1991

An Optimal Parallel Algorithm for Detecting Weak Visibility of a Simple Polygon

Danny Z. Chen

Report Number:
91-008

AN OPTIMAL PARALLEL ALGORITHM FOR DETECTING WEAK VISIBILITY OF A SIMPLE POLYGON

Danny Z. Chen

CSD-TR-91-008
February 1991

An Optimal Parallel Algorithm for Detecting Weak Visibility of a Simple Polygon (Extended Abstract)

Danny Z. Chen*

Abstract

The problem of detecting the weak visibility of an n-vertex simple polygon P is that of finding whether or not P is weakly visible from one of its edges and (if it is) identifying every edge from which P is weakly visible. In this paper, we present an optimal parallel algorithm for solving this problem. Our algorithm runs in $O(\log n)$ time using $O(n/\log n)$ processors in the CREW-PRAM computational model, and is very different from the sequential algorithms for this problem. This algorithm also enables us to optimally solve, in parallel, several other problems on weakly visible polygons.

1 Introduction

Visibility is one of the most fundamental topics in computational geometry. Visibility problems arise in many application areas, such as computer graphics, vision, VLSI design, and robotics. Visibility problems also appear as subproblems in other geometric problems (like finding the shortest obstacle-avoiding paths and computing intersections between geometric figures). Therefore, a great deal of research has been devoted to finding efficient algorithms for solving various visibility problems, in both sequential and parallel computational models.

Weak visibility problems concern visibility with respect to "observers" in the shape of line segments. An important class of weak visibility problems deals with the case where the opaque object is the boundary of a simple polygon. For a point p in a polygon and a line segment s, p is *weakly visible* from s if p is visible from some point on s. An example of such problems is that of computing the region inside a polygon that is weakly visible from a segment. For this problem, many sequential algorithms [4, 12, 14, 18, 23] and a parallel algorithm [11] have been discovered. For more examples of the weak visibility problems on simple polygons, see [2, 3, 6, 13, 15, 22].

This paper considers the problem of detecting the weak visibility of a simple polygon. An n-vertex simple polygon P is *weakly visible* if there exists an edge e of P such that any point of P is weakly visible from e (e is called a *weakly visible edge* of P). The problem of detecting the weak visibility of P is that of finding whether or not P is weakly visible and (if it is) identifying *all* weakly visible edges of P. Note that this problem is a natural generalization of the well-known problem of computing the *kernel* of a simple polygon [19]. (Recall that a point is in the kernel of a polygon iff the whole polygon is visible from that point, and that a polygon with a nonempty kernel is called a *star-shaped* polygon [21].)

Avis and Toussaint [2] first consider the problem of detecting the weak visibility of a simple polygon. They present a sequential linear time algorithm for the following case: check whether or not a polygon P is weakly visible from a specified edge e of P. Another sequential linear time algorithm for this case was recently given in [13]. Using the algorithms in [2, 13], the problem of detecting the weak visibility of P can be trivially solved in $O(n^2)$ time (by checking separately each edge of P), but Sack and Suri [22] succeeded in finding a linear time algorithm for this problem. Our interest here is to solve this problem in parallel. The parallel computational model we use is the CREW-PRAM; this is the synchronous shared-memory model where multiple processors can simultaneously read from the same memory location but at most one processor is allowed to write to a memory location at each time unit.

Based on the observations of Sack and Suri [22], a suboptimal parallel algorithm can be easily obtained by using the recent result of Goodrich *et al.* [11] on constructing a data structure that supports ray-shooting queries [4]. This algorithm first preprocesses P and builds the data structure in $O(\log n)$ time using $O(n)$ processors [11], and then does $O(n)$ ray-shooting queries by using the data structure. The algorithm takes in total $O(\log n)$ time and $O(n \log n)$ work (the work complexity of a parallel algorithm is the total number of operations performed by the algorithm). Obviously, the work complexity of this algorithm is a factor of $\log n$ away from optimality. The sequential algorithm in [22] manages to avoid doing the ray-shooting queries, but that method seems to be inherently sequential.

Our method for obtaining an optimal parallel algorithm is very different from the above approaches. We give geometric insights and parallel techniques which enable us to use the divide-and-conquer strategies and to avoid the difficulty of doing ray-shooting queries. Our algorithm runs in $O(\log n)$ time using $O(n/\log n)$ processors, and is obviously optimal. We also use this algorithm to optimally solve, in parallel, several other problems on weakly visible polygons (such as the shortest paths, triangulation, and one cruising guard [6] problems); these parallel solutions all take $O(\log n)$ time using $O(n/\log n)$ processors and avoid

*Department of Computer Science, Purdue University, West Lafayette, IN 47907. This research was partially supported by the Office of Naval Research under Grants N00014-84-K-0502 and N00014-86-K-0689, the National Science Foundation under Grant DCR-8451393, and the National Library of Medicine under Grant R01-LM05118.

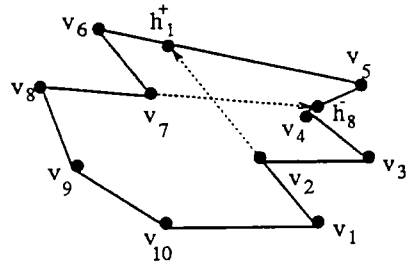

Figure 1: Illustrating HE_i^+ and HE_i^-.

the preprocessing step of triangulating an *arbitrary* simple polygon. (The best known parallel algorithms for triangulating a simple polygon require $O(n)$ processors in the CREW-PRAM [9, 24], or $O(n/\log n)$ processors in the more powerful CRCW-PRAM [10], and $O(\log n)$ time.) The geometric insights we present could be useful in solving other geometric problems.

There are two major subproblems solved by our weak visibility algorithm: (1) identifying all weakly visible edges for a star-shaped polygon whose kernel contains a *convex* vertex, and (2) checking whether or not a polygon is weakly visible from a specified edge (i.e., the case solved by [2, 13]). The solutions to these two subproblems could be interesting in their own right. The problem of detecting the weak visibility of a simple polygon is reduced to the two subproblems. The idea for the reduction is derived from the one used in [22], but our reduction procedure is very different from [22]. Owing to the limitation of space, we leave the reduction and the applications of the algorithm to the full paper.

The rest of this paper consists of 4 sections. Section 2 gives some notations and preliminary results on the weak visibility of a simple polygon. Section 3 discusses several geometric and computational observations needed by the algorithm. Sections 4 and 5 describe the algorithms for solving the two subproblems mentioned above, respectively.

2 Preliminaries

An n-vertex simple polygon P is given as a sequence $(v_1, v_2, \ldots, v_n)$ of its vertices, in the order in which they are visited by a *counterclockwise* walk along the boundary of P, starting from vertex v_1. An edge of P joining v_i and v_{i+1} is denoted by $e_i = \overline{v_i v_{i+1}}$ $(= \overline{v_{i+1} v_i})$, with the convention that $v_{n+1} = v_1$. The boundary of P is denoted by $bd(P)$.

Without loss of generality (WLOG), we assume that no edge of P is vertical and no three consecutive vertices of P are collinear.

Vertex v_i is *convex* if the interior angle of P at v_i is $< \pi$. Edge e_i is *convex* if both v_i and v_{i+1} are convex. For a vertex v_i, if v_{i+1} (resp., v_{i-1}) is nonconvex, then let r_i^+ (resp., r_i^-) be a ray starting at v_i and containing e_i (resp., e_{i-1}). The set of all such r_i^+'s (resp., r_i^-'s) is denoted by $Ray^+(P)$ (resp., $Ray^-(P)$). Let h_i^+ (resp., h_i^-) be the first point at which r_i^+ (resp., r_i^-) hits $bd(P) - e_i$ (resp., $bd(P) - e_{i-1}$) (i.e., v_i is closer to h_i^+ (resp., h_i^-) than to any other point in $r_i^+ \cap (bd(P) - e_i)$ (resp., $r_i^- \cap (bd(P) - e_{i-1})$)). Let h_i^+ (resp., h_i^-) be on $e_j - v_j$ (resp., $e_k - v_k$) for some j (resp., k). Then we call the consecutive edges that are on the portion of $bd(P)$ from v_{i+2} (resp., v_{i-2}) counterclockwise (resp., clockwise) to v_j (resp., v_k) the *counterclockwise* (resp., *clockwise*) *hidden edges* of v_i. The set of counterclockwise (resp., clockwise) hidden edges of v_i is denoted by HE_i^+ (resp., HE_i^-) (see Fig. 1). For examples, in Fig. 1, $HE_1^+ = \{e_3, e_4\}$ and $HE_8^- = \{e_5\}$. Note that HE_i^+ (resp., HE_i^-) can be empty. The union of all HE_i^+ (resp., HE_i^-) is denoted by $HE^+(P)$ (resp., $HE^-(P)$). We henceforth call h_i^+ (resp., h_i^-) the *first-hit* point of r_i^+ (resp., r_i^-). The following lemma characterizes an important property of the weakly visible polygons.

