Computing the Visibility Graph of Points Within a Polygon

Boaz Ben-Moshe
Computer Science
Ben-Gurion University
Beer-Sheva 84105, Israel
benmoshe@cs.bgu.ac.il

Olaf Hall-Holt
Applied Math. & Statistics
Stony Brook University
Stony Brook, NY 11794-3600
olaf@ams.sunysb.edu

Matthew J. Katz
Computer Science
Ben-Gurion University
Beer-Sheva 84105, Israel
matya@cs.bgu.ac.il

Joseph S. B. Mitchell
Applied Math. & Statistics
Stony Brook University
Stony Brook, NY 11794-3600
jsbm@ams.sunysb.edu

ABSTRACT

We study the problem of computing the visibility graph defined by a set $\mathcal{P}$ of n points inside a polygon Q: two points $p, q \in \mathcal{P}$ are joined by an edge if the segment $\overline{pq} \subset Q$. Efficient output-sensitive algorithms are known for the case in which $\mathcal{P}$ is the set of all vertices of Q. We examine the general case in which $\mathcal{P}$ is an arbitrary set of points, interior or on the boundary of Q and study a variety of algorithmic questions. We give an output-sensitive algorithm, which is nearly optimal, when Q is a simple polygon. We introduce a notion of "fat" or "robust" visibility, and give a nearly optimal algorithm for computing visibility graphs according to it, in polygons Q that may have holes. Other results include an algorithm to detect if there are any visible pairs among $\mathcal{P}$, and algorithms for output-sensitive computation of visibility graphs with distance restrictions, *invisibility* graphs, and rectangle visibility graphs.

Categories and Subject Descriptors: F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems—*geometrical problems and computations*

General Terms: Algorithms, Theory

Keywords: Visibility graphs, polygons, illumination, guarding, fatness, output-sensitive algorithms

1. INTRODUCTION

The visibility graph is a fundamental geometric structure useful in many applications, including illumination and rendering, motion planning, pattern recognition, and sensor networks.

Let Q be a polygon with n vertices. Let $\mathcal{P} = \{s_1, \ldots, s_m\}$ be a set of m points (sites) in Q; the points in $\mathcal{P}$ may lie both on the boundary of Q and in the interior of Q. The visibility graph of $\mathcal{P}$ in Q, denoted $\mathrm{VG}_Q(\mathcal{P})$, is the graph whose nodes are the points $\mathcal{P}$ and whose edges connect pairs of nodes that *see* one another within Q (i.e., the segment joining the points lies within Q). See Figure 1 for an example. In this paper, we present efficient algorithms to compute $\mathrm{VG}_Q(\mathcal{P})$.

There has been considerable study of visibility graphs and algorithms to compute them (see, e.g., [24]). Optimal algorithms are known for computing the visibility graph of *all* vertices of Q (i.e., the case that $\mathcal{P} = V$ is the vertex set of Q): one can compute $\mathrm{VG}_Q(V)$ in time $O(n+k)$ if Q is simple, or time $O(n \log n + k)$ if Q has holes, where k is the size of the output. The algorithms for computing $\mathrm{VG}_Q(V)$ exploit the structure of the entire visibility graph $\mathrm{VG}_Q(V)$ in achieving their efficiency, charging off work to the edges.

For our more general problem of computing $\mathrm{VG}_Q(\mathcal{P})$ for *any* set of sites $\mathcal{P}$ within Q, there are two immediate solutions based on existing techniques:

(1) We can build the supergraph, $\mathrm{VG}_Q(V \cup \mathcal{P})$, in time $O((n+m)\log(n+m)+k')$, where k' is the size of the output, and then report only those edges of interest to us (those of $\mathrm{VG}_Q(\mathcal{P})$); however, k' could be enormously greater than the size of the desired output – e.g., k' may be $\Omega(n^2)$, with $n >> m$, while the size of $\mathrm{VG}_Q(\mathcal{P})$ may be very small (e.g., $O(m)$, or $O(1)$). Computing the relative convex hull, C, of $\mathcal{P}$ with respect to Q, and then computing the free bitangents [22] within C reduces the number of superfluous edges computed, but there may still be a substantial discrepancy between the $\Omega(m^2)$ edges that may be reported and the output size (possibly $O(1)$).

(2) We can preprocess Q for ray shooting queries and then conduct a query for every pair of points of $\mathcal{P}$, thereby determining $\mathrm{VG}_Q(\mathcal{P})$. This method takes time $O(m^2 q(n))$, where $q(n)$ is the time for performing a ray-shooting query: $q(n) = O(\log n)$ if Q is simple, and $q(n) = O(\sqrt{n}\log n)$ if Q has holes and we spend $O(n^{3/2}\log n)$ time to preprocess Q. While this method may be far superior to method (1) if $n >> m$, it still requires $\Omega(m^2)$ queries, as it is not output-sensitive.

In this paper, we introduce techniques to achieve output-sensitivity, with a running time close to linear in both the input size $(m+n)$ and the output size (the number, k, of edges of $\mathrm{VG}_Q(\mathcal{P})$), for the case that Q is a simple polygon.

We also introduce a notion of *fat* or *robust* visibility, which models the fact that a point q that is "just barely" visible from p may not be detected by a camera or viewer at p, depending on the resolution of the sensor. In order for q to be fatly (or robustly) visible from p, we require that no matter where q may move within a disk $B_r(q)$ of radius r centered at q, q continues to lie within Q and be visible from p. The radius r of the disk is determined by the distance from p, according to a fatness parameter, α, which is the lower bound on the angle subtended by the disk with respect to a viewer at p.

In many visibility applications, there is a distance restriction on how far one can see. We give an output-sensitive visibility graph algorithm for the case in which each point of $\mathcal{P}$ has an associated range of sight.

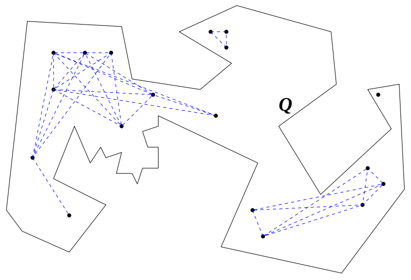

Figure 1. The visibility graph $\mathrm{VG}_Q(\mathcal{P})$ is shown with dashed edges joining pairs of visible sites of $\mathcal{P}$ within a polygon Q.

Summary of Results.

