GUARDING GALLERIES AND TERRAINS

Alon Efrat
Department of Computer Science, University of Arizona
WWW: `http://www.cs.arizona.edu/people/alon`

Sariel Har-Peled
Department of Computer Science, University of Illinois
WWW: `http://www.cs.uiuc.edu/contacts/faculty/harpeled.html`.

Abstract Let P be a simple polygon with n vertices. We say that two points of P see each other if the line segment connecting them lies inside (the closure of) P. In this paper we present efficient approximation algorithms for finding the smallest set S of points of P so that each point of P is seen by at least one point of S. We also present similar algorithms for terrains and polygons with holes.

1. Introduction

The *art gallery problem* [O'R87] is stated as follows: Given a polygon P (the *gallery*), find a smallest set G of points (*guards*) inside P, such that each point in P is seen by at least one of the guards. This problem has been studied extensively in recent years, see, e.g., [O'R83, Agg84, Gho87, Hof90, HKK91, JL93, BS93, BG95], and the recent survey paper by Urrutia [Urr00].

This problem is known to be NP-hard even when P is simple [OS83], and even finding an $(1+\varepsilon)$-approximation (that is, finding a set of guards whose cardinality is at most $1+\varepsilon$ times the optimum) is NP-hard [Eid00]. Ghosh [Gho87] presented a (multiplicative) $O(\log n)$-approximation algorithm that runs in $O(n^5 \log n)$ time, for the case where guards located on vertices (as well as of other types of visibility). Recently, Gonzalez-Banos and Latombe [GBL01] presented an algorithm for a rather restricted version of the art-galley problem, and with much larger set of guards.

Our contribution We present an algorithm for finding in time $O(nc_{opt}^2 \log^4 n)$ a set of vertices that sees P, and its cardinality is within a factor of $O(\log c_{opt})$ from the optimum.

If one allows guards to be placed arbitrarily (not only on vertices), the problem seems to be considerably harder. We present in Section 4 an exact algorithm for this problem, that runs in $O((nc_{opt})^{3(2c_{opt}+1)})$ time. To the best of our knowledge, this is the first exact solution to the problem. The proof follows

R. Baeza-Yates et al. (eds.), *Foundations of Information Technology in the Era of Network and Mobile Computing*

from recent results in algorithmic real algebraic geometry. Thus if the optimum number of guards is a constant then we obtain a polynomial algorithm.

In Section 5 we present an efficient implementation of our approximating algorithm, for the case where the guards are restricted to lie on an arbitrarily dense grid. For this case, we get an $O(\log c_{opt})$-approximation in $O(nc_{opt}^2 \log n \log(nc_{opt}) \log^2 \Delta)$ time, where Δ is the ratio between the diameter of the polygon and the grid size, and c_{opt} is the cardinality of the smallest set of grid-points that sees P. Note that the running time depends on Δ only logarithmically, which implies that we can choose a rather fine grid without paying too large a penalty, so the resulting set of grid-points is likely to cover all of P.

The new algorithms can be extended to handle polygons with h holes, as their VC-dimension is $O(\log h)$ [Val98], yielding an approximation factor of $O(\log h \log(c_{opt} \log h))$. We also show how to solve related problems on terrains: Given a terrain T, find a small set of vertices that see every point of T. This problem has numerous applications in Geographic Information Science (GIS). Our approximation algorithm can be modified for this setting, yields an $O(\log n \log(c_{opt} \log n)) = O(\log n \log \log n)$ approximation factor. Analogous to the case of a simple polygon, these extensions can be modified to find a set of guards that see the whole polygon or terrain, respectively, where the guards are taken from the set of vertices of an arbitrary dense grid. These extensions are described in Section 6.

Our efficient algorithms are the result of obtaining data structures for carefully counting and maintaining the weights of sets of grid points, as described below.

2. Preliminaries

For a point $q \in P$, the *visibility polygon* of q in P, denoted by $Vis(q)$, is the region of all the points of P that q sees. The following observation appeared in [GMMN90].

Observation 1. Let q be any point in P, and let $s \subseteq P$ be a segment. If P is a simple polygon, then the intersection between s and $Vis(q)$ is a (possibly empty) segment.

