LINEAR-TIME ALGORITHMS FOR
WEAKLY-MONOTONE POLYGONS

By

Paul J. Heffernan
Linear-Time Algorithms for Weakly-Monotone Polygons

Paul J. Heffernan*

SORIE, Cornell University
Ithaca, NY 14853

January 18, 1991

Abstract
We introduce a new class of simple polygons, *weakly-monotone*, 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 a new class of polygons, *weakly-monotone*, 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 ([PrS],[LP]). These detection 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 [GJPT] or a star-shaped [ET] 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.

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 [Ch] is conceptually difficult, and too complex to be considered practical. Many of the general algorithms are simpler ([GJPT],[KKT],[To]), but each is super-linear in the worst case. The algorithms of [Ch] and [HeMe] have time bounds that depend on some parameter of the polygon, but each runs in time $O(n \log n)$ 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.

*Supported by an NSF graduate fellowship. The author also thanks DIMACS and Rutgers University for use of facilities.
2 Weakly-Monotone Polygons

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 Figure 1).

We now give some auxiliary definitions, and a more formal definition of weakly-monotone. Given a polygonal chain $c$ and a direction $\theta$, write $c$ as the concatenation of (maximal) subsequences $c = c_1, \ldots, c_k$, each monotone in $\theta + \pi/2$ (the direction perpendicular to $\theta$). We say that the chain $c$ is weakly-monotone in direction $\theta$ if the following holds: any line $l$ in direction $\theta$ that intersects $c$ must do so in such a way that if $p \in c_i$ and $q \in c_j$ are two points of intersection with $p$ preceding $q$ on $l$, then $i \leq j$. If $a$ and $b$ are two points of a polygon $P$, then $PCW(a, b)$ and $PCCW(a, b)$ are the subsequences of $P$ obtained by traversing $P$ from $a$ to $b$ clockwise and counterclockwise, respectively (clockwise is the direction of traversal such that the interior of $P$ lies to the right of each oriented edge of $P$). We say that a polygon $P$ is weakly-monotone in direction $\theta$ with splitting points $s$ and $t$ if $PCW(s, t)$ and $PCCW(s, t)$ are weakly-monotone in direction $\theta$. A polygon monotone in $\theta$ clearly is weakly-monotone in $\theta$.

If a polygon $P$ is weakly-monotone, we can determine a triplet $s, t, \theta$ 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 [ET], where any polygon in the hierarchy can be triangulated by some specific algorithm simpler than that of [Ch]. All classes in the hierarchy are contained in the class of crab-shaped polygons, for which there exists no specific triangulation algorithm. Figure 2 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 [To]; this class neither contains nor is contained in either the crab-shaped or weakly-monotone classes. Figure 3 demonstrates the lack of inclusions. In [Ts], the classes join, fixed, and monotone visibility set are defined, none of which contains nor is contained by the weakly-monotone class. 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. We require some additional definitions.

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 $\theta$. 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 $p_i$ (arithmetic is mod $n$). Let $\phi_i$ denote the direction of edge $e_i$, and $m_i$ the midpoint of $e_i$. For a vertex $p_i$ with incident edges $e_i$ and $e_{i+1}$, the directions $\phi_i$ and $\phi_{i+1}$ 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 $p_i$, $\overline{sw}(p_i)$, to be the smaller (closed) arc. For a subchain $PCCW(a, b)$, $\overline{sw}(a, b) = \cup \overline{sw}(p_i)$, where the union is over all vertices $p_i$ of $PCCW(a, b)$ except $a$ and $b$. The sweep of a subchain $PCCW(a, b)$, denoted $sw(PCCW(a, b))$ or, simply, $sw(a, b)$, is the interior of $\overline{sw}(a, b)$; that is, all directions of $\overline{sw}(a, b)$ but the two boundary directions.

With our definition of sweep, we can give an alternate definition of weakly-monotone: a simple polygon $P$ is weakly-monotone in direction $\theta$ if there exist vertices $s$ and $t$ such that $\theta + \pi \notin sw(PCW(s, t)), \theta + \pi \notin sw(PCCW(s, t))$. Then $PCW(a, b)$ is in the direction $\theta$ and $PCCW(a, b)$ is in the direction $\theta + \pi$. It is easy to see that this is equivalent to the definitions given above.
Figure 1: Example of a weakly-monotone polygon.

Figure 2: Classes of polygons.

(a) weakly-monotone, but not crab-shaped

(b) crab-shaped, but not weakly-monotone
sw(PCCW(s, t)). If θ + π ∈ sw(PCW(s, t)), sw(PCCW(s, t)), then θ ∉ sw(PCW(t, s)), sw(PCCW(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 \( \theta, \theta + \pi \) if there exist vertices \( s \) and \( t \) such that \( sw(PCW(s, t)), sw(PCCW(s, t)) \subseteq (\theta - \pi/2, \theta + \pi/2) \).

Monotone and weakly-monotone polygons are alike not only in definition but also in their detection algorithms. A polygon is not monotone in directions \( \theta, \theta + \pi \) if and only if direction \( \theta + \pi/2 \) or \( \theta + 3\pi/2 \) is swept by some reflex vertex of \( P \). The algorithm of [PrS] 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 \( \theta, \theta + \pi \) if and only if there exists a “reflex” subchain \( PCCW(a, b) \) such that \( \theta, \theta + \pi \in sw(PCCW(a, b)) \).

A simple polygon has two types of vertices: convex and reflex. For points \( a \) and \( b \) on \( P \) (not necessarily vertices), the interior vertices of \( PCCW(a, b) \) are all vertices of \( P \) on \( PCCW(a, b) \) except \( a \) and \( b \). Given a polygon \( P \) and points \( a \) and \( b \) on \( P \), we define \( \Delta \theta(a, b) \) as the sum of the measure of the angle turned (in radians) for all convex interior vertices of \( PCCW(a, b) \) minus the sum for all reflex interior vertices. Basically, \( \Delta \theta(a, b) \) measures the number of radians swept counterclockwise in traversing \( PCCW(a, b) \) from \( a \) to \( b \).