(1) For a simple polygon Q, we show that the $\mathrm{VG}_Q(\mathcal{P})$ can be computed in time $O(n + m\log m \log mn + k)$ using $O(n+m)$ space, where k is the number of edges of $\mathrm{VG}_Q(\mathcal{P})$. The specialization of this result to the case in which the sites are exactly the vertices of Q ($\mathcal{P} = V$) yields an $O(n\log^2 n + k)$ algorithm that nearly matches the optimal $O(n+k)$ algorithm of [18].

(2) For a simple polygon Q and a set $\mathcal{P}$ of m sites in Q each having an associated range (distance) of vision, we show that the range-restricted visibility graph can be computed in time $\tilde{O}(n\log n + m^{3/2} + k)$ time using $O(n + m\log m)$ storage.[1] (There is a space-query time tradeoff that allows the running time to be reduced to roughly $O(n + m^{4/3} + k)$.)

(3) We introduce a natural notion of "robust" visibility, and we show how to compute the robust visibility graph of a set of sites $\mathcal{P}$ within a polygon Q in nearly optimal (output-sensitive) time, even if Q has holes.

[1]Here, $\tilde{O}(\cdot)$ indicates big-Oh notation ignoring polylog factors.

(4) We give algorithms for detecting the emptiness of the edge set of $\mathrm{VG}_Q(\mathcal{P})$, both for the case that Q is any simple polygon and an improved algorithm for the case that Q is a one-dimensional terrain.

(5) We give an efficient algorithm for computing the visibility graph of $\mathcal{P}$ within an arbitrary polygon (possibly with holes) in the case that visibility is *rectangle visibility.*

(6) We give an output-sensitive algorithm for computing *invisibility graphs* for sites $\mathcal{P}$ within a simple polygon Q, whose edge set is the complement of the visible pairs. This algorithm allows one to compute a representation of dense visibility graphs particularly efficiently.

Related Work. Prior algorithms to compute visibility graphs have addressed only the special case in which the sites $\mathcal{P}$ are given as the set V of all vertices of the polygon Q. The first algorithms to construct the visibility graph $\mathrm{VG}_Q(V)$ required time $O(n^2\log n)$ [20], using a radial sweep about each vertex of Q. Welzl [25] and Asano et al. [5] improved the time bound to $O(n^2)$, which is worst-case optimal but not output-sensitive. Hershberger [18] gave an optimal $O(n+k)$ output-sensitive algorithm to compute $\mathrm{VG}_Q(V)$ for a simple polygon Q. For general polygons (with holes), Overmars and Welzl [21] obtained a relatively simple $O(k\log n)$-time method, requiring $O(n)$ space. Then, Ghosh and Mount [13] obtained an $O(k + n\log n)$-time algorithm, using $O(k)$ storage. Pocchiola and Vegter [23] have improved the space bound to optimal using the concept of the *visibility complex*; their algorithm requires time $O(k + n\log n)$ and uses $O(n)$ space.

Preliminaries. Let Q be a polygon in the plane having n vertices; i.e., Q is a connected subset of $\mathbb{R}^2$ whose boundary, ∂Q, is the union of n straight line segments (the *edges* of Q). We consider Q to be a closed region in the plane (it includes its boundary) and assume that the interior of Q is connected. We let h denote the number of holes of Q; thus, ∂Q consists of $h+1$ cycles. Q is a *simple polygon* if $h = 0$.

We let $\mathcal{P} \subset Q$ denote a set of m points within Q. We refer to the points $\mathcal{P}$ as *sites*. Two points, $p, q \in Q$, are *visible* if and only if $\overline{pq} \subset Q$. The *visibility graph*, $\mathrm{VG}_Q(\mathcal{P})$, *of* $\mathcal{P}$ *with respect to* Q is a graph whose nodes are the sites $\mathcal{P}$ and whose edges link pairs of sites that are visible to each other. We let k denote the number of edges of $\mathrm{VG}_Q(\mathcal{P})$.

2. VISIBILITY GRAPHS IN A SIMPLE POLYGON

We consider the case in which Q is a simple polygon ($h = 0$) having n vertices, and $\mathcal{P} = \{s_1, \ldots, s_m\}$ is a set of m sites within Q.

Our algorithm is based on divide-and-conquer. We begin by describing a simple algorithm that is output-sensitive but not as efficient as our main result. We only sketch the method, since we give more details for our improved algorithm.

2.1 An Output-Sensitive Algorithm

We cut Q into two subpolygons, Q_1 and Q_2, using a diagonal, ξ, of Q such that the set of sites $\mathcal{P}$ is partitioned into two subsets, $\mathcal{P}_1 \subset Q_1$ and $\mathcal{P}_2 \subset Q_2$, of approximately the same cardinality (say, $|\mathcal{P}_1|, |\mathcal{P}_2| \geq m/4$). We recursively compute $\mathrm{VG}_{Q_1}(\mathcal{P}_1)$ and $\mathrm{VG}_{Q_2}(\mathcal{P}_2)$. The edges of $\mathrm{VG}_Q(\mathcal{P})$ still to be computed are those that straddle the cut ξ, joining $s_i \in Q_1$ to $s_j \in Q_2$. For each $s_i \in \mathcal{P}$, we determine the subsegment $\sigma_i \subseteq \xi$ of points on the cut ξ that are visible from s_i. Let L_i denote the set of all (infinite) lines passing through s_i that intersect σ_i. Such a family of lines L_i forms a "double wedge", which, in the dual, corresponds to a line segment, L_i^* (under any standard point-line duality transformation). A site $s_i \in Q_1$ is visible to a site $s_j \in Q_2$ if and only if $L_i \cap L_j \neq \emptyset$, in which case there is a unique line $\ell \in L_i \cap L_j$, namely, ℓ is the line through s_i and s_j. See Figure 2. In the dual, then, the visibility of s_i and s_j corresponds to the segment L_i^* intersecting L_j^*, at a point ℓ^* that is dual to line ℓ. Thus, we can compute the set of all visibilities that straddle ξ by computing all of the "red-blue" segment intersections between the "red" segments L_i^* corresponding to sites $\mathcal{P}_1$ and the "blue" segments L_j^* corresponding to sites $\mathcal{P}_2$.

The difficulty now is that the best known algorithms for the bichromatic segment intersection problem require that the union of the red segments is a connected set and that the union of the blue segments is also connected; in such a case, the K red-blue intersections can be reported in time $\tilde{O}(K+r+b)$, for r red segments and b blue segments [7, 10, 17]. (If the unions are not known to be connected, then the intersections can be computed in time $O((r+b)^{4/3}\log(r+b)+K)$ [1, 16].)

