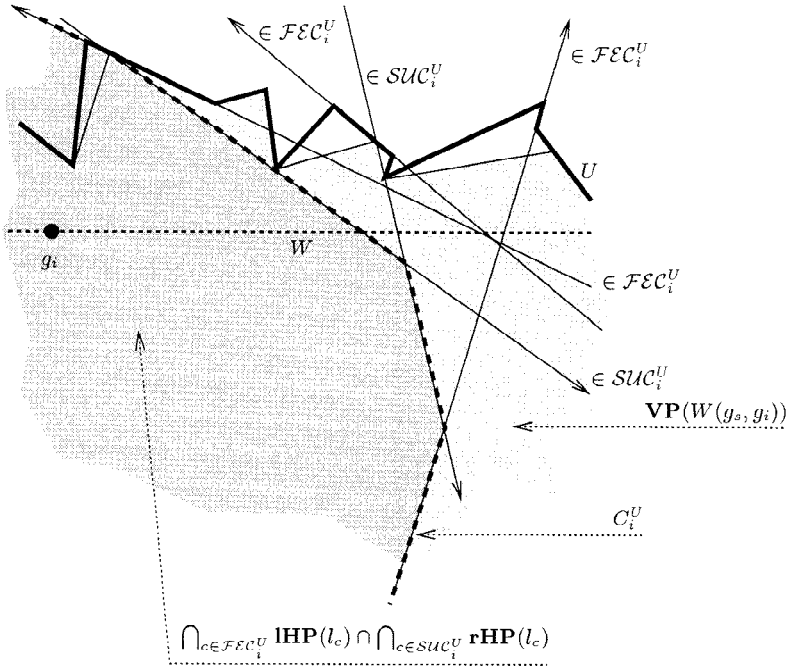

Fig. 14. An example of the curve C_i^U.

$$C_i^D = bd\left(\bigcap_{c\in\mathcal{FEC}_i^D} \mathbf{lHP}(l_c) \cap \bigcap_{c\in\mathcal{SUC}_i^D} \mathbf{lHP}(l_c)\right).$$

The two curves C_i^U and C_i^D can equivalently be viewed as the lower envelope of the cuts in $\mathcal{FEC}_i^U \cup \mathcal{SUC}_i^U$ and the cuts in $\mathcal{FEC}_i^D \cup \mathcal{SUC}_i^D$, respectively; see Figure 14.

By Lemma 4.8 we have that

$$g_{i+1} = nearest_{g_s}(\{W \cap C_i^U, W \cap C_i^D\}).$$

We show how to compute the limit points incrementally, using the curves C_i^U and C_i^D.

Our assumption is, as stated previously, that g_i lies in Δ_{j_i}, and we consider the pyramids $\Delta_{j_i+1}, \Delta_{j_i+2}, \ldots$, in order until we find the limit point in $\Delta_{j_{i+1}}$. Note that $\Delta_{j_{i+1}}$ is not known at the beginning of the computation. We will show how to perform the computation along the upper boundary of each pyramid, i.e., how to find the intersection $C_i^U \cap W \cap \Delta_j$, with $j_i + 1 \leq j \leq j_{i+1}$. The computation along the lower boundary is performed in a similar fashion, simultaneously. The pseudocode of the routine computing the limit point is provided below.

Algorithm *Limit-Point*

Input: A point $g_i \in W$ and an integer j_i such that $g_i \in \Delta_{j_i}$
Output: The point $g_{i+1} = lp(g_i)$ and an integer j_{i+1} such that $g_{i+1} \in \Delta_{j_{i+1}}$