We will now discuss an approach that partitions the boundary of \( P \) by choosing midpoints of edges of \( P \) as the partition points. In this manner every vertex of \( P \) is an interior vertex of some subchain of the partition. For \( a \) and \( b \) midpoints of edges of \( P \), we call \( PCCW(a, b) \) a reflex chain if \( \Delta \theta(a, b) < 0 \). A subchain \( PCCW(a, b) \) is a maximal reflex chain (mrc) if it is a reflex chain and:

- every strict subchain \( PCCW(c, d) \) of \( PCCW(a, b) \) has \( \Delta \theta(c, d) > \Delta \theta(a, b) \) (where \( c \) and \( d \) are midpoints of edges of \( P \));

- every superchain \( PCCW(c, d) \) of \( PCCW(a, b) \) has \( \Delta \theta(c, d) \geq \Delta \theta(a, b) \).

For a reflex-chain \( PCCW(m_i, m_j) \), with \( m_i \) and \( m_j \) the midpoints of edges \( e_i \) and \( e_j \) of \( P \), we call the vertices \( p_i \) and \( p_{j-1} \) the interior bounding vertices of \( PCCW(m_i, m_j) \), and \( p_{i-1} \) and \( p_j \) the exterior bounding vertices. In Figure 4, a maximal reflex chain \( PCCW(m_1, m_5) \) is shown with interior bounding vertices \( p_1 \) and \( p_4 \), and exterior bounding vertices \( p_0 \) and \( p_5 \).

**Lemma 1** The interior bounding vertices of a maximal reflex chain are reflex vertices, and the exterior bounding vertices are convex vertices.

**Proof.** If we have a chain violating the above, we can either add or delete an edge to obtain a superchain or subchain with a lesser \( \Delta \theta \)-value. ☑

**Lemma 2** Maximal reflex chains do not intersect.

**Proof.** Two maximal reflex chains cannot be nested, by the definition. If two maximal reflex chains intersect but are not nested, we have \( PCCW(m_i, m_j) \) and \( PCCW(m_k, m_l) \), where \( i < k \leq j < l \). If \( k = j \),
Figure 3: More classes of Polygons.

Figure 4: A maximal reflex chain.
neither reflex chain is maximal. Otherwise, we have

$$\Delta \theta(m_i, m_j) = \Delta \theta(m_i, m_k) + \Delta \theta(m_k, m_j).$$

Since \( P_{CCW}(m_i, m_j) \) is an mrc, \( \Delta \theta(m_i, m_j) < \Delta \theta(m_k, m_j) \), which implies \( \Delta \theta(m_i, m_k) < 0 \). This implies

$$\Delta \theta(m_i, m_k) = \Delta \theta(m_i, m_k) + \Delta \theta(m_k, m_i) < \Delta \theta(m_k, m_i),$$

contradicting the fact that \( P_{CCW}(m_k, m_i) \) is a maximal reflex chain.  

**Lemma 3** Every reflex vertex belongs to a unique maximal reflex chain.

**Proof.** Every reflex vertex belongs to at most one maximal reflex chain, since maximal reflex chains do not intersect.

Every reflex vertex \( p \) belongs to the reflex chain consisting only of itself, and there are a finite number of subchains of \( P \) containing \( p \), so the minimum \( \Delta \theta \)-value of a subchain containing \( p \) exists and is negative. If two minimum-\( \Delta \theta \)-value subchains containing \( p \) are not nested, their intersection is also a minimum-\( \Delta \theta \)-value subchain; otherwise we have a contradiction similar to the one in the previous proof. Therefore the shortest minimum-\( \Delta \theta \)-value subchain containing \( p \) is a subchain of every minimum-\( \Delta \theta \)-value subchain, and is the unique mrc containing \( p \).

The above lemmas establish that the entire boundary of \( P \) can be divided into alternating pieces of maximal reflex chains and convex chains, where the partition points are midpoints of edges of \( P \). A maximal reflex chain may contain convex vertices, but a convex chain contains no reflex vertices. Note that for a maximal reflex chain \( P_{CCW}(m_i, m_j) \), \( sw(m_i, m_j) \) is an arc of \( -\Delta \theta(m_i, m_j) \) radians if \( \Delta \theta(m_i, m_j) \geq -2\pi \), and \( sw(m_i, m_j) \) is the entire unit circle if \( \Delta \theta(m_i, m_j) < -2\pi \).

In trying to show that a polygon \( P \) is weakly-monotone in a certain direction, we face the task of choosing the splitting vertices, \( s \) and \( t \). We need the following lemma.

**Lemma 4** Given a simple polygon \( P \), if \( P \) is weakly-monotone in direction \( \theta \), it is weakly-monotone in \( \theta \) for splitting vertices \( s \) and \( t \) that do not lie in maximal reflex chains.

**Proof.** Let \( P_{CCW}(m_i, m_j) \) be a maximal reflex chain, where \( m_i \) and \( m_j \) are the midpoints of edges \( e_i \) and \( e_j \) as shown in Figure 5. If \( \Delta \theta(m_i, m_j) < -2\pi \), then \( \Delta \theta(m_j, m_i) > 4\pi \), and regardless of the choice of \( s \) and \( t \), either \( P_{CW}(s, t) \) or \( P_{CCW}(s, t) \) must sweep the entire unit circle of directions. Therefore we assume that \( \Delta \theta(m_i, m_j) \geq -2\pi \).