Lemma 2. Let $G = \{g_1 \dots g_k\}$ be a set of k points in P, and let Vis_i denote the visibility polygon of g_i, for $i = 1, \dots, k$. Let $Vis(G) = \cup_{q \in G} Vis(q)$. Then the complexity of the arrangement $\mathcal{A} = \mathcal{A}(G)$ formed by $\partial Vis_1, \dots, \partial Vis_k$ is $O(nk^2)$. Furthermore, the complexity of the zone of ∂Vis_i in $\mathcal{A}$ is $O(nk\alpha(k))$, for $i = 1, \dots, k$. Here $\alpha(n)$ is the inverse Ackermann function, and is an extremely slowly growing function.

Theorem 1 ([GMMN90]). *$Vis(G)$ is bounded by $O(nk + k^2)$ edges, and this bound is tight in the worst case.*

Efficient construction of $Vis(G)$**.** The bound of Theorem 1 yields the following simple but efficient divide-and-conquer algorithm for constructing $Vis(G)$.

If $|G| = 1$, one can construct $Vis(G)$, the visibility polygon from a single point, in $O(n)$ time [EA81]. Otherwise, divide G into two subsets G_1, G_2 of roughly $k/2$ guards each. Compute recursively the visibility polygons $Vis(G_1)$ and $Vis(G_2)$, and merge them, using a standard line-sweeping procedure [dBvKOS00] to obtain $Vis(G)$. It is easy to see that the running time of this procedure is $O(nk \log k \log n)$.

3. Finding a Small Set of Vertices that Sees P

Let V be the set of n vertices of P. For a point $q \in P$, let $V_q = V \cap Vis(q)$ denote the set of vertices of P that q sees. Let $\mathcal{X} = (V, \mathcal{V})$ be the range space defined by the visibility inside P, where $\mathcal{V} = \left\{ V_q \;\middle|\; q \in P \right\}$. Valtr [Val98] showed that 23 is a upper bound on the VC-dimension of the more general space $\mathcal{Y} = \left(P, \left\{ Vis(q) \;\middle|\; q \in P \right\} \right)$.

Finding a set of guards on the vertices of P that sees all of P, is equivalent to finding a subset U of the vertices of P that hit all the ranges of $\mathcal{V}$. That is $\forall X \in \mathcal{V}, X \cap U \neq \emptyset$. However, since the VC-dimension of $\mathcal{X}$ is bounded, we can use the property that this space has a small ε-net to get an efficient approximation algorithm (see [Cla93, BG95]). We describe next an efficient implementation of this general method for the case of computing a guarding set, i.e. a set of points that sees P. Assume that we have a guess k of the value of c_{opt}. We initialize the value of k to one. We now call the procedure `ComputeGuards`(P, k), depicted in Figure 1, repeatedly. The procedure `ComputeGuards`(P, k) tries to compute a guarding set of P with $O(k \log k)$ guards. If such a call fails, we know that with high probability, our guess of the number of guards needed to guard P (i.e., k) is too small. Thus, we double its value and iterate. Overall, we would perform $O(\log c_{opt})$ calls to `ComputeGuards`. The correctness of this algorithm, and the values of the constants in the big-O — notations follow from the analysis of Clarkson [Cla93] (see also [BG95], and a slightly different presentation in [EHKKRW02]).

We implement `ComputeGuards` using the algorithm of Section 2 to compute the union, and to pick a point outside it, in $O(nk \log n \log k)$ time. Computing the $Vis(q)$ can be done in linear time, using the algorithm of [EA81].

In each call to `ComputeGuards`, the algorithm performs $O(k \log (n/k))$ iterations. Overall, the running time of the algorithm is thus

$$O\left(\sum_{i=1}^{\log c_{opt}} n(2^i)^2 \log n \log \frac{n}{2^i} \log^2 2^i \right) = O\left(n c_{opt}^2 \log n \log \frac{n}{c_{opt}} \log^2 c_{opt} \right).$$

We conclude:

Theorem 2. *Given a simple polygon P with n vertices, one can compute, in $O(nc_{opt}^2 \log n \log (n/c_{opt}) \log^2 c_{opt})$ expected time, a set S of $O(c_{opt} \log c_{opt})$ ver-*

Procedure `ComputeGuards`(P - simple polygon, k - number of guards)

1 Assign weight 1 to all the elements of V, the set of vertices of P.

2 For $i := 1$ to $O(k \log(n/k))$ do:

(a) Pick randomly a set S of $O(k \log k)$ vertices, by choosing each guard randomly and independently from V, according to the weights of the vertices.

