# Determining Weak External Visibility of Polygons in Parallel

Danny Z. Chen*

## Abstract

Given an $n$-vertex simple polygon $P$, the problem of determining the weak external visibility of $P$ is that of deciding whether there is a line segment $s$ such that $s$ does not intersect the interior of $P$ and that every point on the boundary of $P$ is visible from some point on $s$, and (if such a segment $s$ exists) finding the shortest such segment. In this paper, we present an optimal parallel algorithm for solving this problem. Our parallel algorithm runs in $O(\log n)$ time using $O(n/\log n)$ processors in the CREW PRAM computational model. Our approach is very different from that of the sequential solutions for this problem.

## 1 Introduction

In this paper, we deal with the *weak external visibility* of simple polygons, which is defined as follows: For a simple polygon $P$ whose boundary $bd(P)$ is considered to be "opaque", two points $p$ and $q$ not in the interior of $P$ are said to be *externally visible* from each other if and only if the line segment $\overline{pq}$ with endpoints $p$ and $q$ does not intersect the interior of $P$, and $p$ is *weakly externally visible* from an object $B$ if and only if $p$ is externally visible from some point on $B$.

We consider the problem of determining the weak external visibility of a simple polygon (called the WEV problem): Given an $n$-vertex simple polygon $P$, decide whether there is a line segment $s$ such that $s$ does not intersect the interior of $P$ and that every point on $bd(P)$ is weakly externally visible from $s$, and (if such a segment $s$ exists) find the shortest such segment, denoted by $s^*$. Intuitively, if $P$ represents a forbidden territory of polygonal shape in the plane, then $s^*$ is the shortest linear stretch outside $P$ along which a guard has to patrol back and forth in order to keep $P$ completely under surveillance. We present an optimal parallel algorithm for solving the WEV problem. The parallel computational model we use is the CREW PRAM.

Toussaint and Avis [11] first considered the weak external visibility of a polygon. Bhattacharya, Kirkpatrick, and Toussaint [3] showed that the WEV problem can be reduced to the problem of computing the shortest transversal of a set of planar geometric figures. This result [3] and the result on computing the shortest transversals of sets [2] together imply that the WEV problem is solvable sequentially in $O(n(\log n)^2)$ time. Bhattacharya and Toussaint [5] studied the special case of the WEV problem in which the polygon $P$ is convex, and gave an optimal linear time algorithm for it. Bhattacharya, Mukhopadhyay, and Toussaint [4] then extended the result of [5] to the general WEV problem (i.e., $P$ is a simple polygon), and presented an optimal linear time sequential algorithm for it. Interesting characterization of the WEV problem was given in [5, 4]. However, the algorithm in [4] seems to be inherently sequential because it walks, vertex by vertex, along the convex hull of $P$ and, at each convex hull vertex visited in the walk, dynamically maintains the "envelop" structure of certain rays associating with the vertices of $P$ (by adding or eliminating rays).

There is also related work on computing the *weak internal visibility* of simple polygons. (For a point $p$ inside polygon $P$ and an object $B$, $p$ is said to be *weakly internally visible* from $B$ if and only if there is some point $q$ on $B$ such that the segment $\overline{pq}$ does not intersect the exterior of $P$.) Many sequential algorithms and parallel algorithms for solving various problems on weak internal visibility of simple polygons have been designed.

Our parallel algorithm for the WEV problem runs in $O(\log n)$ time using $O(n/\log n)$ CREW PRAM processors. The total work (i.e., the *time* $\times$ *processor product*) of our algorithm matches the optimal linear bound of [4] for this problem. This parallel algorithm is obviously optimal. Our approach is very different from that of [4] since our parallel algorithm is based on a divide-and-conquer strategy and makes use of data structures that support fast parallel maintenance computation of the envelop structures.

Throughout, when we talk about weak external visibility, we will often omit the words "weakly externally" if this does not cause confusion from the context. For example, we will say "$X$ is visible (resp., non-visible) from line segment $s$" instead of "$X$ is weakly externally visible (resp., not weakly externally visible) from line segment $s$."

## 2 Preliminaries

The input to the WEV problem is an $n$-vertex simple polygon $P$; the output is the shortest weakly externally visible segment $s^*$ of $P$ (if $s^*$ exists), or an answer "no" (if $s^*$ does not exist). Polygon $P$ is specified by 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 $bd(P)$ of $P$, starting from vertex $v_1$. Without loss of generality (WLOG), we assume that no three vertices of $P$ are collinear.

The 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 $v_{n+1} = v_1$. A *chain* $C = (q_1, q_2, \ldots, q_k)$ consists of line segments $\overline{q_i q_{i+1}}$, for $i = 1, 2, \ldots, k-1$, and the *length* of $C$, denoted by $|C|$, is the number of line segments on it. The (directed) chain along $bd(P)$ from $v_i$ counterclockwise to $v_j$ is denoted by $bd_{i,j}$. All the chains considered in this paper are *simple*. For a directed chain $C = (q_1, q_2, \ldots, q_k)$, $k \geq 3$, $C$ is said to make only left (resp., right) turns iff every subchain of the form $(q_{i-1}, q_i, q_{i+1})$ makes a left (resp., right) turn, $1 < i < k$.

A polygon $Q$ is said to be *weakly externally visible* [3, 4, 11] if for every point $p$ on $bd(Q)$, there is a ray $r$ originating at $p$ such that $r$ does not intersect the interior of $Q$. For a vertex $v$ of a weakly externally visible polygon $Q$, the *external cone of support* (or simply the *external cone*) of $v$, denoted by $K(v)$, is the wedge whose *apex* is

*Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556, USA. E-mail: chen@cse.nd.edu.