Lemma 1 (Sack and Suri [22]) *Suppose that polygon P is weakly visible from edge e_n and e_n is convex. Then for every $i = 1, 2, \ldots, n$, the following holds: if h_i^+ exists, then h_i^+ is the* first *point on $bd(P) - e_i$ collinear with e_i encountered in the* counterclockwise *walk along $bd(P)$ starting from v_{i+1}, and if h_i^- exists, then h_i^- is the* first *point on $bd(P) - e_{i-1}$ collinear with e_{i-1} encountered in the* clockwise *walk along $bd(P)$ starting from v_{i-1}.*

Proof. See Lemma 3 of [22]. □

It is shown in [22] that the set of weakly visible edges of P, denoted by $WVE(P)$, is equal to $bd(P) - (HE^+(P) \cup HE^-(P))$ (see Theorem 1 of [22]). For convenience, we call the edges in $HE^+(P) \cup HE^-(P)$ the *bad* edges of P. If $HE^+(P)$ and $HE^-(P)$ were available, then $WVE(P)$ could be obtained in the desired complexity bounds. The main difficulty, therefore, is in computing $HE^+(P)$ and $HE^-(P)$. WLOG, we will show the computation for $HE^+(P)$ (the computation for $HE^-(P)$ is similar).

A point p is represented by its x-coordinate and y-coordinate, denoted by $x(p)$ and $y(p)$, respectively. For a line segment s (resp., a ray r), the line containing s (resp., r) is denoted by $l(s)$ (resp., $l(r)$). The slope of a line l (resp., a segment s, a ray r) is denoted by $slope(l)$ (resp., $slope(s)$, $slope(r)$).

The chain on $bd(P)$ from v_i counterclockwise to v_j, $i \neq j$, is denoted by bd_{ij}. The size of a chain C is the number of line segments on C, denoted by $|C|$. For three non-collinear points p, q, and r, we say that the directed chain from p to q to r makes a *left* (resp., *right*) turn if $x(r)(y(p) - y(q)) + y(r)(x(q) - x(p)) + x(p)y(q) - x(q)y(p) > 0$ (resp., < 0). For a directed simple chain $C = (p_1, p_2, \ldots, p_k)$, $k \geq 3$, C is said to only make left (resp., right) turns iff every subchain of the form (p_{i-1}, p_i, p_{i+1}) makes a left (resp., right) turn, $1 < i < k$.

If polygon P is weakly visible from edge e_n and e_n is convex, then for $1 \leq i \leq j \leq n$, the (directed) shortest path from v_i to v_j inside P goes through only the vertices on bd_{ij}, and the shortest path only makes right turns (this fact is shown in [2, 13]). Hence, we call such a shortest path the *internal convex path* of bd_{ij}, and denote it by $ICP(bd_{ij})$.

Let l be a non-vertical line. We say a point p is *above* (resp., *below*) l iff the vertical line passing p intersects l at a point q such that $y(q) < y(p)$ (resp., $y(q) > y(p)$). A segment s is said to be (properly) *above* (resp., *below*) l iff every point of s is above (resp., below) l. The upper (resp., lower) half-plane of l is the half-plane defined by l whose interior points are all above (resp., below) l. A left (resp., right) half-plane of a ray r is the half-plane whose boundary contains r and which is to the left (resp., right) of $l(r)$. For a set L of lines, we use $UPCI(L)$ (resp., $LPCI(L)$) to denote the common intersection of the upper (resp., lower) half-planes for the lines in L.

Internal convex paths and common intersections of half-planes play important role in our algorithm. We will represent the internal convex paths and common intersections of half-planes by a data structure called the *hull tree* [8, 9] or *rank tree* [5]. This data structure supports efficient implementation for the parallel operations of search, concatenation, and split (see [8, 9, 5] for the details).

3 Some Useful Observations

This section gives some useful geometric observations and develops the computational machinery needed by the algorithm.

The following type of tests will be frequently done by the algorithm: given a set L of lines and a line segment s, find (i) whether there is a line $l \in L$ such that s is *below* l, or (ii) whether s is *above all* the lines in L. We call such a test a *lines-vs-segment test* and denote it by $Test(L, s)$.

(Note: Another type of tests which is also needed by the algorithm is: (i') whether there is a line $l \in L$ such that s is *above* l, or (ii') whether s is *below all* the lines in L. The tests for (i') and (ii') are handled similarly to the tests for (i) and (ii). Hence we omit the discussion for (i') and (ii').)

Doing $Test(L, s)$ by a brute force method, which checks segment s against every line in L, is inefficient (it requires $O(|L|)$ work and $O(\log |L|)$ time). We would like to achieve $O(\log |L|)$ time and $O(|L|^{\alpha})$ work for every $Test(L, s)$ done by the algorithm, where α is some constant, $0 < \alpha < 1$. Our method for doing the tests makes use of the common intersection of upper half-planes. It is clear that s is above all the lines in L iff s is properly contained in $UPCI(L)$. However, it is not necessarily true that if s does not intersect $UPCI(L)$, then s is below a line in L. Our solution to the tests is based on the following observation.

Lemma 2 *Given a non-vertical segment s and a set L of lines, if L is partitioned into two subsets L' and L'' such that L' (resp., L'') contains the lines of L whose slopes are all $\geq$ slope(s) (resp., $<$ slope(s)), then the following is true: (i) s is below a line in L iff s does not intersect either $UPCI(L')$ or $UPCI(L'')$, and (ii) s is above all the lines in L iff s is properly contained in* both *$UPCI(L')$ and $UPCI(L'')$.*

Proof. Omitted. See the full paper. □

The computational lemma below follows from Lemma 2.

Lemma 3 *Given a non-vertical segment s and a set L of m lines, suppose that the slopes of all the lines in L are $\geq$ (resp., $\leq$) slope(s) and that $UPCI(L)$ is available. Then using k processors, $Test(L, s)$ can be done in $O(\log m / \log(k+1))$ time if $UPCI(L)$ is stored in an array, and in $O((\log m)^2 / \log(k+1))$ time if $UPCI(L)$ is stored in a rank tree.*

Proof. Omitted. See the full paper. □

Note that in Lemma 3, if $k = O(m^{\alpha})$ for any constant α, $0 < \alpha < 1$, then the time complexities become $O(1)$ (when $UPCI(L)$ is stored in an array) and $O(\log m)$ (when $UPCI(L)$ is stored in a rank tree).

Based on Lemmas 2 and 3, the next lemma describes the basic operation done by the procedure for performing the lines-vs-segment tests.

Lemma 4 *Given a non-vertical segment s and a set L of m lines, suppose that L is partitioned into $m^{1/c}$ subsets L_1, L_2, ..., $L_{m^{1/c}}$ of equal size such that the slope of every line in L_{i+1} is $\geq$ the slopes of all the lines in L_i, and that $UPCI(L_1)$, $UPCI(L_2)$, ..., $UPCI(L_{m^{1/c}})$ are available (each stored in a rank tree), where $c > 1$ is a constant. Then, in $O(\log m)$ time using $O(m^{c'})$ processors, either the result of $Test(L, s)$ is found, or the test range is restricted to a unique L_j (i.e., $Test(L, s)$ is completed by doing $Test(L_j, s)$), where c' is a constant and $1/c < c' < 1$.*

Proof. There are two possible cases: (1) there is a unique subset L_j which contains two lines l' and l'' such that slope(l') $<$ slope(s) $<$ slope(l''), or (2) there is no such L_j. In case (2), we apply Lemma 3 and do $Test(L_i, s)$, in parallel, for each $i = 1, 2, \ldots, m^{1/c}$. Each $Test(L_i, s)$ takes $O(\log m)$ time using $O(m^{\alpha})$ processors (by Lemma 3), where $\alpha = c' - (1/c)$ is a constant and $0 < \alpha < 1$. The answer to $Test(L, s)$ can be easily obtained from the answers to the $Test(L_i, s)$'s (based on Lemma 2), in the desired complexity bounds. In case (1), suppose L_j is the unique subset which gives rise to this case. We first do $Test(L_i, s)$ for each $i \neq j$ (in $O(\log m)$ time using $O(m^{\alpha})$ processors). If the answer to $Test(L, s)$ can be derived from the answers to the $m^{1/c} - 1$ $Test(L_i, s)$'s (e.g., there is a line in $L - L_j$ that is above s), then we are done. Otherwise, the answers to the $m^{1/c} - 1$ $Test(L_i, s)$'s must be combined with the answer to $Test(L_j, s)$ in order to obtain the result for $Test(L, s)$; hence $Test(L, s)$ will be completed by performing $Test(L_j, s)$. □

Note that if Lemma 4 can be recursively applied to $Test(L_j, s)$, then we only need to repeat the use of Lemma 4 a constant number of times in order to reduce the size of the test range to $O(m^{c'})$ (at that point the brute force method can take over). In this way, $Test(L, s)$ is processed in totally $O(\log m)$ time using $O(m^{c'})$ processors.

Lemmas 3 and 4 require that the common intersections of the upper half-planes be available *before* the tests are performed. The computation for the common intersection of m half-planes, in general, requires $O(\log m)$ time and $\Omega(m \log m)$ total work. In our situation, there can be as many as $O(n)$ rays (and thus $O(n)$ half-planes) to consider. It would be impossible to compute the common intersection of $O(n)$ half-planes in $O(\log n)$ time using $O(n/\log n)$ processors if the $O(n)$ rays were arbitrary. Next, we show that if polygon P is weakly visible from a convex edge, then it is possible to obtain a subset of $Ray^+(P)$ (resp., $Ray^-(P)$), denoted by $DR^+(P)$ (resp., $DR^-(P)$), with the following properties: (i) $HE^+(P)$ (resp., $HE^-(P)$) can be computed by using only $DR^+(P)$ (resp., $DR^-(P)$), and (ii) $DR^+(P)$ (resp., $DR^-(P)$) can be easily partitioned into two subsets, each containing rays sorted by slopes. The rays in $DR^+(P)$ (resp., $DR^-(P)$) are called the *dominating rays* of $Ray^+(P)$ (resp., $Ray^-(P)$). We just discuss the case for $DR^+(P)$ (the case for $DR^-(P)$ is similar).