(b) Check if the points of S see all of P, if so, terminate and return S as the set of guards.

(c) Else, find a point $q \in P$ which is not visible from S, and compute Vis_q.

(d) Compute Ω, the sum of weights of vertices in $V \cap Vis(q)$. If $2k\Omega \leq$ the sum of weights of all vertices of P, double the weight of every vertex of $V \cap Vis(q)$.

3 Failure — no solution found.

Figure 1. `ComputeGuards`(P, k) computes with high probability a guarding set of P of $O(k \log k)$ guards, if $k \geq c_{opt}$

tices of V that seems P, where c_{opt} is the cardinality of the minimal set. The quality of approximation is correct with high probability.

Remark: Theorem 2 provides an $O(\log c_{opt})$-approximation to the optimal solution. Previously, only $O(\log n)$-approximation was known [Gho87] in $O(n^5 \log n)$ time. This is especially striking, when one observes that the dependency of the running time of the new algorithm on n is near linear.

4. Exact Algorithm for Fixed Number of Guards

Theorem 3. *A smallest set of guards that can see a given simple polygon P with n edges can be computed in time $O((nk)^{3(2k+1)})$, where k is the size of such an optimal set.*

Proof. This is an easy consequence of known techniques in algorithmic real algebraic geometry. Suppose first that we wish to determine whether there exists a set of k guards that can see the whole of P. This is equivalent to deciding the truth of the following predicate in the first-order theory of the reals:

$$\exists x_1, y_1, x_2, y_2, \ldots, x_k, y_k \forall u, v \mid$$

$$\Big[\mathsf{InP}(u,v) \Longrightarrow (\mathsf{Visib}(x_1, y_1; u, v) \vee \mathsf{Visib}(x_2, y_2; u, v) \vee \cdots \vee \mathsf{Visib}(x_k, y_k; u, v))\Big],$$

where $\mathsf{InP}(u,v)$ is a predicate that is true iff $(u,v) \in P$, and $\mathsf{Visib}(x, y; u, v)$ is a predicate that is true iff (x,y) and (u,v) are visible to each other within

P. Clearly, InP is a Boolean combination of $O(n)$ linear inequalities, whereas $\mathsf{Visib}(x, y; u, v)$ is a Boolean combination of $O(n)$ quadratic inequalities. Hence the whole predicate involves $O(nk)$ polynomials of maximum degree 2, and has only one alternation of quantifiers. Applying the result of [BPR96], deciding the truth of this predicate can be done in time $O((nk)^{3(2k+1)})$. Finding the optimal value of k can then be done by a straightforward unbounded linear search, within asymptotically the same complexity bound. □

5. Unconstrained Locations of Guards

We consider in this section the art gallery problem where the location of the guards inside the polygon is not restricted to vertices. Instead, their location is restricted to lie on a dense grid inside the polygon. Intuitively, if the polygon P is "well-behaved", such a minimum set of guards would be a good approximation (in its cardinality) to the optimal guarding set.

The main idea of our algorithm is that, instead of maintaining the weight of the relevant grid points explicitly, as done in the algorithm for the case of vertices, we exploit the special properties of the grid, and of the weight function defined over the grid points, to maintain those weights implicitly.

Suppose that we are given a simple n-gon P with diameter ≤ 1, a parameter $\varepsilon > 0$, and Γ a grid of square-length ε inside P; that is $\Gamma = P \cap \{(i\varepsilon, j\varepsilon) \mid i, j \in \mathbb{Z}\}$. We present an algorithm that finds a set $G \subseteq \Gamma$ of guards that see all the points of Γ, and its cardinality is $O(c_{opt} \log c_{opt})$ where c_{opt} is the cardinality of a smallest set of vertices of Γ that sees P.

We apply the algorithm of the previous section, with a different scheme for maintaining the weights over the points of Γ, and picking a set of guards in each stage of `ComputeGuards`.