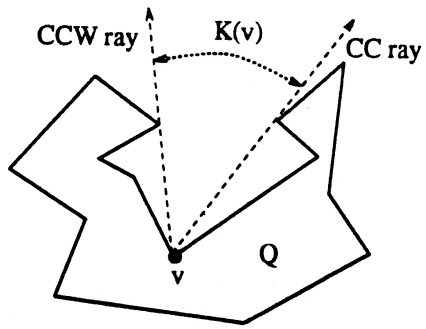


Figure 1: Illustrating the external cone $K(v)$ of vertex $v$.

at $v$ and whose *counterclockwise* (resp., *clockwise*) *bounding ray* is defined as follows: Let $r$ be a ray originating at $p$ such that $r$ does not intersect the interior of $Q$; sweeping $r$ counterclockwise (resp., clockwise) until it for the first time touches a vertex $w$ of $Q$ with $w \neq v$, then the counterclockwise (resp., clockwise) bounding ray of $K(v)$ is the ray $r$ when it touches $w$ (see Figure 1 for an example). We simply call the counterclockwise (resp., clockwise) bounding ray of $K(v)$ the CCW (resp., CC) bounding ray of vertex $v$. Bhattacharya, Kirkpatrick, and Toussaint [3] already showed that any object from which polygon $Q$ is weakly externally visible must intersect all the external cones of $Q$.

Note that determining whether an $n$-vertex polygon is weakly externally visible and computing the external cones of all the vertices of that polygon can be done optimally in parallel, in $O(\log n)$ time with $O(n/\log n)$ CREW PRAM processors, by using Chen's parallel algorithms for determining weak visibility of simple polygons [6, 8]. Hence we assume WLOG that the input polygon $P$ is weakly externally visible and that $K(v_i)$ is available for every vertex $v_i$.

Our algorithm makes use of concave wedges [4], which we review below: Suppose two chains $B_r = (a, q_1, q_2, \ldots, q_k)$ and $B_l = (a, z_1, z_2, \ldots, z_j)$ are given, such that $B_r$ (resp., $B_l$) makes only right (resp., left) turns, $q_k$ and $z_j$ are both at infinity, and $B_r \cap B_l = a$; we call the region from $B_r$ counterclockwise to $B_l$ a *concave wedge*, with *apex* $a$, the *left chain* $B_l$, and *right chain* $B_r$.

For a planar geometric object $B$, let $CH(B)$ denote the convex hull of $B$. Let the vertices of $CH(P)$ for polygon $P$, counterclockwise along $bd(CH(P))$, be $vc_1, vc_2, \ldots, vc_N$, where $N$ is the number of vertices of $CH(P)$, and $ec_i$ be the edge $\overline{vc_i vc_{i+1}}$ of $CH(P)$. Each edge $ec_i = \overline{vc_i vc_{i+1}}$ of $CH(P)$, with $vc_i = v_j$ and $vc_{i+1} = v_k$, is a *chord* of $P$, and the region enclosed by $ec_i$ and the chain $bd_{jk}$ is an *outer pocket* of $P$, denoted by $P_i$ (some of the pockets can consist of only a line segment).

For every vertex $vc_i$ of $CH(P)$, the *antipodal chain* of $vc_i$, denoted by $AC_i$, is defined as follows: Let the tangent of $CH(P)$ parallel to edge $ec_{i-1}$ (resp., $ec_i$) but not containing $ec_{i-1}$ (resp., $ec_i$) touch $CH(P)$ at vertex $s_i$ (resp., $t_i$); then the chain along $bd(CH(P))$ from $s_i$ counterclockwise to $t_i$ is the antipodal chain $AC_i$ of $vc_i$. Note that $AC_i \cap AC_{i+1} = t_i = s_{i+1}$. The vertices of $CH(P)$ on $AC_i$ are the *antipodal vertices* of $vc_i$, and a pair of vertices $vc_i$ and $vc_j$, where $vc_j$ belongs to $AC_i$, is an *antipodal pair* of vertices. A line segment touching an antipodal vertex of $vc_i$ and tangent to $CH(P)$ is said to be *antipodal* to $vc_i$. It is known [4] that the shortest line segment $s^*$ from which $P$ is visible must be antipodal to some vertex of $CH(P)$.

Given polygon $P$, we can compute its convex hull $CH(P)$ in $O(\log n)$ time using $O(n/\log n)$ processors [7]. The antipodal chains of all the vertices of $CH(P)$ can be computed in $O(\log n)$ time using $O(n/\log n)$ processors by parallel merge, based on the fact that the slopes of the edges on the upper (resp., lower) hull of $CH(P)$ are in sorted order along $bd(CH(P))$.

# 3 Geometric Observations and Data Structures

In this section, we discuss some useful geometric insights and data structures needed by our parallel algorithm. Many of the geometric observations in this section were made by Bhattacharya, Mukhopadhyay, and Toussaint [4].

The following lemmas are useful to both the sequential algorithm [4] and our parallel algorithm.