WLOG, we assume that P is weakly visible from convex edge e_n, that e_n is horizontal, and that $l(e_n)$ is below $P - e_n$. We define the polar angle of a ray $r_i^+ \in Ray^+(P)$, denoted by $\alpha(r_i^+)$, as follows: let the starting vertex v_i of r_i^+ be at the origin; then $\alpha(r_i^+)$ is the angle from the positive x-axis counterclockwise to r_i^+. Note that $0 \leq \alpha(r_i^+) < 2\pi$. For rays r_i^+ and r_j^+ in $Ray^+(P)$, $i < j$, we say r_i^+ *dominates* r_j^+ if $\alpha(r_i^+) \geq \alpha(r_j^+)$. Let $DR^+(P)$ consist of the rays in $Ray^+(P)$ that are not dominated by any ray in $Ray^+(P)$. The following lemma characterizes $DR^+(P)$.

Lemma 5 *For rays r_i^+ and r_j^+ in $Ray^+(P)$, $i < j$, if r_i^+ dominates r_j^+, then $HE_j^+ \subseteq HE_k^+$ for some k, $i \leq k < j$.*

Proof. Let Q be the polygon formed by segment $\overline{v_{i+1}h_i^+}$ and the subchain of $bd(P)$ from v_{i+1} counterclockwise to h_i^+ (h_i^+ is the first-hit point of r_i^+). WLOG, assume that $h_i^+ \neq v_j$. There are two possible cases: (a) v_j is in Q (see Fig. 2 (a)), and (b) v_j is not in Q (see Fig. 2 (b)). We first show that in case (a), $HE_j^+ \subseteq HE_i^+$. If HE_j^+ were not a subset of HE_i^+, then h_j^+ would have to be outside Q. For this to happen, r_j^+ must intersect $\overline{v_{i+1}h_i^+}$ before hitting h_j^+ (since v_j is in Q); furthermore, r_j^+ must start in the

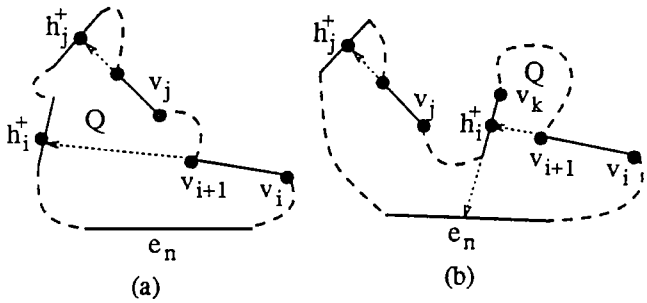

Figure 2: Illustrating the proof of Lemma 5.

right half-plane of r_i^+ and hit h_j^+ in the left half-plane of r_i^+. When $\alpha(r_i^+) - \alpha(r_j^+) \leq \pi$, such an intersection between r_j^+ and $\overline{v_{i+1}h_i^+}$ is impossible because v_j is in the right half-plane of r_i^+ and $\alpha(r_i^+) \geq \alpha(r_j^+)$ (e.g., Fig. 2 (a)). When $\alpha(r_i^+) - \alpha(r_j^+) > \pi$, such an intersection between r_j^+ and $\overline{v_{i+1}h_i^+}$ is also impossible for the following reason. That $\alpha(r_i^+) - \alpha(r_j^+) > \pi$ implies that $\alpha(r_i^+) > \pi$ and $\alpha(r_j^+) < \pi$. If r_j^+ did not intersect $bd(P) - e_j$ before crossing $\overline{v_{i+1}h_i^+}$, then v_j would not be weakly visible from e_n, a contradiction. Hence $HE_j^+ \subseteq HE_i^+$ in case (a).

In case (b), chain $bd_{(i+1)j}$ first intersects the right half-plane of r_i^+ and must later go to the left half-plane of r_i^+ by crossing r_i^+ at h_i^+. Let h_i^+ be on e_l, $l > i+1$. Since P is weakly visible from e_n, the internal convex path $ICP(bd_{(i+1)j})$ makes right turns only. It is not hard to see that there exists a vertex v_k, $l \leq k < j$, such that v_{k+1} is a vertex on $ICP(bd_{(i+1)j})$ and $HE_j^+ \subseteq HE_k^+$ (Fig. 2 (b)). □

Based on Lemma 5, it is easy to compute $DR^+(P)$ in $O(\log n)$ time using $O(n/\log n)$ processors (by doing parallel prefix [16, 17]). Note that the rays in $DR^+(P)$ are sorted by polar angles. We further partition $DR^+(P)$ into two subsets $DR_1^+(P)$ and $DR_2^+(P)$ such that the rays in each subset are sorted by slopes. This partition is done by splitting $DR^+(P)$ using a ray whose polar angle is π. $DR_1^+(P)$ (resp., $DR_2^+(P)$) contains the rays of $DR^+(P)$ whose polar angles are all $\leq$ (resp., $>$) π. From now on, we assume that $DR_1^+(P)$ and $DR_2^+(P)$ are already available.

4 Detecting the Weak Visibility of a Star-Shaped Polygon

This section deals with the following problem: given that P is star-shaped and its kernel contains a convex vertex (say v_1), compute the bad edges in $HE^+(P)$ using $DR^+(P)$. Clearly, P is weakly visible from e_1 (since P is visible from v_1) and e_1 is a convex edge. The algorithm for this problem has two phases. **Phase 1.A** computes the internal convex paths and the data structure storing the common intersections of the half-planes for the rays of $DR^+(P)$. This phase also identifies some bad edges. **Phase 1.B** completes the identification of all bad edges; it makes use of the internal convex paths and the data structure constructed in **Phase 1.A**. The computation consists of two separate parts, one using $DR_1^+(P)$ and the other using $DR_2^+(P)$. Due to the similarity between the two parts, we only discuss the part using $DR_1^+(P)$.

4.1 Phase 1.A

We first sketch the outline and describe the main operation of the algorithm, then we give the analysis and some computational details.

4.1.1 The Outline

We associate each $r_a^+ \in DR_1^+(P)$ with v_a, and denote by R_C the set of those rays in $DR_1^+(P)$ whose starting vertices are on a subchain C of $bd(P)$. The outline below sketches the divide-and-conquer strategies used by this phase.

Input. A subchain C of $bd(P)$ with $|C| = m$, R_C, and a positive integer d.

Step a.1. If $m \leq d$, then use one processor to perform the computation in $O(m)$ time.

Step a.2. If $d < m \leq d^6$, then divide C into two subchains C_1 and C_2 of equal size, and recursively solve the two subproblems on (C_1, R_{C_1}, d) and (C_2, R_{C_2}, d), in parallel. Then perform the computation for C and R_C using the output from the recursive calls on the two subproblems, with m/d processors and in $O(\log m + (d \log d)^{1/2})$ time.

Step a.3. If $m > d^6$, then partition C into $g = (m/d)^{1/3}$ subchains $C_1, C_2, \ldots, C_g$ of size $m^{2/3}d^{1/3}$ each. Then recursively solve the g subproblems, in parallel. Finally, perform the computation for C and R_C using the output from the g recursive calls, with m/d processors and in $O(\log m)$ time.

Observe that, if we could perform the various steps of the above outline within the claimed complexity bounds, then a procedure with such an outline would run in $O(d + \log m)$ time with $O(m/d)$ processors. Choosing $d = \log m$, then the time and processor complexities become $O(\log m)$ and $O(m/\log m)$, respectively. Therefore, a call to the procedure with input $(C, R_C, \log n)$, $|C| = n$, will take $O(\log n)$ time using $O(n/\log n)$ processors.

We must discuss what exactly is computed within the above outline. Let $L(R_C)$ be the set of lines containing the rays in R_C. The following is computed in **Steps a.1–a.3**.

(i) Compute the internal convex path $ICP(C)$.

(ii) Build the data structure which stores the common intersection of the relevant half-planes of $L(R_C)$ (e.g., $UPCI(L(R_C))$) (this data structure is needed for performing the lines-vs-segment tests).

(iii) Do the lines-vs-segment tests to identify the bad edges on C (we may not be able to identify all the bad edges in this phase; some of them are left to **Phase 1.B**).

Since P is star-shaped, in each of the three steps, the internal convex paths for the subchains of $bd(P)$ can be computed in the required complexity bounds by using the algorithm in [5]. By Lemma 5, R_C (thus $L(R_C)$) is sorted by slopes. Hence the common intersection of the relevant half-planes for $L(R_C)$ is also computable in the required complexity bounds (e.g., by the algorithms in [8, 5]). Therefore, we will focus on how to use the internal convex paths and the data structure for the lines-vs-segment tests returned from the recursive calls to identify the bad edges in this phase.

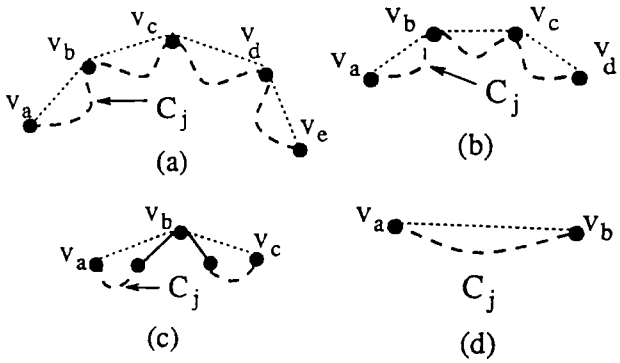

Figure 3: Illustrating the four cases of $|ICP(C_j)|$.