Instead of computing only the red-blue segment intersections, though, we can compute *all* intersections among the dual segments L_i^*, regardless of which subpolygon contains the corresponding site s_i. Each such intersection does correspond to a visible pair of sites; however, the pair of sites may both belong to the same subpolygon. This means that each visible pair is discovered over and over again.

One can argue, though, that a visible pair is rediscovered only $O(\log m)$ times, since in going down the recursion tree, s_i and s_j are on the same side of a cut only $O(\log m)$ times before they are split by a cut. This gives the following result:

PROPOSITION 1. *The simple divide-and-conquer algorithm requires time $O(n + m\log m \log mn + k\log m)$.*

2.2 An Improved Algorithm

We now improve upon our simple divide-and-conquer algorithm in order to remove the factor of $O(\log m)$ that multiplies the output size, k, in the running time, which is caused by over-reporting the visible pairs.

Our improved algorithm makes use of the query structure of Guibas and Hershberger [14], modified in order to account for the presence of sites in the polygon. We first summarize their construction, which was developed to answer shortest path queries in a polygon, and then describe how we modify and extend it to obtain our new results.

In order to build a query structure for shortest paths, we first compute a hierarchical decomposition of the polygon Q by repeatedly splitting Q into two with a diagonal. The collection of diagonals produced in the recursive splitting procedure form the nodes of a binary tree, where the two children of a diagonal are the diagonals splitting its left and right subpolygons. The *factor graph* augments this splitting

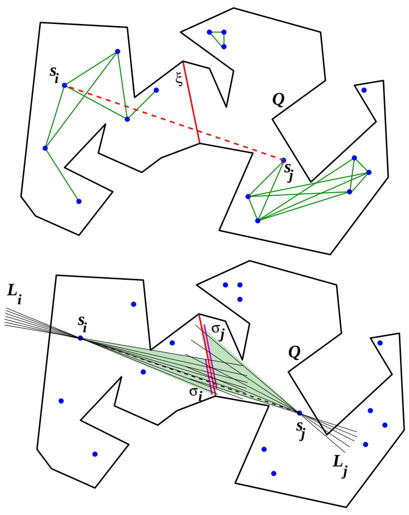

Figure 2. Illustration of the cut ξ, the visible subsegments σ_i, and the set of lines L_i. Site s_i sees site s_j if there is a line in common to L_i and L_j.

tree with an edge between each pair of diagonals d_1, d_2 such that both diagonals lie on the boundary of a subpolygon obtained during the splitting process. That is, if we look at the polygon Q after all the diagonals have been added, then two diagonals will be connected by an edge in the factor graph if and only if they can be connected inside Q by a path that does not cross any diagonals of lesser depth in the splitting tree. For each edge of the factor graph, Guibas and Hershberger compute an hourglass query structure that permits shortest path queries of the following form: given a point p and an hourglass h that connects d_1 and d_2, find the length of the shortest path from p to a point on d_2, assuming that this path passes through d_1. This query requires time proportional to the sum of the heights of the two diagonals in the tree. The construction time to build the factor graph and all its query structures is proportional to the total number of diagonals.

In order to adapt this query structure to the problem of visibility between sites of $\mathcal{P}$, we first choose the sequence of diagonals in such a way that the number of sites on either side of a diagonal is approximately balanced. Some of these diagonals may have endpoints on the interior of edges of the polygon, rather than vertices, so the complexity of the polygon may increase to $\Theta(m+n)$. The recursion continues until there is exactly one site in each leaf subpolygon. We now build the factor graph and the associated hourglass query structures as before, in time $O(m+n)$. We only need those hourglasses that are "open", namely those whose diagonals are mutually visible. In addition, we will only use the following type of query: For a given point p and an hourglass h, determine the parts of each diagonal of h that are visible to p, if any. In order to answer this visibility query, we determine the point at which a ray from p is tangent to a given side of h. Finding this tangent is the basic ingredient of the shortest path query, and thus can be answered in time $O(\log(m+n)) = O(\log mn)$ with the same query structure.

Now to compute the pairs of visible sites, we begin by constructing an appropriate set of dual segments to characterize the part of ray space covered by each site. Consider a ray r from site s, and the initial segment i of r that lies in the interior of Q. This initial segment i will intersect some set D of diagonals. If D is not empty, then let d be the element of D of minimum depth. We associate r with d. That is, we associate the ray r with the first diagonal that would cut the ray inside the polygon, during the original splitting process. Note that if a ray from site s_1 is directed toward another site s_2, then it will be associated with a diagonal, since it will have to cross a diagonal that separates s_1 from s_2. If we thus assign all the rays from a site s to their associated diagonals, we will have a partition of the rays from s into intervals of rays. The set of diagonals thus associated with s cannot include two diagonals of the same depth, and thus the number of ray-space intervals is $O(\log m)$ per site. Computing these intervals can then be accomplished at a cost of $O(\log m \log mn)$ per site.

Now to compute the set of visible pairs of sites, we consider each diagonal d in turn, and compute the intersections between all the ray-intervals associated with d. In the dual space, the ray interval is a segment, so we can compute the set of segment intersections in time $O(k_1 + m_1 \log m_1)$ time, where m_1 is the number of dual segments associated with d, and k_1 is the number of intersections found.

Lemma 1. *A given pair of visible sites will be discovered either once or twice.*

The total cost of the intersection calculation is then $O(k + m \log^2 m)$. This leads to a total running time of $O(n + m \log m \log mn + k)$ to find all k of the visible pairs.

We conclude with our main result of this section:

Theorem 1. *One can construct the visibility graph* $\mathrm{VG}_Q(\mathcal{P})$ *of m sites $\mathcal{P}$ within a simple polygon Q having n vertices in time $O(n + m \log m \log mn + k)$, where k is the number of visible pairs (edges of* $\mathrm{VG}_Q(\mathcal{P})$*).*

Remark: We are able to extend the above theorem to the case of polygons with h holes, at a cost of a factor of $O(h)$ in the time complexity. The extension utilizes a decomposition of the polygon with holes into $O(h)$ corridors and junction triangles (as in [15, 19]).

3. DISTANCE-RESTRICTED VISIBILITY GRAPHS

Consider now the case in which each point $p \in \mathcal{P}$ has an associated *range* of sight, $d_p > 0$, such that an observer at point p can see only up to distance d_p. Then, the *distance-restricted visibility graph*, $\overline{\mathrm{VG}}_Q(\mathcal{P})$, of $\mathcal{P}$ with respect to Q contains a *directed* edge from point $p \in \mathcal{P}$ to point $q \in \mathcal{P}$ if and only if $\overline{pq} \subset Q$ and $|pq| \leq d_p$, where $|pq|$ denotes the Euclidean length of $\overline{pq}$ (see Figure 3).