**Lemma 1** *Let $W$ be a concave wedge bounded by its right chain $B_r = (a, q_1, q_2, \ldots, q_k)$ and left chain $B_l = (a, z_1, z_2, \ldots, z_j)$, and $CP$ be a convex polygon contained in $W$, with its apex $a$ also being a vertex of $CP$. Let $r$ be a fixed ray that originates from $a$ and is contained in $W$. If $\overline{b_l b_r}$ is a line segment lying inside $W$ and tangent to $CP$, with its end vertex $b_l$ (resp., $b_r$) resting on $B_l$ (resp., $B_r$), then the length of $\overline{b_l b_r}$ is a unimodal function of $\theta$, where $\theta$ is the angle from $r$ counterclockwise to the line segment $\overline{b_l b}$, with the point $b$ being the intersection of $r$ and $\overline{b_l b_r}$.*

**Proof.** See Lemma 5 of [4]. □

**Lemma 2** *Let the shortest segment $s^*$ that we seek be antipodal to a vertex $vc_i$ of $CH(P)$ and $s^*$ touch $CH(P)$ at vertex $vc_j$. Then one end point of $s^*$ must be on the* CCW *bounding ray of some vertex on the chain along $bd(P)$ from $vc_i$* counterclockwise *to $vc_j$ and another end point must be on the* CC *bounding ray of some vertex on the chain along $bd(P)$ from $vc_i$* clockwise *to $vc_j$.*

**Proof.** See Lemma 6 of [4]. □

For two vertices $v$ and $w$ of $P$, $v \neq w$, we denote the chain along $bd(P)$ from $v$ counterclockwise (resp., clockwise) to $w$ by $C_{ccw}(v, w)$ (resp., $C_{cc}(v, w)$), and the set of the CCW (resp., CC) bounding rays of all the vertices of $P$ on $C_{ccw}(v, w)$ (resp., $C_{cc}(v, w)$) by $R_{ccw}(v, w)$ (resp., $R_{cc}(v, w)$). Based on Lemma 2, Bhattacharya, Mukhopadhyay, and Toussaint [4] used in their algorithm a geometric structure called *envelop*, which we define below:

**Definition 1** *For two vertices $v$ and $w$ of $P$, let $A$ be the arrangement formed by the rays in $R_{ccw}(v, w)$ (resp., $R_{cc}(v, w)$) from the chain $C_{ccw}(v, w)$ (resp., $C_{cc}(v, w)$). Let $c$ be a cell of the arrangement $A$ characterized as follows: The vertex $v$ is on the boundary of $c$; starting a walk from $v$ along the CCW (resp., CC) bounding ray of $v$, then the walk is on the boundary of $c$ and the interior of $c$ is to the right (resp., left); the walk stays on the boundary of $c$ towards infinity, by making a right (resp., left) turn when a vertex of $A$ is met. The* envelop *of $R_{ccw}(v, w)$ (resp., $R_{cc}(v, w)$), denoted by $Env_{ccw}(v, w)$ (resp., $Env_{cc}(v, w)$), is the chain along which the walk travels.*

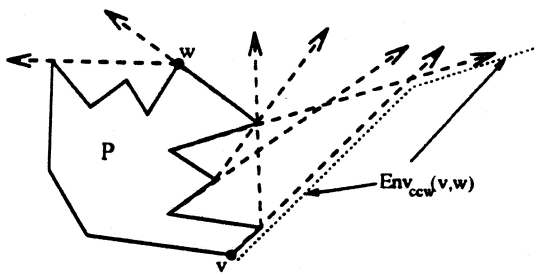


Figure 2: The CCW envelop $Env_{ccw}(v, w)$ for the chain $C_{ccw}(v, w)$.

We call $Env_{ccw}(v, w)$ (resp., $Env_{cc}(v, w)$) a CCW (resp., CC) envelop. Note that $Env_{ccw}(v, w)$ (resp., $Env_{cc}(v, w)$) is either a single ray or a chain that makes only right (resp., left) turns. An example of CCW envelops is given in Figure 2.

It is shown in [4] that, in order to find segment $s^*$, one can consider every antipodal pair of vertices $v$ and $w$ of $CH(P)$ and compute the shortest line segment $s^*_{vw}$ that (1) is antipodal to $v$, (2) is tangent to $CH(P)$ and touches $w$, and (3) has its two endpoints resting respectively on $Env_{ccw}(v, w)$ and $Env_{cc}(v, w)$. From such a segment $s^*_{vw}$, $P$ is certainly visible, and $s^*$ is the shortest among such segments $s^*_{vw}$. The following lemma is a generalization of the statement above.

**Lemma 3** *For a vertex $vc_i$ of $CH(P)$, let its antipodal chain $AC_i$ begin at vertex $s_i$ and end at vertex $t_i$ of $CH(P)$. Let $u$ (resp., $w$) be a vertex of $P$ on the chain $C_{ccw}(vc_i, s_i)$ (resp., $C_{cc}(vc_i, t_i)$). Then the shortest line segment that is antipodal to $vc_i$ (by touching a vertex of $CH(P)$ on $AC_i$), that is tangent to $CH(P)$, and that has its two end points resting respectively on $Env_{ccw}(vc_i, w)$ and $Env_{cc}(vc_i, u)$, is the shortest segment antipodal to $vc_i$ from which $P$ is visible.*

**Proof.** See Lemma 10 of [4]. □

The sequential algorithm in [4] computes and maintains the envelop structure in the following way: (i) Compute, for every outer pocket of $P$ (say from vertex $v_j$ counterclockwise to vertex $v_k$), the envelops $Env_{ccw}(v_j, v_k)$ and $Env_{cc}(v_k, v_j)$, by scanning along the chain $bd_{jk}$; (ii) scan along the boundary of $CH(P)$, maintaining the envelops so that when each antipodal pair of vertices $v$ and $w$ is encountered during the scanning, $Env_{ccw}(v, w)$ and $Env_{cc}(v, w)$ are available for computing the segment $s^*_{vw}$. This approach seems to be very sequential and cannot be used by our parallel algorithm. Therefore, we must use a different method for computing and maintaining the envelops, and for obtaining the segment $s^*_{vw}$ when $Env_{ccw}(v, w)$ and $Env_{cc}(v, w)$ are given. We need the following lemma from [4].

**Lemma 4** *Let $C'$ and $C''$ be two chains on $bd(P)$ whose interior is disjoint from each other. Then the envelop for the CCW (resp., CC) bounding rays of the vertices on $C'$ can intersect the envelop for the CCW (resp., CC) bounding rays of the vertices on $C''$ at most one time.*

**Proof.** See Lemma 8 of [4]. □

The following properties are crucial to our parallel algorithm.

