Linear-time algorithms for weakly-monotone polygons

Paul J. Heffernan*

Department of Mathematical Sciences, Memphis State University, Memphis, TN 38152, USA

Communicated by Godfried Toussaint
Submitted 31 January 1991
Accepted 15 March 1993

Abstract
We introduce the class of weakly-monotone polygons, and give an optimal triangulation algorithm for the class. We also present a simple linear-time detection algorithm, which for input polygon $P$ returns the set of directions in which $P$ is weakly-monotone.

1. Introduction

Much work in computational geometry has focused on special classes of simple polygons. In this paper, we introduce to the hierarchy the class of weakly-monotone polygons, which contains the monotone class. For many classes of polygons, such as monotone and star-shaped, there exist linear-time algorithms for determining if a polygon belongs to the class ([12, 13]). These detections algorithms are of interest for the insight they provide into the structure of polygons. In this paper we present a linear-time detection algorithm for weakly-monotone polygons, which for input polygon $P$ returns the set of directions in which $P$ is weakly-monotone.

A detection algorithm for a special class of polygon takes on added importance with efficient algorithms that operate on the class. For example, there exist simple linear-time algorithms for triangulating a monotone [7] or a star-shaped [5] polygon. In this paper, we present a simple linear-time triangulation algorithm for weakly-monotone polygons, which together with the detection algorithm allows us to triangulate a weakly-monotone polygon in linear time, without prior knowledge of the polygon's weak-monotonicity.

Correspondence to: Paul J. Heffernan, Memphis State University, Department of Mathematical Sciences, Tennessee, Memphis, TN 38152, USA.

* Part of this work was done while the author was an NSF graduate fellow at Cornell University. Also partially supported by the U.S. Army Research Office, Grant No. DAAL03-92-G-0378.

0925-7721/93/$06.00 © 1993 — Elsevier Science Publishers B.V. All rights reserved
Of course, a weakly-monotone polygon, or a star-shaped or monotone polygon, can be triangulated directly by any of the many general polygon triangulation algorithms. However, each of the general methods has a shortcoming. The only optimal algorithm [2] is conceptually difficult, and too complex to be considered practical. Many of the general algorithms are simpler ([7, 11, 14]), but each is super-linear in the worst case. The algorithms of [3] and [10] have time bounds that depend on some parameter of the polygon, but each runs in time $\Theta(n \log n)$ on some weakly-monotone polygons. In this paper we show how to triangulate a weakly-monotone polygon $P$, without prior knowledge of $P$’s weak-monotonicity, with algorithms that are optimal, practical, and conceptually simple.

Section 2 of this paper defines a weakly-monotone polygon, and discusses its place in the hierarchy of simple polygons. Section 3 presents the linear-time detection algorithm for the class. Section 4 describes the linear-time triangulation algorithm for the class. Section 5 is the conclusion.

2. Weakly-monotone polygons

A polygonal chain is a concatenation of line segments in the plane; the segments are called edges and their endpoints vertices. If the chain is closed it is called a polygon. We say that a chain is simple if no two edges intersect except for adjacent edges at their common endpoint. In this paper we shall concern ourselves only with simple polygons, so we will usually drop the modifier. Since a polygon $P$ is a simple closed curve, it partitions $\mathbb{R}^2 \setminus P$ into two regions, one bounded and one unbounded, which we call the interior and the exterior, and denote int($P$) and ext($P$). The segment with endpoints $x$ and $y$ is denoted $\overline{xy}$.

Fig. 1. (a) weakly-monotone but not crab-shaped, (b) crab-shaped but not weakly-monotone.
Suppose we have a polygon $P$ with vertices $s$ and $t$, and a direction $\theta$. Imagine two cars, one which drives clockwise along $P$ from $s$ to $t$, and the other which drives counterclockwise on $P$ from $s$ to $t$. If neither car faces direction $\theta + \pi$ during its drive, we say that $P$ is weakly-monotone in direction $\theta$ for splitting points $s$ and $t$ (see Fig. 1(a)). This definition is not mathematically precise, but it is intuitively helpful; we will give a formal definition in the next section.

If a polygon $P$ is weakly-monotone, we can determine a triplet $s, t$ and $\theta$ for $P$ and triangulate $P$ in linear time, using the algorithms in this paper. There exist specific linear-time triangulation algorithms for many other classes of polygons. A hierarchy of polygons is presented in [5], where any polygon in the hierarchy can be triangulated by some specific algorithm simpler than that of [2]. All classes in the hierarchy are contained in the class of crab-shaped polygons, for which there exists no specific triangulation algorithm. A polygon $P$ is crab-shaped if there is a point $x \notin P$ such that the shortest path from $x$ to $y$ not crossing $P$ is convex, for all $y \in P$. Fig. 1 shows that neither the crab-shaped nor the weakly-monotone class contains the other class. The class of anthropomorphic polygons has a simple triangulation algorithm [14]; a polygon $P$ is anthropomorphic if it has exactly one vertex $p_i$ such that the segment $\overline{p_{i-1}p_{i+1}}$ between its neighboring vertices is completely outside $P$, and two vertices with segments between neighboring vertices lying inside $P$. This class neither contains nor is contained in either the crab-shaped or weakly-monotone classes; Fig. 2 demonstrates the lack of inclusion. In [15], the classes join, fixed, and monotone visibility set are defined, none of which contains nor is contained by the weakly-monotone class. A polygon is monotone in direction $\theta$ if it can be decomposed into two chains such that the intersection between either chain and a line perpendicular to direction $\theta$ is a connected set (i.e. either a point, a segment or the empty set). A polygon is star-shaped if there exists a point $x \in P \cup \text{int}(P)$ such that $\overline{xy} \subset P \cup \text{int}(P)$, for all

Fig. 2. (a) anthropomorphic, but not crab-shaped nor weakly-monotone, (b) crab-shaped and weakly-monotone, but not anthropomorphic.
y \in P$. The weakly-monotone class contains the monotone class and, as we will see, the star-shaped class.