1 $j := j_i$, *done* := **false**, $C_i^U := \emptyset$, $C_i^D := \emptyset$, $vhl^U := \emptyset$, $vhl^D := \emptyset$
2 **while** not *done* **do**
$j := j + 1$
2.1 **if** $vhl^U = \emptyset$ **then**
Let v be the first vertex of U in Δ_j and v' the second vertex of U in Δ_j
if g_i, v, and v' form a left turn **then**
Let vhl^U be the half-line from g_i through v
Compute the half-line shl_1^U from v
Merge shl_1^U to C_i^U
endif
endif
2.2 **if** vhl^U intersects U in Δ_j at p **then**
Compute the half-line shl_2^U from p
Merge shl_2^U to C_i^U
$vhl^U := \emptyset$
endif
2.3 **if** $vhl^U \neq \emptyset$ **then**
Let e be the last edge of U in Δ_j and let hl be the half-line
collinear to e directed toward the interior of **P**
Merge hl to C_i^U
endif
2.4 **if** C_i^U intersects W in Δ_j **then**
Compute the intersection point q_U
endif
2.5 Perform steps corresponding to Steps 2.1–2.4 for the lower boundary D
2.6 **if** at least one of q_U and q_D exists **then**
Let g_{i+1} be the one of q_U and q_D closest to g_i
Let $j_{i+1} := j$
done := **true**
endif
endwhile
3 **return** g_{i+1} and j_{i+1}
End *Limit-Point*

In Step 1 we make initializations of variables we will use. Step 2 is a loop that is traversed for each pyramid after Δ_{j_i} until the limit point is found in $\Delta_{j_{i+1}}$. Step 2.1 tests if there is a window of $\mathbf{VP}(W(g_s, g_i))$ starting at the first vertex v of U, in the current pyramid; see Figure 15(a). If this is so, we let vhl^U be the visibility half-line, i.e., the half-line starting at g_i and passing through v. We also introduce the first supporting half-line shl_1^U collinear to the supporting cut issuing from v, and merge it to the chain C_i^U. We describe how to perform the merging step later. Since v is a vertex of **P**, it corresponds to a node in the augmented shortest path tree rooted at g_t, with v'' being the father node of v in the tree. The half-line shl_1^U is the half line starting at v directed toward v''; see Figure 15(b).

In Step 2.2 we test whether an existing visibility half-line has a further intersection with the boundary of U in the current pyramid. If this is so, we compute the second

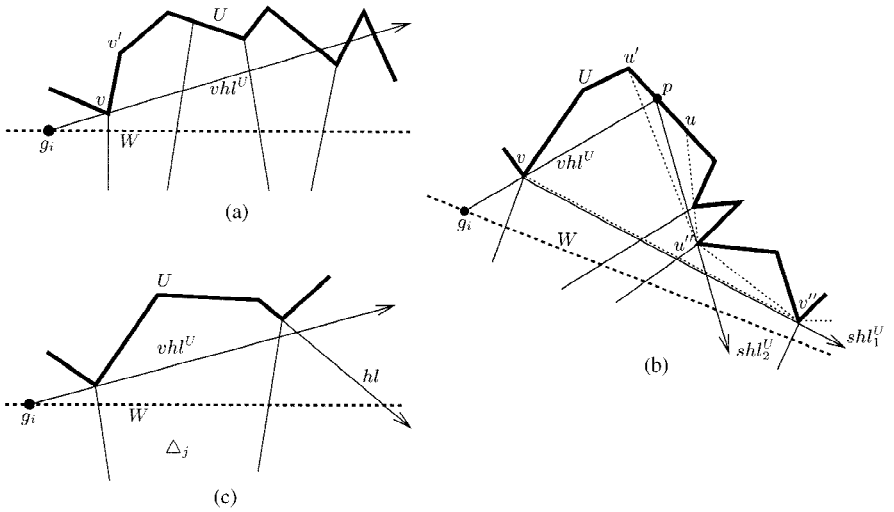

Fig. 15. Illustrating the computation of the limit point of g_i.

supporting half-line shl_2^U collinear with the other supporting cut and merge shl_2^U to the chain C_i^U. The half-line shl_2^U is computed in the following way: Let u and u' be the two asp-vertices closest to p on either side of p, where p is the intersection point of vhl^U and U; see Figure 15(b). We can obtain the nearest common ancestor u'' of u and u' in constant time [14]. The half-line shl_2^U is the half-line starting at p, directed toward u''.