**Lemma 5** *For two vertices $v$ and $w$ of $P$, let $(r_{i_1}, r_{i_2}, \ldots, r_{i_k})$ be the sequence of rays whose portions are visited, in this order, by a walk along $Env_{ccw}(v, w)$ (resp., $Env_{cc}(w, v)$) by starting at $v$ (resp., $w$), where $r_{i_j}$ is the CCW (resp., CC) bounding ray of vertex $v_{i_j}$ of $P$. Then a walk along the chain $C_{ccw}(v, w)$ (resp., $C_{cc}(w, v)$) of $bd(P)$ by starting at $v$ (resp., $w$) visits the vertices in $\{v_{i_j} \mid j = 1, 2, \ldots, k\}$ in the same order as the sequence $(r_{i_1}, r_{i_2}, \ldots, r_{i_k})$, i.e., in the order of $(v_{i_1}, v_{i_2}, \ldots, v_{i_k})$.*

**Proof.** We only prove the case of $Env_{ccw}(v, w)$ (the other case is proved similarly). Note that by the definition of envelops, the first edge of $Env_{ccw}(v, w)$ is on the CCW bounding ray of vertex $v$; hence $r_{i_1}$ is the CCW bounding ray of $v$. The walk along $C_{ccw}(v, w)$ certainly visits $v$ first. Assume that the lemma does not hold. Then there must be two rays $r_{i_a}$ and $r_{i_b}$, with $1 < a < b \leq k$, such that the walk along $C_{ccw}(v, w)$ visits $v_{i_b}$ before $v_{i_a}$, i.e., the chain $C_{ccw}(v, v_{i_b})$ does not contain the vertex $v_{i_a}$. Consider the envelop $Env_{ccw}(v, v_{i_b})$ and the envelop consisting of only the CCW bounding ray of $v_{i_a}$. The above assumption implies that the CCW bounding ray of $v_{i_a}$ intersects $Env_{ccw}(v, v_{i_b})$ at least twice because in the walk along $Env_{ccw}(v, w)$, $r_{i_a}$ is visited *after* $r_{i_1}$ and *before* $r_{i_b}$. But this contradicts with Lemma 4. Hence the lemma holds. □

**Corollary 1** *Let $C$ be a chain on $bd(P)$ and $C'$ a subchain of $C$. Let $Env$ and $Env'$ be the envelops formed by the CCW (resp., CC) bounding rays of the vertices on $C$ and $C'$, respectively. Then $Env \cap Env'$ is either empty or a connected component.*

**Proof.** An immediate consequence of Lemma 5. □

Our parallel algorithm makes use of several data structures: (1) The *rank trees*, for computing and storing the envelop structures (see [7, 8, 10] for a detailed description of rank trees and the parallel operations that they support), and (2) the tree of arrays [6, 8], for handling the maintenance of envelops when the parallel algorithm modifies the envelop for a vertex on $CH(P)$ and then uses the updated envelop for other vertices. More discussion on these data structures will be given in the next section while we present the algorithm.

# 4 The Parallel Algorithm

The idea of our parallel algorithm is as follows: First compute several *global* envelop structures (**Phase One**); then use these global envelops to obtain (the relevant portions of) the envelops needed by every vertex on $CH(P)$, in order to apply Lemma 3 (**Phase Two**); finally, compute the shortest segment antipodal to every vertex on $CH(P)$ from which $P$ is visible, based on Lemmas 1 and 2 (**Phase Three**). Hence our algorithm for the WEV problem consists of three phases, which we present one by one in this section. Among the three, **Phases One** and **Two** are more difficult.

## 4.1 Phase One

Our computation of envelops makes use of rank trees, with the edges of the envelops being stored at the leaves of the trees, one edge per leaf. It is well-known that a rank tree storing certain geometric structure (e.g., envelop, visibility

chain) of a chain $C$ is of height $O(\log |C|)$. The following lemma summarizes the key subroutine for this phase. WLOG, we only discuss the envelops which are formed by CCW bounding rays of the vertices of $P$ (the CC envelops are handled in a similar way).

**Lemma 6** *Given two interior-disjoint chains $C'$ and $C''$ on $bd(P)$ of length $O(k)$ each, let $Env(C')$ and $Env(C'')$ respectively be the CCW envelops of $C'$ and $C''$, each stored in a rank tree. Then by using $k^c$ processors, where $c$ is a constant with $0 < c < 1$, it is possible to decide whether $Env(C')$ and $Env(C'')$ intersect each other, and find the intersection if they do, in $O(\log k)$ time.*

**Proof.** Let $e'$ (resp., $e''$) be the edge on $Env(C')$ (resp., $Env(C'')$) adjacent to the point at infinity. Let $r(s)$ be the CCW bounding ray from which an edge $s$ of a CCW envelop belongs. It is easy to see that if the two envelops intersect each other, then either $r(e')$ must intersect $Env(C'')$ once (by Lemma 4) or $r(e'')$ must intersect $Env(C')$ once. Since the envelops make only right turns, it is trivial to find, in $O(\log k)$ time with $k^c$ processors, whether a ray intersects an envelop of size $O(k)$ stored in a rank tree.