3. Detection of weakly-monotone polygons

In this section we present a linear-time detection algorithm for weak-monotonicity. First we present a formal definition of weakly-monotone.

We consider a polygon $P$ with vertices $p_0, \ldots, p_{n-1}$ ordered counterclockwise on $P$, and edges $e_0, \ldots, e_{n-1}$, where $e_i = \overline{p_ip_{i-1}}$ is directed from $p_{i-1}$ to $p_i$. Let $m_i$ be the midpoint of $e_i$. For points $a$ and $b \in P$, we define $P_{ccw}(a, b)$ as the subchain of $P$ obtained by traversing $P$ counterclockwise from $a$ to $b$, and $P_{cw}(a, b)$ as the one obtained traversing clockwise.

We define the concept of sweep. For this concept we represent directions as polar angles measured in radians on the unit circle in the usual way. Thus, $\theta \in [0, 2\pi)$ for any direction $\theta$ (in some contexts in this paper directions are reduced modulo $2\pi$ and in others they are not—directions are taken mod $2\pi$ when discussing sweep). Let $\theta_i$ denote the direction of edge $e_i$. For a vertex $p_i$ with incident edges $e_i$ and $e_{i+1}$, the directions $\theta_i$ and $\theta_{i+1}$ partition the unit circle into two arcs, one of less than $\pi$ radians and one of more than $\pi$ radians. Define the sweep closure of $p_i$, denoted $\text{sw}(p_i)$, to be the smaller (closed) arc. For a subchain $P_{ccw}(a, b)$ we define $\text{sw}(a, b) = \bigcup \text{sw}(p_i)$, where the union is over all vertices $p_i$ of $P_{ccw}(a, b)$ except $a$ and $b$. The sweep of a subchain $P_{ccw}(a, b)$, denoted $\text{sw}(P_{ccw}(a, b))$ or simply $\text{sw}(a, b)$, is the interior of $\text{sw}(a, b)$; that is, all directions of $\text{sw}(a, b)$ but the two boundary directions.

With our definition of sweep, we can give the formal definition of weakly-monotone: a simple polygon $P$ is weakly-monotone in direction $\theta$ if there exist vertices $s$ and $t$ such that $\theta + \pi \notin \text{sw}(P_{cw}(s, t))$ and $\theta + \pi \notin \text{sw}(P_{ccw}(s, t))$. If $\theta + \pi \notin \text{sw}(P_{cw}(s, t)) \cup \text{sw}(P_{ccw}(s, t))$, then $\theta \notin \text{sw}(P_{cw}(t, s)) \cup \text{sw}(P_{ccw}(t, s))$, so a polygon is weakly-monotone for pairs of opposite directions. Similarly, a polygon is monotone in the traditional sense for pairs of opposite directions. In fact, we can demonstrate the similarity between weakly-monotone and monotone polygons by rephrasing the usual definition of monotone polygons: a polygon $P$ is monotone in directions $\theta$ and $\theta + \pi$ if there exist vertices $s$ and $t$ such that $\text{sw}(P_{cw}(s, t))$ and $\text{sw}(P_{ccw}(s, t)) \subseteq (\theta - \pi/2, \theta + \pi/2)$.

Monotone and weakly-monotone polygons are alike not only in definition but also in their detection algorithms. A polygon is not monotone in directions $\theta$ and $\theta + \pi$ if and only if direction $\theta + \pi/2$ or $\theta + 3\pi/2$ is swept by some reflex vertex of $P$. The algorithm of [13] determines the set of directions in which a polygon is monotone by identifying all directions that are ‘swept backwards’ by reflex angles (the algorithm actually performs the equivalent task of determining which directions are in the sweep of more than one vertex). A similar characterization exists for weakly-monotone polygons: we will prove that a polygon $P$ is not
weakly-monotone in directions \( \theta \) and \( \theta + \pi \) if and only if there exists a ‘reflex’ subchain \( P_{CCW}(a, b) \) that ‘sweeps backwards’ both \( \theta \) and \( \theta + \pi \).

A concern throughout this section is the issue of ‘wrapping around’. This problem results when we designate a vertex as \( p_0 \), the first vertex of the polygon, when of course this designation is completely arbitrary. Our solution is to ‘double’ the polygon. By this we mean that we will consider the polygon to be a sequence of vertices \( p_0, \ldots, p_{n-1}, p_n, \ldots, p_{2n-1}, \) where \( p_{i+n} \) is a copy of \( p_i \) for \( i \in \{0, \ldots, n-1\} \). The concept of the doubled polygon will prove useful in the discussion of this section. For each edge \( e_i \) of the doubled polygon we define a direction value \( \phi_i \) (not taken modulo \( 2\pi \)) as follows. We set \( \phi_0 = \theta_0 \), the direction of edge \( e_0 \). For \( i > 0 \) we define \( \phi_i \) as \( \phi_{i-1} \) plus the size of the angle formed by the directed edges \( e_i \) and \( e_{i-1} \); we consider this angle to be positive (negative) if the common incident vertex \( p_{i-1} \) is convex (reflex). We define \( \Delta \phi(m_i, m_j) = \phi_j - \phi_i \); basically, this quantity measures the net change in direction if one traverses the polygon from \( m_i \) to \( m_j \). If \( \Delta \phi(m_i, m_j) < 0 \) then we call \( P_{CCW}(m_i, m_j) \) a reflex chain, because more turning occurs at reflex vertices. Note that a reflex chain may contain convex vertices.

