Triangulating and guarding realistic polygons

Citation for published version (APA):
https://doi.org/10.1016/j.comgeo.2013.03.005

DOI:
10.1016/j.comgeo.2013.03.005

Document status and date:
Published: 01/01/2014

Document Version:
Accepted manuscript including changes made at the peer-review stage

Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

---

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.

Download date: 03. Dec. 2019
Triangulating and Guarding Realistic Polygons

G. Aloupis∗  P. Bose†  V. Dujmovic‡  C. Gray§  S. Langerman¶  B. Speckmann§

Abstract

We propose a new model of realistic input: \emph{k-guardable} objects. An object is \emph{k-guardable} if its boundary can be seen by \emph{k} guards. We show that \emph{k}-guardable polygons generalize two previously identified classes of realistic input. Following this, we give two simple algorithms for triangulating \emph{k}-guardable polygons. One algorithm requires the guards as input while the other does not. Both take linear time assuming that \emph{k} is constant and both are easily implementable.

1 Introduction

Algorithms and data structures in computational geometry often display their worst-case performance on intricate input configurations that seem artificial or unrealistic when considered in the context of the original problem. Indeed, in “practical” situations, many algorithms and data structures—binary space partitions are a notable example—tend to perform much better than predicted by the theoretical bounds. An attempt to understand this disparity and to quantify “practical” or “normal” with respect to input are the so-called realistic input models [7]. Here one places certain restrictions on the shape and/or distribution of the input objects so that most unusual hypothetical worst-case examples are excluded. Analyzing the algorithm or data structure in question under these input assumptions tends to lead to performance bounds that are much closer to actually observed behavior.

Many realistic input models have been proposed. These include \emph{low-density} scenes [7], where it is assumed that the number of “large” objects intersecting a “small” volume is bounded, and \emph{local} polyhedra [11], where it is assumed that the ratio of lengths between edges coming from a single vertex is limited by a constant. One of the most widely studied realistic input models assumes that input objects are \emph{fat}, that is, they are not arbitrarily long and skinny. There are several ways to characterize fat objects—see Section 3 for formal definitions.

In this paper, we propose a new measure to define realistic input: the number of guards that are required to see the boundary of an input object. We use the term \emph{k-guardable} to denote any object whose boundary can be seen by \emph{k} guards. A rigorous definition of what it means for a guard to see can be found in the next section. In Section 3, we discuss the connection between \emph{k}-guardable polygons and other measures of realistic input. In particular, we show that \emph{(α, β)-covered} polygons are \emph{O(1)}-guardable. \emph{(α, β)}-covered polygons model the intuitive notion of fatness for non-convex input: an \emph{(α, β)}-covered polygon \(P\) has the property that every point \(p \in \partial P\) admits a triangle inside \(P\) with minimum angle at least \(α\) and minimum edge length at least \(β \cdot \text{diam}(P)\) for given constants \(α\) and \(β\).

In Section 4, we describe two algorithms for triangulating \emph{k}-guardable polygons. Our algorithms, which were designed with simplicity in mind, take \(O(kn)\) time, that is, linear time.

∗Carleton University & Université Libre de Bruxelles, greg@cg.scs.carleton.ca.
†Carleton University, jit@cg.scs.carleton.ca.
‡McGill University, vida@cs.mcgill.ca.
§TU Eindhoven, {cgray, speckman}@win.tue.nl. C.G. is supported by the Netherlands’ Organisation for Scientific Research (NWO) under project no. 639.023.301. B.S. is supported by the Netherlands’ Organisation for Scientific Research (NWO) under project no. 639.022.707.
¶Chercheur Qualifié du FNRS, Université Libre de Bruxelles, Belgique, stefan.langerman@ulb.ac.be.
assuming that $k$ is constant. If the *link diameter*—see the next section for a formal definition—of the input polygon is $d$, then the algorithm described in Section 4.1 takes $O(dn)$ time—a slightly stronger result. This algorithm uses the linear-time computation of an edge-visibility polygon as a subroutine. In Section 4.2, we describe an algorithm that uses only scans of the input polygon and stacks, but requires the actual guards as input.