Now suppose $Env(C')$ and $Env(C'')$ do intersect each other once, and we want to compute their actual intersection point. We repeat the following procedure until the intersection is found:

> Choose $k^{c/2}$ edges on $Env(C')$ such that these edges together partition $Env(C')$ into $O(k^{c/2})$ subchains of equal length. For every chosen edge $e$ on $Env(C')$, compute the intersection of the ray $r(e)$ with $Env(C'')$ in $O(\log k)$ time using $k^{c/2}$ processors. The type of the intersection between $r(e)$ and $Env(C'')$ (i.e., $r(e)$ intersects $Env(C'')$ *before*, *on*, or *after* the portion $e$ of $r(e)$) indicates whether the intersection between $Env(C')$ and $Env(C'')$ occurs *before*, *on*, or *after* the edge $e$ on $Env(C')$. Hence we can decide on which one of the $O(k^{c/2})$ subchains of $Env(C')$ the intersection between $Env(C')$ and $Env(C'')$ lies.

By repeating such a procedure $O(c) = O(1)$ times, the intersection between $Env(C')$ and $Env(C'')$ is located. Therefore, the intersection is computable in $O(\log k)$ time using $k^c$ processors. □

**Lemma 7** *Given two interior-disjoint chains $C'$ and $C''$ on $bd(P)$ of length $O(k)$ each, let $Env(C')$ and $Env(C'')$ respectively be the CCW envelops of $C'$ and $C''$, each stored in a rank tree. Then by using one processor, it is possible to decide whether $Env(C')$ and $Env(C'')$ intersect each other, and find the intersection if they do, in $O((\log k)^3)$ time.*

**Proof.** Use the same idea as Lemma 6. Then it is easy to see that we need to repeat the procedure $O(\log k)$ times by using one processor, with each iteration taking $O((\log k)^2)$ time. □

Even though a more efficient procedure may be possible for implementing Lemma 7, it is sufficient for our algorithm to use Lemma 7.

**Phase One** is based on a divide-and-conquer algorithmic structure similar to those of [1, 7, 8], which we outline below:

**Input.** A chain $C$ on $bd(P)$ with $|C| = m$, and a positive integer $d$.

**Output.** The CCW envelop $Env_{ccw}(C)$ of $C$.

**Case 1.1.** If $m \leq d$, then compute $Env_{ccw}(C)$ (stored in a rank tree) in $O(m)$ time with one processor.

**Case 1.2.** If $d < m \leq d^4$, then partition $C$ into two subchains $C_1$ and $C_2$ of equal size, and recursively compute $Env_{ccw}(C_1)$ and $Env_{ccw}(C_2)$, in parallel. Then compute $Env_{ccw}(C)$ from $Env_{ccw}(C_1)$ and $Env_{ccw}(C_2)$, with $m/d$ processors and in $O((\log d)^3)$ time.

**Case 1.3.** If $m > d^4$, 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, compute $Env_{ccw}(C)$ from $Env_{ccw}(C_1)$, $Env_{ccw}(C_2), \ldots, Env_{ccw}(C_g)$, with $m/d = g^3$ processors and in $O(\log m)$ time. (Some additional computation will also be performed within the same complexity bounds, which will be discussed later in this subsection.)

It has been shown [1, 7, 8] that, if we process the various cases of the outline given above within the claimed complexity bounds, then a procedure with such an outline will run in $O(d+\log m)$ time with $O(1+m/d)$ processors. Therefore, a call to the above procedure with input $(C, \log n)$, where $|C| = O(n)$, takes $O(\log n)$ time using $O(n/\log n)$ processors.

**Phase One** consists of two steps: (1) Compute the CCW envelop for every outer pocket of $P$, and (2) compute $O(1)$ global envelops from the envelops of the outer pockets of $P$. By using the sequential algorithm [4] in Case 1.1, Lemma 7 in Case 1.2, and Lemma 6 in Case 1.3, $Env_{ccw}(C)$ for the chain $C = bd(P) \cap P_i$, where $P_i$ is an outer pocket of $P$, can certainly be obtained by the above procedure within the claimed parallel bounds. What we need to discuss are (a) the "additional computation" in the outlined procedure, and (b) the computation of $O(1)$ global envelops.

Since Case 1.2 is a simple case of Case 1.3, it is sufficient for us to discuss Case 1.3. Assume that $Env_{ccw}(C_1)$, $Env_{ccw}(C_2), \ldots, Env_{ccw}(C_g)$ are already available, each stored in a rank tree. We construct a complete binary tree $T_{ccw}(C)$ whose leaves from left to right are associated with chains $C_1, C_2, \ldots, C_g$, respectively. Every node $u$ of $T_{ccw}(C)$ represents the CCW envelop $Env_{ccw}(B(u))$, where $B(u)$ is the union of the chains $C_i$ associated with the descendent leaves of $u$. For example, the leaves of $T_{ccw}(C)$ represent $Env_{ccw}(C_1)$, $Env_{ccw}(C_2), \ldots, Env_{ccw}(C_g)$, and the root $root(T_{ccw}(C))$ of $T_{ccw}(C)$ represents $Env_{ccw}(C)$. Note that $root(T_{ccw}(C_i))$ of the tree $T_{ccw}(C_i)$ returned by the recursive call on the chain $C_i$ becomes a leaf of $T_{ccw}(C)$.

Let the parent of a node $u$ in $T_{ccw}(C)$ be $parent(u)$. By Corollary 1, both $Env_{ccw}(B(u)) \cap Env_{ccw}(B(parent(u)))$ and $Env_{ccw}(B(u)) - (Env_{ccw}(B(u)) \cap Env_{ccw}(B(parent(u))))$ consist of a connected component. Our procedure not only computes $Env_{ccw}(C)$, but also computes $Env_{ccw}(B(u)) - (Env_{ccw}(B(u)) \cap Env_{ccw}(B(parent(u))))$, stored in a rank tree, for every non-root node $u$ of $T_{ccw}(C)$. The next lemma helps the computation for $Env_{ccw}(B(u)) - (Env_{ccw}(B(u)) \cap Env_{ccw}(B(parent(u))))$.

