SCHOOL OF OPERATIONS RESEARCH
AND INDUSTRIAL ENGINEERING
COLLEGE OF ENGINEERING
CORNELL UNIVERSITY
ITHACA, NEW YORK 14853

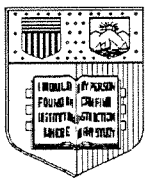
TECHNICAL REPORT NO. 746

June 1987

AN OPTIMAL VISIBILITY ALGORITHM FOR A SIMPLE POLYGON WITH STAR-SHAPED HOLES

By

Esther M. Arkin
Joseph S. B. Mitchell



AN OPTIMAL VISIBILITY ALGORITHM FOR A SIMPLE POLYGON WITH STAR-SHAPED HOLES

Esther M. Arkin and Joseph S. B. Mitchell *

School of Operations Research and Industrial Engineering
Cornell University
Ithaca, NY 14853

Abstract

We give a $\Theta(n + h \log h)$ algorithm for finding the visibility polygon from a point inside a simple polygon with h star-shaped polygonal holes, where n is the total number of vertices describing the bounding polygon and the polygonal holes.

1. Introduction

The visibility polygon problem in the plane is fundamental in computational geometry, with applications in graphics (the hidden-line elimination problem) and shortest path planning. Many researchers have studied the problem of finding the visibility polygon from a point s inside a simple polygon. This can be done in optimal $\Theta(n)$ time [EA, Le83]. Also, the problem in which there are a collection of disjoint simple polygonal "holes" inside the simple polygon has been studied, and there exist algorithms achieving $O(n \log n)$ time [As, Le78, SO], where n is the total number of vertices describing the scene. If we let h be the number of holes, then a simple reduction from sorting gives a lower bound of $\Omega(n + h \log h)$ on the time complexity of finding the visibility polygon [AAGHI, SO]. Since h may be $o(n)$ (and in fact may be *much* smaller than n in practice), it is desirable to find algorithms whose complexity is written in terms of h. In the case of *convex* holes, an optimal algorithm is known [As, AAGHI, LC, Mi]. In the case of simple polygonal holes, it is easy to arrive at an $O(n \log h)$ algorithm [As]. The question remains open whether or not this running time can be improved to match the lower bound of $\Omega(n + h \log h)$.

In this paper we make some progress towards addressing this problem by giving a partial answer: an optimal $\Theta(n + h \log h)$ algorithm for the special case of disjoint *star-shaped* polygonal holes, which is an improvement over the previous $O(n \log h)$ algorithm. Our algorithm exploits the special structure of star-shaped holes and uses the fact that the visibility "profile" of a simple polygon can be obtained in linear time. Since star-shapedness is a generalization of convexity, it seems natural to consider this case. By providing an optimal algorithm for the star-shaped case, we come one step closer to our goal of an optimal time algorithm for the general case of simple polygonal holes.

---

* J. Mitchell is also with the Hughes Artificial Intelligence Center, Hughes Aircraft Company.

Many visibility algorithms require sorting the vertices (say, in their x-coordinates). However, in looking for an optimal algorithm, we must of course avoid sorting the vertices of the polygonal holes (as this will require $\Omega(n \log n)$). We are allowed to sort only $O(h)$ points. Thus, what we are trying to do, in a sense, is "sort" the polygons instead of sorting all of their vertices. With convex polygons, there is a natural definition of an ordering based on whether or not a polygon is "above" another. With nonconvex polygons, it is not clear how to define an ordering of the polygons which will be meaningful for solving visibility problems.

2. Preliminaries

We begin with some definitions. We consider simple polygons to be *closed*. A simple polygon is specified by giving a list of its vertices in order (say, clockwise) around the boundary. We can assume that polygons are given in a doubly-linked list structure. Two points within a simple polygon are said to be *visible* from each other if the straight line segment between them is entirely contained within the polygon. A *star-shaped polygon* is a simple polygon for which there exists a *center*, point C, contained within the polygon such that all vertices of the polygon are visible from C. (This will imply that *all* points in polygon are visible from C.) The set of all possible centers is called the *kernel* of a simple polygon. If the kernel is nonempty, then the polygon is star-shaped. Refer to Figure 1. It is possible to check for star-shapedness by computing the kernel in linear time [LP]. A convex polygon is the special case of a star-shaped polygon in which the kernel is the entire polygon.

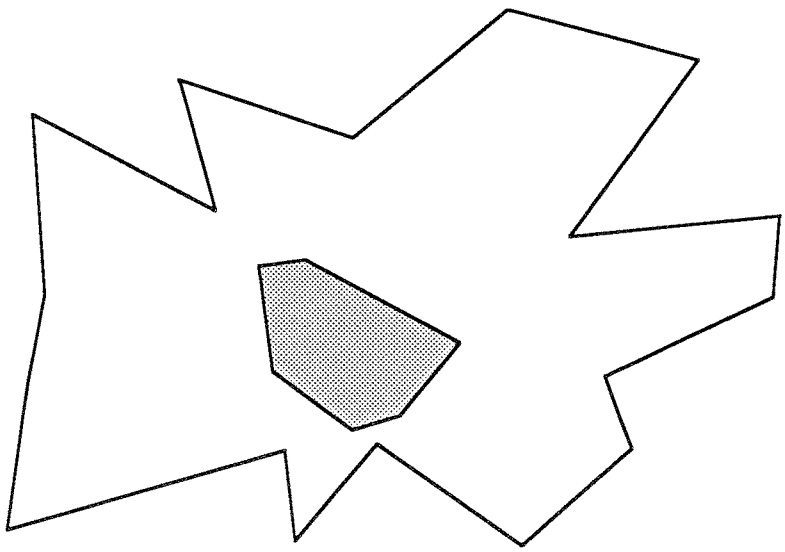


Figure 1. A star-shaped polygon and its kernel.

We are given a (possibly unbounded) simple polygonal "room", P, which contains a set of disjoint star-shaped polygonal "holes", $P_1, \ldots, P_h$. See Figure 2. We are also given a point s from which we are to compute the visibility polygon.