4.1.2 The Main Operation

The following operation is crucial in the conquer stages of **Steps a.2** and **a.3** (say, in **Step a.3**): given $ICP(C_s)$ and $UPCI(L(R_{C_s}))$, $s = 1, 2, \ldots, g$, determine the bad edges on C_j using R_{C_i}, for every pair of i and j, $1 \leq i < j \leq g$. We classify $ICP(C_j)$ into one of four possible cases according to its size, and show how to determine the bad edges on C_j by using R_{C_i} in each case. The classification is as follows.

Case (a). $ICP(C_j)$ has more than 3 segments (see Fig. 3 (a)). Then all the edges on C_j are bad. In Fig. 3 (a), the edges on C_j are all contained in $HE^+_{b-1} \cup HE^-_{d+1}$. Hence there is no need to use R_{C_i} to identify the bad edges on C_j.

Case (b). $ICP(C_j)$ has exactly 3 segments (see Fig. 3 (b)). Then the edges on subchains bd_{ab} and bd_{cd} of C_j are all bad (cf. Fig. 3 (b)), because they are contained in $HE^+_{b-1} \cup HE^-_{c+1}$. Furthermore, if $\overline{v_b v_c}$ is not an edge of P (i.e., $c > b+1$), then all the edges on C_j are bad because the edges on subchain bd_{bc} are also contained in $HE^+_{b-1} \cup HE^-_{c+1}$. The only edge on C_j that may not be bad is $\overline{v_b v_c}$, provided that $c = b + 1$. Thus the problem in this case, when $c = b + 1$, is that of finding whether or not $\overline{v_b v_{b+1}}$ is bad with respect to R_{C_i}.

Case (c). $ICP(C_j)$ has 2 segments (see Fig. 3 (c)). Then clearly all the edges on C_j, except the two edges $\overline{v_{b-1} v_b}$ and $\overline{v_b v_{b+1}}$, are bad (cf. Fig. 3 (c)). We need to check $\overline{v_{b-1} v_b}$ and $\overline{v_b v_{b+1}}$ by using R_{C_i}.

Case (d). $ICP(C_j)$ has 1 segment (see Fig. 4 (d)). If there is a ray $r^+_t \in R_{C_i}$ such that $bd_{ab} \subseteq HE^+_t$, then certainly all the edges on C_j are bad. Otherwise, we might have to "shoot" the rays of R_{C_i} onto C_j to find which of the edges on C_j are bad (this ray-shooting on C_j is to be done in **Phase 1.B**). Thus we need to check whether $bd_{ab} \subseteq HE^+_t$ for a $r^+_t \in R_{C_i}$. This check is done by testing segment $\overline{v_a v_b}$ against the rays in R_{C_i}.

From the discussion above, it is clear that the main computation in **Cases (a)–(d)** is to test an edge of C_j (in **Cases (b)** and **(c)**) or a segment of $ICP(C_j)$ (in **Case (d)**) against the rays in R_{C_i}, in order to find out whether the edge of C_j or C_j itself is bad with respect to R_{C_i}. We call such a test a *bad-segment test*.

We need some notations for describing the solution to the bad-segment tests. WLOG, let v_1 be at the origin and e_1 be on the positive x-axis (hence $P - e_1$ is above $l(e_1)$). The polar angle of a point $p \in bd(P) - v_1$, denoted by $\alpha(p)$, is the one from the positive x-axis counterclockwise to the ray starting at v_1 and going through p. Since v_1 is in the kernel of P and v_1 is convex, it follows that $0 \leq \alpha(p) < \pi$ for each point $p \in bd(P) - v_1$, and that the polar angles of the points on $bd(P) - v_1$, from v_1 counterclockwise to v_n, are in non-decreasing order. For each $r^+_t \in DR^+_1(P)$, h^+_t is on $bd_{(t+2)1}$. For a ray $r^+_t \in DR^+_1(P)$ and a segment s, we say s is properly contained in the *upper-right* (resp., *upper-left*) *quarter-plane* of r^+_t iff (i) s is contained in the intersection of the right (resp., left) half-plane of r^+_t and the left half-plane of the ray starting at v_1 and going through v_{t+1}, and (ii) s does not intersect r^+_t. Observe that if $e_l \in HE^+_t$, then e_l is properly contained in the upper-right quarter-plane of r^+_t.

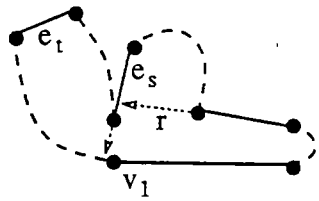

Figure 4: Illustrating the proof of Lemma 6.

WLOG, we assume that for each ray r in R_C, the right half-plane of r is equal to the lower half-plane of line $l(r)$.

We would like to obtain the answer to the bad-segment test on R_{C_i} and a segment s (of C_j or $ICP(C_j)$) by performing a lines-vs-segment test $Test(L(R_{C_i}), s)$ (because we can handle $Test(L(R_C), s)$ by using Lemma 4). In general, however, a bad-segment test cannot be answered by a lines-vs-segment test. For example, line $l(r)$ intersecting a segment s' does not necessarily imply that ray r also intersects s'. Furthermore, even though the half-line defined by a ray $r \in R_{C_i}$ does intersect segment s (of C_j or $ICP(C_j)$), r may first-hit a point on some C_k, $i \leq k < j$, before it intersects s (i.e., C_k blocks r from reaching s if r is viewed as a beam of light emanating from its starting vertex). Therefore, even if the result of $Test(L(R_{C_i}), s)$ does indicate that s is properly contained in the upper-right quarter-plane of r^+_t for a $r^+_t \in R_{C_i}$, the rays of R_{C_i} may be totally blocked from C_j. This means that, in this situation, no edge of C_j truly belongs to HE^+_t for any $r^+_t \in R_{C_i}$, and hence no edge of C_j is bad *with respect to* R_{C_i}. If we had to find out whether or not R_{C_i} is totally blocked from C_j, then for every k, $i \leq k < j$, we might either do $O(|ICP(C_k)|)$ bad-segment tests (for R_{C_i} and each segment of $ICP(C_k)$), or "shoot" each ray of R_{C_i} on $ICP(C_k)$ (by doing a binary search on $ICP(C_k)$). Since we can have $|ICP(C_k)|$ proportional to $|C_k|$ and $|R_{C_i}|$ proportional to $|C_i|$, either method would be too expensive to be performed within the desired complexity bounds. The next lemma saves us from doing these costly computations.

Lemma 6 *If a ray $r \in R_{C_i}$ first-hits some C_k at edge e_s, $i \leq k < j$, and if edge e_w on C_j is properly contained in the upper-right quarter-plane of r, then there exists a vertex v_z on $bd_{(s-1)(w-2)}$ such that $e_w \in HE^+_z$.*

Proof. Chain $bd_{(s-1)w}$ must start in the right half-plane of r. It then intersects r on e_s, and eventually enters the right half-plane of r to join v_w (see Fig. 4). Since P is visible from v_1, $ICP(bd_{sw})$ makes right turns only. Hence there must exist a vertex v_{z+1} on $ICP(bd_{sw})$ such that $s - 1 \leq z \leq w - 2$ and $e_w \in HE^+_z$. □

Lemma 6 implies that if edge e_w of C_j is properly contained in the upper-right quarter-plane of a ray $r \in R_{C_i}$, then e_w is definitely a bad edge. Note that for any k' such that $k' < i$ or $k' > j$, $C_{k'}$ cannot block the rays of R_{C_i} from C_j (by Lemma 1).

The next lemma justifies the use of the lines-vs-segment tests for the bad-segment tests.

Lemma 7 *For $i < j$, a bad-segment test on R_{C_i} and a segment s of C_j or $ICP(C_j)$ can be done by using $Test(L(R_{C_i}), s)$.*

Proof. Let r_t^+ be a ray in R_{C_i}. Recall that by our assumption, the left half-plane of r_t^+ is the upper half-plane of $l(r_t^+)$. The lemma holds if the following is true: (i) s is properly contained in the upper-right quarter-plane of r_t^+ iff s is below $l(r_t^+)$, and (ii) s is properly contained in the upper-left quarter-plane of r_t^+ iff s is above $l(r_t^+)$. We only give the proof for (i) (that for (ii) is very similar). If s is properly contained in the upper-right quarter-plane of r_t^+, then s is below $l(r_t^+)$. If s is below $l(r_t^+)$, then s is properly contained in the lower half-plane of $l(r_t^+)$ (i.e., the right half-plane of r_t^+) and certainly $s \neq \overline{v_t v_{t+1}}$. The facts that $i < j$, that $s \neq \overline{v_t v_{t+1}}$, that P is visible from v_1, and that v_1 is convex, together imply that $\alpha(v_{t+1}) \leq \alpha(p) < \pi$ for each point p on s. This means that s is to the left of the ray r starting at v_1 and going through v_{t+1}, and thus s is in the left half-plane of r. □

If the result of $Test(L(R_{C_i}), s)$ indicates that neither (i) s is below a line in $L(R_{C_i})$ nor (ii) s is above all the lines in $L(R_{C_i})$, then there must be a ray $r \in R_{C_i}$ such that the half-line defined by r intersects s. We need to distinguish two types of intersection between r and s.