Without loss of generality, we assume that \( P \) is oriented so that \( \phi_i \in (\pi, 2\pi] \) and \( \phi_j \in [0, \pi) \). Since \( \Delta \theta(m_i, m_j) = \phi_j - \phi_i \), and a total of \( 2\pi \) radians are turned in traversing \( P \), we have \( \Delta \theta(m_j, m_i) = 2\pi + \phi_i - \phi_j \), where \( 2\pi < \Delta \theta(m_j, m_i) \leq 4\pi \). Suppose we select a vertex \( s \in P_{CCW}(m_i, m_j) \) as a splitting point. We cannot choose another point of \( P_{CCW}(m_i, m_j) \) as the other splitting point \( t \), since this implies that either \( P_{CW}(s, t) \) or \( P_{CCW}(s, t) \) contains all of \( P_{CCW}(m_i, m_j) \) and therefore sweeps through all directions. We therefore try to place \( t \) on \( P_{CW}(m_j, m_i) \).

We choose a splitting point \( t = p_k \) on the chain \( P_{CCW}(m_j, m_i) \). As illustrated in Figure 5, \( sw(P_{CCW}(m_j, t)) \cup sw(P_{CCW}(m_i, t)) \supseteq (\phi_j, \phi_i + \pi) \ (mod 2\pi) \). (If \( \phi_i - \pi > \phi_j \), then we mean to say that \( P_{CCW}(m_i, t) \) and \( P_{CW}(m_i, t) \) sweep the entire unit circle of directions.)

Suppose we replace the splitting point \( s \in P_{CCW}(m_i, m_j) \) with \( p_{i-1} \). We have \( \overline{sw}(P_{CCW}(p_{i-1}, t)) = \overline{sw}(P_{CCW}(p_{i-1}, m_j)) \cup \overline{sw}(P_{CCW}(m_i, t)) \). Because \( P_{CCW}(m_i, m_j) \) is a reflex chain, \( 

4

This gives \( sw(P_{CCW}(p_{i-1}, m_j)) = (\phi_j, \phi_i) \subseteq (\phi_j, \phi_i + \pi) \subseteq sw(P_{CCW}(s, t)) \cup sw(P_{CW}(s, t)) \). Therefore

\[
sw(P_{CCW}(p_{i-1}, t)) \cup sw(P_{CW}(p_{i-1}, t)) \subseteq sw(P_{CCW}(s, t)) \cup sw(P_{CW}(s, t)),
\]

so we can choose the splitting points outside of \( P_{CCW}(m_i, m_j) \) without losing directions of weak-monotonicity.

We will say that a maximal reflex chain \( P_{CCW}(m_i, m_j) \) double-sweeps a pair of opposite directions \( \theta, \theta + \pi \) if \( \theta, \theta + \pi \in sw(m_i, m_j) \). Note that if the orientation of a mrc is reversed, the double-swept pairs are unchanged. The directions double-swept by the mrc \( P_{CCW}(m_1, m_3) \) in Figure 4 are shaded on the pictured unit circle. We can characterize the set of directions in which a polygon is weakly-monotone by the pairs that are double-swept.

Theorem 5

A simple polygon \( P \) is weakly-monotone in the pair of directions \( \theta, \theta + \pi \) if and only if \( \theta \) and \( \theta + \pi \) are not double-swept by any maximal reflex chain.

Proof. Suppose \( \theta \) and \( \theta + \pi \) are double-swept by a maximal reflex chain. Since we choose the splitting points outside of the mrc, one of the subchains contains the mrc, and therefore sweeps \( \theta \) and \( \theta + \pi \), regardless of orientation. Thus, \( P \) is not weakly-monotone in these directions.

Suppose no maximal reflex chain double-sweeps \( \theta \) and \( \theta + \pi \). Consider all vertices of \( P \) that admit tangencies in direction \( \theta \) or \( \theta + \pi \). Assign to each tangency either \( \theta \) or \( \theta + \pi \) by traversing \( P \) counterclockwise and assigning the direction encountered at that vertex. (If an edge \( e_i = \overrightarrow{P_{i-1}P_i} \) faces in direction \( \theta \) or \( \theta + \pi \), consider \( p_{i-1} \) and \( P_i \) tangencies if \( p_{i-1} \) and \( P_i \) are both reflex or both convex.) \( P \) can be split into two subchains such that one subchain contains all \( \theta \) tangencies and the other all \( \theta + \pi \) tangencies.

If this were not true, we would have vertices \( p_i, p_j, p_k \), and \( p_i, i < j < k < l \), where \( p_i \) and \( p_k \) are \( \theta \) tangencies and \( p_j \) and \( p_l \) are \( \theta + \pi \) tangencies. Define \( \Delta \tilde{\theta}(p_i, p_j) = \Delta \theta(p_i, p_j) + m(\theta, \phi_{i+1}) + m(\phi_j, \theta + \pi) \), where \( m(\theta_1, \theta_2) \in (-\pi, \pi) \) such that \( \theta_2 - \theta_1 \equiv m(\theta_1, \theta_2) \pmod{2\pi} \) (see Figure 6). If we define \( \Delta \tilde{\theta} \) similarly for the other three pairs, then \( \Delta \tilde{\theta}(\cdot, \cdot) \equiv \pi \pmod{2\pi} \) for each pair, and \( \sum \Delta \tilde{\theta}(\cdot, \cdot) = 2\pi \). At least one pair, say \( P_{CCW}(p_i, p_j) \), has \( \Delta \tilde{\theta}(p_i, p_j) < 0 \). By making each of \( p_i \) and \( p_j \) either an interior or exterior bounding vertex (depending on whether the vertex is reflex or convex), we obtain a reflex chain that double-sweeps \( \theta \) and \( \theta + \pi \), a contradiction.

If the splitting points are chosen so that the \( \theta \) and \( \theta + \pi \) tangencies are in separate subchains, \( P \) is seen to be weakly-monotone in \( \theta, \theta + \pi \).

Corollary 6

A polygon monotone in any direction is weakly-monotone in every direction.

Corollary 7

A star-shaped polygon is weakly-monotone in every direction.