**Related work.** In 1991 Chazelle [4] presented a linear time algorithm to triangulate any simple polygon. However, after all these years it is still a major open problem in computational geometry to design an implementable linear-time algorithm for triangulation. There are several implementable algorithms which triangulate polygons in near-linear time. For example, Kirkpatrick et al. [16] describe an $O(n \log \log n)$ algorithm and Seidel [22] presents a randomized algorithm which runs in $O(n \log^* n)$ expected time, and Amato et al. [2] present another randomized algorithm that runs in $O(n)$ expected time that is based in large part on Chazelle’s algorithm. We contend that our algorithm is conceptually simpler than the $O(n \log \log n)$ algorithm and that it has a slight advantage over the Seidel algorithm and the algorithm of Amato et al. because it is deterministic. It is also interesting to note that the Seidel algorithm requires $\Omega(n \log n)$ random bits, which makes it theoretically undesirable in some models of computation.

Relationships between shape complexity and the number of steps necessary to triangulate polygons have been investigated before. For example, it has been shown that monotone polygons [23], star-shaped polygons [21], and edge-visible polygons [24] can all be triangulated in linear time by fairly simple algorithms. Other linear algorithms have been given, under the assumption that the number of reflex vertices [13] or the sinuosity [5] is bounded by a constant.

The subject of guarding is also well-studied. The book by O’Rourke [19] gives a good overview of the early results. Perhaps most relevant are the hardness results. Determining whether the entire interior of a polygon $P$ can be seen by at most $k$ guards is NP-complete [20] and NP-hard to approximate within a $(1 + \varepsilon)$ factor [9]. The same results hold when only the boundary of $P$ needs to be guarded: it is NP-complete to determine whether the boundary of $P$ can be seen by at most $k$ guards [17] and this problem is also APX-hard [9]. Finally, it has recently been shown [1] that $O(k)$ guards suffice to see the interior of a polygon if $k$ guards suffice to see the boundary.

Several algorithms and data structures exist for collections of realistic objects. For example, the problem of ray-shooting in an environment consisting of fat objects has been studied extensively [3, 6, 14]. However, there are few results concerning individual realistic objects. We hope that our results on triangulating realistic polygons will encourage further research in this direction.

### 2 Tools and definitions

Throughout this paper let $P$ be a simple polygon with $n$ vertices. We assume that $P$ has no vertical edges. If $P$ has vertical edges, it is easy to rotate it by a small amount until the vertical edges are eliminated.

We denote the interior of $P$ by $\text{int}(P)$, the boundary of $P$ by $\partial P$, and the *diameter* of $P$ by $\text{diam}(P)$. The boundary is considered part of the polygon, that is, $P = \text{int}(P) \cup \partial P$. We say that a point $p$ is in $P$ if $p \in \text{int}(P) \cup \partial P$.

The segment or edge between two points $p$ and $q$ is denoted by $pq$. The same notation implies the direction from $p$ to $q$ if necessary. Two points $p$ and $q$ in $P$ see each other if $\overline{pq} \cap P = \overline{pq}$. If $p$ and $q$ see each other, then we also say that $p$ is visible from $q$ and vice versa. We call a polygon $P$ *$k$-guardable* if there exists a set $G$ of $k$ points in $P$ called *guards* such that every point $p \in \partial P$ can see at least one point in $G$. 


A star-shaped polygon is defined as a polygon that contains a set of points—the kernel—each of which can see the entire polygon. If there exists an edge $pq \subset \partial P$ such that each point in $P$ sees some point on $pq$, then $P$ is weakly edge-visible. The visibility polygon of a point $p \in P$ with respect to $P$, denoted by $VP(p, P)$, is the set of points in $P$ that are visible from $p$—see Figure 1. Visibility polygons are star-shaped and have complexity $O(n)$. We do not describe the procedure for computing the visibility polygon in this paper. The following lemma summarizes the time complexity of computing a visibility polygon.