**Lemma 8** *Let $u$, $w$, and $z$ be three nodes of $T_{ccw}(C)$ such that $w$ is an ancestor of $u$ and $z$ an ancestor of $w$. Then $Env_{ccw}(B(u)) \cap Env_{ccw}(B(w))$ is a connected component; furthermore, $Env_{ccw}(B(u)) \cap Env_{ccw}(B(z)) \subseteq Env_{ccw}(B(u)) \cap Env_{ccw}(B(w))$.*

**Proof.** It follows immediately from Corollary 1. □

We can view the chains $Env_{ccw}(B(u)) \cap Env_{ccw}(B(z))$ and $Env_{ccw}(B(u)) \cap Env_{ccw}(B(w))$ in Lemma 8 as intervals on the chain $Env_{ccw}(B(u))$. Then $Env_{ccw}(B(u)) - (Env_{ccw}(B(u)) \cap Env_{ccw}(B(parent(u))))$, for every non-root node $u$ of $T_{ccw}(C)$, can be computed as follows.

For every $Env_{ccw}(C_i)$ and every node $u$ such that $u$ is a proper ancestor of the leaf associated with $C_i$ in $T_{ccw}(C)$, compute $Env_{ccw}(C_i) \cap Env_{ccw}(B(u))$. Even by using a brute force method, each $Env_{ccw}(C_i)$ has to be involved with $O(\log m)$ ancestors in $T_{ccw}(C)$, which respectively have $g$, $g/2$, $g/4$, ..., 1 descendent leaves. Hence each $Env_{ccw}(C_i)$ has to be involved in the computation with at most $O(g)$ other $Env_{ccw}(C_j)$'s, with $g^2$ processors at its disposal. Thus Lemma 6 can be applied to compute the intersection between every pair of $Env_{ccw}(C_i)$ and $Env_{ccw}(C_j)$. Having found the intersection between $Env_{ccw}(C_i)$ and $Env_{ccw}(C_j)$, where $C_j$ is associated with another descendent leaf of the node $u$, it is trivial to obtain $Env_{ccw}(C_i) \cap Env_{ccw}(B(u))$ from these intersection points on $Env_{ccw}(C_i)$. Given the interval for $Env_{ccw}(C_i) \cap Env_{ccw}(B(u))$, for every proper ancestor $u$ of the leaf storing $Env_{ccw}(C_i)$, we have enough processors to sort these $O(\log m)$ intervals in $O(\log m)$ time. Also in $O(\log m)$ time, we can decide which connected portion of $Env_{ccw}(C_i)$ belongs to $Env_{ccw}(B(u)) - (Env_{ccw}(B(u)) \cap Env_{ccw}(B(parent(u))))$ for every proper ancestor $u$ of the leaf for $C_i$ in $T_{ccw}(C)$, based on Lemma 8. The rank trees storing these connected portions of $Env_{ccw}(C_i)$ can be obtained by applying the parallel split operation [7] on the rank tree storing $Env_{ccw}(C_i)$; the rank tree for $Env_{ccw}(B(u)) - (Env_{ccw}(B(u)) \cap Env_{ccw}(B(parent(u))))$ can be obtained by using the parallel concatenation operation [7] on the pieces that form $Env_{ccw}(B(u)) - (Env_{ccw}(B(u)) \cap Env_{ccw}(B(parent(u))))$. The parallel split and concatenation operations both take $O(\log m)$ time. Hence the information stored in $T_{ccw}(C)$ can be computed in $O(\log m)$ time with $g^3$ processors.

Next, we show how to compute at most six *global envelops* for $bd(P)$. In order to do that, we partition $bd(P)$ into at most three chains, such that for each such chain $C'$, the end vertices of $C'$ are on $CH(P)$ and no vertex $vc_j$ on $C' \cap CH(P)$ has its antipodal chain $AC_j$ that intersects the interior of $C'$ (i.e., $C' \cap AC_j$ consists of at most the end vertices of $C'$). The observations below follow immediately from the definition of antipodal vertices.

- For a vertex $vc_i$ of $CH(P)$, let $vc_k = s_i$ and $C'$ be the chain along $bd(P)$ from $vc_i$ counterclockwise to $vc_k$. Then there is no vertex $vc_j$ on $CH(P) \cap C'$ such that $AC_j$ intersects the interior of $C'$.
- It is possible to partition $bd(P)$ into at most three such chains $C'$ such that the end vertices of $C'$ are both on $CH(P)$.

Given the antipodal chains of the vertices on $CH(P)$, the partition of $bd(P)$ can be easily done in $O(1)$ time with one processor, as follows: Suppose $vc_k = s_1$; then simply partition $bd(P)$ using vertices $vc_1$, $vc_k$, and $s_k$, resulting in at most three chains on $bd(P)$. Let $C'$ be such a chain from the partition of $bd(P)$; then $Env_{ccw}(C')$ and $Env_{ccw}(C)$ are a pair of the global envelops that we would like to compute, where $C = bd(P) - C'$. We complete the exposition of **Phase One** by discussing briefly how to compute the envelop $Env_{ccw}(C)$ and the tree $T_{ccw}(C)$ when the CCW envelops of all the outer pockets of $P$ whose end vertices are both on $C$ are available (the computation for $Env_{ccw}(C')$ and $T_{ccw}(C')$ is similar).

The computation of $Env_{ccw}(C)$ and $T_{ccw}(C)$ is very similar to the computation of the CCW envelop of an outer pocket of $P$, with only one main difference: The sizes of the outer pockets on $C$ can vary. WLOG, let the outer pockets along $C$ be $P_1$, $P_2$, ..., $P_k$. Several minor changes of the procedure for computing the envelop of an outer pocket are needed:

(i) The pockets on $C$ become the leaves of the recursion (their CCW envelops are already given).