We consider the case in which Q is a simple polygon. We first present an alternative algorithm for computing $\mathrm{VG}_Q(\mathcal{P})$. This algorithm is less efficient than the one presented in Section 2, but it can be transformed easily into an algorithm for computing $\overline{\mathrm{VG}}_Q(\mathcal{P})$.

We begin by preprocessing Q for shortest path queries ([14]). This permits us to answer a *wedge query* in time $O(\log n)$: Given a site $s_i \in \mathcal{P}$ and a diagonal ξ of Q, we can determine the segment $\sigma_i \subseteq \xi$ of points on the cut ξ that are visible from s_i. The *wedge* w_i is the set of rays with apex s_i that intersect σ_i.

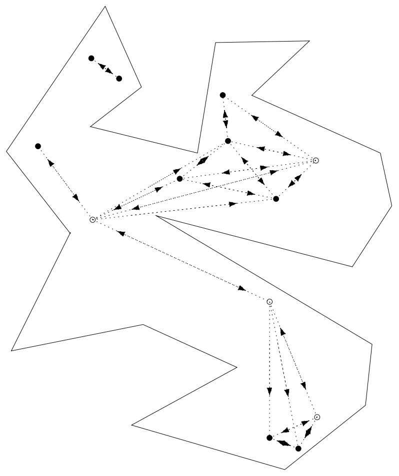

Figure 3. The distance-restricted visibility graph with only two possible ranges of sight: short (solid disks) and long (hollow circles).

Our algorithm uses the divide and conquer paradigm. We find a diagonal ξ that divides Q into two subpolygons, such that each of them contains at least one fourth of the points in $\mathcal{P}$ (refer to Section 2.2). For each point $s_i \in \mathcal{P}$ we compute the wedge w_i and the segment σ_i through which it sees ξ (in total $O(m \log n)$ time).

Without loss of generality we assume that ξ is vertical (since we can simply rotate the scene to achieve this). Let $\mathcal{P}_l$ (resp. $\mathcal{P}_r$) denote the subset of points in $\mathcal{P}$ to the left (resp. to the right) of ξ. Let $\mathcal{S}$ be the set of the m segments $\sigma_i \subseteq \xi$, for $s_i \in \mathcal{P}$. We build a skeleton of a segment tree T for the segments $\mathcal{S}$. We insert the segments of $\mathcal{S}$ into T. Recall that each segment σ_i is stored in $O(\log m)$ nodes of T, such that the canonical segments associated with these nodes consist of a partitioning of σ_i into subsegments. At each node v of T we divide the segments stored in v (i.e., the canonical set of v) into two canonical subsets $\mathcal{S}_v^l$ and $\mathcal{S}_v^r$, so that $\mathcal{S}_v^l$ (resp. $\mathcal{S}_v^r$) includes all of the segments stored in v that are associated with points in $\mathcal{P}_l$ (resp., $\mathcal{P}_r$).

Let $s_i \in \mathcal{P}_l$ and $s_j \in \mathcal{P}_r$, and assume that s_i and s_j are visible to each other (though possibly at a distance greater than $\max\{d_{s_i}, d_{s_j}\}$). Let o be the intersection point between $\overline{s_i s_j}$ and ξ. Clearly $o \in \sigma_i \cap \sigma_j$. The segment σ_i (resp., σ_j) is stored in $O(\log m)$ nodes of T corresponding to $O(\log m)$ canonical segments whose union is σ_i (resp., σ_j). We focus on the canonical segment $\sigma \subseteq \sigma_i$ (resp., $\sigma' \subseteq \sigma_j$) in which o lies. From the definition of a segment tree it follows that either $\sigma \subseteq \sigma'$ or $\sigma' \subseteq \sigma$. Assume, e.g., that $\sigma \subseteq \sigma'$, and let w be the wedge from s_i through σ. Then s_j lies in w.

The opposite direction is also true. That is, if σ (resp., σ') is one of the canonical segments into which σ_i (resp., σ_j) was partitioned, where $s_i \in \mathcal{P}_l$ and $s_j \in \mathcal{P}_r$, and if $\sigma \subseteq \sigma'$, and $s_j \in w$, where w is the wedge from s_i through σ, then s_i and s_j see each other through a point $o \in \sigma$.

Thus, in order to find all pairs (s_i, s_j) of points, $s_i \in \mathcal{P}_l$ and $s_j \in \mathcal{P}_r$, such that s_i and s_j are visible to each other (without the distance restriction), we do the following. For each node v of T, we preprocess the subset of $\mathcal{P}_r$ corresponding to the canonical subset $\mathcal{S}_v^r$ for efficient wedge range searching. Now, for each node v of T we proceed as follows. For each segment $\sigma_i \in \mathcal{S}_v^l$, we perform a range searching query with the wedge from s_i and through the canonical

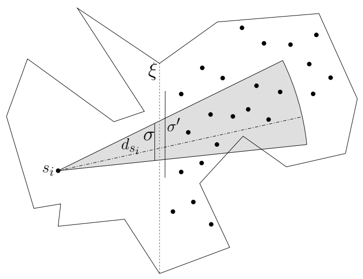

Figure 4. Illustration of a sector.

segment σ of v. We do this query in the range searching data structures of v and of all its ancestors in T. All points that are found are reported to be visible from s_i.

We now switch the roles of left and right and repeat. That is, we preprocess the subsets of $\mathcal{P}_l$ corresponding to the canonical subsets $\mathcal{S}_v^l$ for wedge range searching, and we perform queries with wedges from points in $\mathcal{P}_r$.

At this stage we have found all left-right pairs that are visible to each other, in the usual sense (without distance restriction), and each such pair was discovered at most twice. We now proceed to find all left-left visible pairs and all right-right visible pairs, by processing each of the two parts of the polygon Q recursively.