**Lemma 1 (El Gindy and Avis [10])** $VP(p, P)$ can be computed in $O(n)$ time.

Let $Q$ be a subpolygon of $P$, where all vertices of $Q$ coincide with vertices of $P$, then we call $Q$ a pure subpolygon. If $\partial P$ intersects an edge $w$ of $\partial Q$ only at $w$’s endpoints, then $w$ is called a window of $Q$. Any window $w$ separates $P$ into two subpolygons. The one not containing $Q$ is the pocket of $w$ with respect to $Q$ (see Figure 1). Any vertex added to the polygon (such as the endpoint of a window) is called a Steiner point.

The algorithm by El Gindy and Avis [10], while not trivial, is fairly simple. It involves a single scan of the polygon and a stack. See O’Rourke’s book [19] for a good summary.

A concept related to visibility in a polygon $P$ is the link distance, which we denote by $LD(p, q)$ for two points $p$ and $q$ in $P$. Consider a polygonal path $\pi$ that connects $p$ and $q$ while staying in $\text{int}(P)$. We say that $\pi$ is a minimum link path if it has the fewest number of segments (links) among all such paths. The link distance of $p$ and $q$ is the number of links of a minimum link path between $p$ and $q$. We define the link diameter $d$ of $P$ to be $\max_{p,q \in P} LD(p, q)$. The link diameter of a polygon may be much less than the number of guards required to see its boundary, and is upper bounded by twice the number of guards required to see the boundary. The former fact can be seen in examples based on the “comb” polygon, that have a small link diameter but need a linear number of guards—see Figure 2.

![Figure 1: The visibility polygon $VP(p, P)$ is shaded. $P_w$ is the pocket of $w$ with respect to $VP(p, P)$.

![Figure 2: A polygon with low link diameter that needs $O(n)$ guards.](image)
3 Guarding realistic polygons

In this section we discuss several realistic input models for polygons and their connection to \( k \)-guardable polygons. We first consider the so-called \( \varepsilon \)-good polygons introduced by Valtr [26]. An \( \varepsilon \)-good polygon \( P \) has the property that any point \( p \in P \) can see a constant fraction \( \varepsilon \) of the area of \( P \). Valtr showed that these polygons can be guarded by a constant number of guards. Hence \( \varepsilon \)-good polygons fall naturally in the class of \( k \)-guardable polygons. Kirkpatrick [15] achieved similar results for a related class of polygons, namely polygons \( P \) where any point \( p \in P \) can see a constant fraction \( \varepsilon \) of the length of the boundary of \( P \). These polygons can be guarded by a constant number of guards as well, and hence are \( O(1) \)-guardable polygons.

We now turn our attention to fat polygons. One well-studied variant of fat polygons are the so-called \( \beta \)-fat polygons [27]. A polygon \( P \) is \( \beta \)-fat if for every ball \( b \) whose center is inside \( P \) and which does not contain \( P \) completely, \( \text{volume}(b \cap P) \) is at least \( \beta \cdot \text{volume}(b) \). If \( P \) is convex, then \( \beta \)-fatness captures the intuition of “fat” polygons nicely. However, non-convex “comb” polygons with very thin spikes that are very close together also fall into the class of \( \beta \)-fat polygons. For this reason, Efrat introduced the so-called \((\alpha, \beta)\)-covered polygons [8]. A polygon \( P \) is \((\alpha, \beta)\)-covered if for each point \( p \) on \( \partial P \) there exists a triangle \( T(p) \), called a good triangle of \( p \), such that:

(i) \( p \) is a vertex of \( T(p) \),

(ii) \( T(p) \) is completely contained in \( P \),

(iii) each angle of \( T \) is at least \( \alpha \), and

(iv) the length of each edge of \( T(p) \) is at least \( \beta \cdot \text{diam}(P) \).

It is easy to show that the classes of \((\alpha, \beta)\)-covered polygons and \( \varepsilon \)-good polygons are not equal. Any convex polygon that is not fat is \( \varepsilon \)-good but not \((\alpha, \beta)\)-covered, and the polygon in Figure 3 is \((\alpha, \beta)\)-covered but not \( \varepsilon \)-good.

\[\text{Figure 3: A polygon } P \text{ that is } (\alpha, \beta)\text{-covered but not } \varepsilon\text{-good. By scaling the length of the edges, the central point of } P \text{ can be made to see an arbitrarily small fraction of the area of } P.\]

In the remainder of this section we prove that \((\alpha, \beta)\)-covered polygons can also be guarded by a constant number of guards and hence are \( O(1) \)-guardable polygons.