Suppose that $Test(L(R_{C_i}), s)$ indicates that neither (i) nor (ii) occurs. Let ray r in R_{C_i} intersect s. Let $s = \overline{v_a v_b}$, $a < b$, and let $r(s)$ be the ray starting at v_a and going through v_b. If the starting vertex of r is properly contained in the right half-plane of $r(s)$, then we say r *pseudo-hits* s; otherwise, r *does not pseudo-hit* s. Furthermore, R_{C_i} is said to *pseudo-hit* s if (1) for each ray r' in R_{C_i}, r' intersecting s implies that r' pseudo-hits s, and (2) there is at least one ray in R_{C_i} that pseudo-hits s; otherwise, we say R_{C_i} *does not pseudo-hit* s. We distinguish the types of hits from R_{C_i} on s because only when R_{C_i} does not pseudo-hit s can the rays in R_{C_i} first-hit $bd_{ab} - v_a$ (it is still possible that R_{C_i} is blocked from bd_{ab}). If R_{C_i} pseudo-hits s, then clearly no ray in R_{C_i} can first-hit $bd_{ab} - v_a$.

We define the polar angle of s as $\alpha(s) = \alpha(r(s))$ (with obvious meaning for $\alpha(r(s))$). The following lemma characterizes the types of hits.

Lemma 8 *Suppose that a ray r_l^+ of R_{C_i} intersects a segment s on $ICP(C_j)$, $i < j$. Then r_l^+ pseudo-hits s iff $\alpha(r_l^+) > \alpha(s)$.*

Proof. Omitted. See the full paper. □

The next Lemma shows that the lines-vs-segment tests can be used to find out the type of hits from R_{C_i} on s.

Lemma 9 *The type of hits from R_{C_i} on a segment s of $ICP(C_j)$, $i < j$, can be determined by using the lines-vs-segment tests. in the same complexity bounds as those for $Test(L(R_{C_i}), s)$.*

Proof. Omitted. See the full paper. □

If the result of $Test(L(R_{C_i}), s)$ indicates that s is neither below a line in $L(R_{C_i})$ nor above all the lines in $L(R_{C_i})$, then one of the following three situations occurs: (1) if s is an edge of C_j, then s is not bad with respect to R_{C_i}, or (2) if s is not an edge of C_j and R_{C_i} pseudo-hits s, then bd_{ab} is not bad with respect to R_{C_i} (where $s = \overline{v_a v_b}$, $a < b$), or (3) if s is not an edge of C_j and R_{C_i} does not pseudo-hit s, then $|ICP(C_j)|$ may be **Case (d)** (which is to be handled in **Phase 1.B**).

4.1.3 Performing $Test(L(R_{C_i}), s)$ in Step a.3

We need to discuss how the lines-vs-segment tests are actually performed in the algorithm. We only discuss this for **Step a.3** (**Step a.2** is left to the full paper).

In **Step a.3**, there are $O(g^2) = O((m/d)^{2/3})$ pairs of C_i and C_j, $i < j$, where $m = |C|$. For each pair of C_i and C_j, we need to do $Test(L(R_{C_i}), s)$ for $O(1)$ segments s on $ICP(C_j)$, if $ICP(C_j)$ is not of **Case (a)**. There are totally $O(m/d)$ processors available. Thus $O(g) = O((m/d)^{1/3})$ processors are allocated to each pair of C_i and C_j. It suffices to show how to do one $Test(L(R_{C_i}), s)$ using $O(g)$ processors.

Our primary tool is Lemma 4. In **Step a.3**, to perform $Test(L(R_{C_i}), s)$ in $O(\log m)$ time, we need to achieve two things: (i) in performing the test, Lemma 4 is recursively applied only $O(1)$ times, and (ii) the size of the test range, after (i) is done, is reduced to $O((m/d)^{1/3})$ (so that the brute force method can take over).

In the conquer stage of **Step a.3**, $UPCI(L(R_{C_i}))$ is available (stored in a rank tree), for each $i = 1, 2, \ldots, g$, and we compute $UPCI(L(R_C))$ from the $UPCI(L(R_{C_i}))$'s. Because $L(R_C)$ is sorted by slopes, for each i, $bd(UPCI(L(R_{C_i}))) \cap bd(UPCI(L(R_C)))$ consists of at most *one* connected component, and $bd(UPCI(L(R_{C_i}))) - bd(UPCI(L(R_C)))$ consists of at most *two* connected components. After $UPCI(L(R_C))$ is computed, we still retain the (at most) two connected components of $bd(UPCI(L(R_{C_i}))) - bd(UPCI(L(R_C)))$ in two separate rank trees. This structure can be readily maintained in every recursion level of **Step a.3**. Using this structure, it is easy to obtain the information about each $UPCI(L(R_{C_i}))$ (either from $UPCI(L(R_C))$ or from the two connected components of $bd(UPCI(L(R_{C_i}))) - bd(UPCI(L(R_C)))$). Furthermore, for a subchain C_i^k of C_i, the information about $UPCI(L(R_{C_i^k}))$ can be obtained from $UPCI(L(R_{C_i}))$ and $UPCI(L(R_{C_i^k})) - UPCI(L(R_{C_i}))$. In general, the information about the common intersection of the half-planes at each level of the algorithm can be obtained from the information stored at its ancestor levels.

WLOG, we assume that in the recursive call on (C_i, R_{C_i}, d), C_i was partitioned into $O((m/d)^{2/9})$ subchains (the case where C_i was as in **Step a.2** can be easily taken care of within the desired complexity bounds). In performing $Test(L(R_{C_i}), s)$, we repeat the following two steps. (i) Apply Lemma 4 to $Test(L(R_{C_i}), s)$ using $O((m/d)^{1/3})$ processors. (Note that the information on $UPCI(L(R_{C_i^k}))$, for each subchain C_i^k of C_i, is obtained using the structure described in the previous paragraph.) (ii) Either we have found the answer to $Test(L(R_{C_i}), s)$ in (i), or we have reduced the test to a unique subchain C_i^k of C_i, in which case we continue the test by repeating (i) for $Test(L(R_{C_i^k}), s)$.

The size of C is m at the current recursion level of the algorithm. The following can be easily proved by induction: at the i-th recursion level below, $i \geq 1$, the chain size is $O(m^{f(i)}d^{1-f(i)})$ and the chain is partitioned into $O((m/d)^{f(i)(1/3)})$ subchains (for the $(i+1)$-th level), where $f(i) = (2/3)^i$. We want to stop the recursive procedure for $Test(L(R_{C_i}), s)$ when we reach the structure for the i-th level below (in the recursion tree of **Phase 1.A**) for some $i \geq 1$, such that the size of the chain at that level is $\leq (m/d)^{1/3}$. Hence we have

$$(m/d)^{1/3} \geq m^{f(i)}d^{1-f(i)},$$

which is equivalent to

$$m^{(1/3)-f(i)} \geq d^{(4/3)-f(i)}.$$

Choosing i to be 5, the inequality becomes

$$m^{49/243} \geq d^{1+(49/243)},$$

and it holds as long as $m \geq d^6$. Thus, the test range size is reduced to $O((m/d)^{1/3})$ by going down at most *five* levels in the recursion tree of **Phase 1.A**.

4.1.4 The Procedure for Step a.3

We only sketch the computation for **Step a.3**, as follows. (1) For every pair of C_i and C_j such that $i < j$ and $|ICP(C_j)|$ is not **Case (a)**, do $Test(L(R_{C_i}), s)$ for each segment s on $ICP(C_j)$. (2) For every R_{C_i}, find all segments s such that s is tested in (1) and s is *not below* any line in $L(R_{C_i})$; furthermore, among these segments s, find $s_i = \overline{v_a v_b}$ such that the vertex indices a and b for s_i are no bigger than the vertex indices for any of these segments, where $a < b$ (by Lemma 1, R_{C_i} cannot reach any of these segments other than s_i). Let s_i be on $ICP(C_{j'})$ for some j'. (3) If $|ICP(C_{j'})|$ is **Cases (b)** or **(c)**, or s_i is above all the lines in $L(R_{C_i})$, or R_{C_i} pseudo-hits s_i, then the bad edges on the subchain of $bd(P)$ from the last vertex of C_i counterclockwise to v_b (on $C_{j'}$) can be determined; otherwise, $|ICP(C_{j'})|$ is **Case (d)** and R_{C_i} does not pseudo-hit s_i (in this case, we will decide in **Phase 1.B** the bad edges on the subchain of $bd(P)$ from C_i counterclockwise to $C_{j'}$).

4.2 Phase 1.B

By using the structure built in **Phase 1.A**, this phase finishes the identification of the edges in $HE^+(P)$. **Phase 1.B** has the same algorithmic outline as **Phase 1.A**. Its computation follows the recursion tree of **Phase 1.A**, level by level, from the root down to the leaves.

The following operation is typical in this phase: identify the bad edges on C' by using R_C, where $C \neq C'$ and $|ICP(C')|$ is **Case (d)**. What we do is to shoot R_C onto each subchain C'_i of C'. Thus, we encounter one of the four cases at each $ICP(C'_i)$, and we may have to continue the recursive computation on a unique C'_j where $|ICP(C'_j)|$ is again **Case (d)**.

We will not go into the details of this procedure, since these details are similar to those in **Phase 1.A**. But we would like to point out one thing that must be handled carefully in this top-down procedure, as follows. If one keeps using the same data structure for R_C (i.e., $UPCI(L(R_C))$) in the upcoming recursion levels below C', then the same $O(\log |C|)$ time will be spent at each level. This will not give an $O(\log n)$ time algorithm; instead, the time bound so resulted will be $O(\log n \log \log n)$. To avoid this inefficiency, what we do when shooting R_C onto C' is to partition C into $g = (|C|/d)^{1/3}$ subchains $C_1, C_2, \ldots, C_g$ (as in **Phase 1.A**). Then, by using the structures for the $UPCI(L(R_{C_k}))$'s (rather than $UPCI(L(R_C))$), every R_{C_k} is independently shot onto each of the C'_i's. Note that each $UPCI(L(R_{C_k}))$ can be recovered from $UPCI(L(R_C))$ and from the (at most) two connected components of $UPCI(L(R_{C_k})) - UPCI(L(R_C))$ in the desired complexity bounds.