The algorithm described above computes the visibility graph $\mathrm{VG}_Q(\mathcal{P})$, but it can be transformed easily into an algorithm for computing the distance-restricted visibility graph, $\overline{\mathrm{VG}}_Q(\mathcal{P})$. Instead of preprocessing the subsets of $\mathcal{P}_r$ (resp., $\mathcal{P}_l$) corresponding to the canonical subset $\mathcal{S}_v^r$ (resp., $\mathcal{S}_v^l$) for wedge range searching, we preprocess them for efficient *sector* range searching, where a sector is obtained by intersecting a wedge with a disk centered at the origin of the wedge. The query ranges associated with point $s_i \in \mathcal{P}$ will be sectors of radius d_{s_i}, each of which is the intersection of two halfplanes and a disk. See Figure 4. Thus, we can use known results (see, e.g., [2, 4]) in multi-level range search data structures to answer queries in time (roughly) output size plus $O(m^{1/2})$: the first two levels of the structure apply halfspace range searching to be able to report all points in the query wedge as a union of canonical sets, and the third level of the structure reports those points in the canonical sets that lie within the query disk.

THEOREM 2. *Let Q be a simple polygon with n vertices, and let $\mathcal{P}$ be a set of m points in Q with associated ranges of sights. Then $\overline{\mathrm{VG}}_Q(\mathcal{P})$ can be computed in $\tilde{O}(n + m^{3/2} + k)$ time, using roughly $O(n + m)$ storage.*

If we can afford to use more storage space, we can reduce the running time, using the known space-query tradeoff of range searching.

Notice that if the number k of edges in $\mathrm{VG}_Q(\mathcal{P})$ is greater than $m^{3/2}$, and the number $\overline{k}$ of edges in $\overline{\mathrm{VG}}_Q(\mathcal{P})$ is significantly smaller than k, then the above algorithm is more efficient than the algorithm for computing $\mathrm{VG}_Q(\mathcal{P})$ (which could then be scanned for edges that are too long).

4. ROBUST VISIBILITY

In this section we assume that Q is a polygon, possibly with holes, consisting of n vertices in total, and that $\mathcal{P}$ is a set

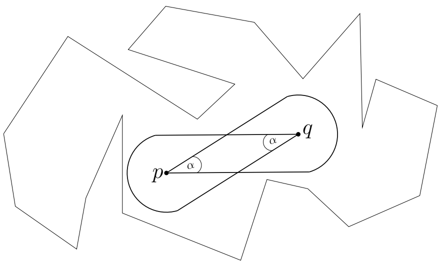

Figure 5. p and q are α-robustly visible to each other.

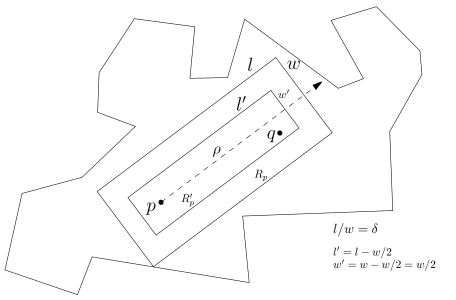

Figure 6. The rectangles R_p and R'_p.

of m points in Q. We say that $p \in \mathcal{P}$ *α-robustly* (or *α-fatly*) *sees* $q \in \mathcal{P}$, for a given angle $\alpha > 0$, if the "ice cream cone" with p as apex, $\overline{pq}$ as axis, and α as opening angle is completely contained in Q. Refer to Figure 5. This notion of robust visibility models the situation in which a camera or sensor at p can only detect q if there is a "fat" wedge (e.g., indicative of the angular resolution of the sensor) of unobstructed space containing q, so that if q is perturbed within a neighborhood, it is still seen from p.

We define the α-robust visibility graph, $\mathrm{VG}_Q^\alpha(\mathcal{P})$, to be the (undirected) graph whose nodes are the sites $\mathcal{P}$ and whose edges link a pair of sites $p, q \in \mathcal{P}$ if and only if p α-robustly sees q and q α-robustly sees p, in which case we say that p and q are α-robustly visible to each other. We wish to find all k edges of $\mathrm{VG}_Q^\alpha(\mathcal{P})$. We present an algorithm that finds all these pairs, possibly together with some other pairs; however, the additional pairs reported by our algorithm are guaranteed also to be robustly visible, just with a slightly different parameter: every pair reported by the algorithm is (α/c)-robustly visible to each other, for an appropriate constant c. The running time of the algorithm is $O(n + m \log mn + k')$ (resp., $O(n \log n + m \log mn + k')$), where k' is the number of pairs reported, for the case that Q is a simple polygon (resp., polygon with holes). (The dependence on α is a factor of $O(1/\alpha)$.)

The Algorithm

We start by choosing a uniform sample of $O(1/\alpha)$ orientations, such that, the angle between any orientation and the sample orientation that forms the smallest angle with it is at most β, where $\beta = f(\alpha)$ is an appropriate parameter. We now repeat the following algorithm for each of the sample orientations ρ.

Let δ be an appropriate parameter, depending on α and β. We say that a rectangle is of orientation ρ if its (directed) medial axis is of orientation ρ. (The medial axis of a rectangle of width W and length $L \geq W$ is a line segment

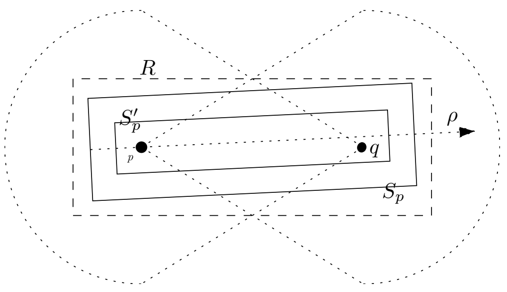

Figure 7. The algorithm finds all pairs p, q that are α-robustly visible to each other.

of length $L - W$ positioned in the center of the rectangle, aligned with the longer sides. It is the locus of points that are centers of maximal disks within the rectangle.) We call the origin of the (directed) medial axis the *focus* point of the rectangle. For each point $p \in \mathcal{P}$ we compute a maximal rectangle $R_p \subseteq Q$ of orientation ρ, with p at its focus point and with aspect ratio δ. See Figure 6. This can be done in $O(\log n)$ time per p, after preprocessing Q in time $O(n)$ (if Q is simple [12]) or $O(n \log n)$ (if Q has holes). For this step, we compute the Voronoi diagram (see, e.g., [6, 11]) of Q according to the convex distance function defined by a rectangle of aspect ratio δ with origin at the focus point, and we preprocess the diagram for point location queries. Then, for a given $p \in \mathcal{P}$, the Voronoi diagram reports the boundary element(s) (edge or vertex) of Q that is in contact with R_p, allowing us to compute R_p.

Let R'_p denote the rectangle contained in R_p, such that the distance between each of the sides of R'_p and the corresponding side of R_p is $w/4$, where w is the width of R_p. We now perform an orthogonal range searching query with R'_p to report all points in $\mathcal{P}$ that lie in R'_p. For each such point q, we output the pair p, q.

