Shortest paths and visibility inside triangulated curved polygons with applications to conic visibility *

Jesús García-López
Dpto. Matemática Aplicada
E. U. Informática
Universidad Politécnica de Madrid
e-mail: jglopez@eui.upm.es

Pedro A. Ramos†
Dpto. Matemática Aplicada
Facultad de Informática
Universidad Politécnica de Madrid
e-mail: pramos@fi.upm.es

June 8, 1998

Abstract

In this paper we present linear time algorithms for computing the shortest path tree from a point and the weak visibility polygon of an arc inside a triangulated curved polygon. We present also a linear time algorithm for computing the planar subdivision (in the parametric space) of the set of lines emanating from a fixed arc, such that each face of the subdivision corresponds to rays hitting the same arc of the polygon. Although these results, which involve nontrivial generalizations of known results for rectilinear polygons, may have some interest on its own right, the main result of this paper is a linear time algorithm for computing the conic (circular, elliptic, parabolic and hyperbolic) visibility polygon of a point inside a simple polygon. The main advantage of our technique over previous results on circular visibility is that it provides a simple, unified approach to conic visibility. Finally, we present a linear time algorithm for computing the planar subdivision, in the parametric space, of two-parametric families of conic rays emanating from a fixed point, such that each face of the subdivision corresponds to conic rays hitting the same edge of the polygon. All these algorithms are asymptotically optimal.

Key words. Computational geometry, Planar subdivision, Shortest path, Ray shooting, Curved polygon, Curved triangulation, Circular visibility, Conic visibility.

1 Introduction

The problems of shortest paths and visibility inside simple polygons have been extensively studied in the last two decades. Actually, they are in the core of the origins of Computational Geometry. There are optimal algorithms for computing the shortest path tree from a point (Guibas *et al.* [12]), the visibility polygon from a point (ElGindy and Avis [8] and Lee [13]), the weak visibility polygon from an edge (Guibas *et al.* [12], Lee and Lin [14] and Toussaint [19]). There are also optimal algorithms to decide if a polygon is star-shaped (Lee and Preparata [15]), to decide if a polygon is weakly visible from an edge (Avis and Toussaint [3]) and to triangulate a polygon (Chazelle [4]). Finally, the problem of computing the first intersection of an arbitrary ray in $O(\log n)$ time has

*This research is partially supported by DGES grant PB96-0005-C02-01.

†Corresponding author

been solved with $O(n \log n)$ preprocessing time by Chazelle and Guibas [5] and with optimal $O(n)$ preprocessing time by Guibas *et al.* [12].

There are also a number of variations on this subject. Among them, *circular visibility* was introduced by Agarwal and Sharir [2]. Besides being a natural extension of linear visibility, circular visibility can model some physical systems, like trajectories of electrically charged particles in a uniform magnetic field. In that paper, the problem of computing the circular visibility polygon from a point is solved in $O(n \log n)$ time and the problem of computing the first intersection of a circular ray emanating from a fixed point is solved in $O(\log n)$ time with $O(n \log n)$ preprocessing. This results have been improved by Chou and Woo [6], who give a linear time algorithm for computing the circular visibility polygon from a point and by Agarwal and Sharir [1], who solve the general ray tracing problem, that is, computing the first intersection of a circular ray starting from an arbitrary point. The first intersection can be computed in $O(\log^4 n)$ time with $O(n \log^3 n)$ preprocessing.

Other possible variations involve to generalize the concept of rectilinear polygon to the concept of curved polygon. The definition of curved polygon that we are going to follow in this paper is the *splinegon* introduced by Dobkin and Souvaine [7]. There are other non-equivalent definitions like the *pseudo-polygons* introduced by O'Rourke and Streinu [17].

Generalizations of rectilinear visibility to circular visibility (and, more generally, to conic visibility) and to visibility inside curved polygons can be closely related via some geometric transformations as proposed in [9, 10]. In order to attack circular visibility from a point p, we consider the *inversion* with respect to p, whose analytic expression is

$$x^* = \frac{x - p}{||x - p||^2} + p.$$

As we shall see, the properties of this transformation will reduce the problem of circular visibility from a point p to the problem of linear visibility outside a polygon whose edges are circular arcs, because circles passing through p are transformed in lines, the boundary of the polygon is transformed in a polygon with circular arcs as edges and the interior of the polygon is mapped to the exterior. p is a singular point of the transformation that is mapped to ∞ and, therefore, the circular visibility polygon of p is mapped to the set of points that are linearly visible from ∞ outside the curved polygon.

In the same way, a circular ray emanating from p is mapped in a rectilinear ray emanating from ∞ and the problem of circular ray shooting from p is reduced to the problem of rectilinear ray shooting from an edge of a curved polygon.

This transformation can be generalized to attack elliptic, parabolic and hyperbolic visibility with the same technique, if we restrict ourselves to two-parametric families of conics passing through a fixed point. From this point of view, first-degree (linear) visibility, is generalized to second-degree (conic) visibility, which is a natural extension from the algebraic point of view. Moreover, the field of potential applications also grows, because conic visibility allows us to model a larger class of physical systems like, for example, parabolic trajectories of objects under the gravity or trajectories of charged particles inside electromagnetic fields created by other particles.

In this way, we reduce the problem of conic visibility in rectilinear polygons to the problem of linear visibility in curved polygons, which involve nontrivial generalizations of the techniques used to solve the problem in simple polygons and have some interest on its own right.

In order to compute visibility from an edge in a curved polygon, we shall need to compute the shortest path tree (which is, as well as in the rectilinear case, the key tool of the method) and the planar subdivision, in the parametric space of lines, such that each face of the subdivision corresponds to rays, with origin in an edge of the polygon, and hitting the same edge.

The rest of the paper is organized as follows: in the next section we introduce formal definitions and basics results, in section 3 we present an algorithm for computing pseudo-triangulations, in

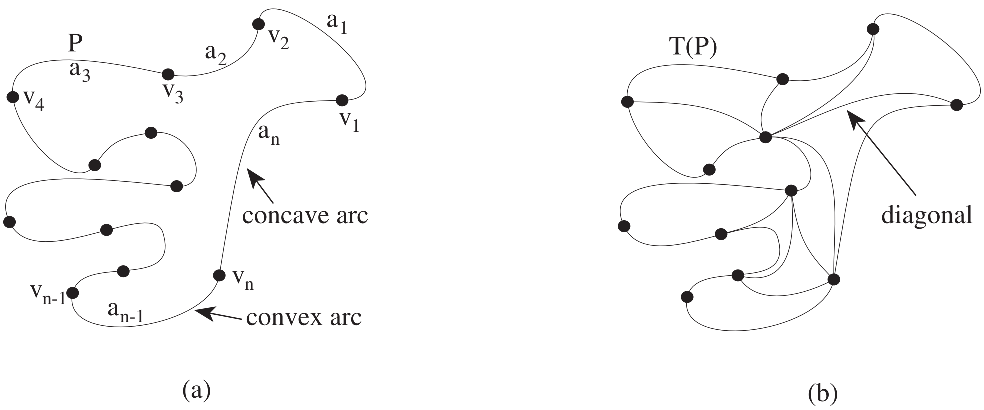


Figure 1: Curved polygon and curved triangulation.

section 4 we deal with the problem of computing the shortest path tree, in section 5 we apply this results to the original problems and, finally, we conclude in section 6.