(ii) In the divide stage, compute $sum_i = \sum_{j=1}^{i} |P_j|$ (by parallel prefix), and divide the chain $C$ at $P_{i-1} \cap P_i$ if $sum_{i-1} \leq j * m^{2/3} d^{1/3} < sum_i$ for some integer $j = 1, 2, \ldots, g$.

(iii) "Schedule" for every pocket $P_j$ of $C$ so that its CCW envelop can participate in the computation of $Env_{ccw}(C)$ at the appropriate level of the recursion. Let $m = |C|$ at level 0 of the recursion. Then the following can be easily proved by induction: At the $i$-th level, $i \geq 1$, the chain size is $O(m^{f(i)} d^{1-f(i)})$, where $f(i) = (2/3)^i$. Therefore, all we need to do is to let the CCW envelop of $P_j$ participate in the computation of $Env_{ccw}(C)$ at the $i$-th recursion level such that $m^{f(i)} d^{1-f(i)} \leq |P_j| < m^{f(i-1)} d^{1-f(i-1)}$.

(iv) The tree $T_{ccw}(C)$ is built in the same way. I.e., build a complete binary tree whose leaves are either the envelops returned from recursive calls or envelops of pockets of $C$, regardless of the sizes of the chains from which these envelops were obtained. $T_{ccw}(C)$ so constructed is obviously of height $O(\log m) = O(\log n)$.

This concludes the discussion of **Phase One**.

## 4.2 Phase Two

Let $C'$ be a chain obtained in **Phase One** such that there is no vertex $vc_j$ of $CH(P) \cap C'$ whose $AC_j$ intersects the interior of $C'$, and let $C = bd(P) - C'$. This phase accomplishes the following task: Given the envelops $Env_{ccw}(C)$, $Env_{ccw}(C')$, $Env_{cc}(C)$, and $Env_{cc}(C')$, and the trees $T_{ccw}(C')$ and $T_{cc}(C')$ (the trees $T_{ccw}(C)$ and $T_{cc}(C)$ are actually not needed), obtain the relevant portions of the CCW and CC envelops needed by every vertex of $CH(P) \cap C'$, so that Lemma 3 can be applied. The procedure for this task is based on the tree-of-arrays data structure used in [6, 8].

Below we first discuss the main idea and the tree-of-arrays structure for our solution to this phase; then we show how this data structure is used in a divide-and-conquer approach, to compute the relevant portions of the CCW and CC envelops needed by all the vertices of $CH(P) \cap C'$. As in the previous subsection, we focus on the case with the CCW envelops (the CC envelops are computed in a similar way).

### 4.2.1 Main Idea and Tree-of-Arrays Structure

In **Phase One**, we build the binary tree $T_{ccw}(C')$ which is of height $O(\log n)$ and has $O(n/d) = O(n/\log n)$ nodes. The root of $T_{ccw}(C')$ stores $Env_{ccw}(C')$. Each non-root node $u$ of $T_{ccw}(C')$ stores at most one connected envelop $Env_{ccw}(B(u)) - (Env_{ccw}(B(u)) \cap Env_{ccw}(B(parent(u))))$ with a rank tree, where $B(u)$ is the union of the subchains of $C'$ associated with the descendent leaves of $u$. Hence there are altogether $O(n/\log n)$ envelops stored at the nodes of $T_{ccw}(C')$, and the total sum of the sizes of the envelops stored at all the nodes of $T_{ccw}(C')$ is $O(n)$.

Let the chain $C = bd(P) - C'$ be from vertex $z$ counterclockwise to vertex $z'$, i.e., $C \cap C' = \{z, z'\}$. The idea for **Phase Two** is based on the following observations.

(I) Let $vc_i$ be a vertex of $CH(P) \cap C'$ and $vc_k$ be a vertex of $CH(P) \cap C$. Then by Lemma 3, the relevant

portion of $Env_{ccw}(vc_i, vc_k)$ for computing the shortest segment antipodal to $vc_i$ from which $P$ is visible can be obtained from $Env_{ccw}(vc_i, z')$. Furthermore, the envelop $Env_{ccw}(vc_i, z')$ can be partitioned into two parts: One is a subchain of $Env_{ccw}(z, z') = Env_{ccw}(C)$ (which is already available); another is a subchain of $Env_{ccw}(vc_i, z)$ (which may not be readily available but its information is stored in $T_{ccw}(C')$).

(II) Let $vc_i$ and $vc_j$ be two vertices of $CH(P) \cap C'$, $i \neq j$. Then the relevant portion of the CCW envelop needed by $vc_i$ is interior-disjoint from that needed by $vc_j$. This implies that the total sum of the sizes of the relevant portions of the CCW envelops needed by all the vertices of $CH(P) \cap C'$ is $O(n)$. We only show that the portions on $Env_{ccw}(C)$ needed by $vc_i$ and $vc_j$ are interior-disjoint (the case with $Env_{ccw}(C')$ is proved in a similar way). Recall that the antipodal chain $AC_i$ of $vc_i$ is defined by vertices $s_i$ and $t_i$, where $s_i$ (resp., $t_i$) is the vertex at which the tangent $t(ec_{i-1})$ (resp., $t(ec_i)$) parallel to the edge $ec_{i-1}$ (resp., $ec_i$) but not containing $ec_{i-1}$ (resp., $ec_i$) touches $CH(P)$. Suppose the edges of $CH(P)$ are all directed counterclockwise. Let $r(t(ec_{i-1}))$ (resp., $r(t(ec_i))$) be the ray originating from $s_i$ (resp., $t_i$), parallel to $ec_{i-1}$ (resp., $ec_i$), and having the same direction as $ec_{i-1}$ (resp., $ec_i$). Then the portion on $Env_{ccw}(C)$ needed by $vc_i$ is contained by the subchain of $Env_{ccw}(C)$ from the point hit by $r(t(ec_{i-1}))$ to the point hit by $r(t(ec_i))$. The proof follows from the facts that the polar angles of the rays $r(t(ec_{i-1}))$ and $r(t(ec_i))$ for all the vertices $vc_i$ of $CH(P) \cap C'$ are in sorted order along $bd(CH(P))$ and that $Env_{ccw}(C)$ makes only right turns.