The Analysis

We prove the following lemma

LEMMA 2. *1. If $p, q \in \mathcal{P}$ are α-robustly visible to each other, then the algorithm above will report the pair p, q; and,*

2. There exists a constant c, such that, if a, b is a pair reported by the algorithm above, then a, b are (α/c)-robustly visible to each other.

PROOF. (Sketch) In this extended abstract, we only give a sketch of the proofs without detailing the computation of the explicit values of the parameters. Consider the first claim. Assume $p, q \in \mathcal{P}$ are α-robustly visible to each other, and assume, e.g., that $\overline{pq}$ is horizontal with p to the left of q. Let ρ be the sample orientation that forms the smallest angle with the positive x-axis. We know that this angle is at most β. Let R be the axis-aligned rectangle, depicted in Figure 7, of length $|pq| + r$ and width r, where r is the radius of the ice cream balls of p and q. We show that there exists a rectangle S_p of orientation ρ, with p at its focus point and with aspect ratio δ, such that (i) S_p is contained in R, and (ii) $q \in S'_p$, where S'_p is the rectangle contained in S_p, such that the distance between each of the sides of S'_p and the corresponding side of S_p is $w/4$, and w is the width of S_p. Since $S_p \subseteq R$ and $R \subseteq Q$, and since R_p is maximal, we know that $S_p \subseteq R_p$, and therefore q necessarily lies in R'_p which contains S'_p.

Consider now the second claim. Since the pair a, b was found by the algorithm, we know that both a and b lie in a rectangle, R'_a, that is a shrunk copy of a larger rectangle R_a (of aspect ratio δ), where, say, a lies at its focus point. We also know that the larger rectangle R_a (which is obtained from R'_a by moving each of its sides away from the center a distance of $w'/2$, where w' is the width of R'_a), is contained in Q. This implies, with the right choice of parameters, that there exists a constant c such that a, b are (α/c)-robustly visible to each other. □

Concerning the efficiency of the algorithm, it is easy to see that each pair that is reported by the algorithm above is rediscovered only a constant number of times ($O(1/\alpha)$), and therefore the total running time of the algorithm is $O(n + m \log mn + k')$ (resp., $O(n \log n + m \log mn + k')$) if Q is a simple polygon (resp., a polygon with holes).

THEOREM 3. *Let Q be a polygon, possibly with holes, consisting of n edges in total, and let $\mathcal{P}$ be a set of m points in Q. One can compute in $O(n \log n + m \log mn + k')$ time (or $O(n + m \log mn + k')$, if Q is simple) all pairs in $\mathcal{P}$ that are α-robustly visible, possibly together with some other pairs that are (α/c)-robustly visible, for some constant c. Here, k' is the number of (α/c)-robustly visible pairs.*

5. VISIBILITY DETECTION

We consider now the problem of detecting whether or not there is *any* pair of sites in $\mathcal{P}$ that see one another; i.e., we want to detect whether the edge set of $\mathrm{VG}_Q(\mathcal{P})$ is empty, and, if it is not, produce a witness visible pair. This problem of detecting "visibility independence" of a point set arises in applications of sensor coverage (guarding) and visibility-preserving terrain simplifications [8].

For a simple polygon Q visibility detection can be done by applying the simple version of the algorithm of Section 2, just applying a line segment intersection detection algorithm (for the dual segments) instead of a reporting algorithm. The algorithm terminates upon discovery of a visible pair. We thus have

THEOREM 4. *For a simple polygon Q having n vertices and a set $\mathcal{P}$ of m sites within Q, we can detect if the edge set of $\mathrm{VG}_Q(\mathcal{P})$ is empty in time $O(n + m \log m \log mn)$.*

The rest of the section is devoted to the special case of terrains in the plane, which arises in the visibility-preserving terrain simplification applications. For this case, we present a specialized algorithm which is slightly more efficient and which exploits the geometric properties of terrains.

Let T be a terrain of size n in the plane. That is, T is an x-monotone polygonal chain with n vertices. Let $\mathcal{P}$ be a set of m points above T. Our goal is to determine whether there exists a pair $\{p, p'\}$ of points in $\mathcal{P}$, such that p and p' see each other. We shall assume that between any two points in $\mathcal{P}$ there is at least one vertex of T, because otherwise there are clearly two points in $\mathcal{P}$ that see each other. (We can easily check in $O(n + m)$ time whether this assumption holds, assuming T and $\mathcal{P}$ are already sorted by x.)

Preliminaries

Our solution is based on the following claim.

Claim 1. *Let p and q be two points in $\mathcal{P}$, q to the right of p, that do not see each other. Let u be a vertex of T to the right of q that is visible from both p and q. Then for any point a (on or above T) to the right of u, if a is visible from q then it is also visible from p.*

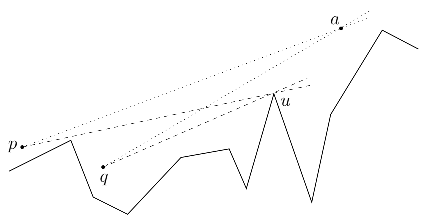

Figure 8. Proof of Claim 1.

Proof. Since p and q do not see each other and since u is visible from p, u must lie above the line through p and q; see Figure 8. Notice that for any point c above the line through p and q and to the right of q we have that segment $\overline{pc}$ is above segment $\overline{qc}$. Now let a be a point on or above T to the right of u, and assume a is visible from q. Then a must lie above the ray emanating from q and passing through u (and therefore it also lies above the ray emanating from p and passing through u). We need to show that the segment $\overline{pa}$ does not intersect T. Since $\overline{pa}$ lies above $\overline{qa}$, $\overline{pa}$ does not intersect T between $q.x$ and $a.x$, and since $\overline{pa}$ lies above $\overline{pu}$, $\overline{pa}$ does not intersect T between $p.x$ and $q.x$. $\square$

Corollary 1. *Under the conditions of the claim above, if we want to determine whether there exists a point $a \in \mathcal{P}$ to the right of u that is visible from either p or q, then it is enough to consider p; we may forget about q.*

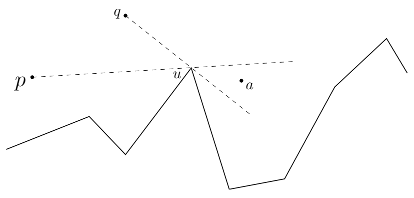

Figure 9. The requirement that p and q do not see each other is necessary.