2 Preliminaries

A *curved polygon* P is described by an ordered list of arcs $a_1, a_2 \ldots, a_n$ and the vertices of P are denoted $v_1, v_2, \ldots, v_n$, where $v_i = a_{i-1} \cap a_i$ (hereafter, indices are understood modulo n). The polygon is oriented in positive (counterclockwise) direction and we assume that the arcs $a_1, a_2 \ldots, a_n$ define a Jordan curve which is the boundary of the polygon and will be denoted ∂P. We also assume that each a_i is contained in the boundary of its own convex hull and say that an arc is convex if it rotates left or concave if it rotates right when described in the prescribed direction (see Figure 1.a).

A *curved triangulation* $T(P)$ of the curved polygon P is a partition of the polygon in $n-2$ curved triangles such that each arc of the triangulation is also contained in the boundary of its own convex hull (see Figure 1.b). The *diagonals* of the triangulation are the arcs which are not part of ∂P. Like in the linear case, a curved triangulation has $n-3$ diagonals and the dual graph is a tree, that we shall denote $D(T)$. However, unlike in the rectilinear case, there are curved polygons that do not have curved triangulations.

Given two points $p, q \in P$, the shortest path between p and q, denoted $\pi(p,q)$, is defined as the curve of minimum total length joining p and q without intersecting the exterior of P. Because P is simply connected, $\pi(p,q)$ is always unique. Moreover, it is easy to see that $\pi(p,q)$ is composed of a finite number of segments and curved arcs and that the curved arcs are always part of ∂P (see Figure 2.a). Given a point $p \in P$, the shortest path tree for p, denoted $\pi(p)$, is the union of the shortest paths from p to all the vertices of P (see Figure 2.b).

Lemma 1 *Let P be a curved polygon with n arcs and $p, q \in P$:*

i) The number of arcs of $\pi(p,q)$ is at most $2n+1$.

ii) The number of arcs of $\pi(p)$ is at most $3n$.

Proof: The intersection between $\pi(p,q)$ and each arc of ∂P is connected and, therefore, the number of curved arcs of $\pi(p,q)$ is at most n. For the rectilinear segments of the path, we observe

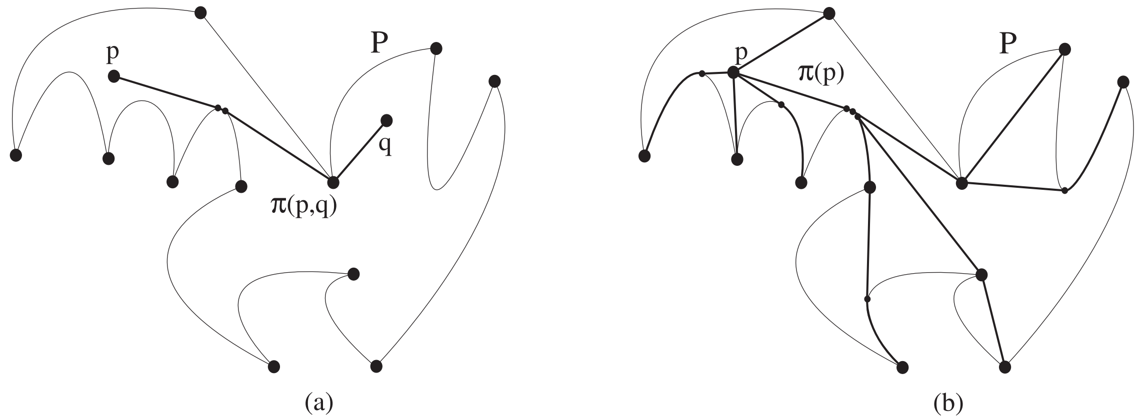


Figure 2: Shortest path and shortest path tree.

that they are separated either for vertices of P or for sub arcs of ∂P. Because the number of such vertices and arcs is at most n (a sub arc for each concave edge of ∂P and a vertex for each convex edge of ∂P), we conclude that the number of rectilinear arcs of $\pi(p,q)$ is at most $n+1$ and thus the total number of arcs is at most $2n+1$.

For the second part of the lemma, we are going to bound the number of vertices of $\pi(p)$, which has $n+1$ fixed vertices at p and the n vertices of the polygon and a number of *additional* vertices in the interior of concave arcs of ∂P. Let $\mathcal{E}$ the set of edges of $\pi(p)$ with at least one additional vertex and direct each edge of $\mathcal{E}$ from p to the leaves. We observe that the number of edges of $\mathcal{E}$ incident in a concave arc of ∂P is at most 2. Moreover, both edges are tangent to the arc and each of them is directed towards a different vertex of the arc. Finally, we observe that for two consecutive arcs, it is not possible to have edges directed to the common vertex in both arcs and that, if the common vertex has in degree one, none of them exists. Therefore, we have shown that $|\mathcal{E}| \leq n$ and that the number of additional vertices is $2n$. □

The main difficulty to compute shortest paths in a curved polygon using an incremental algorithm is that, unlike in the rectilinear polygon case, curved triangulations are not the appropriate tool. This becomes evident with the observation that diagonals are considered as part of the boundaries of intermediate sub polygons and are then barriers for visibility while this is not the case in the final polygon. Therefore, the final result can be incorrect when a diagonal is a concave edge of an intermediate polygon. In order to overcome this difficulty, we introduce the concept of *pseudo-triangulation*.

Given a curved polygon P and a triangulation $T(P)$, we call pseudo-triangulation of P associated to $T(P)$ to the partition $S(P)$ obtained by substituting each diagonal of $T(P)$ by the shortest path between the end-points of the diagonal. The number of faces of $S(P)$ is at most $n-2$ (some of the faces of $T(P)$ can collapse in this process, see Figure 3 for an example).