We say that a vertex \( p_i \) is a \((\theta \text{ and } \theta + \pi)\)-tangency if a line in direction \( \theta \) through point \( p_i \) is locally tangent to the polygon (if an edge \( e_i = p_{i-1}p_i \) faces in direction \( \theta \) or \( \theta + \pi \), consider \( p_{i-1} \) and \( p_i \) to be tangencies if \( p_{i-1} \) and \( p_i \) are both reflex or both convex). Each \((\theta \text{ and } \theta + \pi)\)-tangency is specifically either a \(\theta\)-tangency or a \((\theta + \pi)\)-tangency; we assign to each tangency either \( \theta \) or \( \theta + \pi \) by traversing \( P \) counterclockwise and assigning the direction encountered at that vertex.

If \( p_i \) is a \((\theta \text{ and } \theta + \pi)\)-tangency, then there is a (unique) value between \( \phi_i \) and \( \phi_{i+1} \) that is equal to \( \theta \) modulo \( \pi \); we set \( \tilde{\phi}(\theta, p_i) \) equal to this value. For two vertices \( p_i \) and \( p_j \) with \( i < j \) that are \((\theta \text{ and } \theta + \pi)\)-tangencies, we define \( \Delta \tilde{\phi}(\theta, p_i, p_j) = \tilde{\phi}(\theta, p_j) - \tilde{\phi}(\theta, p_i) \). The value \( \Delta \tilde{\phi}(\theta, p_i, p_j) \) is always a multiple of \( \pi \), and it measures the net change in direction when traversing from \( p_i \) to \( p_j \) if one begins and ends in the directions of the tangencies.

As mentioned above, a chain \( P_{CCW}(m_i, m_j) \) is a reflex chain if \( \Delta \phi(m_i, m_j) < 0 \). We say that the reflex chain back-sweeps the arc of directions \( (\phi_j, \phi_i) \) taken modulo \( 2\pi \) (if \( \Delta \phi(m_i, m_j) < -2\pi \) then the reflex chain back-sweeps the entire unit circle of directions). If directions \( \theta \) and \( \theta + \pi \) are both back-swept by a reflex chain, then we say that the reflex chain double-back-sweeps the pair. The following lemma is the basis for our weakly-monotone detection algorithm. It states that a polygon \( P \) is weakly-monotone in a pair of directions \( \theta \) and \( \theta + \pi \) if and only if no reflex chain double-back-sweeps the pair.

**Lemma 1.** A polygon \( P \) is weakly-monotone in a pair of directions \( \theta \) and \( \theta + \pi \) if and only if no reflex chain double-back-sweeps the directions \( \theta \) and \( \theta + \pi \).

**Proof.** (Refer to Fig. 3.) Suppose \( \theta \) and \( \theta + \pi \) are double-back-swept by a reflex
Then the reflex chain contains a pair of $(\theta$ and $\theta + \pi)$-tangencies $p_i$ and $p_j$ such that $i<j$ and $\Delta \phi(\theta, p_i, p_j) = -\pi$. We will assume without loss of generality that $p_i$ is a $\theta$-tangency (which implies that $p_j$ is a $(\theta + \pi)$-tangency). This implies $\Delta \phi(\theta, p_i, p_{i+n}) = 3\pi$, which means there exist vertices $p_k$ and $p_l$ such that $i<j<k<l$ and $p_k$ is a $\theta$-tangency while $p_l$ is a $(\theta + \pi)$-tangency, as in Fig. 3. In other words, as we traverse $P$ we encounter the four vertices $p_i, p_j, p_k$ and $p_l$ in the order stated, where vertices alternate between being $\theta$-tangencies and $(\theta + \pi)$-tangencies. An exhaustive inspection reveals that no matter how the splitting points $s$ and $t$ are chosen, one of $P_{CCW}(s, t)$ and $P_{CCW}(t, s)$ will contain both a $\theta$-tangency and a $(\theta + \pi)$-tangency, or both $P_{CCW}(s, t)$ and $P_{CCW}(t, s)$ will contain a $(\theta$ and $\theta + \pi)$-tangency of the same type; either event prevents $P$ from being weakly-monotone in directions $\theta$ and $\theta + \pi$ for the given splitting points, therefore we have that $P$ is not weakly-monotone in $\theta$ and $\theta + \pi$.

Suppose $P$ is not weakly-monotone in directions $\theta$ and $\theta + \pi$. This means that the $\theta$-tangencies and the $(\theta + \pi)$-tangencies cannot be partitioned into separate subchains. This implies the existence of vertices $p_i, p_j, p_k$ and $p_l$ that appear on $P$ in the order stated, where $p_i$ and $p_k$ are $\theta$-tangencies and $p_j$ and $p_l$ are $(\theta + \pi)$-tangencies, as in Fig. 3 (the lack of such a configuration would allow a pair of legal splitting points). Since each of $\Delta \phi(\theta, p_i, p_j), \Delta \phi(\theta, p_j, p_k), \Delta \phi(\theta, p_k, p_l)$ and $\Delta \phi(\theta, p_l, p_{i+n})$ equals $\pi$ modulo $2\pi$, and the sum of these four quantities is $2\pi$, at least one of them (say $\Delta \phi(\theta, p_i, p_j)$) is negative. By either including or excluding each of $p_i$ and $p_j$, we obtain a reflex chain that double-back-sweeps $\theta$ and $\theta + \pi$ (if an edge incident to $p_i$ or $p_j$ lies in direction $\theta$ or $\theta + \pi$ then we may have to include or exclude additional adjacent vertices). □

Lemma 1 provides us with a strategy for determining the set of weakly-monotone directions: traverse $P$ (twice), searching for all directions double-back-swept by reflex chains. We now present the algorithm Detect, which for input polygon $P$ returns all directions of weak-monotonicity.
Algorithm Detect

We consider the sequence of edges $e_0, \ldots, e_{2n-1}$ encountered while twice traversing $P$ counterclockwise, from the starting edge $e_0$. We must traverse $P$ twice because of wraparound: the initial edge $e_0$ could be in a reflex chain. We define the front direction, $f_P(j) = \max_{i=0,\ldots,j} \phi_i$, where $e_j$ is the current edge.