Theorem 5 provides us with a strategy for determining the set of weakly-monotone directions: find all maximal reflex chains, and for each mrc eliminate the appropriate pair of opposite arcs of directions. As illustrated in Figure 7, a polygon \( P \) can have \( O(n) \) pairs of arcs of weakly-monotone directions. We now present the algorithm \textbf{Detect}, which for input polygon \( P \) returns all directions of weak-monotonicity.

Algorithm \textbf{Detect}.

We define a \textit{turning function}, \( \Theta_P \), whose domain is the sequence of vertices encountered while twice traversing \( P \) counterclockwise, from the starting vertex \( p_0 \). (We distinguish between a vertex \( p_i \) encountered on the first traversal of \( P \) and the copy of that vertex, \( p_{i+n} \), encountered on the second traversal.)
Figure 5: Proof of Lemma 4.

Figure 6: Proof of Lemma 5.
We define $\Theta_P(p_0) = \phi_0$ and $\Theta_P(p_i) = \Delta\theta(a, p_i) + \Theta_P(p_0)$, for $i \geq 1$, where $a \in e_0$. We call the turning function the current direction. We also define the front direction, $f_P(p_i) = \max_{j=0,\ldots,i} \Theta_P(p_j)$.

The algorithm proceeds as follows. Beginning at $p_0$, traverse $P$ twice, updating $\Theta_P$ and $f_P$ at the vertices. Whenever $\Theta_P(p_i) \neq f_P(p_i)$ for the current vertex $p_i$, we are in a reflex chain, and if $\Theta_P(p_i) + \pi < f_P(p_i)$, then the reflex chain double-sweeps some directions. If we encounter a vertex $p_j$ such that $\Theta_P(p_{j-1}) = f_P(p_{j-1})$ but $\Theta_P(p_j) < f_P(p_j)$ ($= \Theta_P(p_{j-1})$), we store $\phi_{j-1}$ (refer to Figure 8).

Upon encountering vertex $p_k$ such that $\Theta_P(p_k) + \pi < f_P(p_k)$ ($= \Theta_P(p_{j-1})$), we eliminate the open intervals ($\phi_k + \pi, \phi_{j-1}$) and ($\phi_k, \phi_{j-1} - \pi$) (taken modulo $2\pi$) as possible directions of weak monotonicity. Until $\Theta_P(p_i) > \Theta_P(p_{j-1})$ for the current vertex $p_i$, we retain the direction $\phi_{j-1}$, enlarging the eliminated intervals until reaching a mrc. When once again $\Theta_P(p_i) = f_P(p_i)$, we discard $\phi_{j-1}$, and thereby begin looking for the next mrc. We must traverse $P$ twice because the initial vertex $p_0$ could be in a mrc.

The algorithm outputs the set of weakly-monotone directions, which is simply those directions not eliminated by the traversals of $P$.

End of Detect.

The intervals of directions eliminated by Algorithm Detect are encountered in sorted angular order. This means that Detect outputs a sorted list of arcs of weakly-monotone directions, using only $O(n)$ time.

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 $\theta \in [\theta_i, \theta'_i]$. 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 $\theta, \theta + \pi$, we can query in $O(\log n)$ time whether this is a weakly-monotone pair, and if it is we also return a valid pair of splitting points. These query times are optimal in the sense that a polygon can have $O(n)$ pairs of opposite weakly-monotone cones, and each pair can require a distinct pair of splitting points (as in Figure 7).

Our definition of weak-monotonicity relaxes the traditional definition of monotonicity on a polygonal chain. The notion of a $\phi$-monotone chain, as introduced by [CRS], considers a continuum of restrictions on the sweep of the chain. A chain $c$ is $\phi$-monotone in direction $\theta$ if $sw(c) \subseteq (\theta - \phi/2, \theta + \phi/2)$. 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 [Ch] triangulates a general polygon in linear time, the following method for weakly-monotone polygons is considerably simpler. The algorithm actually produces the horizontal
Figure 7: A polygon with many pairs of arcs of weakly-monotone directions.

Figure 8: Algorithm Detect.
visibility map of the polygon. If a triangulation is desired, it can be obtained by the brief algorithm of [FM].

We are given a polygon $P$, splitting points $s$ and $t$, and a direction $\theta$ such that $P_{CW}(s,t)$ and $P_{CCW}(s,t)$ are weakly-monotone in direction $\theta$ (the previous section describes how to compute a triplet $s, t, \theta$ for a polygon $P$). Assume without loss of generality that $\theta = 0$. Let $S$ be the ray with root $s$ in direction $\pi$ and let $T$ be the ray with root $t$ in direction $0$. Note that $S$ and $T$ intersect $P_{CW}(s,t)$ and $P_{CCW}(s,t)$ only at $s$ and $t$, because of the weak-monotonicity of the two chains. Let $c_T = S \cup P_{CW}(s,t) \cup T$ and $c_B = S \cup P_{CCW}(s,t) \cup T$. Then $c_T$ and $c_B$ are infinite, simple, polygonal chains, weakly-monotone in $\theta = 0$. Let $C_T$ and $C_B$ represent the regions below $c_T$ and above $c_B$, respectively. We have $P = C_T \cap C_B$. Figure 9 shows the chain $c_T$ and the region $C_T$ corresponding to the polygon $P$ with splitting vertices $s$ and $t$.

If $p$ is a vertex of a chain $c$, define $p$.prev and $p$.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 $p$. In this way, $c_T$ and $c_B$ are infinite chains from $s$.prev to $t$.next, where $x_{s.prev} = -\infty$ and $x_{t.next} = +\infty$. A point $p$ of a chain $c$ is a peak if (1) $y_p > y_p$.prev, and (2) $y_p > y_p$.next, or $y_p = y_p$.next > $y_{p.next.next}$; $p$ is a valley of $c$ if it satisfies (1) and (2) with the inequalities reversed (see Figure 10).