Lemma 2 *Let P be a curved polygon with n arcs and let $S(P)$ be a pseudo-triangulation of P. Then we have:*

i) The number of vertices of $S(P)$ is at most $3n-6$.

ii) The number of arcs of $S(P)$ is at most $4n-9$.

Proof: We present a constructive proof, which is also the key for the algorithm in the next

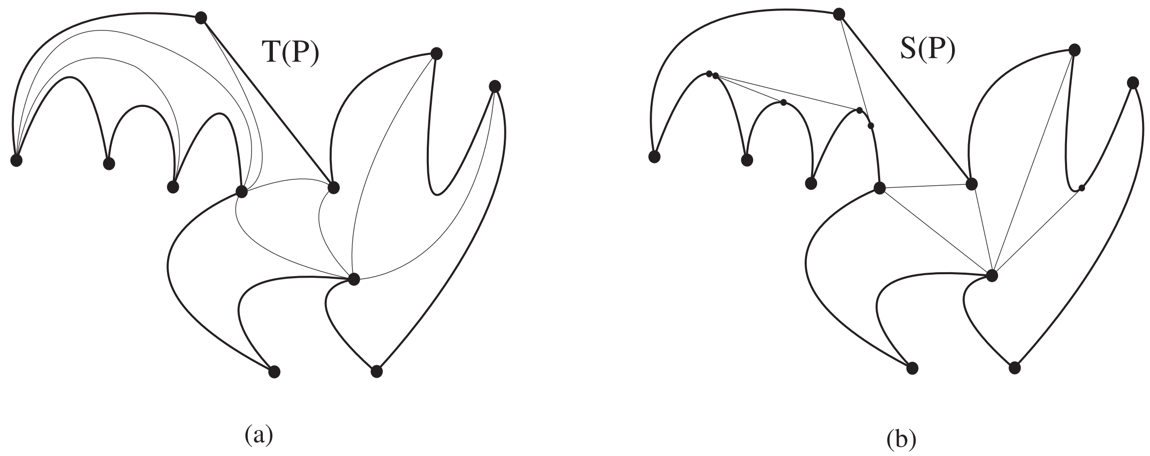


Figure 3: Curved triangulation and pseudo-triangulation.

section. Let $D(T)$ be the dual tree of the triangulation $T(P)$ with the edges oriented from the locally concave to the locally convex triangle.

We substitute each diagonal for the corresponding shortest path in the following way: we choose a sink of $D(T)$ (i.e. a vertex with out degree zero) and substitute the diagonal corresponding to one of its edges by the shortest path between the vertices. In this way, three new arcs and two new vertices can be generated (see Figure 4.a). The triangle becomes a *generalized triangle*, because one of its edges has been substituted by a concave chain.

If the diagonal that is going to be deleted belongs to a generalized triangle, the number of new vertices and arcs is the same (see Figure 4.b) and, therefore, the total number of generated arcs is, at most, $3(n-3)$ which, all along with the n arcs of the boundary, give the bound of $4n-9$. In the same way, the number of generated vertices is, at most, $2(n-3)$ for a total of, at most, $3n-6$ vertices. □

3 Pseudo-triangulations

In order to simplify the notation, here and hereafter we call polygons and triangulations to curved polygons and curved triangulations, respectively. The first step in order to compute the shortest path tree of a point in a triangulated polygon is to compute a pseudo-triangulations from the given triangulation. This process follow the same ideas introduced in the proof of Lemma 2.

Given a pseudo-triangulation of a polygon, the vertices of the polygon are called *principal vertices*, the faces of the pseudo-triangulation are called *generalized triangles* and the interior arcs are called *diagonal segments*. A *generalized diagonal* is a curve contained in the boundary of a generalized triangle t such that it does not contain any principal vertex in its interior, is contained in the boundary of its own convex hull (i.e. it is convex or concave as seen from t) and is maximal, i.e. it is not contained in any other curve with the same properties. Generalized triangles are defined by three generalized diagonals.

The algorithm proceeds by substituting diagonals in the triangulation by generalized diagonals, imitating the following physical process: in the beginning, all the boundary arcs and diagonals are considered rigid and then, one at a time, interior diagonals are considered as rubber bands with fixed end-points and let them reach the equilibrium state. This process gives us a pseudo-triangulation if the order of processing of the diagonals is carefully chosen from the directed dual graph of the

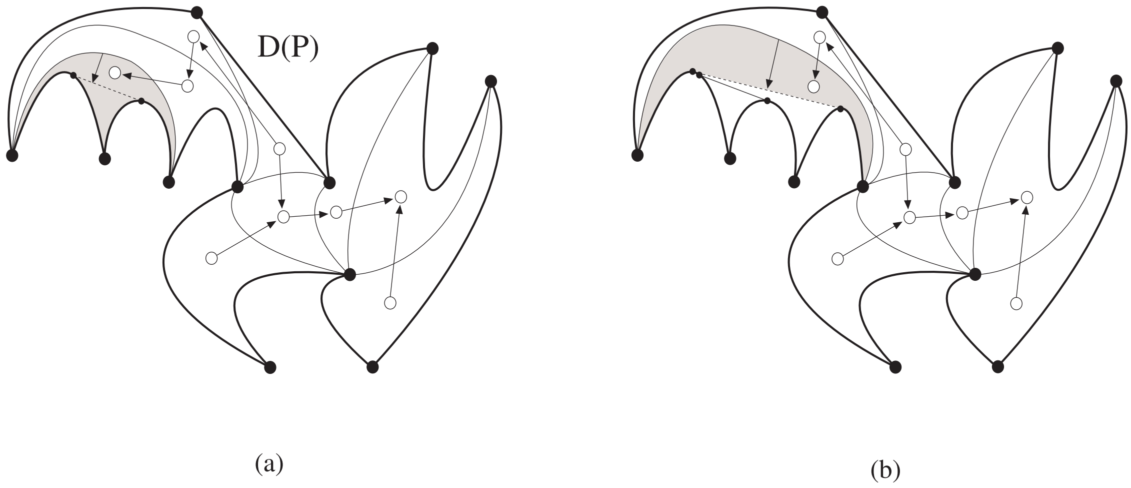


Figure 4: Illustration of the algorithm Pseudo_Triangulation.

triangulation.

Algorithm Pseudo_Triangulation

Input A polygon P with triangulation $T(P)$.

Output Pseudo-triangulation $S(P)$.

begin

Step 0 Initialize $S = T$.

Step 1 Compute the dual tree $D(T)$ and direct each edge from the locally concave triangle to the locally convex triangle.