Let \( P \) be an \((\alpha, \beta)\)-covered polygon with diameter 1 and let \( T \) be a good triangle inside \( P \). One of the vertices of \( T \) must have an angle that is at most \( \pi/3 \). Call this vertex \( p \). We construct a circle \( C \) with radius \( \beta/2 \) and center \( p \) and place \( \lceil 4\pi/\alpha \rceil \) guards evenly spaced around the boundary of \( C \). We call this set of guards \( G_p \). Since the edges of \( T \) are at least \( \beta \) long and the angle at \( p \) is at least \( \alpha \), \( T \) must contain at least one guard from \( G_p \). The fact that the angle at \( p \) is at most \( \pi/3 \) implies that the edge of \( T \) across from \( p \) can not cross \( C \).

Let \( C' \) be the circle with radius \( \beta/4 \) centered at \( p \).

**Lemma 2** Let \( T \) be a good triangle with a vertex whose angle is at most \( \pi/3 \) inside the circle \( C' \). Then \( T \) contains at least one guard from \( G_p \).
Proof. Let $v$ be the vertex of $T$ that lies inside $C'$. Since $T$ is a good triangle, all of its edges have length at least $\beta$. Also, all of its angles are at least $\alpha$. Thus $\alpha$ is at most $\pi/3$, and no other vertex of $T$ is inside $C$.

Let $r$ be the ray that bisects the angle at $v$ and that is partially contained inside $T$. Assume that $T$ contains no guards from $G_p$. Then we can rigidly shift $T$ by moving $v$ along $r$ until it is on the boundary of $C'$, without introducing guards into $T$. Thus we can restrict to considering only good triangles that have a vertex on $C'$, and which do not intersect $C'$. We will show that such triangles must contain a guard, thus contradicting the assumption just made.

To maximize the angle at $v$ while avoiding the inclusion of a guard in $T$, it is best to position $r$ so that it bisects the segment connecting two consecutive guards from $G_p$. We now show that even in this configuration, pictured in Figure 4, there must be a guard from $G_p$ in $T$.

We arbitrarily choose two consecutive guards $g_1, g_2$, and denote the length of the segment $g_1g_2$ by $2\delta$. Hence $\tan(\alpha/4) = 2\delta/\beta$. Let the angle $\angle g_1vg_2$ be denoted by $2\theta$. We have $\tan \theta = 4\delta/\beta$. Therefore, $\tan \theta = 2 \tan(\alpha/4)$. Since $0 < \alpha/4 \leq \pi/12 < \pi/4$, we have $0 < 2 \tan(\alpha/4) < \tan(\alpha/2)$ (by double-angle identities for $\tan$). This implies that $\tan \theta < \tan(\alpha/2)$ and hence $2\theta < \alpha$. It follows that $T$ must contain the segment $g_1g_2$.

Any good triangle with a vertex on the boundary of $C'$ and a different choice of $r$ will also contain a guard. Thus the original good triangle examined could not have been empty. \hfill \Box

Lemma 2 almost directly provides a guarding set for $\partial P$.

**Theorem 3** Let $P$ be a simple $(\alpha, \beta)$-covered polygon. The boundary of $P$ can be guarded by $\lceil 4\pi/\alpha \rceil \lceil 2\sqrt{2}/\beta \rceil^2$ guards.