For $c_T$, we construct a partial horizontal visibility map, $T(c_T)$, called the tree of pockets (see Figure 11). $T(c_T)$ is a binary tree, where each node is associated with a subregion of $C_T$ and a subchain of $c_T$. The root node is associated with the entire region $C_T$ and the entire chain $c_T$. The tree is recursively defined as follows. Consider a node $\nu$ of the tree. If the associated subchain, $c(\nu)$, has no valleys, then $\nu$ is a leaf. Otherwise, the node $\nu$ has children, and we let $w$ represent the valley of $c(\nu)$ with least $y$-coordinate. (Consider $s$ a valley if $y_s < y_{s.next}$, and $t$ a valley if $y_t < y_{t.prev}$.) Imagine “firing bullets” horizontally left and right from $w$, allowing each bullet to go until it hits $c_T$ (or allowing it to go until infinity, if necessary). If the left bullet stops at $w_l$, then the left child of the node is associated with the subchain of $c_T$ from $w_l$ to $w$, and with the subregion defined by its subchain plus $w_l\nu w$. Similarly, the right child is associated with the subchain from $w$ to $w_r$, the point where the bullet fired right from $w$ hits $c_T$. This procedure partitions all of $C_T$ into horizontal slabs.

The regions associated with the nodes of the tree are called pockets, and the horizontal segment of the bullet path that forms the lower border of a pocket is called its bottom. A pocket minus its children, i.e. a horizontal slab, is called a strict pocket. If $\nu$ is a pocket, let $\tilde{\nu}$ denote the corresponding strict pocket, and $\nu_l$ and $\nu_r$ the left and right endpoints of the bottom of $\nu$. Each node in the tree stores its bottom, and pointers to its parent and children. Note that the strict pockets are a partitioning of $C_T$ into polygons monotone in the vertical direction.

The procedure Pocket constructs the tree of pockets $T(c_T)$ onto the chain $c_T$. That is, each left (right) endpoint $\nu_l$ ($\nu_r$) of a pocket bottom is made a vertex of $c_T$, is marked as being a bottom endpoint, and has a pointer to the node $\nu$ of $T(c_T)$. Figure 12 shows $c_T$ with its tree of pockets; all vertices of $c_T$ are marked, both original and added, and the endpoints of bottoms are shown with pointers to the appropriate node of $T(c_T)$. With this construction, we can traverse $c_T$ and always know in which pocket we are. Note that a pocket bottom always has as one endpoint a valley and as the other a point of $c_T$ that is not an original vertex (assuming nondegeneracy). Clearly the number of vertices added to $c_T$ in constructing $T(c_T)$ is bounded by the number of original vertices, and procedure Pocket can be performed by a single linear-time traversal of $c_T$. 


Figure 9: The chain $c_T$ and the region $C_T$

Figure 10: Examples of valleys ($p$ and $q$) and peaks ($p$.next).

Figure 11: The tree of pockets
A variation of **Pocket** constructs \( T(c_B) \), for \( c_B \), where the pockets lie below the horizontal bullet paths, which are therefore called **tops** instead of bottoms. The triangulation algorithm consists of constructing \( T(c_T) \) and \( T(c_B) \) with **Pocket**, and then inputting \((c_T,T(c_T))\) and \((c_B,T(c_B))\) to procedure **Partition**. The procedure **Partition** produces a partitioning of \( P \) into polygons monotone in the vertical direction.

For the purposes of the following discussion, consider a pocket \( \nu \) not to contain \( \overline{FT} \), its bottom (top). The construction of **Partition** is based closely on the unique shortest \((s,t)\)-path in \( P \), which we call \( \pi \). Every peak of \( \pi \) is a vertex of \( \pi \), and is therefore a vertex \( z \) of \( P \). Specifically, \( z \) must be a reflex vertex of \( P \) and a peak of \( P \), implying that \( z \) is a peak of \( c_B \). Therefore every peak of \( \pi \) is a peak of \( c_B \), and similarly every valley of \( \pi \) is a valley of \( c_T \). A partial converse holds:

**Lemma 8** A point \( z \) is a peak of \( \pi \) if and only if \( z \) is the highest point of \( c_B \) in \( \nu \), where \( \nu \) is the strict pocket of \( c_T \) containing \( z \).

**Proof.** Suppose \( z \) is a peak of \( \pi \). Then \( z \) is a peak of \( c_B \), and \( z \in \nu \) for some unique strict pocket \( \nu \) of \( c_T \). If \( \nu \) contains another peak of \( \pi \), \( z' \), then there exists a valley of \( \pi \), \( z'' \), lying between \( z \) and \( z' \). But since \( z'' \notin c_T \), this requires \( c_T \) to enter the interior of \( \nu \), a contradiction (see Figure 13(a)). Therefore \( z \) is the unique peak of \( \pi \) in \( \nu \), and is thus the highest point of \( c_B \) in \( \nu \).

If a peak \( z \in \nu \) is the highest point of \( c_B \) in \( \nu \), then \( P \setminus \nu \) consists of two or more connected components, with \( s \) and \( t \) in different components (see Figure 13(b)). The point \( z \) is therefore a peak of \( \pi \). □

A lemma analogous to the one above holds for pockets of \( c_B \) and valleys of \( \pi \).

The previous lemma motivates procedure **Partition**. If \( z \) is a peak (valley) of \( c_B \) \( (c_T) \), let \( z^* \) be the point of \( P \) encountered by a bullet shot due right from \( z \). Let \( z_1, \ldots, z_{m-1} \) be the horizontal tangencies (peaks and valleys) of \( \pi \), in order. **Partition** will construct the chords \( \overline{z_iz_{i+1}} \), for \( i = 1, \ldots, m - 1 \). Also, any bottom (top) of \( c_T \) \( (c_B) \) not intersected by \( c_B \) \( (c_T) \) is a partition chord. Figure 14 shows a weakly-monotone polygon \( P \) whose shortest \((s,t)\)-path \( \pi \) has three horizontal tangencies; the chords constructed by **Partition** are shown with dashed lines.