The range space for this problem is defined as follows: Let $\tilde{V}$ denote the set of vertices of P, and the set of vertices of Γ. Let L be the set of lines passing through pairs of vertices of $\tilde{V}$. Let $\mathcal{X}$ be the set of all intersection points of lines of L. Let the range space $\Sigma = (\tilde{V}, \{\tilde{V} \cap \mathit{Vis}(p) | p \in \Gamma\})$. We do not construct Σ explicitly, as it is not necessary. It is not hard to see that $S \subseteq \Gamma$ sees P if and only if S sees $\mathcal{X}$. Assume that $S \subseteq \Gamma$ does not see P. Let K be a connected component of $P \setminus \mathit{Vis}(S)$. Observe that since P is a close set, the edges of the closure of $\mathit{Vis}(S)$ which are not edges of P, are not edges of $\mathit{Vis}(S)$. Thus each vertex of K (which is also a vertex of $\mathcal{X}$) is not seen by any guard of S.

The weights of the points of Γ are maintained by a subdivision $\mathcal{A}_i$ of P, so that the weight $w(f)$ assigned to all the points of Γ inside a face f of $\mathcal{A}_i$ is the same, where i is the current iteration of `ComputeGuards`. We associate with a face f of $\mathcal{A}_i$ the quantities $n(f) = \Gamma \cap f$, namely, the number of grid points of Γ inside f, $w(f)$, which is the weight assigned to each point of $\Gamma \cap f$, and $W(f) = w(f) \cdot n(f)$, which is the overall weight of f. Initially $\mathcal{A}_0$ consists of a single cell, namely all of P. In the i-th iteration of `ComputeGuards`, we pick at random as set S_i of vertices of Γ, according to their weights. This is done by first picking the face f of $\mathcal{A}_{i-1}$ from which a point $g \in S_i$ is to be picked, and then picking g uniformly from $f \cap \Gamma$. Next we compute the polygon $\mathit{Vis}(S_i)$

and check as in Section 4 if it covers P. If $P \neq Vis(S_i)$ we find a vertex q_i of $P \setminus Vis(S_i)$. As mentioned above, $q_i \in \tilde{V}$. We compute the visibility polygon $Vis(q_i)$, computes the total weight Ω of points of $\Gamma \cap Vis(q_i)$ (details described below) and if $2k\Omega \leq W(\Gamma)$, we insert $\partial Vis(q_i)$ into $\mathcal{A}_{i-1}$, splitting some faces of $\mathcal{A}_{i-1}$ and forming a new arrangement $\mathcal{A}_i$. We double (in an implicit fashion) the weight of $\Gamma \cap Vis(q_i)$. In Section 5.1 we explain how to find the number of grid-point of Γ inside a face f, how to split f and and how to pick a grid point at random from $\Gamma \cap f$ uniformly (note that all grid-points of $\Gamma \cap f$ have the same weight). We next explain how to insert $Vis(q_i)$ and maintain the weights of the faces of $\mathcal{A}_i$.

We assume for simplicity of exposition that each face f is a triangle (if not, when we compute f we also compute a triangulation of it, and pick a triangle from this triangulation. This does not effect the overall complexity of the algorithm, and we omit the tedious but straightforward details).

To explain how to efficiently maintain the weights, we need the following lemma, whose proof is postponed to the end of this section.

Lemma 3. Let $S = \{x_i\}_1^n$ be a set of n points on a line, where each point x_i is associated with a weight w_i. There is an augmented search tree $\mathcal{T}$ that supports the following operations in time $O(\log n)$:

insert(x_i, w_i) — insert a new point x_i into S, with an assisted weight w_i.

modify(x_i, w_i, w_i') — change the weight of x_i from w_i to w_i'.

pick — pick a point x_i at random from S, with probability $w_i / \sum_{j=1}^n w_j$.

interval_sum(x, y) — report the sum of weights of the points in $S \cap [x, y]$.

interval_double(x, y) — double the weight of each point of $S \cap [x, y]$.

Let the weight of a cell of $\mathcal{A}_{i-1}$ be the sum of weights of grid points inside this cell. By Theorem 1 the arrangement $\mathcal{A}$ consists of $O(nk^2 + k^2)$ edges, which we call *arrangement-edges*. These edges lie on one of the $O(nk)$ edges of the original polygons $Vis(q_j)$, $(1 \leq j < i)$ which we call *long edges*. We replace long edge e by two copies of e, so that each copy bounds faces of $\mathcal{A}_{i-1}$ only on one of its sides (analogously to half-edges in the description of the DCEL data structure [dBvKOS00]). We denote these edges *polygon-edges*. We construct the tree $\mathcal{T}_i$ of Lemma 3 for each polygon edge e_i, where the keys stored in that tree are the vertices of $\mathcal{A}_{i-1}$ along e_i. Each vertex v of $\mathcal{A}$ appears on four polygon-edges adjacent to v. In each of them, v is stored twice (with the same coordinate), corresponding to two of the four cells of $\mathcal{A}_{i-1}$ adjacent to v. The weight of the copy of v corresponding to a cell c is $\omega_c/(2m_c)$, where ω_c is the total weight of c, and m_c is the number of vertices of c. As easily checked, the sum of weights of vertices corresponding to c, summed over all data structure $\mathcal{T}_i$ for all edges e_i in $\mathcal{A}_{i-1}$ is ω_c. Let the *total weight* of a polygon-edge e_i denote the sum of weights of vertices on e_i. To pick a face of $\mathcal{A}_{i-1}$ at random, we first pick a polygon edge polygon edge bounding the face.