In Step 2.3 we test whether there are forward essential cuts with respect to g_s that start in a pocket of $\mathbf{VP}(W(g_s, g_i))$. Since all vertices of U in the current pyramid are convex, with the possible exception of the two end vertices, our only interest lies in the last edge e of U in the current pyramid; see Figure 15(c). The half-line hl is the half-line collinear to e starting at the end vertex of U in this pyramid and directed so that g_i lies to the right of hl so as to correspond with the direction of the supporting cuts described above. Note that a half-line hl issuing from U will have opposite direction with respect to the associated forward essential cut.

Each half-line is merged with the chain $C_i^U = \langle c_1, c_2, \ldots, c_{k-1}, c_k \rangle$ where $c_1, \ldots, c_{k-1}$ are line segments and c_k is a half-line, using the algorithm *Merge-to-Upper-Chain*.

Algorithm *Merge-to-Upper-Chain*

Input: The chain C_i^U and a half-line hl to be merged with C_i^U under certain conditions
Output: The new upper chain C_i^U

1 **if** C_i^U is empty **then**
  $C_i^U := \langle hl \rangle$
  endif
2 **if** $C_i^U = \langle c_1, c_2, \ldots, c_{k-1}, c_k \rangle$ **then**
  set $k' := k$ and set *done* := **false**

2.1 **while not** (*done*) **do**
Let l be the line collinear to $c_{k'}$ directed so that g_i lies to the right of l
if hl has points to the right of l **then**
let l' be the line collinear to hl
if l' intersects $c_{k'}$ at the point p **then**
if $c_{k'} = [a, b]$ or $c_{k'}$ is a half-line starting at a **then**
let hl' be the half-line starting at p with the same direction as hl and
set $C_i^U := \langle c_1, \ldots, c_{k'-1}, [a, p], hl' \rangle$
endif
else /* **if** l' does not intersect $c_{k'}$ **then** */
$k' := k' - 1$
endif
else /* **if** hl has all its points to the left of l **then** */
done := **true**
endif
endwhile
endif
return C_i^U
End *Merge-to-Upper-Chain*

The straight walk $(\mathcal{U}, \mathcal{D})$ specifies one parameter partial functions $\mathcal{W}$, $\mathcal{C}_i^U$, and $\mathcal{C}_i^D$ defined by

$$\begin{array}{lll} \mathcal{W}\colon [0,1] \to W & \text{such that} & \mathcal{W}(x) = W \cap [\mathcal{U}(x), \mathcal{D}(x)], \\ \mathcal{C}_i^U\colon [0,1] \to C_i^U & \text{such that} & \mathcal{C}_i^U(x) = C_i^U \cap [\mathcal{U}(x), \mathcal{D}(x)], \\ \mathcal{C}_i^D\colon [0,1] \to C_i^D & \text{such that} & \mathcal{C}_i^D(x) = C_i^D \cap [\mathcal{U}(x), \mathcal{D}(x)]. \end{array}$$

It follows from the fact that W is polygonal, straight, and has m edges, together with the fact that $\mathcal{U}$ and $\mathcal{D}$ are piecewise linear and monotone, that $\mathcal{W}$ is monotone and consists of $O(n+m)$ functions of the form $\mathcal{A}(x)/\mathcal{B}(x)$, where $\mathcal{A}$ is a quadratic function and $\mathcal{B}$ is a linear function of subintervals to [0, 1]. Similarly, each function $\mathcal{C}_i^U$ and $\mathcal{C}_i^D$ consists of $O(n)$ functions of the same type.