The procedure **Partition** simultaneously traverses \( c_T \) and \( c_B \) from \( s \) to \( t \), using pointers \( p \) and \( q \). At any moment, points \( p \in c_T \) and \( q \in c_B \) are visible and at the same \( y \)-coordinate, such that \( \pi \) contains exactly one point on \( \overline{pq} \). In this way, as \( p \) and \( q \) traverse \( c_T \) and \( c_B \) from \( s \) to \( t \), they trace a subpolygon of \( P \) containing \( \pi \). Specifically, every peak of \( \pi \) is encountered by \( q \) and every valley by \( p \). **Partition** alternates between two subprocedures, **Moving-Up** and **Moving-Down**, which correspond to the times when the \( y \)-coordinate of \( p \) and \( q \) is increasing and decreasing, respectively. We discuss **Moving-Up**, which identifies all peaks of \( \pi \). Upon calling **Moving-Up**, the procedure has identified \( s = z_0, \ldots, z_{i-1} \), for \( i \geq 1 \), where \( z_{i-1} \) is a valley of \( \pi \) (or \( s \)). **Moving-Up** will find \( z_i \), which either is a peak of \( \pi \) or is \( t \). At any moment, we have \( y_p = y_q = y_a \) and \( x_p < x_q < x_a \), where \( a \) is an auxiliary pointer. The point \( a \) is the point of \( c_T \) opposite \( p \); i.e. \( a \) and \( p \) are in the same strict pocket and \( y_p = y_a \). We also have \( CPT(c_B) \), the pocket of \( c_T \) \( (c_B) \) containing \( p \) \( (q) \). In the example of Figure 15, we have found \( z_1 \) and are looking for \( z_2 \); we begin with \( p = z_1 \), \( q = z_1^* \), and \( a = \nu_t \), where \( \nu \) is the pocket of \( c_T \) with \( \nu_t = z_1 \).

**Moving-Up** proceeds with \( q \) traversing forwards on \( c_B \), while \( p \) traverses forwards and \( a \) traverses backwards on \( c_T \). Since we update \( p \) and \( a \), we can mark each bottom of \( c_T \) intersected by \( q \). When \( q \) encounters a peak \( z \) of \( c_B \), \( q \) is the right endpoint of the top of \( CPT \). If \( b \) is the right endpoint of the top of the right sibling of \( CPT \), then \( z_a < z_b \) if and only if \( q \) is the highest point of \( c_B \) in \( CPT \), which,
Figure 12: $c_T$ and the tree of pockets.

Figure 13: Proof of Lemma $\Theta$. 

(a)

(b)
Figure 14: The partition of P.

Figure 15: Description of Moving-Up
by Lemma 8, is equivalent to \( q \) being a peak of \( \pi \). By traversing \( q \) forwards until encountering such a peak (or until encountering \( \ell \)), we will locate \( z_i \). When \( q = z_i \), we have \( a = z_i' \), so the segment \( z_i z_i' \) is added to the list of partition chords. Then \( p \) is set to \( a = z_i' \), and procedure \textbf{Moving-Down} is called.

In our example (Figure 15), \( q \) first encounters peak \( z' \), where \( x_{y'} < x_{z'} \), and later encounters \( z'' \), with \( x_{z''} < x_{z'''} \); therefore \( z_2 = z'' \), and \( z_2^* = a'' \). The weak-monotonicity of \( c_T \) and \( c_B \) implies that the total work performed by \textbf{Partition} is proportional to the size of \( (c_T, T(c_T)) \) and \( (c_B, T(c_B)) \).

The final partition of \( P \) consists of all segments \( z_i z_i^* \) constructed by \textbf{Partition}, and all unmarked tops and bottoms. An unmarked bottom (top) is any bottom (top) not crossed by \( q \) (\( p \)); these are exactly the bottoms (tops) not crossed by \( c_B \) (\( c_T \)). For \( i = 1, \ldots, m \), let \( P_i \) be the partition subpolygon with \( z_{i-1} \) and \( z_i \) on its boundary (see Figure 16). For each \( P_i \), the points \( p \) and \( q \) trace the portion of \( P_i \)'s boundary lying between \( y_{z_i-1} \) and \( y_{z_i} \). Since \( p \) and \( q \) move monotonically in the vertical direction during each call to \textbf{Moving-Up} or \textbf{Moving-Down}, it is not hard to see that each \( P_i \) is \( y \)-monotone. All other polygons of the partition are strict pockets of either \( c_T \) or \( c_B \), and are therefore \( y \)-monotone. No polygon of the partition contains non-horizontal edges that are not part of the boundary of \( P \); therefore, by computing the horizontal visibility map of each \( y \)-monotone polygon in the partition, we obtain the horizontal visibility map of \( P \). If a triangulation of \( P \) is desired, it can be obtained from the horizontal visibility map by the algorithm of [FM]. The result is a triangulation of \( P \) obtained by conceptually simple procedures that have \( O(n) \) time bounds with small constants.

**Theorem 9** The algorithm \textbf{Triangulate} triangulates an \( n \)-vertex polygon \( P \) that is weakly-monotone with respect to \( s, t, \) and \( \theta \) in \( O(n) \) time.

**Algorithm \textbf{Triangulate}(\( P; s, t, \theta \)).**

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

Call \textbf{Pocket}(\( c_T \)) and \textbf{Pocket}(\( c_B \)) to obtain \( T(c_T) \) and \( T(c_B) \), respectively.

Call \textbf{Partition}(\( (c_T, T(c_T)), (c_B, T(c_B)) \)).

End of \textbf{Triangulate}.

**Procedure \textbf{Pocket}(\( c_T \)).**

The input is \( c_T \), an infinite chain weakly-monotone in direction \( \theta = 0 \). We will traverse the chain from \( s.\text{prev} \) to \( t.\text{next} \), keeping a pointer on the current point, \( p \), and one on the current pocket, \( CP \). For each pocket, we construct left and right endpoints of the bottom, a parent pointer, and possibly left and right child pointers. There are two starting cases. After the appropriate starting case calls subprocedure \textbf{Down}, the procedure continuously calls \textbf{Down} until termination.