The algorithm proceeds as follows. Beginning at $e_0$, traverse $P$ twice, updating $f_P$. Whenever $\phi_j \neq f_P(j)$ for the current edge $e_j$, we are in a reflex chain, and if $\phi_j + \pi < f_P(j)$, then the reflex chain double-back-sweeps some directions. If we encounter an edge $e_j$ such that $\phi_{j-1} = f_P(j-1)$ but $\phi_j < f_P(j)$ (i.e., $\phi_{j-1}$), then we store $\beta = \phi_{j-1}$ and set $\alpha \leftarrow \phi_j$. Upon encountering edge $e_k$ such that $\phi_{k-1} = f_P(k-1) + \pi < f_P(j)$, we update $\alpha \leftarrow \phi_k$. When $\phi_j = f_P(j)$ for the current edge $e_j$, we check if $\alpha + \pi < \beta$, and if so we eliminate the interval of directions $(\alpha + \pi, \beta)$.

The algorithm outputs the set of weakly-monotone directions, which is all directions $\theta$ such that neither $\theta$ nor $\theta + \pi$ has been eliminated by the traversals of $P$.

End of Detect

Let us discuss the correctness of the algorithm. Whenever $\phi_j \neq f_P(j)$ for the current edge $e_j$, we are in a reflex chain. There exists an edge $e_i$ with $i \in \{0, \ldots, j\}$ such that $\phi_i = f_P(i)$ and $P_{CCW}(m_i, m_j)$ is a reflex chain that back-sweeps $(\phi_j, \phi_i)$. Therefore when the algorithm eliminates an interval of directions $(\alpha + \pi, \beta)$, these directions are in fact double-back-swept by a reflex chain. Later the algorithm eliminates the partners of these directions, which are those found in the interval $(\alpha, \beta - \pi)$. Thus, every direction deemed by the algorithm not to be a weakly-monotone direction is indeed not weakly-monotone.

We must now show that every non-weakly-monotone direction is found by the algorithm. We will show that every reflex chain is accounted for. Consider a reflex chain $P_{CCW}(m_k, m_j)$ (because we think of $P$ as the doubled polygon, any reflex chain can be represented as $P_{CCW}(m_k, m_j)$ for $0 \leq k < j \leq 2n-1$). At some moment $e_j$ is the current edge, and at this time the algorithm has $\phi_j$ and $f_P(j)$. Since $f_P(j) = \max_{i=0,\ldots,j} \phi_i$, we have $\phi_k \leq f_P(j)$. Thus the set of directions double-back-swept by the reflex chain $P_{CCW}(m_k, m_j)$, which is $(\phi_j, \phi_k)$, is a subset of $(\phi_j, f_P(j))$. This implies that all directions that should be eliminated as candidates for being weakly-monotone are eliminated.

We discuss the manner in which the algorithm processes the intervals of eliminated directions. The traversal of $P$ produces a list of intervals, $(\alpha_1 + \pi, \beta_1), \ldots, (\alpha_m + \pi, \beta_m)$—at this time the directions are not taken modulo $2\pi$. The intervals may overlap; however, since the values $\beta_1, \ldots, \beta_m$ occur in sorted order we can compute the union of the intervals by means of a single traversal of the intervals. Now, we wish to reduce the intervals modulo $2\pi$. Since $P$ is traversed twice, the range of directions in the intervals exceeds $6\pi$ only if all directions are double-back-swept, so the reduction modulo $2\pi$ can be performed by merging a constant number of sorted lists. Finally, for any direction $\theta$
eliminated we must eliminate $\theta + \pi$, which is accomplished by adding $\pi$ to a copy of the intervals and merging the copy with the original list. The output is a list in sorted order of the intervals of directions from 0 to $2\pi$ of the non-weakly-monotone directions.

All operations of the algorithm can be performed in time proportional to $n$, the number of vertices of the polygon. We summarize our results.

**Theorem 2.** Given a simple polygon $P$ with $n$ vertices, Algorithm Detect determines in $O(n)$ time all directions in which $P$ is weakly-monotone.

We can actually do more with our detection algorithm. We saw earlier that if $\theta$ and $\theta + \pi$ are a pair of weakly-monotone directions, then the $\theta$-tangencies and $(\theta + \pi)$-tangencies lie in disjoint subchains of $P$. It is in fact true that if $[\theta_i, \theta'_i]$ is a weakly-monotone interval of directions, then the $[\theta_i, \theta'_i]$-tangencies and the $[\theta_i + \pi, \theta'_i + \pi]$-tangencies lie in disjoint subchains, where a $[\theta_i, \theta'_i]$-tangency is any $\theta$-tangency for a value $\theta \in [\theta_i, \theta'_i]$. Therefore, there exist vertices $s_i$ and $t_i$ such that for any $\theta \in [\theta_i, \theta'_i]$, $P$ is weakly-monotone in $\theta$ with respect to splitting points $s_i$ and $t_i$. Furthermore, if $\{(\theta_i, \theta'_i) : 1 \leq i \leq k\}$ is the set of all (maximal) weakly-monotone intervals, then we can compute $s_i$ and $t_i$ for $i = 1, \ldots, k$ in $O(n)$ total time. This is possible because the first $[\theta_i, \theta'_i]$-tangencies (i.e. the first $\theta_i$-tangencies) of the intervals are ordered counterclockwise on $P$ by the directions $\theta_i$; similarly, the last $[\theta_i, \theta'_i]$-tangencies (i.e. the last $\theta'_i$-tangencies) are ordered on $P$ by $\theta'_i$. In this way, we can preprocess $P$ in $O(n)$ time such that, if we are given a pair of directions $\theta$ and $\theta + \pi$, we can query in $O(\log n)$ time whether this is a weakly-monotone pair, and if it is we also can return a valid pair of splitting points. These query times are optimal in the sense that a polygon can have $\Theta(n)$ pairs of opposite weakly-monotone cones, and each pair can require a distinct pair of splitting points (as in Fig. 4).