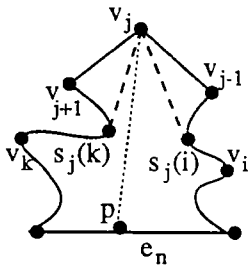

Figure 5: Illustrating Lemma 10.

5 Checking the Weak Visibility of a Polygon from an Edge

This section concerns the following problem: check whether or not a simple polygon P is weakly visible from a specified edge e of P (i.e., the case solved sequentially in [2, 13]). We show how to solve this problem in $O(\log n)$ time using $O(n/\log n)$ processors. Our solution consists of two parts: a preprocess and a two-phase procedure. WLOG, let $e = e_n$, e_n be on the x-axis, and $x(v_1) > x(v_n)$. The preprocess reduces the problem to that of checking the weak visibility of a simple polygon from a *convex* edge. Also, the preprocess checks to make sure that for every i, $1 < i < n$, none of the following local conditions is satisfied: (i) e does not intersect the left half-plane of $r(e_i)$, and (ii) $\alpha(r(e_{i-1})) < \pi$, $\alpha(r(e_i)) > \pi$, and v_i is nonconvex, where $r(e_i)$ is the ray starting at v_i and containing e_i. (For any i, P is not weakly visible from e if either (i) or (ii) is satisfied.) The two-phase procedure handles the problem of checking the weak visibility of P from a convex edge (say, e_n).

5.1 The Basic Idea

We still use $ICP(bd_{i,j})$ to represent the directed shortest path from v_i to v_j that goes through *only* the vertices of $bd_{i,j}$ (i.e., the computation of $ICP(bd_{i,j})$ is based only on $bd_{i,j}$ and disregards $bd(P) - bd_{i,j}$). Note that in general, $ICP(bd_{i,j})$ does not necessarily make consistent right turns for $i < j$; furthermore, $ICP(bd_{i,j})$ may even intersect the exterior of P (because it can intersect $bd(P) - bd_{i,j}$).

The lemma below gives the basic idea for solving this weak visibility problem. For $i < i'$ (resp., $i > i''$), let $s_i(i')$ (resp., $s_i(i'')$) be the segment on $ICP(bd_{i,i'})$ (resp., $ICP(bd_{i'',i})$) that contains v_i.

Lemma 10 *If P is weakly visible from e_n, then for any i, j, and k, $1 \leq i < j < k \leq n$, a scan of the interior angle of P at v_j, from edge e_{j-1} clockwise to edge e_j, encounters e_{j-1}, $s_j(i)$, $s_j(k)$, and e_j, in that order (cf. Fig. 5).*

Proof. Omitted. See the full paper. □

By Lemma 10, if $s_j(i)$ and $s_j(k)$ are not in correct order with e_{j-1} and e_j for some i, j, and k, $i < j < k$, then P is not weakly visible from e. If they are in correct order, then there is a ray (say, the one starting at v_j and containing $s_j(i)$) separating $ICP(bd_{i,j})$ from $ICP(bd_{j,k})$. Let the ray starting at v_j and containing $s_j(i)$ be denoted by $r(s_j(i))$. If $r(s_j(i))$ separates $ICP(bd_{i,j})$ from $ICP(bd_{j,k})$, then $ICP(bd_{i,k})$ can be computed efficiently from $ICP(bd_{i,j})$ and $ICP(bd_{j,k})$. This is the idea used in the recursive algorithm.

It is known that if there is a ray (e.g., $r(s_j(1))$) separating $ICP(bd_{1,j})$ from $ICP(bd_{j,n})$ for every j, $1 < j < n$, then P is weakly visible from e (e.g., see [12]). Our ultimate goal,

therefore, is to compute $s_j(1)$ and $s_j(n)$ for every j and check their relative order based on Lemma 10. Clearly, our main problem is to compute all the $s_j(1)$'s and $s_j(n)$'s.

The algorithm for computing the $s_j(1)$'s and $s_j(n)$'s consists of two phases. **Phase 2.A** uses Lemma 10 to compute the internal convex paths on certain subchains of $bd(P)$ (hence at every recursion level, either it succeeds in computing $ICP(C)$ for a subchain C of $bd(P)$, or it concludes that P is not weakly visible from e). After the first phase is completed, if P is still not known to be non-weakly visible from e, then the algorithm proceeds with **Phase 2.B**. For every j, **Phase 2.B** constructs $ICP(bd_{1j})$ and $ICP(bd_{jn})$ (by using the internal convex paths obtained in **Phase 2.A**) and reports $s_j(1)$ and $s_j(n)$.

We need to characterize $ICP(bd_{ij})$ before going to the two-phase algorithm. For bd_{ij}, $i < j$, we say $ICP(bd_{ij})$ is *in order* if the vertices of $ICP(bd_{ij})$ are in increasing order of vertex indices in the walk of $ICP(bd_{ij})$ from v_i to v_j. For a segment $s = \overline{v_a v_b}$, $a < b$, let $r(s)$ be the ray starting at v_a and containing s. The following lemmas enable us to compute the internal convex paths efficiently.

Lemma 11 *Suppose P is not known to be non-weakly visible from e. For $i < j$, if $ICP(bd_{ij})$ is in order and makes only right turns, then for two consecutive segments $\overline{v_a v_b}$ and $\overline{v_b v_c}$ on $ICP(bd_{ij})$, $a < b < c$, $\alpha(r(\overline{v_b v_c})) < \alpha(r(\overline{v_a v_b}))$.*

Proof. If $\alpha(r(\overline{v_a v_b})) \geq \pi$, then the lemma is obviously true because $ICP(bd_{ij})$ is in order and makes only right turns. If $\alpha(r(\overline{v_a v_b})) < \pi$, then either $\alpha(r(\overline{v_b v_c})) < \alpha(r(\overline{v_a v_b}))$ or $\alpha(r(\overline{v_b v_c})) > \alpha(r(\overline{v_a v_b})) + \pi$ (since $ICP(bd_{ij})$ makes only right turns). But $\alpha(r(\overline{v_b v_c})) > \alpha(r(\overline{v_a v_b})) + \pi$ and $ICP(bd_{ij})$ being in order imply that the local condition (ii) tested in the preprocess is satisfied at v_b, a contradiction. □

Corollary 1 *Suppose P is not known to be non-weakly visible from e. For $i < j$, if $ICP(bd_{ij})$ is in order and makes only right turns, then the polar angles of the segments on $ICP(bd_{ij})$ are in sorted order.*

Proof. An immediate consequence of Lemma 11. □

Lemma 12 *Suppose P is not known to be non-weakly visible from e. For i, j, and k, $1 \leq i < j < k \leq n$, if both $ICP(bd_{ij})$ and $ICP(bd_{jk})$ are in order and make only right turns, then $ICP(bd_{ik})$ is in order and makes only right turns.*

Proof. Omitted. See the full paper. □

Lemma 13 *Suppose P is not known to be non-weakly visible from e. For $i < j$, if $ICP(bd_{ij})$ is in order and makes only right turns, then $ICP(bd_{ij})$ can be partitioned into at most two subpaths C' and C'' such that C' (resp., C'') is completely on the convex hull of C' (resp., C'').*

Proof. Let l be the horizontal line tangent to $ICP(bd_{ij})$ (let l touch $ICP(bd_{ij})$ at v_a), such that $ICP(bd_{ij}) - v_a$ is above l. By Corollary 1, the polar angles of the segments on $ICP(bd_{ia})$ (resp., $ICP(bd_{aj})$) are all $\geq \pi$ (resp., $\leq \pi$). Hence, $ICP(bd_{ia})$ (resp., $ICP(bd_{aj})$) is completely on the convex hull of $ICP(bd_{ia})$ (resp., $ICP(bd_{aj})$). □

5.2 Phase 2.A

This phase consists of three steps: **Step c.1**, **Step c.2**, and **Step c.3**. Its algorithmic outline and recurrences for the time and processor complexities are similar to those of **Phase 1.A**. Given input $(C, |C|, d)$ to each recursive call, where C is a subchain of bd_{1n}, either an answer "no" (indicating that P is already known to be non-weakly visible from e) or $ICP(C)$ is returned. All internal convex paths in this phase are stored in rank trees. Here we only discuss the computation in **Step c.3**.

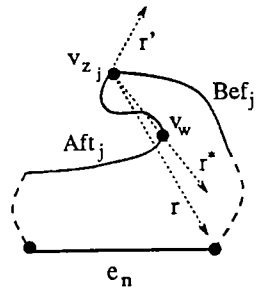

Figure 6: An example for illustrating Lemma 14.