The point \( s.\text{prev} \) is the first point of the infinite chain, so \( x_{s.\text{prev}} = -\infty \). The ray from \( s \) through \( s.\text{prev} \) is horizontal, so \( x_s = x_{s.\text{prev}} \). The starting cases are (refer to Figure 17):

\( y_s < y_s.\text{next} \): We initialize a pocket and make it \( CP \). Make \( s \) its left endpoint. Create a left sibling of the \( CP \), and make \( s \) both the left and right endpoints (this pocket is empty). Create a parent pocket for \( CP \) and its sibling. Let \( p \) be the first peak in \( c_T \). Call \textbf{Down}(\( p \)) (see Figure 17(a)).

\( y_s > y_s.\text{next} \): We traverse to the first valley, and call it \( q \). Create a parent pocket with 2 children. The left child has bottom \((-\infty, y_q), q \), and the right child has left endpoint \( q \). The right child is \( CP \). Let \( p \) be the first peak after \( q \). Call \textbf{Down}(\( p \)) (see Figure 17(b)).
Figure 16: The subpolygon $P_i$.

Figure 17: Starting cases of Pocket
End of Pocket.

Procedure Down\((p, CP)\).

The point \(p\) is the highest point of a pocket with no children. Traversing the chain backwards from \(p\), keeping a pointer on the current point \(l\), gives the left side of this down-pocket. Similarly, traversing forwards from \(p\), keeping a pointer on the current point \(r\), gives the right side. Perform a double-traversal down the two sides in leapfrog fashion to keep the two traversal points at the same approximate \(y\)-coordinate. Recall that \(s(t)\) is considered a valley if \(y_s < y_{s,\text{next}}\) \((y_t < y_{t,\text{prev}})\). Two cases can occur.

Case (1) The left side hits a valley, \(l\). Refer to Figure 18.

Denote by \(r'\) the point near \(r\) on the right side such that \(y_{r'} = y_r\). Make \(r'\) the right endpoint of \(CP\). \(CP\) is a right sibling. If \(l \neq s\) and the left sibling of \(CP\) has a left endpoint, \(l'\), not at infinity (Figure 18(a)), then set \(l \leftarrow l'\), set \(CP\) to the parent, and continue the double traversal.

If \(l = s\) (Figure 18(b)), or if \(l \neq s\) (Figure 18(c)) and the left sibling has a left endpoint at infinity, then let \(r'\) be the point on the right side near \(r\) such that \(y_l = y_{r'}\), and let \(r'\) be the right endpoint of \(CP\). Move \(CP\) to the parent, move \(r\) to the next valley, and let \(CP\) have bottom \(((-\infty, y_r), r)\). Now create a parent and right sibling for \(CP\), and make \(r\) the left endpoint of the right sibling. Make the right sibling the new \(CP\). Let \(p\) be the first peak after \(r\). Call Down\((p, CP)\).

Case (2) The right side hits a valley, \(r\). Refer to Figure 19.

Create left and right children for \(CP\). If \(CP\) already has children, then they become the children of the newly-created left child.

Make \(r\) the right endpoint of the left child and the left endpoint of the right child. Let \(l'\) be the point on the left side near \(l\) such that \(y_l = y_{r'}\). Make \(l'\) the left endpoint of the left child.

If \(r \neq t\) (Figure 19(a)), then let \(CP\) be the right child, let \(p\) be the first valley after \(r\), and call Down\((p, CP)\).

If \(r = t\) (Figure 19(b)), then \(r = t\) is the right endpoint of the right child (i.e. the right child is empty). Exit Down and Pocket.

End of Down.

Procedure Partition\(((c_T, T(c_T)), (c_B, T(c_B)))\).

The input is \((c_T, T(c_T))\) and \((c_B, T(c_B))\), two infinite chains monotone in \(\theta = 0\), with their corresponding trees of pockets. The chains intersect only at the infinite rays \(S\) and \(T\), and the region between \(s\) and \(t\) defines the polygon \(P\). The procedure traverses \(c_T\) and \(c_B\) from \(s\) to \(t\) using pointers \(p\) and \(q\), respectively. It also uses two auxiliary pointers, \(a\) and \(b\). The pointer \(CP_T\) points to the node of the strict pocket of \(c_T\) that contains \(p\). Similarly, \(CP_B\) is the strict pocket containing \(q\).

The algorithm will find in order the points of horizontal tangency on the shortest \((s, t)\)-path in \(P\): \(z_1, \ldots, z_{m-1}\). This gives a list \(s = z_0, z_1, \ldots, z_{m-1}, z_m = t\). For each \(z_i, i = 1, \ldots, m - 1\), a horizontal segment in \(P\) is drawn from \(z_i\) to a point \(z_i'\) on the other path, with \(z_i\) the left endpoint of the segment.

As a final step, all bottoms (tops) of \(c_T\) (\(c_B\)) not intersected by \(c_B\) (\(c_T\)) are added to the partition.

Let \(p \leftarrow s, q \leftarrow s\). There are three initial cases (refer to Figure 20):

1. \(y_{p,\text{next}}, y_{q,\text{next}} > y_t\). Then \(s\) is the left endpoint of a pocket of \(c_T\). Initialize \(CP_T\) and \(CP_B\) accordingly, and call the subprocedure Moving-Up, where \(a\) is initialized to be the right endpoint of \(CP_T\) (see Figure 20(a)).
Figure 18: Procedure Down: Case (1)

Figure 19: Procedure Down: Case (2)
2. \(y_p .next < y_q\). Then \(s\) is the left endpoint of a pocket of \(c_B\). Initialize \(CP_T\) and \(CP_B\) accordingly, and call the subprocedure \textbf{Moving-Down}, where \(b\) is initialized to be the right endpoint of \(CP_B\) (see Figure 20(b)).