Step 2 While $D \neq \emptyset$:

a) Select $v(t)$, a vertex of D with out degree zero (corresponding to the triangle $t \in T$).

b) Select $(u_1u_2)^\frown$, convex diagonal of t (u_3 denotes the remaining principal vertex).

c) Let x be the bridge of $\pi(u_1, u_2)$:

i) If $(u_1u_2)^\frown$ and $(u_2u_3)^\frown$ are convex in t (Figure 5.a), $x = \overline{u_1u_2}$ or $x = \overline{u_1u_3}, \overline{u_3u_2}$.

ii) If $(u_2u_3)^\frown$ is concave and $(u_3u_1)^\frown$ is convex (Figure 5.b), let z be the point of support of u_1 in $(u_2u_3)^\frown$. Then $x = \overline{zu_1}$.

iii) If $(u_2u_3)^\frown$ is convex and $(u_3u_1)^\frown$ is concave (Figure 5.c), let z be the point of support of u_2 in $(u_3u_1)^\frown$. Then $x = \overline{u_2}z$.

iv) If $(u_2u_3)^\frown$ and $(u_3u_1)^\frown$ are concave (Figure 5.d), let $\overline{z_1z_2}$ be the segment of support of $(u_2u_3)^\frown$ and $(u_3u_1)^\frown$. Then $x = \overline{z_1z_2}$.

d) In S, delete $(u_1u_2)^\frown$ and insert $\pi(u_1u_2)$. In D, delete $v(t)$.

end

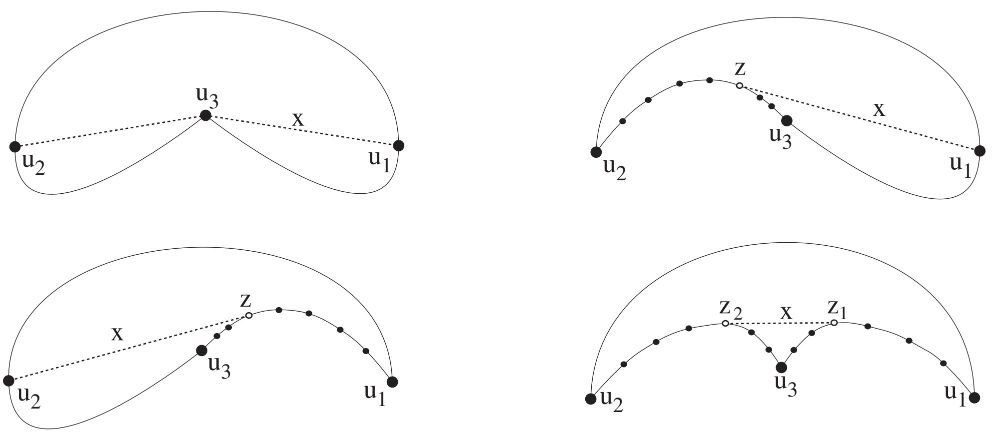


Figure 5: Computing generalized diagonals.

Theorem 1 *The algorithm Pseudo_Triangulation computes a pseudo-triangulation of P in optimal linear time.*

Proof: In order to prove the correctness of the algorithm, we make the following observations:

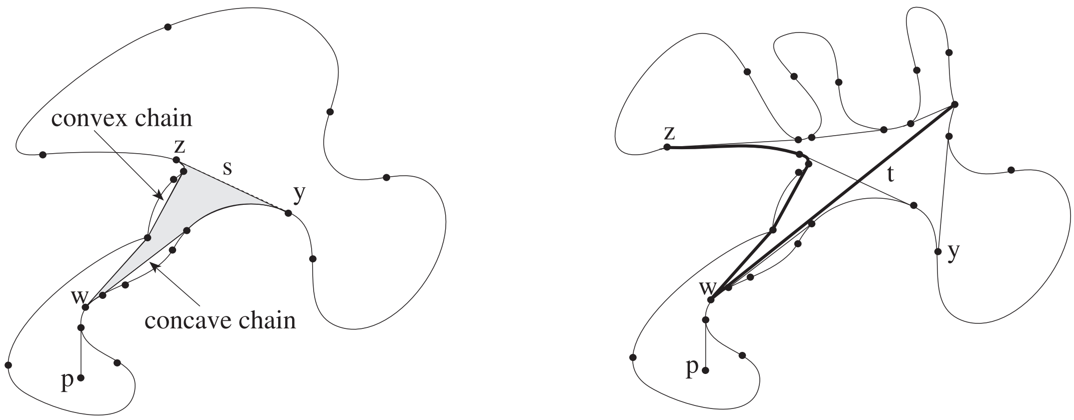
- Step 2.a. In a nonempty, directed tree there always exists at least one sink.
- Step 2.b. D has no vertices with degree zero because of step 2.d and $v(t)$ has out degree zero. Therefore, t has at least one convex diagonal.
- Step 2.c. In order to show that $\pi(u_1, u_2)$ is computed correctly, we start by observing that $\pi(u_1, u_2)$ is inside t. Suppose, on the contrary, that there exists $y \in \pi(u_1, u_2)$ such that $y \notin t$. Then, $\pi(u_1, u_2)$ cuts twice an arc of t that has to be a concave (from t) diagonal, contradicting the fact that the out degree of $v(t)$ is zero. Finally, we observe that the computation of $\pi(u_1, u_2)$ is reduced to compute lines of support because generalized diagonals are convex or concave.

The rest of the steps of the algorithm are obvious. Furthermore, because all the diagonals are substituted by shortest paths, the result is a pseudo-triangulation of P.

The complexity of step 1 is $O(n)$. For the step 2, we observe that, in each iteration, the complexity of 2.a and 2.b is $O(1)$, 2.c has complexity proportional to the number of arcs between u_3 and the point(s) of support and 2.d has complexity proportional to the degree of $v(t)$. Only 2.c requires an additional observation: the arcs processed when looking for the segment x are not processed again, because they cannot belong to generalized diagonals of generalized triangles that will be processed later on. Therefore, the complexity of step 2 is also linear. We close the proof observing that, because S has linear size, the algorithm is asymptotically optimal. $\square$

4 Shortest path tree

In this section we present an incremental algorithm for computing the shortest path tree of a point inside a polygon following the ideas of Lee and Preparata [16] and Guibas *et al.* [12]. First, we compute the tree inside the generalized triangle containing the point, taking into account all the vertices of the pseudo-triangulation. Then, we extend the tree to an adjacent generalized triangle, until the whole polygon has been processed.



(a) (b)

Figure 6: Examples of funnels.