Our definition of weak-monotonicity relaxes the traditional definition of monotonicity on a polygonal chain. The notion of a $\phi$-monotone chain, as introduced in [1], considers a continuum of restrictions on the sweep of the chain.

![Fig. 4. A polygon with many pairs of arcs of weakly-monotone directions.](image-url)
A chain \( c \) is \( \phi \)-monotone in direction \( \theta \) if \( \text{sw}(c) \subseteq (\theta - \phi/2, \theta + \phi/2) \). Note that the usual definition of monotonicity corresponds to \( \pi \)-monotonicity, and weak-monotonicity to \( 2\pi \)-monotonicity. The results of this section can be extended to test for \( \phi \)-monotonicity for any \( \phi \) such that \( \pi \leq \phi \leq 2\pi \).

4. Triangulation of weakly-monotone polygons

In this section we describe a simple, linear-time algorithm for triangulating a weakly-monotone polygon. While the algorithm of [2] triangulates a general polygon in linear time, the following method for weakly-monotone polygons is considerably less complex.

We are given a simple polygon \( P \), splitting points \( s \) and \( t \), and a direction \( \theta \) such that \( P_{\text{cw}}(s, t) \) and \( P_{\text{ccw}}(s, t) \) are weakly-monotone in direction \( \theta \) (the previous section describes how to compute a triplet \( s, t \) and \( \theta \) for a polygon \( P \)). Assume without loss of generality that \( \theta = 0 \). For simplicity, we will denote \( P_{\text{cw}}(s, t) \) as \( c_T \) and \( P_{\text{ccw}}(s, t) \) as \( c_B \).

For ease of exposition, the following discussion makes an assumption: no edge of \( P \) is horizontal. This will simplify the definitions and the presentation of several steps of the algorithm. The assumption is not necessary, however. If each maximal horizontal subchain of \( c_T \) and \( c_B \) is viewed as being just a single vertex, then the following definitions and algorithm apply without change. Viewing a horizontal chain as a single vertex does not introduce difficulties because, as we shall see, we are concerned only with horizontal visibility.

If \( p \) is a vertex of a chain \( c \in \{c_T, c_B\} \), define \( p.\text{prev} \) and \( p.\text{next} \) to be the vertices preceding and succeeding \( p \), respectively, on \( c \). Let \( x_p \) and \( y_p \) represent the \( x \)- and \( y \)-coordinates of the point \( p \). A vertex \( p \) (where \( p \notin \{s, t\} \)) is a peak of \( c \) if \( y_p > y_{p.\text{prev}} \) and \( y_p > y_{p.\text{next}} \), and \( p \) is a valley of \( c \) if it satisfies the above conditions with the inequalities reversed. Together, the peaks and valleys comprise the horizontal tangencies.

For a segment \( ab \), let \( \text{int}(ab) = ab \setminus \{a, b\} \). A chord of \( P \) is a segment \( ab \) such that \( a, b \in P \) and \( \text{int}(ab) \subseteq \text{int}(P) \). A horizontal chord is a chord \( ab \) with \( y_a = y_b \).

The horizontal visibility map of a simple polygon \( P \), denoted \( \text{HVM}(P) \), is the collection of all horizontal chords with a vertex of \( P \) as an endpoint (see Fig. 5). This structure tells what any vertex of \( P \) sees when it looks horizontally left or right in the interior of \( P \). We will concentrate on computing a subset of \( \text{HVM}(P) \)—we will compute all horizontal chords of horizontal tangencies of \( P \) (we denote this \( p-\text{HVM}(P) \)). We claim that the full map \( \text{HVM}(P) \) can be easily computed from this partial map. To see this, first note that the horizontal chords of horizontal tangencies are exactly those emanating from valleys of \( c_T \) and peaks of \( c_B \), since a horizontal tangency \( p \) that contradicts this rule has \( \pi \in \text{sw}(p) \), a contradiction of the fact that \( c_T \) and \( c_B \) are weakly-monotone in direction \( \theta = 0 \). If we insert these horizontal chords, we obtain a partitioning of \( P \) into subpolygons.
which are monotone in the vertical direction. One can obtain a full HVM for such a subpolygon by simultaneously traversing its left and right sides; this is true because the vertical monotonicity of the subpolygon implies that any horizontal line intersects the left and right sides each at most once.

We see from the argument above that HVM(P) can be easily obtained from p-HVM(P) in O(n) time. Furthermore, it is possible to transform HVM(P) into a triangulation of P in O(n) time, using the brief algorithm of [6]. Therefore, we will focus our attention on computing p-HVM(P).

Our method of computing p-HVM(P) consists of two parts. In the first, called procedure Pocket, we build partial horizontal visibility maps for cT and cB separately (see Fig. 6). In the second, procedure Merge, we merge this visibility information to obtain p-HVM(P). For the chain cT, Pocket assigns to each valley v some visibility information. We define \( \ell(v, c_T) \) as the point \( q \) of \( c_T \) with \( y_q = y_v \) and \( x_q < x_v \) such that \( \text{int}(v, \ell(v, c_T)) \cap c_T = \emptyset \) (if there is no such point \( q \), then \( \ell(v, c_T) = (-\infty, y_v) \)). In other words, \( \ell(v, c_T) \) is the point seen by \( v \) as it looks horizontally to the left with respect to \( c_T \). The point \( r(v, c_T) \) lying to the right of \( v \) is defined in a similar manner. For each valley \( v \), Pocket computes \( \ell(v, c_T) \) and \( r(v, c_T) \), inserts these points as vertices into \( c_T \), and constructs pointers between

![Fig. 5. (a) HVM(P), (b) p-HVM(P).](image)

![Fig. 6. (a) p-HVM(cT), (b) p-HVM(cB).](image)
Algorithm Triangulate \((P; s, t, \theta)\)

Rotate \(P\) so that \(\theta = 0\).