Picking an polygon edge e. This is accomplished by maintaining a tree $\tilde{\mathcal{T}}$ storing a representative point x_i for each polygon-edge e_i, where the weight of x_i is the total-weight of e_i. Similarly to the data structure of Lemma 3, $\tilde{\mathcal{T}}$ stores for each node μ the variable W_μ maintaining the sum of total-weights of the polygon-edges stored at the subtree rooted by μ. Maintaining W_μ upon changing the total weight of one of the polygon-edges in μ's subset is done in a routine bottom up fashion. Picking an edge e_i is done similar to Lemma 3. Both operation are doable in time $O(\log n)$.

Inserting a new polygon $Vis(q_i)$. We find a cell c of $\mathcal{A}$ containing a point of ∂Vis_i, which is also a vertex of P. This is easy to accomplish by maintaining which cell of $\mathcal{A}$ contains every vertex of P, so all is left to do is finding a vertex of P that sees the point corresponding (the "center" of) $Vis(q_i)$. It is known that the complexity of the zone of $\partial Vis(q_i)$ in $\mathcal{A}$ is only $O(ni\alpha(n))$. We follow $\partial Vis(q_i)$ through these cells that it intersects, splitting each cell we pass through. We compute, using the operations on the discrete hull described in Lemma 4 the number of points in the new cells, and update the weights accordingly, and the number of vertices along the boundaries of these cells.

Next we apply, for each tree $\mathcal{T}_i$ associated with e_i, the operation $sum(x_1, x_2)$ in order to compute the value of Ω defined above. If $2c_{opt}\Omega \leq$ the sum of weights of all points of Γ we double the weight of all the vertices of cells encapsulated in Vis_i, but applying $Interval\_double(x_1, x_2)$ operations described above to each of the trees associated with polygon edges. After a triangle is split, we need to compute the number of grid points inside each of the new triangles. This is required for calculating the weights of the new triangles. This is accomplished by the data structure of Lemma 4, and add a factor of $O(\log^2 \Delta)$, where Δ is the ratio between the diameter of the polygon and the grid size. Thus the time needed for the ith iteration is $O(ni\alpha(n) \log n(\log i + \log^2 \Delta))$.

We perform exponential search for the value of c_{opt} by performing $O(\log c_{opt})$ calls to `ComputeGuards`(P, k), where k is always $O(c_{opt})$, we conclude after omitting details due to lack of space

Theorem 4. *Given a simple polygon P with n vertices, one can spread a grid Γ inside P, and compute an $O(\log c_{opt})$-approximation to the smallest subset of Γ that sees P. The expected running time of the algorithm is $O\left(nc_{opt}^2 \log c_{opt} \log (nc_{opt}) \log^2 \Delta\right)$, where Δ is the ratio between the diameter of the polygon and the grid size.*

5.1. Range Searching on a Grid

Lemma 4. Let T be a triangle in the plane, and let Γ be a grid inside T. The boundary of DiscreteHull$(T) = \mathcal{CH}(\Gamma \cap T)$ and the number of points of Γ inside T can be computed in $O(\log \Delta)$ time, where Δ is the ratio between the diameter of the T and the grid size.

Proof. The boundary of the discrete hull $C_T = \mathcal{CH}(T \cap \Gamma)$ can be computed in $O(\log \Delta)$ time [KS96, HP98]. One can compute (in the same time complexity), the number M_T of points of Γ on the boundary of C_T, and Area(C_T). Now,

using Pick's Theorem one can now derive a precise closed formula on the number of grid points in $\Gamma \cap T$. Thus, the number of points of Γ inside T can be computed in $O(\log \Delta)$ time. □

Lemma 5. Let T, Γ and Δ be as in Lemma 4. One can pick randomly and uniformly a point from $T \cap \Gamma$ in $O(\log^2 \Delta)$ time.