Let y and z the end-points of a diagonal segment s and w the point where curves $\pi(p, y)$ and $\pi(p, z)$ separate. The *funnel* of s, denoted $f(s)$, is the pair of curves $\pi(w, y)$ and $\pi(w, z)$, w is the *vertex* of the funnel and s the *lid* of the funnel. When oriented from the vertex, one of the curves of $f(s)$ is convex and the other one is concave (see Figure 6.a). The funnel of a sub arc of a generalized diagonal is defined in an analogous way (see Figure 6.b).

Let t be the generalized triangle adjacent to $f(s)$. The main step of the algorithm is to compute, starting from $f(s)$, the shortest paths to all the vertices of t. Let $(u_1u_2)\hat{}$ be the generalized diagonal containing s. First, we add $(zu_1)\hat{}$ and $(yu_2)\hat{}$ to $f(s)$ (see Figure 7.a) and then we compute the shortest paths to the rest of vertices of t, starting from the third vertex of t (see Figure 7.b).

In order to achieve linear complexity, we need to represent the funnels as *finger trees*, introduced by Guibas *et al.* [11]. Let L be an ordered list with n elements, L_1 is the sublist with the k first elements of L and L_2 the sublist containing the rest of the elements of L. If F, F_1 and F_2 denote finger trees storing these lists, the following operations take $O(\log(\min\{k, n-k\}))$ time:

- Find the k-th element in F;
- Split F into F_1 and F_2;
- Merge F_1 and F_2 to obtain F.

Algorithm Shortest_Path_Tree

Input A polygon P, a point $p \in P$ and a pseudo-triangulation $S(P)$.

Output The shortest path tree $\pi(p)$.

begin

Step 0 Initialize $\pi = \{p\}$ and $F = \emptyset$.

Step 1 Locate the generalized triangle t_0 containing p.

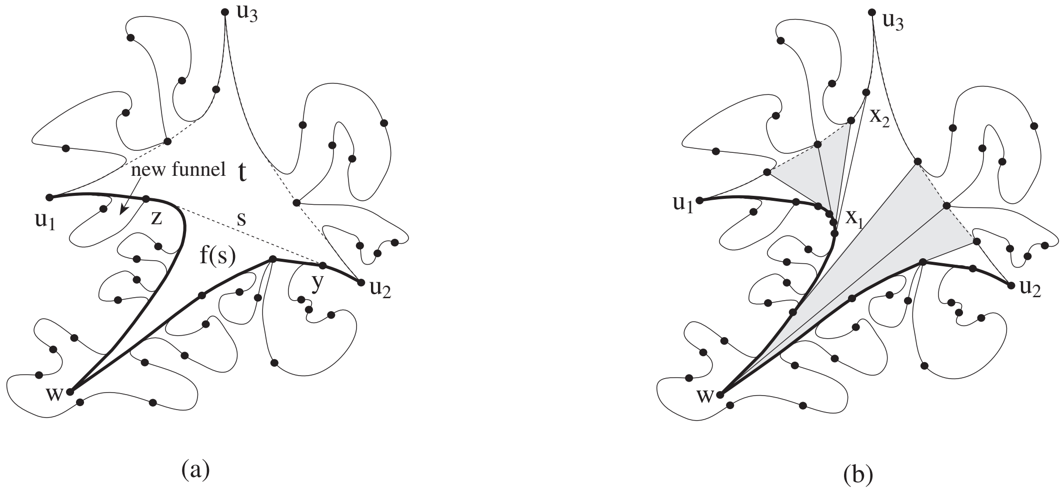


Figure 7: Computing shortest paths.

Step 2 Compute the shortest paths from p to the vertices of t_0.

Step 3 For each diagonal segment $s \in \partial t_0$, compute $f(s)$ and insert it in F.

Step 4 While $F \neq \emptyset$:

a) Select a funnel $f(s) \in F$ (we assume that $s = \overline{yz}$ with the positive orientation of $f(s)$).

b) Select the generalized triangle t adjacent to $f(s)$. Let u_1, u_2 and u_3 be the vertices of t and assume that $s \in (u_1 u_2)^\frown$.

c) Let C_1 and C_2 be the chains of ∂t between y and u_2 and between z and u_1, respectively (observe that C_1 has positive orientation while C_2 has negative orientation). Insert C_1 and C_2 in $\pi(p)$ and $f(s)$.

d) Construct the funnels corresponding to the diagonal segments of C_1 and C_2 and insert them in F.

e) For u_3:

i) Compute $\pi(w, u_3)$, where w is the vertex of $f(s)$:

α) Compute the bridge $\overline{x_1 x_2}$ defining $\pi(w, u_3)$ (see Figure 7.b).

β) Divide the funnel $f(s)$ into $f(s_1)$ and $f(s_2)$ in such a way that $f(s_1)$ is on the right of $\pi(w, u_3)$ (oriented from w to u_3).

ii) Let C_3 be the chain of $\pi(w, u_3)$ between x_2 and u_3. Construct the funnels corresponding to the segments of C_3 and add them to F.

iii) Iterating the process of step 4.e.i, compute the shortest paths between w and the vertices of the arc $(u_1 x_2)^\frown$ inserting in F the funnels that are generated in the process.

iv) Repeat previous step for the vertices of the arc $(u_2 u_3)^\frown$.

end

Theorem 2 *The algorithm Shortest_Path_Tree computes $\pi(p)$ in optimal linear time.*

Proof: In order to prove the correctness of the algorithm, we make the following observations:

- Step 4.b. Because s is a diagonal segment, there exists a triangle t adjacent to s.

- Step 4.c. In order to add C_1 and C_2 to $f(s)$, it is enough to construct finger tree for these chains and add them to the finger tree of $f(s)$.

- Step 4.e.i. We compute the segments of support between $(u_2u_3)^\frown$ and $(u_3u_1)^\frown$ to $f(s)$ and choose the appropriate one.

- Step 4.e.ii. We proceed in the same way as in step 4.c.

- Step 4.e.iii. This step is a repeated application of step 4.e.i.

The complexity of steps 1, 2 and 3 is clearly $O(n)$. For step 4, we first analyze the complexity of a single iteration: steps 4.a and 4.b have $O(1)$ complexity, while steps 4.c and 4.d have complexity $O(|C_1|+|C_2|)$. Furthermore, the step 4.e.i requires $O(|(u_2u_3)^\frown|+|(u_3u_1)^\frown|+\min\{\,|f(s_1)|,|f(s_2)|\,\})$ time and, finally, the step 4.e.ii has complexity $O(|C_3|)$. We shall deal with the complexity of step 4.e.iii together with the global complexity of step 4.e.i.