Remark: The requirement that p and q do not see each other in the claim above is necessary, as can be seen in Figure 9. In this figure p and q can see each other, but there exists a point a that is visible from q and not from p. The reason why the requirement that p does not see q is necessary is that it forces u and the point a to be above the line through p and q.

The Algorithm

Our algorithm sweeps the input scene from left to right, with a vertical line l, until either a point of $\mathcal{P}$ that is visible from some point to its left is encountered, or the rightmost vertex of T is encountered. During the sweep the algorithm maintains a set $\mathcal{S}$ of rightward directed rays. Initially this set is empty; it is updated whenever l passes through a point in $\mathcal{P}$ or a vertex of T. For each point $p \in \mathcal{P}$ to the left of l, let ρ_p be the minimum angle ray that emanates from p and hits l before entering T. (The angle is measured with respect to the negative y-axis.) The set $\mathcal{S}$ is an appropriate subset of these rays. Let $\mathcal{P}_{\mathcal{S}}$ denote the subset of $\mathcal{P}$ corresponding to the rays in $\mathcal{S}$. With each ray $\rho_p \in \mathcal{S}$ we also keep the vertex of T that determines its angle.

The algorithm uses two auxiliary data structures. The first data structure D_1 stores a (dynamic) set of halfplanes H_1 and supports queries of the following form: Given a point a, determine whether a belongs to the intersection of the halfplanes in H_1. The set H_1 will be the set of halfplanes lying below the lines containing the rays in $\mathcal{S}$. Assuming a lies on l (and is on or above T), if a belongs to the intersection of the halfplanes in H_1, then it is not seen by any of the points in $\mathcal{P}_{\mathcal{S}}$, and therefore, as we shall see, it is not seen by any of the points in $\mathcal{P}$ to the left of l.

The second data structure D_2 stores a (dynamic) set of halfplanes H_2 and supports queries of the following form: Given a point w, find all halfplanes in H_2 containing w. The set H_2 will be the set of halfplanes lying above the lines containing the rays in $\mathcal{S}$. Assuming w lies on l (and is on or above T), then the reported halfplanes correspond to the points in $\mathcal{P}_{\mathcal{S}}$ that see w.

The best known bounds for the first data structure are due to Brodal and Jacob [9], who present a linear-size dynamic data structure that supports some basic queries on the convex hull of a set of points in the plane. Both the update time and the query time of their data structure is $O(\log m)$. As for the second data structure (which in the dual setting is simply a dynamic data structure for half-plane range searching), we have the following bounds using the results of Brodal and Jacob: Storage – $O(m)$, update – $O(m)$, Query – $O(\log m + k \log m)$.

We now describe the operations performed by the sweep algorithm for each of the two types of events.

l passes through a point a of $\mathcal{P}$. We first check whether a is seen by one of the points in $\mathcal{P}_{\mathcal{S}}$ by performing a query in the first data structure D_1. If a does not lie in the intersection of the halfplanes in H_1 (i.e., the halfplanes lying below the lines containing the rays in $\mathcal{S}$), then it is seen by some point in $\mathcal{P}_{\mathcal{S}}$ and we are done. Otherwise, we add the ray ρ_a to $\mathcal{S}$, where ρ_a is the ray emanating from a and passing through the vertex immediately to the right of a. We also update the sets H_1 (of D_1) and H_2 of D_2 accordingly (where H_2 is the set of halfplanes lying above the lines containing the rays in $\mathcal{S}$).

l passes through a vertex w of $\mathcal{T}$. We use the second data structure D_2 to find all k_w points in $\mathcal{P}_{\mathcal{S}}$ that see w. For each of these points p we need to update its corresponding ray ρ_p in $\mathcal{S}$, by computing the ray that emanates from p and passes through w. According to the claim above we may delete all these rays from $\mathcal{S}$, except for the one that forms the smallest angle to the right of w with respect to the downward vertical ray from w. (It is easy to see that this ray is necessarily the ray whose origin is the leftmost.) We thus update the sets H_1 and H_2 accordingly.

The Analysis

We first prove the correctness of the algorithm above. Then we analyze it to obtain bounds on its space and time consumption.

Lemma 3. *For any location of the sweep line l, let a be a point on l (on or above T) and let p be a point in $\mathcal{P}_{\mathcal{S}}$. Then a is visible from p if and only if ρ_p intersects l not above a.*

Proof. From the description of the algorithm it is clear that ρ_p is the minimum angle ray that hits l before entering T. $\square$

Lemma 4. *If there exist two points in $\mathcal{P}$ that see each other, then the algorithm will detect this fact.*

Proof. Let q be the leftmost point in $\mathcal{P}$ that is visible from a point of $\mathcal{P}$ to its left, and let $p \in \mathcal{P}$ be the leftmost point that sees q. We show that the pair $\{p, q\}$ will be detected by the algorithm. More precisely, we show that p is in $\mathcal{P}_{\mathcal{S}}$ when l reaches q. Indeed, if p is removed from $\mathcal{P}_{\mathcal{S}}$ before l reaches q, then the point $p' \in \mathcal{P}$, that is "responsible" for p's removal, must lie to the left of p and must see q, contradicting our assumption concerning p. $\square$

In order to bound the running time, we notice that a point in $\mathcal{P}$ is inserted only once to the set $\mathcal{P}_{\mathcal{S}}$, so the sum of k_w, over all vertices w of T, is $O(m)$.

Theorem 5. *Let T be a terrain in the plane of size n, and let $\mathcal{P}$ be a set of m points above T. Then one can detect if the edge set of $\mathrm{VG}_T(\mathcal{P})$ is empty in time $O((n+m)\log m)$ using $O(n+m)$ space.*

6. RECTANGLE VISIBILITY

Consider now the notion of *rectangle visibility*, in which we say that p and q are *r-visible* to one another if and only if the minimum enclosing axis-parallel rectangle, $R(p, q)$, lies inside Q. Based on an algorithm that combines our divide-and-conquer techniques with orthogonal range searching data structures, we establish the following result (whose proof is deferred to the full paper):

Theorem 6. *Let Q be a polygon, possibly with holes, consisting of n edges in total, and let $\mathcal{P}$ be a set of m points in Q. One can compute in $O(n \log n + m \log mn + k)$ time all k pairs in $\mathcal{P}$ that are r-visible to one another.*

7. INVISIBILITY GRAPHS

A visibility graph can be specified by listing its k edges (the visible pairs of sites); however, an alternative specification is to list the *complementary* graph edges, giving those $\bar{k}$ pairs of sites that are *invisible* to each other. Since $k + \bar{k} = \binom{m}{2}$, there are situations in which the visibility graph is particularly dense and it would be advantageous to compute the invisibility graph in output-sensitive time, depending on $\bar{k}$. (Agarwal et al. [3] have studied the related problem of compact representations of a visibility graph, as a union of a small number of complete subgraphs.)