Call \(\text{Pocket}(c_T)\) and \(\text{Pocket}(c_B)\).

Call \(\text{Merge}(p\text{-HVM}(c_T), p\text{-HVM}(c_B))\).

Refine \(p\text{-HVM}(P)\) to \(HVM(P)\), using the method described above.

Transform \(HVM(P)\) into a triangulation of \(P\), using the method of [6].

End of Triangulate

Procedure \(\text{Pocket}(c_T)\)

The input is \(c_T\), a simple infinite chain weakly-monotone in direction \(\theta = 0\). We will traverse the chain from its start point \(s\) to its end point \(t\). For each valley \(v\), we insert the vertices \(\ell(v, c_T)\) and \(r(v, c_T)\) into \(c_T\) (if these points are not at infinity), and we construct pointers between \(v\) and \(\ell(v, c_T)\) and between \(v\) and \(r(v, c_T)\). We insert a point \(T(S, c_T)\) if \(y_i < y_{i-1}\) and we insert \(e(t, c_T)\) if \(y_{i+1} < y_{i-1}\). There are two starting cases. After the appropriate starting case calls sub-procedure \(\text{Down}\), the procedure continuously calls \(\text{Down}\) until termination.

The starting cases are:

1. \(y_i < y_{s.next}\): Let \(p\) be the first peak in \(c_T\). Call \(\text{Down}(p)\).

2. \(y_i > y_{s.next}\): We traverse to the first valley, and call it \(v\). Set \(\ell(v, c_T) \leftarrow (-\infty, y_v)\).

Let \(p\) be the first peak after \(v\). Call \(\text{Down}(p)\).

End of \(\text{Pocket}\)

Procedure \(\text{Down}(p)\)

We set \(a \leftarrow p\) and \(b \leftarrow p\), and simultaneously traverse with \(a\) and \(b\), going backwards with \(a\) and forward with \(b\), keeping \(a\) and \(b\) at the same approximate \(y\)-coordinate. Three cases can occur. Refer to Fig. 7.

Case (1): The left side, \(a\), encounters a valley or encounters \(s\).

Denote by \(b'\) the point near \(b\) on the right side such that \(y_{b'} = y_a\). Set \(r(a, c_T) \leftarrow b'\).

If \(a \neq s\) and \(\ell(a, c_T)\) is not a point at infinity (Fig. 7(a)), then: set \(a \leftarrow \ell(a, c_T)\) and \(b \leftarrow b'\), and continue the double-traversal.
Else \((a = s\) or \(\ell(a, c_T)\) is a point at infinity, as in Fig. 7(b)): let \(v\) be the first valley after \(b\), set \(\ell(v, c_T) \leftarrow (-\infty, y_v)\), let \(p\) be the first peak after \(v\), and call Down\((p)\).

Case (2): The right side, \(b\), encounters a valley (Fig. 7(c)).
Denote by \(a'\) the point on the left side near \(a\) such that \(y_{a'} = y_b\). Set \(\ell(b, c_T) \leftarrow a'\).
Let \(p\) be the first peak after \(b\), and call Down\((p)\).

Case (3): The right side, \(b\), encounters \(t\).
Denote by \(a'\) the point on the left side near \(a\) such that \(y_{a'} = y_t\). Set \(\ell(t, c_T) \leftarrow a'\).
For each valley \(v\) such that \(r(v, c_T)\) has not been assigned a value, set \(r(v, c_T) \leftarrow (\infty, y_v)\). (If \(y_s < y_{s,\text{next}}\) and \(r(s, c_B)\) has not been assigned a value, then set \(r(s, c_T) \leftarrow (\infty, y_s)\).) Terminate Down and Pocket.
End of Down

Procedure Merge \((p-HVM(c_T), p-HVM(c_B))\)
The input is \(p-HVM(c_T)\) and \(p-HVM(c_B)\), the modified versions of \(c_T\) and \(c_B\), where \(\ell(v, c_T)\) and \(r(v, c_T)\) for each valley \(v\) of \(c_T\) and \(\ell(p, c_B)\) and \(r(p, c_B)\) for each peak \(p\) of \(c_B\) have been inserted as vertices, and the appropriate pointers have been constructed. (For simplicity, we will refer to \(p-HVM(c_T)\) and \(p-HVM(c_B)\) simply as \(c_T\) and \(c_B\).) The procedure traverses \(c_T\) and \(c_B\) simultaneously with pointers \(d\) and \(e\), respectively, where the \(y\)-coordinates of the pointers are kept approximately equal. It also uses auxiliary pointers \(f\) (with \(f \in c_T\)) and \(g\) (with \(g \in c_B\). The procedure is always in one of two sub-procedures, Moving-Up and Moving-Down, according to whether the \(y\)-coordinate of the pointers is increasing or decreasing at the time. the output is \(p-HVM(P)\).

Let \(d \leftarrow s\) and \(e \leftarrow s\). There are three initial cases.