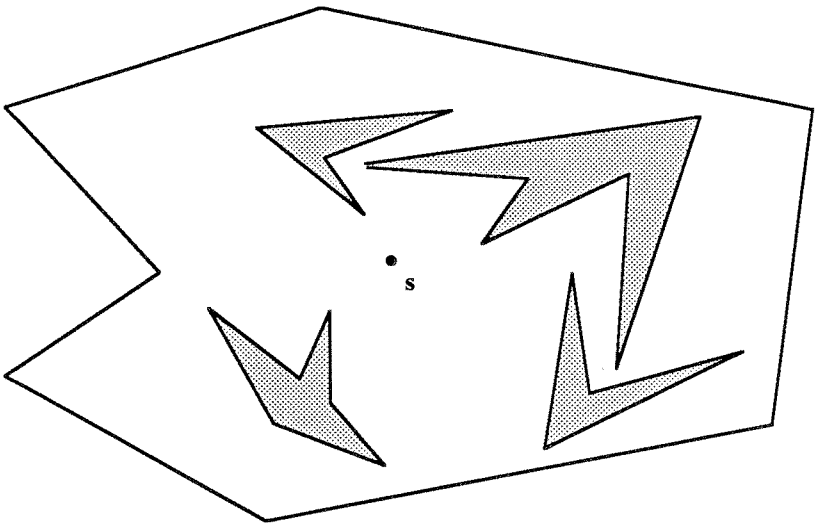


Figure 2. A polygonal room with star-shaped holes.

We claim that it suffices to assume that the bounding simple polygonal room is an arbitrarily large rectangle that encompasses all of the holes. This follows since we can separately compute the visibility polygon from s within the room P (assuming no holes) and compute the visibility polygon from s in a very large rectangular room with the holes P_i, and then we can merge the two results in linear time by simply advancing around each visibility polygon, keeping track of the closest edges to s. Thus, we concentrate on the case of computing visibility from s in the presence of the "obstacles" P_i.

We also claim that we can map the visibility problem into the problem of finding the *lower envelope* of a set of objects. See Figure 3. The visibility problem from point s is asking for the set of obstacle points which are first hit by following rays out from s. If we think of the problem in the (θ, r) plane (where θ is the angle about s and r is distance from s), then the problem is to find the "lower envelope" of the transformed obstacles in the (θ, r) plane. Sweeping a ray angularly around s corresponds to sweeping a vertical line across the transformed problem, and a point p will be visible to s if and only if it lies *below* all obstacles in the transformed problem. The transformation into the (θ, r) space begins by "splitting" the obstacle space along the horizontal ray out of s to the right. This allows us to concentrate on the problem over the interval $[0, 2\pi]$, avoiding problems of wraparound. (Note that in splitting the obstacles, we will at most double the number of vertices and obstacles.) Of course, this transformation into polar coordinates about s does not map line segments into line segments, but it does preserve the basic relationships among the obstacles. (The fact that polygonal objects are not mapped into polygons will not cause difficulties for visibility problems.) Note that another way to view this transformation is to think of the point s at being very far away (at $(0, -\infty)$), below the set of obstacles. Our algorithm and its analysis can easily be cast in the original obstacle space, but we find it easier for the exposition to consider the transformed problem.

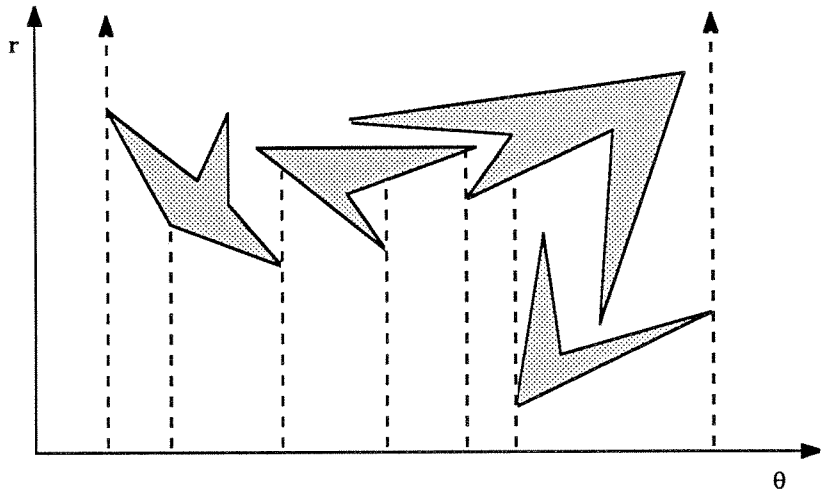


Figure 3. Transforming the visibility problem into a lower envelope problem.

3. The Convex Case

The case in which all polygonal holes are convex has been shown to have an optimal ($\Theta(n+h\log h)$) algorithm [AAGHI, LC, Mi]. The idea is simply to replace each hole by its "chord" (the segment joining the left and right points of tangency with respect to the observer s), and then to apply the known (optimal) algorithm to the set of h segments (noting that $n = 2h$ in this case). Since the original holes were disjoint, the chords will also be disjoint. The chords can be found either in time $O(n)$ (by marching around the boundary of each hole) or in time $O(h\log n)$ (by doing a binary search on each hole boundary). Next, we can compute visibility from s among the h chords in time $O(h\log h)$. Once the visibility of the chords is known, the actual tracing of the visibility polygon can be done in a straightforward manner in time $O(K)$, where K is the output size (the number of vertices visible from s). Thus, the algorithm runs in a total time of either $O(K + h\log n)$ or of $O(n + h\log h)$, depending on which method is used to find the points of tangency.