Proof. Let B_0 be a bounding box of T. In the i-th stage, we split B_{i-1} vertically in the middle by a line ℓ_i that does not pass through points of the grid Γ, let B_i^R, B_i^L be the resulting two boxes to the right and left of ℓ_i, respectively. We can compute in $O(\log \Delta)$ time, by Lemma 4, the number of grid points in $w_i^R = B_i^R \cap T \cap \Gamma$, and in $w_i^L = B_i^L \cap T \cap \Gamma$. We now decide to B_i to be either B_i^R or B_i^L randomly according to their weights w_i^L, w_i^R. We can stop as soon as a single vertical grid line crosses our box B_i, as we can uniformly pick a grid point along this vertical grid line that lies inside T. Overall, this process clearly takes $O(\log^2 \Delta)$ time. □

5.2. Proof of Lemma 3

Proof. We maintain a sorted balanced tree $\mathcal{T}$, whose leaves are associate with the values x_i. Let $\pi(v, \mu)$ denote the path connecting node v to node μ, where v is an ancestor of μ. Each internal node μ maintains its *multiplicative factor* M_μ, initially 1.

The weight of a point x_i associated with the leaf μ equals $\prod_{\xi \in \pi(root(\mathcal{T}), \mu)} M_\xi$. We assign for each internal node μ the variable σ_μ, which equals

$$\sigma_\mu = \sum_{v \text{ descendent leaf of } \mu} \prod_{\xi \in \pi(\mu, v)} M_\xi$$

As easily observed, the sum of weights of the leaves in the subtree rooted at a node μ equals $\sigma_\mu \cdot \prod_{\xi \in \pi(root(\mathcal{T}), \mu)} M_\xi$. We next explain how to perform a "*pick*" operation: Assume that we already decided that point x_i to be picked belongs to the subtree $\mathcal{T}_\mu$ of a node μ, and we next decide whether x_i belongs to the left subtree of $\mathcal{T}_{left(\mu)}$, where $left(\mu)$ is the left child of μ. Observe that the probability of picking a point from the left subtree of μ equals

$$\frac{\sum \text{weights of leaves in } \mathcal{T}_{left(\mu)}}{\sum \text{weights of leaves in } \mathcal{T}_\mu} = \frac{\sigma_{left(\mu)} \prod_{\xi \in \pi(root(\mathcal{T}), left(\mu))} M_\xi}{\sigma_\mu \prod_{\xi \in \pi(root(\mathcal{T}), \mu)} M_\xi} = \frac{\sigma_{left(\mu)} M_{left(\mu)}}{\sigma_\mu}$$

This suggests the following approach to find a leaf x_i. We branch from the root to one of its children μ with the probabilities given above. Thus we perform a pick operation in time $O(\log n)$.

To support *Interval_double*(x_1, x_2), and *Interval_sum*(x_1, x_2) we first locate the set X of *canonical* nodes μ with the property that all descendent leaves of μ lie in the range $[x_1, x_2]$, but the parent of μ does not have this property. It is well known (see e.g. [dBvKOS00]) that we can visit all nodes in X in time $O(\log n)$. In the case of *Interval_double*(x_1, x_2) we just double M_μ for each $\mu \in X$.

In the case of *Interval_sum*(x_1, x_2) we use the equation above for computing the sum of weights of the points of each subtree μ for $\mu \in X$. Since we can visit all of them in $O(\log n)$ time, this is also the time required for this operation. □

6. Polygons with Holes and Terrains

The algorithms introduced in the previous sections, can easily be modified to solve visibility problems in more complicated "galleries', such as polygons with holes, or terrains. The modifications needed are only in the bounds on the complexity of the arrangements of visibility regions, and in way we compute them, and in the approximation factor.

Visibility in a Polygon with holes Let P be a polygon with n vertices and h holes. Let $\{q_1 \dots q_k\}$ be a set of points and let Vis_i denote the visibility polygon of q_i,

We claim that the complexity of the arrangement forms by the visibility polygons $\{Vis_1 \dots Vis_k\}$ is $O(nk^2h)$. This follows from the following argument. The boundary of Vis_i consists of $n+h$ edges which are not on ∂P. Every such edge can intersect ∂Vis_j in $\leq 2h$ points. Thus the total number of intersection points on ∂Vis_i is $\leq nhk$, and summing this bounds for all i yields the asserted bound.