In Step 2.4 we compute the possible intersection point of C_i^U and W in the current pyramid. If the current pyramid is Δ_j we begin by computing the subsets $C_i^U \cap \Delta_j$ and $W \cap \Delta_j$. We have that $\Delta_j = \bigcup_{x_{j-1} \le x \le x_j} [\mathcal{U}(x), \mathcal{D}(x)]$ and therefore to find $C_i^U \cap \Delta_j$ we only have to find the intersection points $\mathcal{C}_i^U(x_{j-1}) = C_i^U \cap [\mathcal{U}(x_{j-1}), \mathcal{D}(x_{j-1})]$ and $\mathcal{C}_i^U(x_j) = C_i^U \cap [\mathcal{U}(x_j), \mathcal{D}(x_j)]$, for $\mathcal{C}_i^U$. Similarly for $W \cap \Delta_j$, we have to find the intersection points $\mathcal{W}(x_{j-1}) = W \cap [\mathcal{U}(x_{j-1}), \mathcal{D}(x_{j-1})]$ and $\mathcal{W}(x_j) = W \cap [\mathcal{U}(x_j), \mathcal{D}(x_j)]$, for $\mathcal{W}$. To get the intersection point between C_i^U and W in the pyramid, we solve the equation $\mathcal{W}(x) = \mathcal{C}_i^U(x)$, for x, which produces the point $q_U = \mathcal{W}(x)$. The time to perform the operations is linear in the size of the pyramid, the parts of C_i^U and W in the pyramid.

In Step 2.5 we perform these same operations for the lower boundary of the current pyramid, i.e., computing the set $C_i^D \cap W \cap \Delta_j$, for $j_{i+1} \le j \le j_i + 1$. We determine the half-lines vhl^D, shl_1^D, shl_2^D, and the half-lines corresponding to forward essential cuts, to

get the chain C_i^D, and compute the possible intersection point q_D of C_i^D and W in the current pyramid.

The last step of the loop, Step 2.6, determines if the limit point has been found, and, if so, exits the procedure.

LEMMA 5.1. *A watchman route in a straight walkable polygon needs at most n vision points.*

PROOF. The algorithm positions the first vision point g_1 in pyramid Δ_{j_1} which means that g_1 sees all the pyramids $\Delta_1, \ldots, \Delta_{j_1}$ and symmetrically the last vision point g_k is placed in Δ_{j_k} implying that the last pyramids $\Delta_{j_k}, \ldots, \Delta_l$ are seen by g_k. Furthermore, we know that each of the pyramids $\Delta_{j_1+1}, \ldots, \Delta_{j_k-1}$ contains points of the watchman route, and, hence, one vision point in each pyramid will suffice to guard the whole polygon because each pyramid is convex. Since there are no more than n pyramids, no more than n vision points are needed. □

THEOREM 3. *The algorithm Optimum-Vision-Points-in-Straight-Walkable-Polygon computes an optimum set of vision points on a straight watchman route W, in a straight walkable polygon* **P**. *The algorithm uses $O(n+m)$ time and storage, where n is the size of* **P** *and m is the size of W.*

PROOF. The correctness of the algorithm follows from the construction and Lemma 4.4, so it only remains to analyze the complexity.

Step 1 takes $O(n+m)$ time since we can compute the augmented shortest path tree from a point in $O(n)$ time. The subdivision into pyramids also takes $O(n)$ time, if the two sets $\mathcal{U}$ and $\mathcal{D}$, each consisting of $O(n)$ linear functions, are scanned from s to t. Computing g_1 and $\mathcal{W}$ takes $O(n+m)$ time since U, D, and W are each scanned a constant number of times.

To show that Step 2 takes $O(n+m)$ time, we show that, if a vision point g_i lies in Δ_{j_i} and its limit point g_{i+1} lies in $\Delta_{j_{i+1}}$, the time to compute g_{i+1} is bounded by $O(\sum_{j_i \le j \le j_{i+1}} |\Delta_j| + |W \cap \Delta_j|)$, but this follows easily since only the boundary of the pyramids $\Delta_{j_i}, \Delta_{j_i+1}, \ldots, \Delta_{j_{i+1}}$ and the portion of W in these pyramids are traversed a constant number of times during the construction. Since all of these pyramids are seen by the vision point together with its limit point we do not have to consider any of these pyramids again, but can continue the computation from $\Delta_{j_{i+1}}$. Hence, the total time for the construction is linear in the size of the input. □

6. Conclusion

6.1. *Remarks.* We have shown a method with which we can compute an optimum set of vision points on a given collapsed watchman route in a street. We have also presented a faster version of the algorithm that works for straight walkable polygons with straight watchman routes.