For the case of star-shaped polygonal holes, the obvious generalization of the above method fails, as is shown in Figure 4. The problem is that there may be stars which are partially visible to s but whose "chords" are totally obscured. (A chord should now be defined as a chain of at most two segments which goes from the left point of tangency, to a center of the star, then to the right point of tangency.) The critical property of convex holes was that object P_i is visible from s at angle θ if and only if the chord of P_i (with respect to s) is visible from s at angle θ when only chords are considered to be obstacles. This property does not hold for star-shaped polygons, as there may be "points" or "fingers" of a star which "poke through" the "gaps" between holes. We overcome the difficulty of these fingers by devising a scheme to check the gaps for potential overhangs.

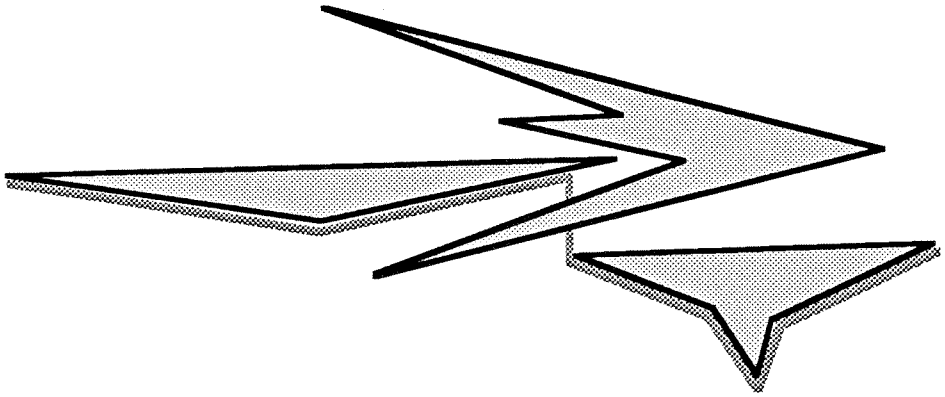


Figure 4. Complications of applying the straightforward algorithm.

4. The Algorithm for Star-Shaped Holes

Our algorithm begins with a preprocessing step. First, we compute the kernel of each hole P_i. This takes time $O(n)$ and can serve as a check that the data is indeed as claimed (that all holes are star-shaped). Now, each kernel is a nonempty convex polygon, and we select an arbitrary center C_i from the kernel of hole P_i by choosing, say, the rightmost point of the kernel (in the event of a tie, select the rightmost point which has maximal y-coordinate). (We are assuming here that we are addressing the lower envelope problem, so that rightmost points correspond to right tangency points from s.) Next, we "split" each of the star-shaped holes into "half stars" by drawing a vertical line through each C_i. See Figure 5. This separates each hole into a "left star" (that portion of the hole which lies to the left of the vertical line) and a "right star" (that portion of the hole to the right of the vertical line). Clearly, both the right and left stars are star shaped polygons with center C_i. If the center C_i is the rightmost point of P_i, then the right star of P_i will be empty. We can perform the splitting operation by simply finding the points u_i and v_i at which the vertical line intersects each hole P_i. (Note that the star-shapedness of P_i guarantees that any line through a center point intersects the boundary of P_i in at most two points.) A straightforward search finds all pairs of points u_i and v_i in total time $O(n)$; thus, the splitting can be done in time $O(n)$.

We concentrate on finding the visibility among the collection of left stars. The visibility among the right stars is found similarly, and then the two can be merged in linear time at the end. We will therefore assume that $P_1, \ldots, P_h$ are star-shaped polygons which have a rightmost vertex which serves as a center.

One further simplification that we will adopt for the exposition is that we can replace each left star by a "stick figure" consisting of the center C_i and segments ("spokes") which join C_i to each vertex of the left star. See Figure 6. Then, once visibility among the stick figures is known, it is an easy task to obtain visibility among the original obstacles. (The visible portion of a spoke $\overline{C_i v}$ has a unique obstacle edge immediately below it, namely the edge that comes immediately after v in the counterclockwise enumeration of the boundary of P_i.) In the resulting stick figure for each left star we can discard the two vertical spokes

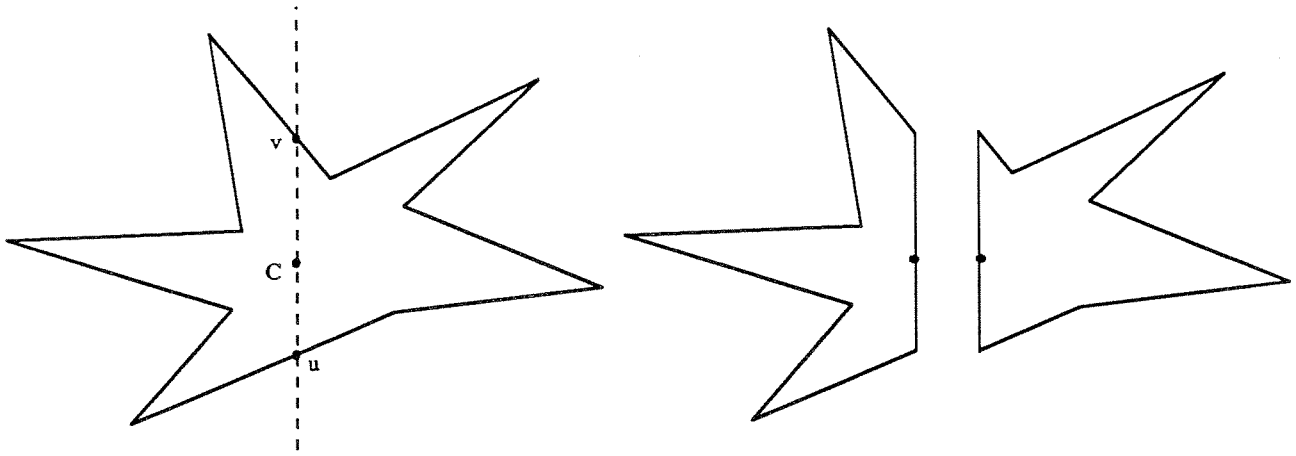


Figure 5. Star splitting.