The global complexity of steps 4.a and 4.b is $O(n)$. Steps 4.c and 4.d have also $O(n)$ global complexity because the chains C_1 and C_2 take part in a single iteration and the $O(n)$ funnels have constant complexity. For the same reasons, the overall complexity of step 4.e.ii is $O(n)$.

In step 4.e.i we compute the segment of support between $f(s)$ and the curve γ, where $\gamma = (u_1u_2)^\frown \cup (u_2u_3)^\frown$ if both are concave in t, $\gamma = (u_2u_3)^\frown$ if only $(u_2u_3)^\frown$ is concave, $\gamma = (u_3u_1)^\frown$ if only $(u_3u_1)^\frown$ is concave and $\gamma = \{u_3\}$ if both are convex. In order to do this efficiently, we need to construct a tree for γ and, after that, the segment of support can be computed in $O(\log(|\gamma|) + \log(\min\{|f(s_1)|,|f(s_2)|\}))$ time. The global complexity of $O(\log(|\gamma|)$ is $O(n)$ because each diagonal is processed at most once. In step 4.e.iii we compute the segments of support of the funnels corresponding to the vertices of $(u_2u_3)^\frown$ and $(u_3u_1)^\frown$, so it only remains to analyze the complexity of the subdivision of the funnels.

This analysis is analogous to the one in [12]: let $T(n,m)$ be the number of operations of dividing all the funnels, starting with a funnel with m arcs and n pending divisions. This function verifies the following recursive relation:

$$T(n,m) \leq T(n_1,m_1) + T(n_2,m_2) + c\log(\min\{\,m1, m2\,\})$$

where c is a positive constant and logarithms are in base 2. Furthermore, we have that $n_1+n_2 = n-1$ and $m_1+m_2 \leq m+4$. Taking into account that for the cases with $n+m \leq 7$, $T(n,m) \leq 1$, we can use induction to show that

$$T(n,m) \leq \begin{cases} 1 & \text{if } n+m \leq 7 \\ c(n+m-\log m-4) & \text{otherwise} \end{cases}$$

and, therefore the total complexity of steps 4.e.i and 4.e.iii is $O(n)$.

Finally, it is obvious that the algorithm is asymptotically optimal because $\pi(p)$ has linear size. □

5 Applications

Shortest path trees in curved polygons can be applied to compute the weak visibility polygon from an arc and to solve the ray tracing problem. They can also be applied to compute conic (circular, elliptic, parabolic or hyperbolic) visibility polygons of a point inside a rectilinear polygon and to solve the ray tracing problem for conic rays emanating from a fixed point.

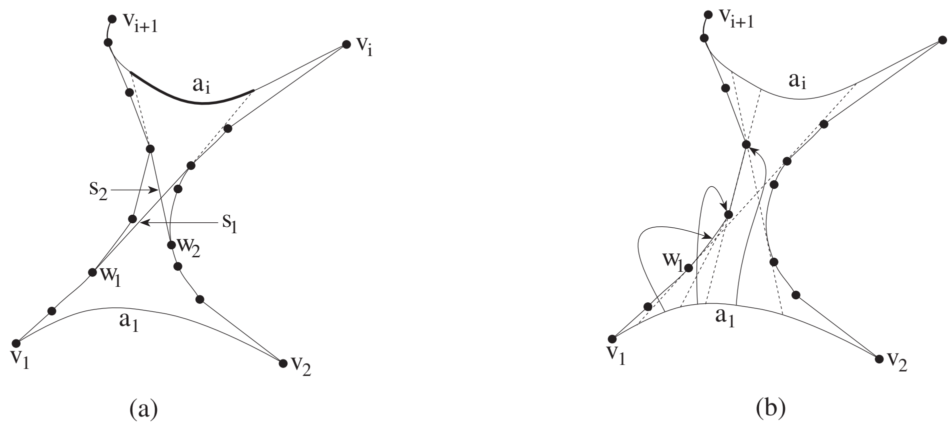


Figure 8: Contribution of a_i to $V(a_1)$ and subdivision of a_1.

5.1 Weak visibility polygon from an arc

We say that the point q is weakly visibly from the arc a if there exists a point $p \in a$ such that p and q are visible. The weak visibility polygon of a, denoted $V(a)$, is the set of points that are weakly visible from a. $V(a)$ is a curved polygon whose boundary contains segments and arcs and sub arcs of ∂P. Furthermore, arcs of ∂P appear in $V(a)$ in the same order as in P and each arc of P contributes to $V(a)$ with at most one arc. Hereafter, we assume that $a = a_1$. The following observation, which is well known for rectilinear polygons, can be easily generalized to curved polygons: a_i contributes to $V(a_1)$ if and only if $\pi(v_1, v_{i+1}) \cap \pi(v_2, v_i) = \emptyset$. Furthermore, let w_1 be the last common point of $\pi(v_1, v_i)$ and $\pi(v_1, v_{i+1})$ and w_2 the last common point of $\pi(v_2, v_i)$ and $\pi(v_2, v_{i+1})$. Let s_1 and s_2 be the first segments of $\pi(w_1, v_i)$ and $\pi(w_2, v_{i+1})$, respectively. Then, the sub arc of a_i visible from a_1 can be obtained computing the first intersections of the rays defined by s_1 and s_2 with a_i (see Figure 8.a).

With this observations and the previous lemma in mind, we can use $\pi(v_1)$ and $\pi(v_2)$ in order to compute $V(a_1)$ in linear time. Then we have shown:

Theorem 3 *The problem of computing the weak visibility polygon from an arc of a triangulated curved polygon has linear complexity.*

5.2 Ray tracing

The set of lines in the plane is a biparametric family. Although it is not well defined for vertical lines, the most common parameterization maps the line $r : y = ax + b$ to the point (a, b). In order to solve the ray tracing problem, we compute the *visibility diagram* of the polygon, which is a subdivision of the parametric space of lines, such that each face of the subdivision corresponds to rays hitting the same arc of the polygon. Therefore, the problem of ray tracing is reduced to the problem of point location in the visibility diagram. In this section we show how the visibility diagram of an arc (say a_1) of a curved polygon can be computed in linear time. First, we need some observations.