The *Optimum-Vision-Points-in-Street* algorithm uses explicit ray shooting to find the

vision points. This makes for heavy time consumption when we compute each vision point, but the algorithm requires only linear time preprocessing. If we use more preprocessing time, the cost of computing the vision points can be reduced. Instead of doing explicit ray shooting operations, we do them implicitly, by first computing a visibility graph structure and using this structure to guide the placement of the vision points. However, this version of the algorithm would only be more efficient when the number of vision points is very large.

In fact, we can use the method to compute optimum guard covers for certain polygons. The method consists of two steps:

1. Show that the given polygon **P** is a street.
2. Show how to obtain a collapsed watchman route W such that some set $\mathcal{OPT}(\mathbf{P}) \subseteq W$.

The algorithm runs in $O(kn \log(n+m) + m) = O(n^2 \log(n+m) + m)$ time since k, the number of guards, is $O(n)$.

If we can show that the polygon is straight walkable and that the watchman route is straight, then our algorithm runs in optimal linear time. This can be done for spiral polygons, where it is easy to see that there is an optimum guard cover on the convex chain of a spiral polygon [24], [25]. We can also find optimum guard covers for histogram and alp polygons in linear time. Here it is easy to establish that the base is a straight watchman route that allows an optimum guard cover, thus yielding the result [4], [24].

6.2. *Open Problems.* In the first part of this work, we discuss the problem of placing static point guards inside polygons. We introduce the concept of one-dimensional guarding, meaning that guards have to be positioned on a given closed curve called the watchman route. These guard points are called vision points. In this setting, we investigate the problem of computing optimum vision points, i.e., positions for a minimum set of vision points. This differs from previous definitions of guarding where the guards are only restricted to lie inside the polygon, and the objective is to find an optimum guard cover.

The algorithm we present for streets is probably not optimal. However, we feel that the important breakthrough is to show that there exist efficient algorithms for the optimum vision points problem in streets. Efficiency in this setting is somewhat undefined. The complexity of the algorithm depends heavily on the size of the output, as described in the beginning of Section 4, which can have arbitrary size. Hence, we can never have an algorithm for the optimum vision points problem that is polynomial in the size of the input only. If we let QP, the set of *Quasi-Polynomial* problems, be the set of problems that have polynomial time complexity in the input and output size, we have shown that the optimum vision points problem for streets lies in QP, and this is the best we can hope for.

Furthermore, a linear time algorithm to solve the problem for straight walkable polygons is presented under the assumption that the watchman route is straight. If the watchman route is not straight, the problem can be solved in polynomial time, since the output size is $O(n)$ and the algorithm for streets also works in this, more restricted, case. We also show that the optimum guard covering problem is NP-hard for the slightly less restricted class of streets. A summary of the complexity of guarding is shown in Table 1.

To indicate further research problems, it would be interesting to narrow the gap

Table 1. Complexity of guarding, for some classes of simple polygons.

	Polygon classes				
Cover type	Spiral	Alp	Str. walk.	Street	Simple
Vision points	Linear	Linear	Linear/P	QP	NP-hard
Guard cover	Linear	Linear	Unknown	NP-hard	NP-hard

between the tractable and intractable guarding problems further. Other polygon classes such as monotone and walkable polygons may yield interesting information that could be useful to show further complexity results.

References

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

[2] S. Carlsson, H. Jonsson. Computing a Shortest Watchman Path in a Simple Polygon in Polynomial Time. In *Proc. Workshop on Algorithms and Data Structures*, *WADS* '95, pages 122–134. Lecture Notes in Computer Science, vol. 955. Springer-Verlag, Berlin, 1995.

[3] S. Carlsson, H. Jonsson, B. J. Nilsson. Finding the Shortest Watchman Route in a Simple Polygon. In *Proc.* 4*th International Symposium on Algorithms and Computation*, *ISAAC* '93, pages 58–67. Lecture Notes in Computer Science, vol. 762. Springer-Verlag, Berlin, 1993.