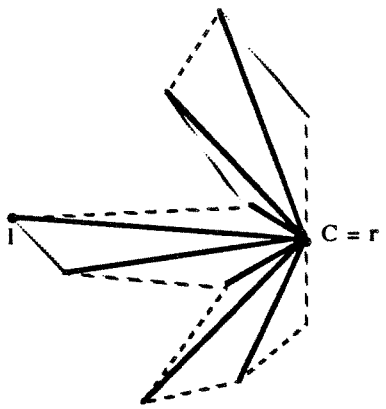


Figure 6. Making a star into a stick figure.

out of C_i. From now on, we will assume that the obstacles are left star stick figures.

Next, we compute the *visibility profile* of each P_i. The visibility profile of P_i is defined to be the visible portion of P_i's boundary, together with the "shadow lines" induced by visibility from s (thus, it is the portion of the boundary of the visibility polygon from s with respect to the single hole P_i which lies between the left and right tangency points of P_i). Each such visibility profile can be computed in linear time by the algorithm of [EA, Le83]. We denote by π_i the visibility profile of P_i. Note that, due to the nonconvexity of the holes, the visibility profiles can intersect. Each profile is a chain of points which lie on the boundary of the corresponding polygon, with vertical "jumps" at those points (called "profile vertices") where the obstacle "casts a shadow upon itself". See Figure 7.

Our problem is then to merge the profiles so as to find the overall *visibility envelope*, Π, which describes the boundary of the visibility polygon from s. A straightforward merge of these h profiles leads to an

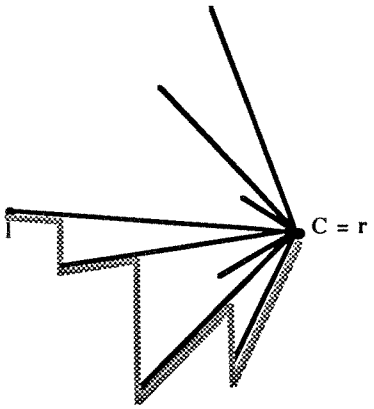


Figure 7. An obstacle profile.

$O(n \log h)$ algorithm, which works for arbitrary disjoint polygonal holes, not exploiting the fact that the holes are known to be star-shaped (this same result was obtained by [As]).

Each obstacle has an associated *chord* $\overline{l_i r_i}$ which joins its extreme left point to its extreme right point. We compute the visibility among the chords to obtain the *Lowest Chord Profile* (LCP). The LCP is simply the result of computing the lowest envelope of the chords (in time $O(h \log h)$), and then replacing each section of visible chord with the corresponding section of the profile of the object. This second step requires time $O(n)$, as it can be done in one pass over the data, comparing the lowest chord envelope with obstacle profiles.

The LCP is a (non strictly) monotone chain from right to left, with vertical segments occurring at each vertex. The vertical segments of the LCP which join a vertex of one profile with a point on another profile are called *gaps*. Any discrepency between the LCP and the lowest envelope of the obstacles is a result of protrusions of obstacle spokes through the gaps. We call these protrusions *overhangs*. The leftmost point of an overhang is called an *overhang endpoint*. The lower envelope of an overhang will be called the *overhang profile*. Refer to Figure 8. The next phase of the algorithm is a "clean-up" step which sweeps the plane from right to left, determining the effect of overhangs on the LCP.

The plane sweep proceeds as follows. We assume that the points l_i, r_i have been sorted according to their x-coordinates; let XLIST be the resulting list of $2h$ sorted numbers. At any particular stage, we have available the *Current Lowest Envelope* (CLE), which is either an obstacle profile (for the current lowest obstacle) or an overhang profile. We let e be the edge of the CLE which is currently intersected by the vertical sweep line. Our algorithm builds a list, Π, of points (from right to left) which constitute the desired lowest envelope of the obstacles. Initially, Π is empty, and the initial position of the event line is at the rightmost vertex of the scene (the first point in the XLIST). We can take the initial CLE to be a single line segment e which is horizontal, very wide, and situated above all obstacles. The sweep algorithm concludes when the sweep line encounters the leftmost vertex of the scene (the last point in the XLIST).

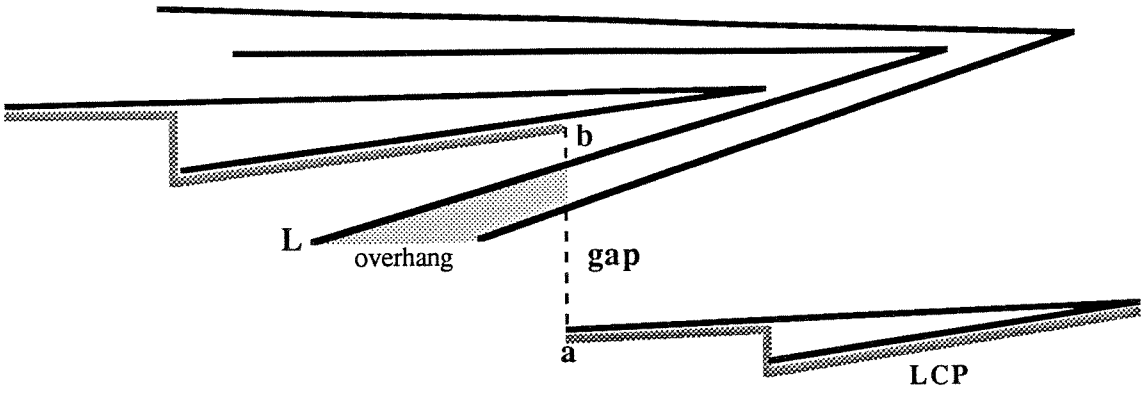


Figure 8. An overhang protrudes through the Lowest Chord Profile.

We maintain a stack of overhang profiles, the OVERHANG-STACK, which is initially empty. This is necessary since while sweeping across an overhang profile we may encounter new gaps and need to follow a new overhang. But when we have completely swept across one overhang (that is, when we encounter an overhang endpoint), we may need to return to following the last overhang. See Figure 9.