Next we consider vertex visibility in a polygon with holes. Since the VC-dimension of the problem is $\Theta(1+\log h)$, as shown in [Val98] the approximation factor increases to $O(\log h \log(c_{opt} \log h))$. See [BG95] for details. Regarding the case where guards can be located inside P, Analogous version to Section 5 yields, after the obvious modification

Theorem 5. *Let P be a given polygon with n vertices in total and h holes.*

- *We can find a set G of $O(c_{opt} \log n \log(c_{opt} \log n))$ vertices of P that sees P, where c_{opt} is the cardinality of the optimal solution. The running time is $O(nh\, c_{opt}^3 polylog\, n)$.*
- *Let Γ be a grid inside P. Then we can find a set G of $O(c_{opt} \log h \log(c_{opt} \log n))$ vertices of Γ that sees P. The running time is $O(nhc_{opt}^3 polylog\, n \log^2 \Delta))$, where Δ is the ratio between the diameter of the polygon and the grid size.*

Visibility in Terrains Let T be a (triangulated) terrain of n triangles. We can also modify our algorithm in order to find a set S of vertices of T that sees T. Clearly, $O(\log n)$ bounds the VC-dimension of the set system obtained by assigning to each point $q \in T$ the points of T that q sees. This follows from the following observation: Assume d is the VC-dimension of the problem, and let S be a set of points d of T which is shatterable under visibility. That is, for every $S' \subseteq S$ there is a point $g_{S'}$ on T such that $S' = S \cap Vis(g_{S'})$.

The visibility region $Vis(g_{S'})$ can be described as the union of $\Theta(n^2)$ triangles in T, each fully contained inside a face of T, where the boundary of each such triangle Δ is either the boundary of a triangle of T, or of the intersection of T with the the plane h_r passing through p and through an edge r of T. Since

there are $O(n)$ edges r in T, and each plane h_r intersects a triangle of T along a straight segment, the $\Theta(n^2)$ bound on the complexity of $Vis(p)$ follows. The total number of edges of $Vis(q)$ for all $q \in S$ is $O(dn^2)$, and overlaying the boundaries of $Vis(p)$ for each $q \in S$ imposes a subdivision $\mathcal{S}$ of T into $O(d^2n^4)$ regions, where if two points x_1, x_2 of T lie in the same region of $\mathcal{S}$, then they see the same subset of S. Since S is shattered under visibility, the number of regions in $\mathcal{S}$ is at least 2^d implying $d = O(\log n)$.

In [dB93] de Berg showed that the complexity of the arrangement $\mathcal{A}$ forms by the visibility polygons of a set G of k guards is $O(n^2k^2)$. Plugging the upper bound into our algorithm, and skipping obvious details, we obtain a running time of $O(n^2c_{opt}^3 polylog\, n \log^2 \Delta)$ where Δ is the ratio between the diameter of the terrain and the grid size. This improves the recent $O(n^8)$-algorithm of Eidenbenz [Eid02], who obtained a slightly better approximation factor of $O(\log n)$.

Note that if guards are allowed to be located only on vertices of the terrain, then the use of the grid is not needed, and the running time is improved to $O(n^2c_{opt}^3 polylog\ n)$. To summerize

Theorem 6.

- *Given a terrain T of n triangles, we can find in time $O(n^2c_{opt}^3 polylog\ n)$ a set S of vertices of T that see T, where $|S|$ is within a factor of $O(\log n \log\log n)$ of the minimum.*

- *Given a terrain T of n triangles, and a grid Γ placed on each triangle of T, we can find in time $O(n^2c_{opt}^3 polylog\ n \log^2 \Delta)$ a set S of vertices of Γ that see T, where $|S|$ is within a factor of $O(\log n \log\log n)$ of the minimum, and Δ is the ratio between the diameter of the terrain and the grid size.*

Remark: Currently we are working on methods which allow us to find small sets of guards inside polygons and terrains, *without* using the grid. This would improve the running time, and simplify the algorithm significantly. Preliminary results obtained so far look very promising, and will be reported separately.

Acknowledgments We would like to thank Will Evans, Stephen Kobourov, T.M. Murali and Micha Sharir for very helpful discussions. We also thank Micha Sharir for the result of Section 4.