[4] S. Carlsson, B. J. Nilsson, S. Ntafos. Optimum Guard Covers and m-Watchmen Routes for Restricted Polygons. *International Journal of Computational Geometry and Applications*, 3(1):85–105, 1993.

[5] B. Chazelle, H. Edelsbrunner, M. Grigni, L. Guibas, J. Hershberger, M. Sharir, J. Snoeyink. Ray Shooting in Polygons using Geodesic Triangulations. In *Proc.* 18*th ICALP*, pages 661–673, 1991.

[6] B. Chazelle, L. Guibas. Visibility and Intersection Problems in Plane Geometry. *Discrete and Computational Geometry*, 4:551–581, 1989.

[7] W. Chin, S. Ntafos. Optimum Watchman Routes. *Information Processing Letters*, 28:39–44, 1988.

[8] V. Chvátal. A Combinatorial Theorem in Plane Geometry. *Journal of Combinatorial Theory, Series B*, 13(6):395–398, 1975.

[9] D. Das, P. J. Heffernan, G. Narasimhan. LR-Visibility in Polygons. In *Proc.* 5*th Canadian Conference on Computational Geometry*, pages 303–308, 1993.

[10] H. ElGindy, D. Avis. A Linear Algorithm for Computing the Visibility Polygon from a Point. *Journal of Algorithms*, 2:186–197, 1981.

[11] S. Fisk. A Short Proof of Chvátal's Watchman Theorem. *Journal of Combinatorial Theory, Series B*, 24:374, 1978.

[12] M. R. Garey, D. S. Johnson. *Computers and Intractability*: *A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, CA, 1979.

[13] L. Guibas, J. Hershberger, D. Leven, M. Sharir, R. Tarjan. Linear Time Algorithms for Visibility and Shortest Path Problems inside Triangulated Simple Polygons. *Algorithmica*, 2:209–233, 1987.

[14] D. Harel, R. E. Tarjan. Fast Algorithms for Finding Nearest Common Ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.

[15] P. J. Heffernan. An Optimal Algorithm for the Two-Guard Problem. In *Proc.* 9*th ACM Symposium on Computational Geometry*, pages 348–358, 1993.

[16] J. Hershberger, S. Suri. A Pedestrian Approach to Ray Shooting: Shoot a Ray, Take a Walk. In *Proc. SODA*, pages 54–63, 1993.'

[17] C. Icking, R. Klein. The Two Guards Problem. In *Proc.* 7*th ACM Symposium on Computational Geometry*, pages 166–175, 1991.

[18] B. Joe, R. B. Simpson. Correction to Lee's Visibility Polygon Algorithm. *BIT*, 27:458–473, 1987.
[19] J. M. Keil, J.-R. Sack. Minimum Decompositions of Polygonal Objects. In G. T. Toussaint, editor, *Computational Geometry*, pages 197–216. North-Holland, Amsterdam, 1985.
[20] R. Klein. Walking an Unknown Street with Bounded Detour. *Computational Geometry*: *Theory and Applications*, 1(6):325–351, 1992.
[21] D. T. Lee. Visibility of a Simple Polygon. *Computer Vision, Graphics, and Image Processing*, 22:207–221, 1983.
[22] D. T. Lee, A. K. Lin. Computational Complexity of Art Gallery Problems. *IEEE Transactions on Information Theory*, IT-32:276–282, 1986.
[23] D. T. Lee, F. P. Preparata. Euclidean Shortest Paths in the Presence of Rectilinear Barriers. *Networks*, 14:393–410, 1984.
[24] B. J. Nilsson. Guarding Art Galleries—Methods for Mobile Guards. Ph.D. thesis, Lund University, 1995.
[25] B. J. Nilsson, D. Wood. Watchmen Routes in Spiral Polygons. Technical Report LU-CS-TR:90–55, Dept. of Computer Science, Lund University, 1990. An extended abstract of a preliminary version appears in *Proc. 2nd Canadian Conference on Computational Geometry*, pages 269–272, 1990.
[26] J. O'Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, Oxford, 1987.
[27] T. C. Shermer. Recent Results in Art Galleries. *Proceedings of the IEEE*, pages 1384–1399, September 1992.