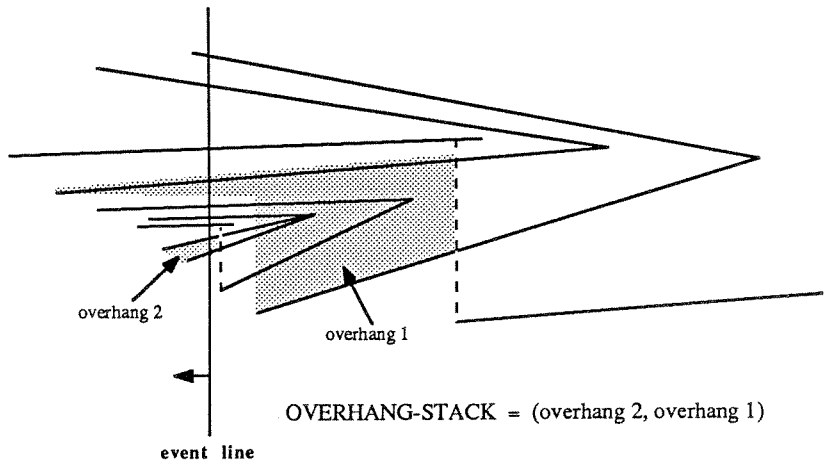


Figure 9. The overhang stack.

Events for the sweep algorithm are defined by four types of points that the sweep line may encounter: (1) a right extreme point r, (2) a left extreme point l, (3) an *overhang endpoint* L, or (4) a *profile vertex* v, which is simply a vertex on the CLE. We consider each case in turn. Refer to Figure 10.

PLANE SWEEP ALGORITHM

(1). **(A right extreme point r is encountered.)** There are two subcases depending on the relative

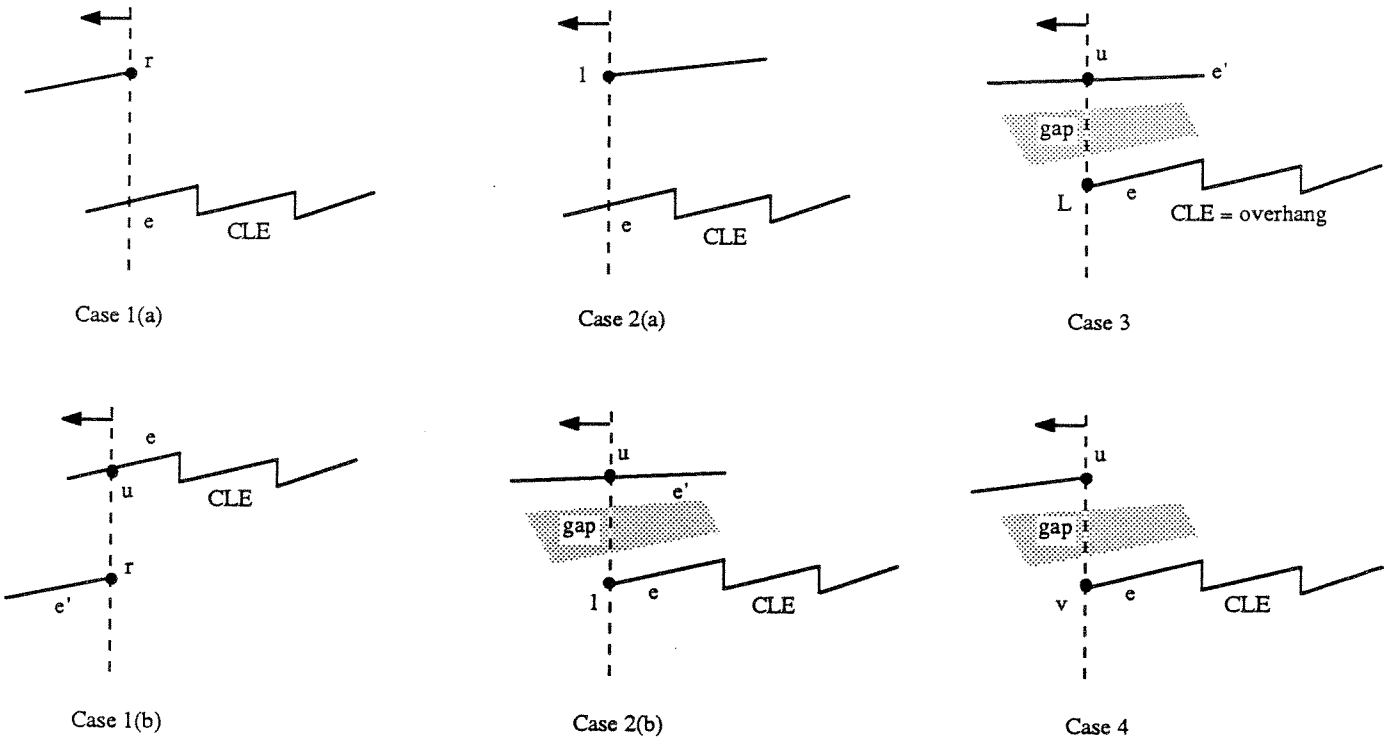


Figure 10. Various cases for the plane sweep events.

positions of e and r:

(a). (r **is above** e) Do nothing. The effect of the obstacle containing r will be discovered when and if the obstacle protrudes through a gap. [Claim: r is not visible, but the point on e below r is visible.]

(b). (r **is below** e) Let u be the point of intersection between the event line and e. Set $\Pi = \Pi \cup (u, r)$. Set the CLE to be the profile of the obstacle containing point r (so that e becomes the segment which proceeds left out of r). [Claim: r is visible.]

(2). **(A left extreme point l is encountered.)** There are two subcases depending on the relative positions of e and l:

(a). (l **is above** e) Do nothing. The obstacle containing l need not be considered further. [Claim: l is not visible.]