References

[Agg84] A. Aggarwal. *The art gallery problem: Its variations, applications, and algorithmic aspects.* PhD thesis, Dept. of Comput. Sci., Johns Hopkins University, Baltimore, MD, 1984.

[BG95] H. Brönnimann and M. T. Goodrich. Almost optimal set covers in finite VC-dimension. *Discrete Comput. Geom.*, 14:263–279, 1995.

[BPR96] S. Basu, R. Pollack, and M.-F. Roy. On the combinatorial and algebraic complexity of quantifier elimination. *J. ACM*, 43:1002–1045, 1996.

[BS93] I. Bjorling-Sachs, *Variations on the Art Gallery Theorem.* PhD thesis, Rutgers University, 1993.

[Cha91] B. Chazelle, Triangulating a simple polygon in linear time. *Discrete Comput. Geom.* , 6 (1991) 485–524.

[Cla93] K.L. Clarkson. Algorithms for polytope covering and approximation. In *Proc. 3rd Workshop Algorithms Data Struct.*, LNCS 709, 246–252, 1993.

[dB93] M. de Berg. Generalized hidden surface removal. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 1–10, 1993.

[dBvKOS00] M. de Berg, M. van Kreveld, M. H. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, 2nd edition, 2000.

[EHKKRW02] A. Efrat, F. Hoffmann, K. Kriegel, C. Knauer, G. Rote and C. Wenk, Covering Shapes by Ellipses *ACM-SIAM Symposium on Discrete Algorithms*, 2002, 453–454.

[EA81] H. ElGindy and D. Avis. A linear algorithm for computing the visibility polygon from a point. *J. Algorithms*, 2:186–197, 1981.

[Eid00] S. Eidenbenz. *(In-)Approximability of Visibility Problems on Polygons and Terrains.* Phd thesis, diss. ETH no. 13683, 2000.

[Eid02] S. Eidenbenz. Approximation algorithms for Terrain Guarding *Information Processing Letters (IPL)* 82 (2002) 99-105.

[GBL01] Hector Gonzalez-Banos and Jean-Claude Latombe. A randomized art-gallery algorithm for sensor placement. In *Proc. 17th Annu. ACM Sympos. Comput. Geom.*, pages 232–240, 2001.

[Gho87] S. K. Ghosh. Approximation algorithms for art gallery problems. In *Proc. Canadian Inform. Process. Soc. Congress*, 1987.

[GMMN90] L. Gewali, A. Meng, Joseph S. B. Mitchell, and S. Ntafos. Path planning in $0/1/\infty$ weighted regions with applications. *ORSA J. Comput.*, 2(3):253–272, 1990.

[HKK91] F. Hoffmann, M. Kaufmann, and K. Kriegel. The art gallery theorem for polygons with holes. In *Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci.*, (1991) 39–48.

[Hof90] F. Hoffmann. On the rectilinear art gallery problem. In *Proc. 17th Internat. Colloq. Automata Lang. Program.*, LBNCS 443, 717–728, 1990.

[HP98] S. Har-Peled. An output sensitive algorithm for discrete convex hulls. *Comput. Geom. Theory Appl.*, 10:125–138, 1998.

[JL93] G. F. Jennings and W. J. Lenhart. An art gallery theorem for line segments in the plane. In G. T. Toussaint, editor, *Pattern Recognition Letters Special Issue on Computational Geometry*, 1993.

[KS96] S. Kahan and J. Snoeyink. On the bit complexity of minimum link paths: Superquadratic algorithms for problems solvable in linear time. In *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, 151–158, 1996.

[O'R83] J. O'Rourke. An alternative proof of the rectilinear art gallery theorem. *J. Geom.*, 21:118–130, 1983.

[O'R87] J. O'Rourke. *Art Gallery Theorems and Algorithms.* The International Series of Monographs on Computer Science. Oxford University Press, New York, NY, 1987.

[OS83] J. O'Rourke and K. J. Supowit. Some NP-hard polygon decomposition problems. *IEEE Trans. Inform. Theory*, IT-30:181–190, 1983.

[Urr00] J. Urrutia. Art gallery and illumination problems. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, pages 973–1027. North-Holland, 2000.

[Val98] P. Valtr. Guarding galleries where no point sees a small area. *Israel J. Math*, 104:1–16, 1998.