In **Step c.3**, a chain $C = bd_{st}$ is partitioned into $g = (m/d)^{1/3}$ subchains $C_1, C_2, \ldots, C_g$, where $|C| = m$ and $s < t$. There are $O(g^3)$ processors available. After the recursive calls for the C_i's, if P is not known to be non-weakly visible from e, then $ICP(C_1), ICP(C_2), \ldots, ICP(C_g)$ are all available. Let v_{z_j} be the common vertex of C_j and C_{j+1}, for each $j = 1, 2 \ldots, g-1$. For each v_{z_j}, let Bef_j (resp., Aft_j) denote the subchain of C *before* (resp., *after*) v_{z_j}, that is, $Bef_j = bd_{sz_j}$ (resp., $Aft_j = bd_{z_j t}$). One important operation in this step is to find $s_{z_j}(s)$ and $s_{z_j}(t)$ for each v_{z_j}. This is because if $s_{z_j}(s)$ and $s_{z_j}(t)$ are in correct order with e_{z_j-1} and e_{z_j} (cf. Lemma 10) for each v_{z_j}, then there is a ray separating every pair of $ICP(C_i)$ and $ICP(C_j)$, which makes the computation of $ICP(C)$ possible. Note that $ICP(Bef_j)$ and $ICP(Aft_j)$ are in general not explicitly available for the computation of $s_{z_j}(s)$ and $s_{z_j}(t)$. We only explain how to use Aft_j to compute $s_{z_j}(t)$ (the case for $s_{z_j}(s)$ is similar).

The following lemma is useful in computing the $s_{z_j}(s)$'s.

Lemma 14 *Suppose that P is not known to be non-weakly visible from e and that every $ICP(C_k)$ is in order and makes only right turns. For a v_{z_j}, let r be the ray starting at v_{z_j} and going through v_1, and let r' be the ray on line $l(\overline{v_{z_j} v_n})$ which starts at v_{z_j} and does not contain v_n (see Fig. 6). For any $i \in \{j+1, j+2, \ldots, g\}$, if the intersection of the left half-plane $lfp(r)$ of r and the right half-plane $rtp(r')$ of r' intersects $ICP(C_i) - v_{z_j}$, then P is not weakly visible from e.*

Proof. Omitted. See the full paper. □

For example, in Fig. 6, $ICP(C_i) - v_{z_j}$ intersects $lfp(r) \cap rtp(r')$ in such a way that a vertex v_w on $ICP(C_i) - v_{z_j}$ is contained in $lfp(r) \cap rtp(r')$. Let r^* be the ray starting at v_{z_j} and going through v_w. Then e_n does not intersect the left half-plane of r^*. Hence P is not weakly visible from e_n.

Observe that in **Phase 2.A**, if P is not known to be non-weakly visible from e, then all the $ICP(C_k)$'s have the properties stated in Lemmas 11 and 12 (that all the $ICP(C_k)$'s are in order and make only right turns can be easily proved by induction by using Lemma 12). Therefore, we can check whether each $ICP(C_i) - v_{z_j}$ intersects $lfp(r) \cap rtp(r')$ in $O(\log m)$ time using $O(g)$ processors.

We compute the $s_{z_j}(t)$'s as follows. We first do the checking based on Lemma 14 for every v_{z_j} (in $O(\log m)$ time using $O(g^2)$ processors). Suppose P is still not known to be non-weakly visible from e, then we compute the common tangent between v_{z_j} and $ICP(C_i)$ for each $C_i \subseteq Aft_j$, by

using the ray starting at v_{z_j} and containing $\overline{v_1 v_{z_j}}$ as a ray separating v_{z_j} from $ICP(C_i)$. This is easily computed in $O(\log m)$ time using $O(g)$ processors for each C_i. Suppose among the $O(g)$ common tangents so obtained, the tangent between v_{z_j} and $ICP(C_{i'})$ is the one first encountered if we use a ray originating at v_{z_j} to scan around v_{z_j} clockwise, by starting at $\overline{v_1 v_{z_j}}$. Then $s_{z_j}(t)$ is the common tangent $\overline{v_{z_j} v_b}$ between v_{z_j} and $ICP(C_{i'})$ (v_b is on $ICP(C_{i'})$). Therefore, each $s_{z_j}(t)$ is computable in $O(\log m)$ time using $O(g^2)$ processors.

Given $s_{z_j}(s)$ and $s_{z_j}(t)$ for each v_{z_j}, if $s_{z_j}(s)$ and $s_{z_j}(t)$ are in correct order with e_{z_j-1} and e_{z_j} (cf. Lemma 10), then the ray $r(s_{z_j}(s))$ (starting at v_{z_j} and containing $s_{z_j}(s)$) separates Bef_j from Aft_j, and hence it separates $ICP(C_i)$ from $ICP(C_k)$ for each $C_i \subseteq Bef_j$ and each $C_k \subseteq Aft_j$. After $r(s_{z_j}(s))$ is available, we can compute the common tangent between each pair of $ICP(C_i)$ and $ICP(C_k)$ in $O(\log m)$ time using $O(g)$ processors (by using [8, 5]).

Next, using the $O(g^2)$ common tangents so obtained, we build a *complete binary tree of internal convex paths* whose leaves are associated with $C_1, C_2, \ldots, C_g$, respectively. The root of the tree is associated with $bd_{st} = C$ and it stores $ICP(bd_{st})$ (in a rank tree).

The complete binary tree of internal convex paths above is denoted by T_C. The root of T_C is denoted by $root(T_C)$ and the left (resp., right) child of an internal node u of T_C is denoted by $lch(u)$ (resp., $rch(u)$). The information associated with each node of T_C is as follows. $root(T_C)$ is associated with bd_{st} and it stores $ICP(bd_{st})$. For an internal node u, the subchain of C associated with u is the union of the subchains associated with the descendent leaves of u. Suppose that the subchain associated with u is bd_{ac} and the subchains associated with $lch(u)$ and $rch(u)$ are respectively bd_{ab} and bd_{bc}, $a < b < c$. Observe that $ICP(bd_{ab}) - ICP(bd_{ac})$ (resp., $ICP(bd_{bc}) - ICP(bd_{ac})$) consists of at most one connected component. The information stored at $lch(u)$ (resp., $rch(u)$) is $ICP(bd_{ab}) - ICP(bd_{ac})$ (resp., $ICP(bd_{bc}) - ICP(bd_{ac})$), represented by a rank tree. The height of T_C is $O(\log m)$. This tree structure has been used in [9] for triangulating a *one-sided* monotone polygon (in [9], each node of the tree stores a portion of the convex hull for its associated subchain; see [9] for the definitions and the details). The construction of T_C is done by an algorithm in [9, 5], in $O(\log m)$ time using $O(g^3)$ processors.

In total, the procedure for **Step c.3** requires $O(\log m)$ time and $O(g^3)$ processors.

5.3 Phase 2.B

At the root of the recursion tree of **Phase 2.A**, if P has been decided to be non-weakly visible from e, then the algorithm stops. Otherwise, we proceed with **Phase 2.B** to compute $s_j(1)$ and $s_j(n)$ for every j. WLOG, we just show how to compute the $s_j(1)$'s (the computation for the $s_j(n)$'s is similar).

In **Phase 2.A**, we have constructed a complete binary tree of the internal convex paths (cf. Step **c.3**). We denote this tree by T. T has $O(n/d) = O(n/\log n)$ nodes. The root of T stores $ICP(bd_{1n})$. Each non-root node uses a rank tree to store at most one connected portion of the internal convex path for the subchain of bd_{1n} associated with that node. Hence there are totally $O(n/\log n)$ internal convex paths stored at the nodes of T.

The algorithmic structure of **Phase 2.B** is the same as the structure of tree T. The procedure follows a top-down paradigm. It starts at $root(T)$, then goes to the two children

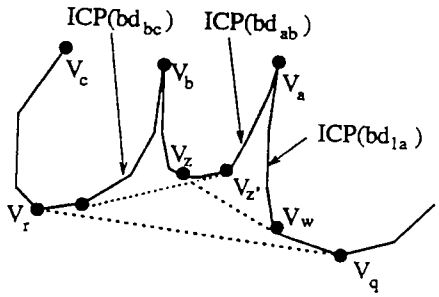

Figure 7: Illustrating Lemma 15.

of $root(T)$, and so on, level by level, until the leaves of T are reached. Because the height of T is $O(\log n)$, we need to process each level of T (in most part of the procedure) in $O(1)$ time, in order to achieve an $O(\log n)$ time algorithm.

Let $u \neq root(T)$ be an internal node of T. Let the chains associated with u, $lch(u)$, and $rch(u)$ be respectively bd_{ac}, bd_{ab}, and bd_{bc}, $1 < a < b < c$. The main computation of **Phase 2.B** is based on the lemma below. WLOG, we assume that up to the level of u, P is not known to be non-weakly visible from e.

Lemma 15 *Let the common tangent between $ICP(bd_{1a})$ and $ICP(bd_{ac})$ touch $ICP(bd_{1a})$ at v_q and $ICP(bd_{ac})$ at v_r, the common tangent between $ICP(bd_{1a})$ and $ICP(bd_{ab})$ touch $ICP(bd_{1a})$ at v_w and $ICP(bd_{ab})$ at v_z, and the common tangent between $ICP(bd_{ab})$ and $ICP(bd_{bc})$ touch $ICP(bd_{ab})$ at $v_{z'}$ (see Fig. 7). Then the common tangent between $ICP(bd_{1a})$ and a vertex on bd_{ab} touches $ICP(bd_{1a})$ on $ICP(bd_{wa})$ (with $w \geq q$), and the common tangent between $ICP(bd_{1b})$ and a vertex on bd_{bc} touches $ICP(bd_{1b})$ either on $ICP(bd_{qb})$ ($= ICP(bd_{qw}) \cup \overline{v_w v_z} \cup ICP(bd_{zb})$) (when $v_r \in ICP(bd_{bc})$) or on $ICP(bd_{z'b})$ (when $v_r \in ICP(bd_{ab})$).*

Proof. Omitted. See the full paper. □

We call $ICP(bd_{qa})$ in Lemma 15 the *left* internal convex path to the chain (i.e., bd_{ac}) associated with u and denote it by $LICP_u$. Likewise, $LICP_{lch(u)} = ICP(bd_{wa})$ and $LICP_{rch(u)} = ICP(bd_{qb})$ (when $v_r \in ICP(bd_{bc})$). Observe that $LICP_{lch(u)}$ and $LICP_{rch(u)}$ are disjoint except possibly at v_w. For convenience, we also let ICP_u denote $ICP(bd_{ac})$.