(b). (l **is the left endpoint of** e) We have reached the left endpoint of the CLE, so we must decide where to go next. Set CLE to the result of calling *Gap-Check*(l, u), where u is the lower of the two points $LCP(x(l)^-)$ and $\mathcal{O}(x(l))$. We use the notation that $LCP(x(l)^-)$ is the point of the LCP just to the left of l (which will be a point directly above l, on some other obstacle), and $\mathcal{O}(x(l))$ is the point on overhang $\mathcal{O} = top$(OVERHANG-STACK) at x coordinate $x(l)$ (directly above l). (If OVERHANG-STACK is empty, define $\mathcal{O}(x(l)) = +\infty$.) *Gap-Check*$(l, u)$ returns a

new profile, which is either an overhang profile, if there exists an overhang protruding through gap $\overline{lu}$, or it returns the profile of the obstacle containing the point of the LCP above l. [Claim: The overhang returned by *Gap-Check*(l, u) is visible at $x(l)^-$.] Finally, set $\Pi = \Pi \cup (l, CLE(x(l)^-))$. [Claim: l cannot be below e.]

(3). **(An overhang endpoint L is encountered.)** Then, the CLE is the profile of some overhang, and L is the left endpoint of e. Pop the OVERHANG-STACK. Set CLE to the result of calling *Gap-Check*(L, u), where u is the lower of the two points $LCP(x(L)^-)$ and $\mathcal{O}(x(L))$, where $\mathcal{O}$ is the element at the top of the OVERHANG-STACK. (Again, if OVERHANG-STACK is empty, define $\mathcal{O}(x(L)) = +\infty$.) Finally, set $\Pi = \Pi \cup (L, CLE(x(L)^-))$.

(4). **(A profile vertex v is encountered.)** Potentially, there can be an overhang passing through the gap above v, so we must again call *Gap-Check*. Set CLE to the result of calling *Gap-Check*(v, u), where u is the lower of the two points $LCP(x(v)^-) = CLE(x(v)^-)$ and $\mathcal{O}(x(v))$, where $\mathcal{O}$ is the element at the top of the OVERHANG-STACK. (Again, if OVERHANG-STACK is empty, define $\mathcal{O}(x(v)) = +\infty$.) If there is no overhang protruding through the gap, then the CLE will remain unchanged. Finally, set $\Pi = \Pi \cup (v, CLE(x(v)^-))$.

In all cases, the next event is determined (in constant time) by comparing the location of the next point in the CLE (which is in right-to-left order) with the next point in the XLIST of chord endpoints; the closer of the two will be the new event location.

Note that in order to do evaluations of the form $LCP(x)$ or $\mathcal{O}(x)$, we must find the point of the LCP or of an overhang at a particular x coordinate. We do this by keeping a pointer to a position along each such chain where we last did evaluation on the chain. The pointers will always be advancing to the left. When we need to evaluate the value of a profile, we can start at the position of the pointer and simply walk along the chain to the left until we get to coordinate x, and we then leave the pointer at this new position. Since the sweep line always advances to the left, we will never need to back up to do an evaluation. This ensures an $O(n)$ bound on all such evaluations.

We complete the description of the algorithm by describing *Gap-Check*. *Gap-Check* (a, b) takes as input a gap $\overline{ab}$, which is a vertical line segment from a point a on one obstacle to a point b on a possibly different obstacle above a. The output is a profile, either an overhang profile if an overhang protrudes through the gap, or the obstacle profile of the obstacle containing b. *Gap-Check* is the heart of the algorithm, and requires the most care to assure that we do not do more than $O(n)$ work.

The method requires that we keep track of "marks" on each obstacle. Each obstacle begins with its mark at its right endpoint. The marks will advance around the obstacles in the clockwise direction. Each call to *Gap-Check* advances the marks around some of the obstacles. In the process of advancing a mark, we keep track of times that the obstacle boundary crosses through the gap $\overline{ab}$. Portions of obstacles that

protrude through the gap are recorded in a list PROTRUSIONS. The segments in the PROTRUSIONS list constitute the overhang through the gap. Once we have the list of all protrusions, we can determine the overhang profile by calling the linear-time visibility algorithm for a single simple polygon: the polygon constructed from the protrusions by clipping them at $\overline{ab}$ and joining them along $\overline{ab}$ to form a simple figure.

The search for protrusions begins with the right endpoint, r, of the obstacle containing b. (We shall use the notation that $obs(b)$ is the obstacle containing point b.) We assume that the obstacle chords have been preprocessed so that each right endpoint points to the chord that is immediately above it. (If no chord is above it, then we can think of it pointing as to $+\infty$, or we can create a dummy chord that sits above all obstacles and "catches" such pointers.) This preprocessing can be done in time $O(h \log h)$ by a standard plane sweep. Let $g(r)$ denote the obstacle to which right endpoint r points and let u_r be the point directly above r on the chord of $g(r)$. Now, we advance from r, the right endpoint of $obs(b)$, to the right endpoint of $obs(g(r))$. We advance the mark clockwise around $obs(g(r))$ until it passes through the vertical gap $\overline{ru_r}$ (the right endpoint r will then be declared "DONE"). Then, we follow the pointer from the right endpoint of $obs(g(r))$ to the chord above it, and repeat the procedure, following a staircase path up and to the right. This continues until we find an obstacle whose right endpoint has already been declared *DONE*. We define a right endpoint r to be DONE if the mark on the obstacle $obs(g(r))$ has already been advanced around $obs(g(r))$ until it passed through the gap $\overline{ru_r}$. The meaning of a right endpoint being DONE is that, when such a point is encountered, we will be assured that all possible protrusions through the original gap have been detected. More formally, we give the details of *Gap-Check* below. Refer to Figure 11.