1. \(y_d\text{.next} > y_s\) and \(y_e\text{.next} > y_s\). Call Moving-Up\((s, s)\).
2. \(y_d\text{.next} < y_s\) and \(y_e\text{.next} < y_s\). Call Moving-Down\((s, s)\).
3. \(y_d\text{.next} > y_s\) and \(y_e\text{.next} < y_s\). If \(x_{r(s, c_B)} < x_{r(s, c_T)}\) then call Moving-Up\((s, r(s, c_B))\), else call Moving-Down\((r(s, c_T), s)\).
Note that $y_{d, \text{next}} < y_s$ and $y_{e, \text{next}} > y_s$ cannot occur.

End of Merge

**Procedure Moving-Up($d, e$)**

We have $d \in c_T$ and $e \in c_B$ such that $d$ is a valley (or $d = s$) and $de$ is a horizontal chord of $P$, i.e. $y_d = y_e$ and $\text{int}(de) \subseteq \text{int}(P)$. We traverse simultaneously with $d$ and $e$, where $d$ proceeds forwards on $c_T$ and $e$ proceeds forwards on $c_B$. Also, we initialize a pointer $f$ to $r(d, c_T)$, and include $f$ in the simultaneous traversal by proceeding backwards on $c_I$ with $f$. Under the simultaneous traversal, the $y$-coordinates of $d$, $e$ and $f$ remain approximately equal. (Note that $f$ can be a pointer to a point at infinity, in which case $f = (\infty, y_d)$.) Several events can occur. Refer to Fig. 8.

**Case (1):** The pointer $d$ encounters the point $\ell(v, c_T)$ for some valley $v \in c_T$.

Denote by $e'$ the point of $c_B$ near $e$ such that $y_{e'} = y_d$.

If $x_{e'} < x_{e'}$ (Fig. 8(a)) then set $\ell(v) \leftarrow \ell(v, c_T)$ and $r(v) \leftarrow e'$, and set $d \leftarrow v$ and continue; else ($x_{e'} > x_{e'}$, as in Fig. 8(b)) set $f \leftarrow v$ and continue.

**Case (2):** The pointer $e$ encounters a peak.

Denote by $d'$ and $f'$ the points of $c_T$ near $d$ and $f$ such that $y_{d'} = y_e = y_{f'}$. Set $\ell(e) \leftarrow d'$.

If $x_{r(e, c_B)} < x_{f'}$ (Fig. 8(c)) then set $r(e) \leftarrow r(e, c_B)$, set $e \leftarrow r(e, c_B)$ and continue; else ($x_{r(e, c_B)} > x_{f'}$, as in Fig. 8(d)) set $r(e) \leftarrow f'$ and call Moving-Down($f'$, $e$).

**Case (3):** The pointer $e$ encounters $t$.

For all valleys $v \in c_T$ such that $\ell(v)$ and $r(v)$ have not been assigned values, set

![Fig. 8. Procedure Moving-Up.](image-url)
\( \ell(v) \leftarrow \ell(v, c_T) \) and \( r(v) \leftarrow r(v, c_T) \) (similarly for peaks of \( c_B \)). Terminate Moving-Up and Merge.

End of Moving-Up

We discuss the correctness of the algorithm. We require the following lemma.

**Lemma 3.** Given a simple chain \( c \) that is weakly-monotone in direction \( \theta \), a line \( \bar{l} \) in direction \( \theta \), and three distinct points \( u, v, w \in c \cap \bar{l} \). The ordering of the points determined by their positions on \( \bar{l} \) is the same as or is the reverse of the ordering determined by their positions on \( c \).

**Proof.** An ordering on \( c \) which is not the same as or the reverse of the ordering on \( \bar{l} \) would result in \( \theta + \pi \in \text{sw}(c) \), a contradiction of the fact that \( c \) is weakly-monotone in direction \( \theta \). □

We discuss procedure Pocket for \( c_T \); an analogous discussion applies for \( c_B \).

The procedure Pocket requires \( O(n) \) time, since no vertex is traversed more than once forward and once backward. Pocket maintains the invariant that at any moment \( \ell(v, c_T) \) has been correctly determined for each valley \( v \) that has already been traversed, and \( r(v, c_T) \) has been correctly determined for each valley \( v \) such that \( r(v, c_T) \) has been traversed. When the pointer \( a \) encounters a valley, Pocket computes \( b' \) and sets \( r(a, c_T) \leftarrow b' \). This is a correct construction if \( \text{int}(ab') \cap c_T = \emptyset \). By the behavior of Pocket, we know that no point of \( c_T \) between \( a \) and \( b' \) intersects \( ab' \), and by Lemma 3 we know that no point of \( c_T \) preceding \( b' \) or succeeding \( b' \) intersects \( ab' \). A similar argument establishes that when pointer \( b \) encounters a valley, the value \( \ell(b, c_T) \) is correctly computed.

**Definition 1.** A mixed horizontal chord of \( P \) is a horizontal chord \( \bar{uv} \) of \( P \) such that \( u \) and \( v \) are on different chains; that is \( u \in c_T \) and \( v \in c_B \), or \( v \in c_T \) and \( u \in c_B \).

**Lemma 4.** If \( \bar{uv} \) is a mixed horizontal chord of \( P \), then at some moment of procedure Merge, we have \( \bar{de} = \bar{uv} \).