The computation based on Lemma 15 involves the following main operations: (i) computing the common tangent between two internal convex paths, (ii) splitting an internal convex path into two subpaths, and (iii) combining two internal convex paths together to form another path. We need to perform each of these operations in $O(1)$ time for most part of the algorithm. Obviously, an appropriate data structure for representing the internal convex paths is essential in this process.

We can no longer use rank trees to represent the internal convex paths because of their logarithmic heights. Instead, we represent the internal convex path stored at each node of T by an array. The conversion of the rank tree representation into the array representation for the internal convex paths is easy to do in $O(\log n)$ time using $O(n/\log n)$ processors. We henceforth assume that the internal convex path stored at each node of T is represented by an array.

The computation on a node u of T involves ICP_u and $LICP_u$, which need to be represented in such a way that enables us to compute their common tangent in $O(1)$ time.

Recall that $lch(u)$ (resp., $rch(u)$) stores the portion of $ICP_{lch(u)}$ (resp., $ICP_{rch(u)}$) that is not on ICP_u. Only $root(T)$ has its internal convex path, i.e., $ICP(bd_{1n})$, stored in a single array. $lch(root(T))$ may have one portion

of $ICP(bd_{1(n/2)})$ stored in $root(T)$. The left child of $lch(root(T))$ may have one portion of $ICP(bd_{1(n/4)})$ stored in $lch(root(T))$, which may again have a portion stored in $root(T)$. In general, a node u may have a portion of ICP_u stored in each of its ancestors in T. That is, ICP_u is obtained from the $O(\log n)$ arrays stored at its ancestors. Therefore, we represent ICP_u by using $O(\log n)$ subarrays.

Let ICP_u be represented by $A_u(1), A_u(2), \ldots, A_u(k)$, in order, where each $A_u(i)$ is a subarray of an array stored at an ancestor of u. Each $A_u(i)$ is specified by two pointers, one pointing to the first element of $A_u(i)$ and the other pointing to the last element of $A_u(i)$. Suppose those $2k$ pointers are available in the beginning of the computation at u, where u is an internal node of T. In the process at u, we split ICP_u at $v_{z'}$, where $v_{z'}$ (resp., $v_{z''}$) on $ICP_{lch(u)}$ (resp., $ICP_{rch(u)}$) is the endpoint of the common tangent between $ICP_{lch(u)}$ and $ICP_{rch(u)}$, as follows. Let $v_{z'}$ be contained in $A_u(i)$ for some i. Then $A_u(i)$ is split at $v_{z'}$ into two subarrays A'_u and A''_u, such that $v_{z'}$ is the last element of A'_u and $v_{z''}$ is the first element of A''_u. Let the representation of $ICP_{lch(u)}$ be the union of $ICP_{lch(u)} \cap ICP_u$ (represented by $O(\log n)$ pieces from u) and $ICP_{lch(u)} - ICP_u$ (one single piece stored at $lch(u)$). That is, $ICP_{lch(u)}$ is represented by $A_u(1), A_u(2), \ldots, A_u(i-1), A'_u$, and $B_{lch(u)}$, in order, where $B_{lch(u)}$ is the array representing $ICP_{lch(u)} - ICP_u$. The similar thing is done for $ICP_{rch(u)}$.

We associate with ICP_u k *size* parameters $size_u(1), size_u(2), \ldots, size_u(k)$, where $size_u(i) = |A_u(1)| + \cdots + |A_u(i)|$. Using the size parameters, we can quickly access the j-th vertex on ICP_u for any j (say, in $O(1)$ time using $O((\log n)^{1/2})$ processors). When ICP_u is split to form $ICP_{lch(u)}$ and $ICP_{rch(u)}$, the size parameters for the representation of $ICP_{rch(u)} \cap ICP_u$ can be easily updated (in $O(1)$ time using $O(\log n)$ processors) because we just need to subtract/add a same number from/to all the parameters in the list for $rch(u)$ and then add a new parameter (for $ICP_{rch(u)} - ICP_u$) to the beginning of the list. The update on the parameters for $lch(u)$ is even easier (only a new term is added to the end of the parameter list).

It can be shown that $LICP_u$ is also represented by $O(\log n)$ subarrays. Hence the "split" and "combine" operations on $LICP_u$ are also the same as on ICP_u.

With these representations for the ICP_u's and $LICP_u$'s, the common tangent between every pair of ICP_u and $LICP_u$, at each level of T (in most part of the algorithm), can be computed in $O(1)$ time and $O(n/\log n)$ processors by using [1].

Acknowledgement. The author would like to thank Professor Mikhail Atallah for his great support and encouragement to this research, and for many inspirational discussions.

References

[1] M. J. Atallah and M. T. Goodrich. "Parallel algorithms for some functions of two convex polygons," *Algorithmica*, 3 (1988), pp. 535–548.

[2] D. Avis and G. T. Toussaint. "An optimal algorithm for determining the visibility polygon from an edge," *IEEE Trans. Comput.*, C-30 (12) (1981), pp. 910–914.

[3] B. K. Bhattacharya, A. Mukhopadhyay, and G. Toussaint. "A linear time algorithm for computing the shortest line segment from which a polygon is weakly externally visible," *Proc. WADS*, 1991, Ottawa, Canada, pp. 412–424.

[4] B. Chazelle and L. J. Guibas. "Visibility and intersection problems in plane geometry," *Disc. Comput. Geo.*, 4 (1989), pp. 551–581.

[5] D. Z. Chen. "Efficient geometric algorithms in the EREW-PRAM," *Proc. 28-th Annual Allerton Conf. Commun., Contr., and Comput.*, 1990, pp. 818–827.

[6] Y. T. Ching, M. T. Ko, and H. Y. Tu. "On the cruising guard problems," Technical Report, 1989, Institute of Information Science, Academia Sinica, Taipei, Taiwan.

[7] R. Cole and M. T. Goodrich. "Optimal parallel algorithms for polygon and point-set problems," *Proc. 4-th Annual ACM Symp. Comput. Geo.*, 1988, pp. 211–220. (To appear in *Algorithmica*.)

[8] M. T. Goodrich. "Finding the convex hull of a sorted point set in parallel," *IPL*, 26 (1987/88), pp. 173–179.

[9] M. T. Goodrich. "Triangulating a polygon in parallel," *J. of Algorithms*, 10 (1989), pp. 327–351.

[10] M. T. Goodrich. "Planar separators and parallel polygon triangulation," to appear in the *24-th Annual ACM STOC*, 1992, Victoria, Canada.

[11] M. T. Goodrich, S. B. Shauck, and S. Guha. "Parallel methods for visibility and shortest path problems in simple polygons (Preliminary version)," *Proc. 6-th Annual ACM Symp. Comput. Geo.*, 1990, pp. 73–82.

[12] L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. Tarjan. "Linear time algorithms for visibility and shortest paths problems inside triangulated simple polygons," *Algorithmica*, 2 (1987), pp. 209–233.

[13] S. K. Ghosh, A. Maheshwari, S. P. Pal, S. Saluja, and C. E. V. Madhavan. "Characterizing weak visibility polygons and related problems," Tech. Rep., No. IISc-CSA-90-1, 1990, Dept. Comput. Sci. and Auto., Indian Institute of Science.

[14] P. J. Heffernan and J. S. B. Mitchell. "Structured visibility profiles with applications to problems in simple polygons," *Proc. 6-th Annual ACM Symp. Comput. Geo.*, 1990, pp. 53–62.

[15] Y. Ke. "Detecting the weak visibility of a simple polygon and related problems," Manuscript, Dept. of Comput. Sci., The Johns Hopkins University, 1988.

[16] C. P. Kruskal, L. Rudolph and M. Snir. "The power of parallel prefix," *IEEE Trans. Comput.*, C-34 (1985), pp. 965–968.

[17] R. E. Ladner and M. J. Fischer. "Parallel prefix computation," *JACM*, 27 (4) (1980), pp. 831–838.

[18] D. T. Lee and A. Lin. "Computing the visibility polygon from an edge," Tech. Rep., Northwestern Univ., 1984.

[19] D. T. Lee and F. P. Preparata. "An optimal algorithm for finding the kernel of a polygon," *JACM*, 26 (1979), pp. 415–421.

[20] M. H. Overmars and J. van Leeuwen. "Maintenance of configurations in the plane," *J. Comput. and Syst. Sci.*, 23 (1981), pp. 166–204.

[21] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, New York, 1985.

[22] J.-R. Sack and S. Suri. "An optimal algorithm for detecting weak visibility of a polygon," *IEEE Trans. Comput.*, C-39 (10) (1990), pp. 1213–1219.

[23] G. T. Toussaint. "A linear-time algorithm for solving the strong hidden-line problem in a simple polygon," *Pattern Recognition letters*, 4 (1986), pp. 449–451.

[24] C. K. Yap. "Parallel triangulation of a polygon in two calls to the trapezoidal map," *Algorithmica*, 3 (1988), pp. 279–288.