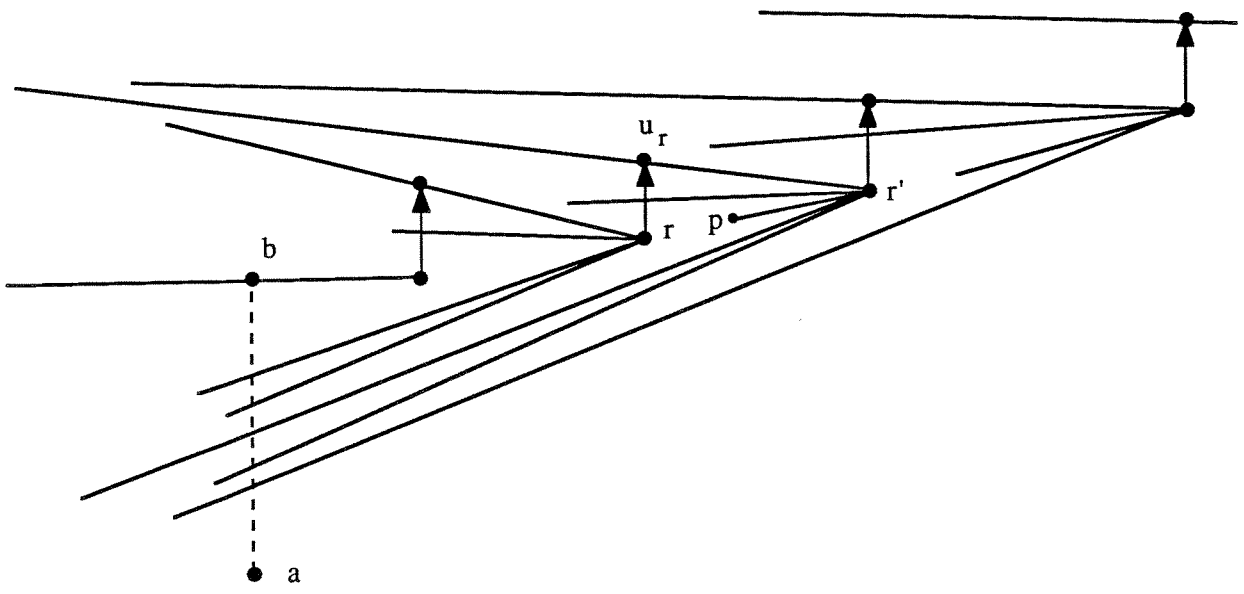


Figure 11. Gap-Check (a, b).

Procedure *Gap-Check* (a, b)

(0). Let r be the right endpoint of $obs(b)$. Initialize PROTRUSIONS to be empty.

(1). If r is DONE, then go to step (2). Otherwise, let p be the mark on obstacle $obs(u_r)$ and let r' be the right endpoint of $obs(u_r)$. Initialize a variable LIST to be empty. Advance p clockwise around $obs(u_r)$ until $\overline{r'p}$ intersects $\overline{ru_r}$ (which must happen since p will eventually get to the left endpoint of the chord of $obs(u_r)$). During this advance, check whether or not the segment $\overline{r'p}$ intersects the gap $\overline{ab}$; if it does, then push the segment $\overline{r'p}$ onto the LIST. When done advancing the mark, label r DONE. Set PROTRUSIONS = append(PROTRUSIONS, LIST). Set $r = r'$. Go to (1).

(2). If PROTRUSIONS is empty, then return the profile of $obs(b)$. Otherwise, we need to find the overhang profile caused by the PROTRUSIONS. First, note that PROTRUSIONS is a list of segments crossing $\overline{ab}$, ordered from top to bottom. Clip each segment at $\overline{ab}$, and join the segments along the edge $\overline{ab}$. This forms a simple figure, which is called an overhang, of size |PROTRUSIONS|. Compute the lowest envelope of this simple figure by the standard method [EA, Le83] in time $O(|\text{PROTRUSIONS}|)$. Let L be the leftmost point of the resulting profile, and return this profile (keeping the point L associated with the overhang profile).

5. Proof of Correctness

Our goal in this section is to prove that the algorithm correctly computes the visibility envelope Π. We wish to prove the following proposition:

Proposition 1 *The visibility envelope Π is correct to the right of the sweep line at any event position. In particular, this implies that at the completion of the algorithm, the entire visibility envelope Π has been correctly computed.*

Proof: The proof is by induction on the event number of the sweep. Initially, the sweep line is at the extreme right of the obstacle space, so the claim holds trivially. We now assume that the claim holds for the event position x_E, and we shall show that it will also hold for the next event position $x_{E'}$. This will be shown in the following sequence of lemmas. ∎

Our task is to show that from one event to the next, we follow the visibility envelope. First, by our definition of event point, we know that nothing important can occur strictly between x_E and $x_{E'}$:

Lemma 2 *If the CLE is visible at $(x_E)^-$, then it will be visible for all $x \in (x_{E'}, x_E)$.*

Proof: This claim follows from the facts that the obstacle are disjoint and that the definition of event point includes any vertices on the CLE. Thus, the portion of the CLE between x_E and $x_{E'}$ is just a segment e. If part of this segment were blocked, say at position $x \in (x_E, x_{E'})$, then the right endpoint of the object blocking it would have to be outside the interval $(x_E, x_{E'})$ (otherwise, there would have been a right endpoint encountered by the sweep line). The blocking object cannot intersect e, so it must also block the CLE at $(x_E)^-$, a contradiction. ∎

Our induction hypothesis is that the CLE is visible at positions $x \geq x_E$. We must now show that the CLE is visible at position $(x_E)^-$, the position just to the left of the event.

Lemma 3 Gap-Check *works correctly. Specifically, if* Gap-Check (a, b) *returns the profile of* $obs(b)$*, then* $obs(b)$ *is visible at* $(x_E)^-$*. If* Gap-Check *returns an overhang profile, then the overhang profile is visible at* $(x_E)^-$*. Also, for all values of* x *between* x_E *and* $x(L)$ *(the position of the overhang endpoint), the overhang profile is not blocked by any obstacle whose right endpoint is right of* x_E*.*

Proof: Suppose that the overhang profile at $(x_E)^-$ is not visible. What obstacle could be blocking it? There are two possibilities: An obstacle that was not one of those encountered during the march carried out by *Gap-Check*, or an obstacle which was, but not found to be blocking.