3. \(y_p .next > y_s, y_q .next < y_s\). Then \(s\) is the left endpoint of non-empty pockets \(\nu^T\) and \(\nu^B\) of \(c_T\) and \(c_B\), respectively. If \(\nu^B\) has a shorter lid than \(\nu^T\) (i.e. if \(x_{\nu^B} < x_{\nu^T}\)), then call \textbf{Moving-Up}, with \(p = s, q = \nu^B, a = \nu^T\) (see Figure 20(c)). Otherwise call \textbf{Moving-Down}, with \(q = s, p = \nu^T, b = \nu^B\).

Note that \(y_p .next < y_s, y_q .next \geq y_s\) cannot occur.

End of \textbf{Partition}.

Procedure \textbf{Moving-Up}.

We start with \(z_{i-1}\). We will find \(z_i\), and draw a segment \(z_i z_i'\).

We have \(p\) lying to the left of \(q\), where \(p\) is the left endpoint of \(CP_T\). We will use another pointer, \(a\), which is initialized to the right endpoint of \(CP_T\).

The general step of \textbf{Moving-Up} is to consider \(q .next\):

\textbf{Case (1)} \(y_q < y_q .next\) or \(y_q = y_q .next < y(q .next) .next\).

We update \(p\) and \(a\) (refer to Figure 21). Traverse forward from \(p\) up to the left side of \(CP_T\) until reaching a point \(p'\) above \(q .next\). Similarly, traverse backwards from \(a\), up to the right side of \(CP_T\), until reaching a point \(a'\) above \(q\). If during the traversal up the left side, \(p\) hits the left endpoint of the left child of \(CP_T\) (equivalently, if during the traversal up the right side, \(a\) hits the right endpoint of the right endpoint of the right child), then determine whether \((q)(q .next)\) intersects the bottom of the left or of the right child and update \(CP_T\) and \(p\) or \(a\) accordingly. Once we reach points \(p'\) and \(a'\), we update \(p \leftarrow p'\) and \(a \leftarrow a'\).

When the traversals end, we have new points \(p\) and \(a\), such that \(y_{q .next} \leq y_p, y_a\). Set \(q \leftarrow q .next\), update \(CP_B\) if necessary, and return to the general step of \textbf{Moving-Up}.

\textbf{Case (2)} \(y_q > y_q .next\) or \(y_q = y_q .next > y(q .next) .next\).

Then \(q\) is a peak. The point \(q\) is the right endpoint of \(CP_B\), and the left endpoint of the sibling of \(CP_B\). Let \(b\) be the right endpoint of the sibling (if \(b\) is at infinity, then replace segment \(q b\) below with ray \(q b\)).

Determine whether \(q b\) intersects \((a)(a .next)\). If \textbf{NO} (Figure 22(a)), then \(q \leftarrow b\); update \(CP_B\) to be the parent, and return to the general step of \textbf{Moving-Up}.

If \textbf{YES} (Figure 22(b)), then call the intersection point \(d\). Set \(z_i \leftarrow q, z_i' \leftarrow d\), and add the segment \(z_i z_i'\) to the partition of \(P\). Switch \(CP_B\) to the sibling. Call procedure \textbf{Moving-Down}, initializing \(p \leftarrow d, q \leftarrow q, b \leftarrow b\).

End of \textbf{Moving-Up}.

Procedure \textbf{Moving-Down}.

We have \(q\) lying to the left of \(p\), where \(q\) is the left endpoint of \(CP_B\). We will use another pointer, \(b\), initialized to be the right endpoint of \(CP_B\).

The general step of \textbf{Moving-Down} is to consider \(p .next\):

\textbf{Case (1)} \(y_p > y_p .next\) or \(y_p = y_p .next > y(p .next) .next\).
Figure 20: Starting cases of Partition
Figure 21: Procedure Moving-Up: Case (1)

Figure 22: Procedure Moving-Up: Case (2)
Then scan to update \( q \) and \( b \), updating \( CP_B \) if necessary, in a manner similar to Case (1) of Moving-Up. Set \( p \leftarrow p_.next \), update \( CP_T \) if necessary, and return to the general step of Moving-Down.

**Case (2)** \( y_p < y_.next \) or \( y_p = y_.next < y(p_.next).next \).

Then \( p \) is the right endpoint of \( CP_T \), and also the left endpoint of the sibling of \( CP_T \). Let \( a \) be the right endpoint of the sibling of \( CP_T \) (if \( a \) is at infinity, then replace segment \( \overline{pa} \) below with ray \( \overrightarrow{pa} \)). Determine whether \( \overline{pa} \) intersects \( \overline{b(a).next} \). If NO, then update \( CP_T \) to the parent, set \( p \leftarrow a \), and return to the general step of Moving-Down.

If YES, then call the intersection point \( d \). Switch \( CP_T \) to the sibling, and call procedure Moving-Up, initializing \( q \leftarrow d, p \leftarrow p, a \leftarrow a \).

End of Moving-Down.

5 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 theorems and algorithms of this paper extend to well-behaved curved chains; for example, splines and [DS].

Much attention has been given recently to parallel algorithms in computational geometry. Triangulation exhibits a gap between the best known sequential and parallel algorithms. Whereas there exists an optimal sequential triangulation algorithm, no parallel algorithm attains optimality for general polygons. We mention here that 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 computation model, and the weakly-monotone polygon triangulation problem can be solved optimally in the CREW PRAM model [HeMa]. Therefore, in contrast to general polygons, there is no performance gap between sequential and parallel triangulation for weakly-monotone polygons (even without prior knowledge of a polygon's weak-monotonicity).

We close with an open question. Chazelle and Incerpi [CI] 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 [CI] could be used as a linear-time algorithm in place of the triangulation algorithm of this paper; however, the algorithm of this paper is simpler, 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 [CI] would have an accompanying detection algorithm, just as the detection algorithm of this paper accompanies the triangulation algorithm.
References


[HeMa] P.J. Hefferman and A. Maheshwari, Manuscript.