The rays hitting a_i separate the convex chains $\pi(v_1, v_{i+1})$ and $\pi(v_2, v_i)$. This set of lines can be described by subdividing the arc a_1 into sub arcs such that in each sub arc the tangents to the convex chains are supported by the same element (vertex or arc) of the chains (see Figure 8.b). In order to compute this subdivision, we only need to travel along the chains $\pi(w_1, v_{i+1})$ and

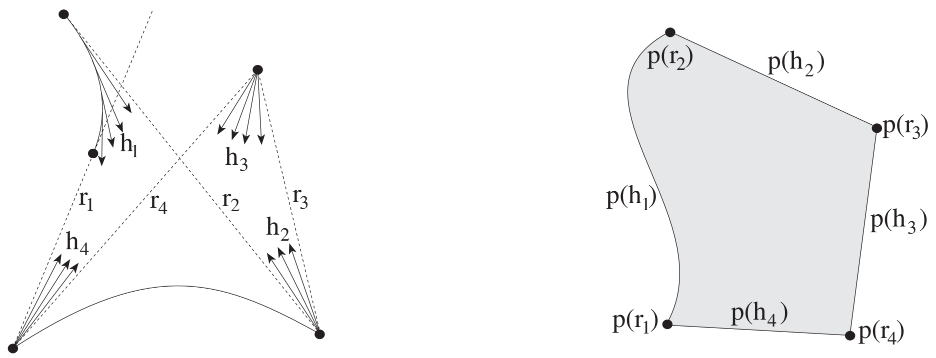


Figure 9: Constructing the subdivision.

$\pi(w_2, v_i)$. The complexity of this process is $O(|f_1(a_i)| + |f_2(a_i)|$, where $f_1(a_i)$ and $f_2(a_i)$ are the funnels defined by $\pi(w_1, v_i)$ and $\pi(w_1, v_{i+1})$ and by $\pi(w_2, v_i)$ and $\pi(w_2, v_{i+1})$ respectively. The following lemma shows that this process has linear complexity:

Lemma 3 ([12]) *Given the shortest path tree of a point p inside a curved polygon P with n arcs, and the funnels $f(a_i)$ defined as above, we have that $\sum_{i=2}^{n} |f(a_i)| = O(n)$.*

We observe that the lines hitting a_i and emanating from a sub arc of a_1 correspond, in the parametric space, to a curved quadrilateral. The arcs of the boundary of this quadrilateral are the curves corresponding to the families of lines tangent to the same element of the convex chains and the lines emanating from an end-point of the sub arc (see Figure 9. Computing the union of the quadrilaterals defined by a_i we obtain the face of the subdivision corresponding to the lines hitting a_i and we repeat the process for the rest of the arcs of the polygon.

From the previous lemma and these observations, we have that the subdivision can be computed in $O(n)$ time. Then, we have shown:

Theorem 4 *The visibility diagram of an arc of a triangulated curved polygon can be computed in linear time.*

5.3 Conic visibility

Let C be a family of conics in the plane. We say that two points p and q are C-visible inside a *rectilinear* polygon P if there exists a conic $c \in C$ passing through both points and such that at least one of the (possibly two) arcs joining p and q is contained inside P. If C is the set of circles, we are dealing with circular visibility. If C is other three-parametric family of conics, such us axis parallel parabolas, axis parallel ellipsis with constant eccentricity or axis parallel hyperbolas with constant eccentricity, we obtain some variations of conic visibility which, up to our knowledge, were first introduced by García [9].

The key idea is to use properties of conics in order to establish a one to one correspondence between curves of the family C passing through a fixed point and lines of the plane. More specifically, assume that p is the origin of the coordinate system and the conics are parallel to the coordinate axis. For the parabolic case, the transformation

$$(x, y) \longmapsto (x, y)^* = \frac{1}{x^2}(x, y)$$

maps the parabola $y = ax^2 + bx$ to the line $y = bx + a$, and conversely. For the elliptic and hyperbolic case, if $b \neq 0$, the transformation

$$(x, y) \longmapsto (x, y)^* = \frac{1}{x^2 + by^2}(x, y)$$

maps the ellipse or hyperbola

$$x^2 + by^2 = 2x_0 x + 2by_0 y$$

(centered at $(x_0 y_0)$ and passing through the origin) to the line

$$y = -\frac{x_0}{by_0}x + \frac{1}{2by_0},$$

and conversely. Therefore, these transformations map a conic ray passing through p to rectilinear rays and the region which is C-visible from p inside the polygon P to the region outside P^* which is linearly visible from ∞. It is worth noting that if we want to compute parabolic visibility in order to study trajectories of objects under the gravity, we have to restrict ourselves to parabolas $y = ax^2 + bx$ with $a < 0$ and, therefore, we have to restrict linear visibility in the $*$-plane to lines $y = bx + a$ with $a < 0$.

Thus, the steps for computing the C-visibility polygon of a point p inside a rectilinear polygon P, denoted $V_C(p)$, are the following:

a) Compute P^*.

b) Compute $CH(P^*)$ (see [9, 18]).

c) Substitute the lids of the pockets of $CH(P^*)$ for arcs of conics of the family C passing through p and not intersecting ∂P^*.

d) Triangulate the pockets. To do this, we come back to the original polygon, triangulate it in linear time using Chazelle's algorithm [4] and return to the *-plane.

e) Compute the weak visibility polygons from the lids of the pockets and, using that, the set of points outside P^* that are visible from ∞, denoted $V_{P^*}(\infty)$.

f) Compute $V_C(p) = (V_{P^*}(\infty))^*$.

In order to solve the ray tracing problem from a fixed point, we compute the C-visibility diagram of the point in an analogous way: steps a-e are the same and, after that, we compute the visibility diagram for rectilinear rays emanating from ∞. To do so, we compute separately the rays that hit an arc of $CH(P^*)$, the rays that hit a lid of a pocket and the rays that do not intersect P^*.

We summarize the results of this subsection in the following theorem:

Theorem 5 *Let C be a family of axis parallel, constant eccentricity conics. The C-visibility polygon and the C-visibility diagram of a point inside a simple polygon can be computed in linear time.*

6 Concluding remarks

In this paper we have presented optimal linear time algorithms for shortest paths and visibility problems inside triangulated curved polygons and for conic visibility inside rectilinear polygons. We have shown that these two problems are equivalent if we restrict ourselves to conics passing through a fixed point.