**Proof.** The pointers \( d \) and \( e \) traverse the chains \( c_T \) and \( c_B \), respectively, monotonically from \( s \) to \( t \), with occasional 'skips' to points further forward. Thus we can say that \( \bar{de} \) assumes the value of every mixed horizontal chord, with the exception of those omitted as a result of the skipping of \( d \) or \( e \). Consider a case of \( d \) skipping. We have points \( d', e \) and \( f' \), where \( x_{d'} < x_e < x_{f'} \) and \( y_{d'} = y_e = y_{f'} \), and \( \text{int}(d'e), \text{int}(ef') \subset \text{int}(P) \). Therefore any horizontal chord formed with an endpoint on the subchain of \( c_T \) between \( d' \) and \( f' \) must have both endpoints on this subchain, which implies that the 'skip' step of setting \( d \leftarrow f' \) does not result in Merge missing any mixed horizontal chords. A similar argument applies for the case of \( e \) skipping. □
The correctness of \textit{Merge} follows from the above lemma and the visibility properties maintained by \textit{Merge}. Whenever \textit{Moving-Up} encounters a peak or valley that is an endpoint of a mixed horizontal chord, it determines the points \(\ell(\cdot)\) and \(r(\cdot)\). At all times the points \(d\) and \(f\) are visible with respect to \(c_T\), and \(d\) and \(e\) are visible with respect to the entire polygon \(P\), so whenever \textit{Moving-Up} updates a value \(\ell(\cdot)\) or \(r(\cdot)\) it does so correctly; likewise \textit{Moving-Down} makes only correct updates. Lemma 4 states that all situations that require updating are noticed by either \textit{Moving-Up} or \textit{Moving-Down}. The procedure \textit{Merge} requires \(O(n)\) time because no point is traversed more than once forward and once backward.

The following discussion may aid an intuitive understanding of procedure \textit{Merge}. Consider the shortest path in \(P\) from \(s\) to \(t\), which we denote \(SP(s, t)\). Each point of \(SP(s, t)\) is contained by a horizontal chord; we see that this chord is a mixed horizontal chord, since no point on a non-mixed horizontal chord can be on \(SP(s, t)\). Conversely, each mixed horizontal chord contains a point of \(SP(s, t)\). If a point of \(SP(s, t)\) is a horizontal tangency of \(SP(s, t)\), then it is either a valley of \(c_T\) or a peak of \(c_B\) and it lies on two mixed horizontal chords (one to the left and one to the right). If a point of \(SP(s, t)\) is not a horizontal tangency, then it lies on a unique mixed horizontal chord.

A natural ordering is defined on the points of \(SP(s, t)\) by their distance from \(s\), i.e. the order in which the points are encountered when one traverses \(SP(s, t)\) from \(s\) to \(t\). Similarly, the mixed horizontal chords are ordered by the order in which they assume the value of the current chord \(\overline{de}\) in procedure \textit{Merge}. These two orderings are in fact the same, under the correspondence between mixed horizontal chords and points of \(SP(s, t)\). We therefore can think of \textit{Merge} as a traversal of \(SP(s, t)\), where at any moment we see one of \(c_T\) and \(c_B\) to the left and the other to the right, and where \textit{Merge} switches between \textit{Moving-Up} and \textit{Moving-Down} at every peak or valley of \(SP(s, t)\). Procedure \textit{Merge} notices exactly those visibility pointers in need of updating, since a point of \(c_T(c_B)\) horizontally sees a point of \(c_B(c_T)\) if and only if it horizontally sees a point of \(SP(s, t)\).

In summary, the procedure \textit{Merge} constructs \(p\text{-HVM}(P)\), a partitioning of \(P\) into polygons monotone in the vertical direction. In turn, \(p\text{-HVM}(P)\) is refined to \(HVM(P)\) by performing a simultaneous traversal on each monotone subpolygon, and a triangulation is obtained from \(HVM(P)\) by the method of \cite{6}. All steps require \(O(n)\) time. We have the following result.

\textbf{Theorem 5.} The algorithm \textit{Triangulate} triangulates an \(n\)-vertex polygon \(P\) that is weakly-monotone with respect to \(s\), \(t\) and \(\theta\) in \(O(n)\) time.

\section{Conclusion}

In this paper, we have introduced weakly-monotone polygons, and given a
linear-time algorithm for determining the set of directions in which a polygon is weakly-monotone. This detection algorithm allows us to use the simple triangulation algorithm for weakly-monotone polygons that is given in this paper, without prior knowledge of a polygon's weak-monotonicity.

We mention several extensions of this work, and an open question. Throughout this paper we have assumed that the input is a polygonal chain—a concatenation of straight line segments. In fact, the discussion of this paper regarding detection extends to well-behaved curved chains; for example, splines [4].

Much attention has been given recently to parallel algorithms in computational geometry. The methods of this paper can be transformed into optimal parallel algorithms: the weakly-monotone detection problem can be solved optimally (i.e. \(O(\log n)\) time and \(O(n/\log n)\) processors) in the EREW PRAM computational model, and the weakly-monotone polygon triangulation problem can be solved optimally in the CREW PRAM model [9]. While the general polygon triangulation problem has been solved to optimality by Goodrich [8], we argue (as in the sequential case) that the special-case algorithm for weakly-monotone polygons is less complex.

We close with an open question. Chazelle and Incerpi [3] define the sinuosity of a polygon, and give an algorithm to triangulate a polygon with sinuosity \(s\) in time \(O(n \log s)\). The sinuosity is dependent on the orientation of the polygon; in fact, a polygon can have \(s = 1\) for one orientation and \(s = \Omega(n)\) for another. A polygon \(P\) being weakly-monotone in direction \(\theta\) is equivalent to \(P\) having sinuosity \(s - 1\) when oriented so that \(\theta\) is the horizontal direction. (Thus, the algorithm of [3] could be used as a linear-time algorithm in place of the triangulation algorithm of this paper, however, the algorithm of this paper is less complex, since it is tailored especially to weakly-monotone polygons.) An interesting open question is whether one can construct an efficient algorithm to determine the orientation of \(P\) that admits the minimum sinuosity. If \(s\) represents this 'true' sinuosity of \(P\), we would like such an algorithm to run in \(O(n \log s)\) time, so that the triangulation algorithm of [3] would have an accompanying detection algorithm, just as the detection algorithm of this paper accompanies the triangulation algorithm.

Acknowledgment

The author would like to thank the anonymous referees for many helpful comments.

References

Linear-time algorithms for weakly-monotone polygons