Based on these observations, the idea for **Phase Two** is as follows: Compute the relevant portions on $Env_{ccw}(vc_i, z')$ for the vertices $vc_i$ of $CH(P) \cap C'$ that are respectively the subchains of $Env_{ccw}(C)$ and $Env_{ccw}(vc_i, z)$; this computation is done by tracing the tree $T_{ccw}(C')$ in a top-down fashion, level by level, and constructing (the relevant portions of) $Env_{ccw}(vc_i, z)$ for the vertices $vc_i$ associated with each level of $T_{ccw}(C')$.

Observation (I) above implies that the way that our procedure uses the envelop $Env_{ccw}(C)$ is static. Hence we simply convert $Env_{ccw}(C)$ from a rank tree representation into an array representation. Let the array storing $Env_{ccw}(C)$ be $A(C)$. Obtaining $A(C)$ from the rank tree storing $Env_{ccw}(C)$ can be easily done in $O(\log n)$ time using $O(n/\log n)$ processors [8, 6, 10].

Also by Observation (I), we need to compute $Env_{ccw}(vc_i, z)$ by using the information stored in $T_{ccw}(C')$. In order to compute $Env_{ccw}(vc_i, z)$ efficiently, we would like to convert the rank trees stored at the nodes of $T_{ccw}(C')$ into arrays, i.e., each envelop stored in a rank tree of $T_{ccw}(C')$ is converted into an array. This conversion operation can also be done in $O(\log n)$ time using $O(n/\log n)$ processors [8, 6]. For every pocket $P_i$ on $C'$, let chain $C_i = P_i \cap P$. We let the leaves of $T_{ccw}(C')$ be the nodes that are associated with the chains $C_i$, and store each envelop $Env_{ccw}(C_i)$ in an array. We still use $T_{ccw}(C')$ to denote the tree whose envelops are stored in arrays. $T_{ccw}(C')$ is the *tree of arrays* that we need.

### 4.2.2 Algorithm for Phase Two

The algorithm for this phase traces the tree $T_{ccw}(C')$ in a top-down fashion, level by level, and constructs the relevant portions of $Env_{ccw}(z, z')$ and $Env_{ccw}(vc_i, z)$ for the vertices $vc_i$ of $CH(P) \cap C'$ that are associated with each level of $T_{ccw}(C')$. Since the height of $T_{ccw}(C')$ is $O(\log n)$, we must process the majority of the levels of $T_{ccw}(C')$ in $O(1)$ time per level. The tree-of-arrays data structure enables us to achieve that. This parallel technique has been used in [8, 6] to compute the cones of support of a weakly visible polygon. The details of this procedure are left to the full paper.

## 4.3 Phase Three

Given the relevant portions of the CCW and CC envelops needed by every vertex $v$ of $CH(P)$, this phase applies Lemmas 1 and 2 to compute the shortest segment antipodal to $v$ from which $P$ is visible. The relevant portions of the envelops for $v$ are represented by a set of arrays (obtained from **Phase Two**). The algorithm for this phase is based on an idea for parallel search in arrays and it is relatively simple comparing to the previous two phases. The details of this procedure are also left to the full paper.

# References


[1] M. J. Atallah, D. Z. Chen, and H. Wagener. "An optimal parallel algorithm for the visibility of a simple polygon from a point," *Journal of the ACM*, 38 (3) (1991), pp. 516–533.

[2] B. K. Bhattacharya, J. Czysowicz, P. Egyed, G. T. Toussaint, I. Stojmenovic, and J. Urrutia. "Computing shortest transversals of sets," *Proc. 7th Annual ACM Symp. Computational Geometry*, 1991, pp. 71–80.

[3] B. K. Bhattacharya, D. G. Kirkpatrick, and G. T. Toussaint. "Determining sector visibility of a polygon," *Proc. 5th Annual ACM Symp. Computational Geometry*, 1989, pp. 247–254.

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

[5] B. K. Bhattacharya and G. T. Toussaint. "Computing shortest transversals," Tech. Report SOCS 90.6, McGill University, April 1990.

[6] D. Z. Chen. "Parallel techniques for paths, visibility, and related problems," Ph.D. thesis, Technical Report No. 92-051, Dept. of Computer Sciences, Purdue University, July 1992.

[7] D. Z. Chen. "Efficient geometric algorithms on the EREW-PRAM," *Proc. 28th Annual Allerton Conf. on Communication, Control, and Computing*, Monticello, Illinois, 1990, pp. 818–827. Accepted to *IEEE Trans. on Parallel and Distributed Systems.*

[8] D. Z. Chen. "An optimal parallel algorithm for detecting weak visibility of a simple polygon," *Proc. 8th Annual ACM Symp. on Computational Geometry*, 1992, pp. 63–72. Accepted to *International Journal of Computational Geometry and Applications.*

[9] D. Z. Chen. "Optimally computing the shortest weakly visible subedge of a simple polygon," *Lecture Notes in Computer Science, No. 762: Proc. of the Fourth Annual International Symp. on Algorithms and Computation*, Hong Kong, December 1993, pp. 323–332.

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

[11] G. T. Toussaint and D. Avis. "On a convex hull algorithm for polygons and its applications to triangulation problems," *Pattern Recognition*, 15 (1) (1982), pp. 23–29.