In the first case, the blocking obstacle must have its chord above b in such a way that the staircase march did not encounter it. This implies either that its right endpoint lies above the staircase march (implying that there is no way for a spoke to protrude through the gap $\overline{ab}$ without intersecting either another obstacle or one of the vertical "risers" along the staircase), or that the staircase march was stopped before encountering this chord. In the latter case, there must have been a previous call to *Gap-Check* whose staircase march encountered the chord. This implies that any spoke that protrudes through $\overline{ab}$ would have been part of a previous overhang. This contradicts the definition of b, which takes into account the LCP as well as the overhangs already found.

The second case can occur only if the mark on the blocking obstacle had already been advanced beyond the protruding spoke, which again means that a previous call to *Gap-Check* would have considered the spoke as part of an overhang. This contradicts the fact that the effect of such an overhang was taken into account in the definition of b.

The rest of the claim follows by similar arguments. ∎

Since every time there is a vertical gap, our algorithm calls *Gap-Check* to determine the next CLE, we conclude that the algorithm will select the correct CLE after event x_E. Then, combining this result with Lemma 2, our proposition is proved.

We must now analyze the complexity of the algorithm. The initial work of computing visibility among the chords and establishing the pointers from each right endpoint r to the chord directly above it can be done in time $O(h \log h)$, using a standard plane sweep across the chords. The obstacle profiles are computed in total time $O(n)$ using the linear-time algorithms for visibility of a single simple polygon. The conversion

of the chord visibility to the LCP is then easily accomplished in time $O(n)$. In the main sweep from right to left, we do not rely on the n vertices being sorted. Instead, the next event is always found in constant time from the CLE point list and the XLIST of sorted endpoints.

The majority of the work takes place in *Gap-Check*, where we must follow pointers from chord to chord and advance marks around obstacles. But the crucial property is that we never advance a mark past a vertex more than once, so the total cost of all mark advancement is $O(n)$. The computation of the visibility profile of an overhang takes time linear in the number of protrusions. Since no protrusion occurs in more than one overhang, we get a total linear bound on all visibility calculations on overhangs. Thus, we have shown the following theorem.

Theorem 4 *It is possible to compute the visibility polygon from a point in a simple polygonal room with h star-shaped holes in time $\Theta(n + h \log h)$ and linear space.*

6. Conclusion

We have devised an optimal $\Theta(n+h \log h)$ algorithm for computing visibility from a point among a collection of star-shaped obstacles. The obvious open problem is to solve the problem for arbitrary simple polygonal obstacles.

One consequence of our algorithm is that if the the holes are simple polygons each of which has a known covering by at most k star-shaped polygons, then the visibility problem is solved in time $O(n + hk \log hk)$, which is optimal if k is bounded. (Here, k can be thought of as the maximum number of guards needed to observe any hole.)

An interesting open question is whether or not our method can be used to get an optimal $(\Theta(K+h \log h))$ algorithm for computing the visibility graph of a set of star-shaped polygons. Note that the recent result of [GM] gives an $O(K + n \log n)$ algorithm for the visibility graph of a set of polygons, but that this bound is known to be tight only in the case that $h = \Omega(n)$.

Another interesting open question is how to generalize our results to the case of visibility from a line *segment* (a florescent light bulb source) rather than from a point. For the case of a set of n disjoint line segment obstacles, Suri and O'Rourke [SO] give a lower bound of $\Omega(n^4)$ and devise an optimal algorithm for the explicit construction of the boundary of the visibility region. In terms of the number of holes, h, however, the lower bound may be written as $\Omega(n + h^4)$. For the case of convex holes, we suspect that the straightforward generalization of Suri and O'Rourke's algorithm yields an optimal bound. However, it is not clear that our algorithm as presented in this paper generalizes to yield an optimal visibility algorithm from a segment.

7. Bibliography

[AM] E.M. Arkin and J.S.B. Mitchell, "An Optimal Visibility Algorithm for a Simple Polygon With Star-

Shaped Holes", Manuscript, School of Operations Research and Industrial Engineering, Cornell University, October, 1986.

[As] T. Asano, "Efficient Algorithms for Finding the Visibility Polygons for a Polygonal Region With Holes", *Transactions of IECE of Japan*, Vol. E-68 (1985), pp. 557-559.

[AAGHI] T. Asano, T. Asano, L. Guibas, J. Hershberger, and H. Imai, "Visibility of Disjoint Polygons", *Algorithmica*, Vol. 1, (1986), pp. 49-63.

[EA] H.A. El Gindy and D. Avis, "A Linear Algorithm for Computing the Visibility Polygon From a Point", *Journal of Algorithms*, Vol. 2 (1981), pp. 186-197.

[GM] S.K. Ghosh and D.M. Mount, Private communication, 1987.

[Le78] D.T. Lee, "Proximity and Reachability in the Plane", Ph.D. Thesis, Technical Report ACT-12, Coordinated Science Laboratory, University of Illinois, Nov. 1978.

[Le83] D.T. Lee, "Visibility of a SImple Polygon", *Computer Vision, Graphics, and Image Processing*, Vol. 22 (1983), pp. 207-221.

[LC] D.T. Lee and I.M. Chen, "Display of Visible Edges of a Set of Convex Polygons", in *Computational Geometry*, Edited by G.T. Toussaint. North Holland, Amsterdam, 1985.

[LP] D.T. Lee and F.P. Preparata, "An Optimal Algorithm for Finding the Kernel of a Polygon", *Journal of the ACM*, Vol. 26 (1979), pp. 415-421.

[Mi] J.S.B. Mitchell, "Planning Shortest Paths", PhD Thesis, Department of Operations Research, Stanford University, August, 1986. (Available as Research Report 561, Artificial Intelligence Series, No. 1, Hughes Research Laboratories, Malibu, CA.)

[SO] S. Suri and J. O'Rourke, "Worst-Case Optimal Algorithms For Constructing Visibility Polygons With Holes", *Proc. Second Annual ACM Conference on Computational Geometry*, Yorktown Heights, NY, June 1986, pp. 14-23.