We conclude with two open problems:

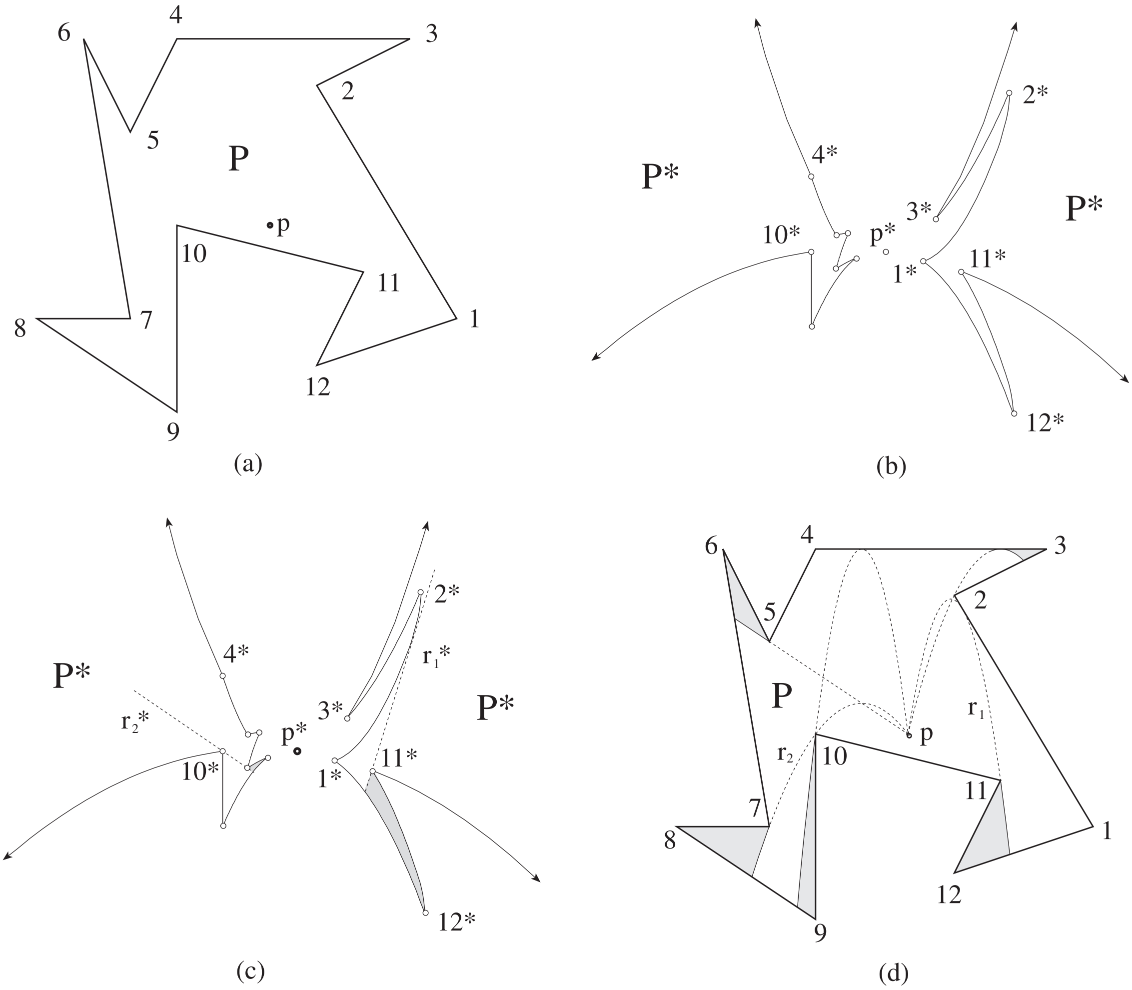


Figure 10: Downward parabolic visibility from p. In this case, $(\partial P)^*$ is not connected and P is mapped to the regions of the $*$-plane not containing the OY-axis (figure (b)). In figure (c) two rays defining regions that are not visible from ∞ are shown. Finnaly, figure (d) shows the parabolic rays defining the boundary of the downward parabolic visibility polygon, which is the not-shaded part of P.

- The ray shooting problem in curved polygons when rays can emanate from an arbitrary edge may be solved using similiar techniques.
- These techniques do not seem to be useful in the circular ray shooting problem when rays can emanate from an arbitrary point so the general conic ray shooting problem needs some different ideas.

7 Acknowledgements

We thank Prof. G. Toussaint for bringing the circular visibility problem to our attention.

References

[1] P. K. Agarwal and M. Sharir. Circle shooting in a simple polygon. *J. Algorithms*, 14:69–87, 1993.

[2] P. K. Agarwal and M. Sharir. Circular visibility of a simple polygon from a fixed point. *Internat. J. Comput. Geom. Appl.*, 3:1–25, 1993.

[3] D. Avis and G. T. Toussaint. An optimal algorithm for determining the visibility of a polygon from an edge. *IEEE Trans. Comput.*, C-30:910–1014, 1981.

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

[5] B. Chazelle and L. J. Guibas. Visibility and intersection problems in plane geometry. *Discrete Comput. Geom.*, 4:551–581, 1989.

[6] S. Y. Chou and T. C. Woo. A linear-time algorithm for constructing a circular visibility diagram. *Algorithmica*, 14:203–228, 1995.

[7] D. P. Dobkin and D. L. Souvaine. Computational geometry in a curved world. *Algorithmica*, 5:421–457, 1990.

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

[9] J. García-López. *Problemas algorítmico-combinatorios de visibilidad.* Ph.D. thesis (in spanish), Facultad de Informática, Universidad Politécnica de Madrid, Madrid, Spain, 1995.

[10] J. García-López and P. Ramos. Circular visibility and separability. In *Proc. 5th Canad. Conf. Comput. Geom.*, pages 18–23, 1993.

[11] L. Guibas, E. M. McCreight, M. F. Plass, and J. R. Roberts. A new representation for linear lists. In *Proc. 9th Annu. ACM Sympos. Theory Comput.*, pages 49–60, 1977.

[12] L. J. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2:209–233, 1987.

[13] D. T. Lee. Visibility of a simple polygon. *Comput. Vision Graph. Image Process.*, 22:207–221, 1983.

[14] D. T. Lee and A. K. Lin. Computing the visibility polygon from an edge. *Comput. Vision Graph. Image Process.*, 34:1–19, 1986.

[15] D. T. Lee and F. P. Preparata. An optimal algorithm for finding the kernel of a polygon. *J. ACM*, 26:415–421, 1979.

[16] D. T. Lee and F. P. Preparata. Euclidean shortest paths in the presence of rectilinear barriers. *Networks*, 14:393–410, 1984.

[17] J. O'Rourke and I. Streinu. Pseudo-visibility graphs in pseudo-polygons: Part I. Technical Report 041, Dept. Comput. Sci., Smith College, Northampton, MA, January 1996. Slightly revised Apr. 1996.

[18] A. A. Schäffer and C. J. Van Wyk. Convex hulls of piecewise-smooth Jordan curves. *J. Algorithms*, 8:66–94, 1987.

[19] G. T. Toussaint. A linear-time algorithm for solving the strong hidden-line problem in a simple polygon. *Pattern Recogn. Lett.*, 4:449–451, 1986.