We observe that the algorithms described in Section 3 (for computing the visibility graph and the distance-restricted visibility graph in a simple polygon) can be modified for the purpose of computing the invisibility graph. We only need to replace the wedge range searching (resp., the sector range searching, in the case of distance-restricted visibility) with range searching with the complement of a wedge (resp., with the complement of sector). We thus obtain:

Theorem 7. *In a simple polygon Q having n vertices, the invisibility graph of m points $\mathcal{P}$ within Q can be computed in time $\tilde{O}(n + m^{3/2} + \bar{k})$, where $\bar{k}$ is the number of invisible pairs of sites in $\mathcal{P}$. In addition, the invisibility graph can be computed within the same time bound in the case that each point in $\mathcal{P}$ has an associated range of sight, d_p.*

As for the algorithms in Section 3, we can improve the running time to $\tilde{O}(n + m^{4/3} + \bar{k})$ at the cost of increasing the storage space to $m^{4/3}$. We also note that in the unrestricted case, the segment tree and the range searching can be replaced by red-blue segment intersection, yielding the same $\tilde{O}$ bounds.

Acknowledgements

M. Katz and J. Mitchell are partially supported by grant No. 2000160 from the U.S.-Israel Binational Science Foundation. M. Katz and B. Ben-Moshe are partially supported by the MAGNET program of the Israel Ministry of Industry and Trade (LSRT consortium). J. Mitchell acknowledges support from Honda Fundamental Research Lab, NASA Ames Research (NAG2-1620), the National Science Foundation (CCR-0098172), and Metron Aviation. O. Hall-Holt is partially supported by the National Science Foundation (DMS-9983196).

References

[1] P. K. Agarwal. Partitioning arrangements of lines: II. Applications. *Discrete Comput. Geom.*, 5:533–573, 1990.

[2] P. K. Agarwal. Range searching. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 31, pages 575–598. CRC Press LLC, Boca Raton, FL, 1997.

[3] P. K. Agarwal, N. Alon, B. Aronov, and S. Suri. Can visibility graphs be represented compactly? *Discrete Comput. Geom.*, 12:347–365, 1994.

[4] P. K. Agarwal and J. Erickson. Geometric range searching and its relatives. In B. Chazelle, J. E. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational Geometry*, volume 223 of *Contemporary Mathematics*, pages 1–56. American Mathematical Society, Providence, RI, 1999.

[5] T. Asano, T. Asano, L. J. Guibas, J. Hershberger, and H. Imai. Visibility of disjoint polygons. *Algorithmica*, 1:49–63, 1986.

[6] F. Aurenhammer and R. Klein. Voronoi diagrams. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 201–290. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[7] J. Basch, L. J. Guibas, and G. D. Ramkumar. Reporting red-blue intersections between two sets of connected line segments. In *Proc. 4th Annu. European Sympos. Algorithms*, volume 1136 of *Lecture Notes Comput. Sci.*, pages 302–319. Springer-Verlag, 1996.

[8] B. Ben-Moshe, M. J. Katz, J. S. B. Mitchell, and Y. Nir. Visibility preserving terrain simplification – An experimental study. In *Proc. 18th Annu. ACM Sympos. Comput. Geom.*, pages 303–311, 2002.

[9] G. S. Brodal and R. Jacob. Dynamic planar convex hull. In *Proc. 43rd IEEE Sympos. Foundations of Computer Science*, pages 617–626, 2002.

[10] T. M. Chan. Dynamic planar convex hull operations in near-logarithmic amortized time. *Journal of the ACM*, 48(1):1–12, 2001.

[11] L. P. Chew and R. L. Drysdale, III. Voronoi diagrams based on convex distance functions. In *Proc. 1st Annu. ACM Sympos. Comput. Geom.*, pages 235–244, 1985.

[12] F. Chin, J. Snoeyink, and C. A. Wang. Finding the medial axis of a simple polygon in linear time. *Discrete Comput. Geom.*, 21(3):405–420, 1999.

[13] S. K. Ghosh and D. M. Mount. An output-sensitive algorithm for computing visibility graphs. *SIAM J. Comput.*, 20:888–910, 1991.

[14] L. J. Guibas and J. Hershberger. Optimal shortest path queries in a simple polygon. *J. Comput. Syst. Sci.*, 39(2):126–152, Oct. 1989.

[15] L. J. Guibas, J. E. Hershberger, J. S. B. Mitchell, and J. S. Snoeyink. Approximating polygons and subdivisions with minimum link paths. *Internat. J. Comput. Geom. Appl.*, 3(4):383–415, Dec. 1993.

[16] L. J. Guibas, M. H. Overmars, and M. Sharir. Intersecting line segments, ray shooting, and other applications of geometric partitioning techniques. In *Proc. 1st Scand. Workshop Algorithm Theory*, volume 318 of *Lecture Notes Comput. Sci.*, pages 64–73. Springer-Verlag, 1988.

[17] S. Har-Peled and M. Sharir. Online point location in planar arrangements and its applications. *Discrete Comput. Geom.*, 26:19–40, 2001.

[18] J. Hershberger. An optimal visibility graph algorithm for triangulated simple polygons. *Algorithmica*, 4:141–155, 1989.

[19] S. Kapoor, S. N. Maheshwari, and J. S. B. Mitchell. An efficient algorithm for Euclidean shortest paths among polygonal obstacles in the plane. *Discrete Comput. Geom.*, 18:377–383, 1997.

[20] D. T. Lee. Proximity and reachability in the plane. Report R-831, Dept. Elect. Engrg., Univ. Illinois, Urbana, IL, 1978.

[21] M. H. Overmars and E. Welzl. New methods for computing visibility graphs. In *Proc. 4th Annu. ACM Sympos. Comput. Geom.*, pages 164–171, 1988.

[22] M. Pocchiola and G. Vegter. Computing the visibility graph via pseudo-triangulations. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 248–257, 1995.

[23] M. Pocchiola and G. Vegter. Topologically sweeping visibility complexes via pseudo-triangulations. *Discrete Comput. Geom.*, 16:419–453, Dec. 1996.

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

[25] E. Welzl. Constructing the visibility graph for n line segments in $O(n^2)$ time. *Inform. Process. Lett.*, 20:167–171, 1985.