Proof. Assume without loss of generality that the diameter of $P$ is 1. Thus, $P$ has a bounding square $B$ with area 1. The circle $C'$ in the guarding set $G_p$ from Lemma 2 contains a square with area $\beta^2/8$. We cover $B$ by $\lceil 2\sqrt{2}/\beta \rceil^2$ such squares that are each surrounded by a copy of $G_p$ centered at the center of the square. Since every vertex of every good triangle is contained in at least one such square, and every point of $\partial P$ has a good triangle, this must be a guarding set by Lemma 2. Since each copy of $G_p$ contains $\lceil 4\pi/\alpha \rceil$ guards, we need at most $\lceil 4\pi/\alpha \rceil \lceil 2\sqrt{2}/\beta \rceil^2$ guards to guard $\partial P$. \hfill \Box

4 Triangulating $k$-guardable polygons

We present two algorithms that triangulate a $k$-guardable polygon. The first algorithm does not require the guards as input. However, it uses a subroutine that is fairly complicated. The second algorithm only uses visibility-polygon computation as a subroutine, but it requires the
guards as input. This may seem strange at first given the hardness results mentioned in the introduction. However, given the results of the previous section for \((\alpha, \beta)\)-covered polygons, we can easily find a small guarding set in linear time for certain fat polygons.

4.1 Triangulating without guards

In many situations where triangulation is desired, it may be unrealistic to expect a set of guards as part of the input. In this section we show how to triangulate a \(k\)-guardable polygon in \(O(kn)\) time without knowing the guards. The most complicated procedure used in our algorithm is computing the visibility polygon from an edge in linear time [12]. We begin with some new notation and definitions.

The edge-visibility polygon, \(EVP(e, P)\), of an edge \(e\) with respect to polygon \(P\) consists of all points in \(P\) that are visible from at least one point on \(e\). We sometimes call \(EVP(e, P)\) the weak visibility polygon of the edge \(e\) if the polygon is clear from the context. We define an extended edge-visibility polygon of \(e\) with respect to \(P\), denoted by \(EEVP(e, P)\), to be the smallest (in terms of the number of edges) pure subpolygon of \(P\) that contains \(EVP(e, P)\). These concepts are illustrated in Figure 5.

![Figure 5: (a) The weak visibility polygon of the dotted edge. (b) The associated extended edge visible polygon. \(EEVP(e, P)\) is the union of the light and dark gray regions.](image)

The geodesic between two points in \(P\) is the shortest polygonal path connecting them that is contained in \(P\). The vertices of a geodesic (except possibly the first and last) belong to \(\partial P\)—see Figure 6.

![Figure 6: The geodesic from \(x\) to \(v\).](image)

Below, we show that Melkman's algorithm [18] can find a specific type of geodesic related to finding the \(EEVP\) of a polygon.
Lemma 4 Let $x$ be a vertex of polygon $P$ and let $y$ be a point on edge $\overline{wv} \in P$. If $y$ sees $x$, then the geodesic between $x$ and $v$: (a) is a convex chain and entirely visible from $y$, and (b) can be computed in $O(n)$ time.

Proof. Property (a) holds trivially if $x$ sees $v$. Consider the case where $x$ does not see $v$. Then, the triangle $(x, y, v)$, denoted by $T$, must contain at least one vertex of $P$ in its interior. Let $I$ be all the vertices of $P$ inside $T$ and let $CH(I)$ be the convex hull of $I$. The path $S = \partial CH(I) \setminus \overline{wv}$ is the geodesic from $x$ to $v$. Any other path from $x$ to $v$ inside $T$ can be shortened. Thus, property (a) holds.

To prove property (b), note that since the geodesic we seek is entirely visible from $y$ by part (a) it is fully contained in $VP(y, P)$. We compute $VP(y, P)$ in linear time. Consider the polygonal chain from $x$ to $v$ along $\partial VP(y, P)$ that avoids $y$. By construction of $VP(y, P)$, the shortest path from $x$ to $v$ is part of the convex hull of this chain. Using Melkman’s algorithm, we compute the convex hull of a simple polygonal chain in linear time. □

Finally, a weakly edge-visible polygon can be triangulated using a very simple algorithm known as the Graham scan. The following lemma formalizes that.

Lemma 5 (Toussaint and Avis [24]) Any weakly edge-visible polygon can be triangulated in linear time.

We now show how to compute and triangulate an extended edge visibility polygon, which is the main subroutine of our algorithm.

Lemma 6 $EEVP(e, P)$ can be computed and triangulated in $O(n)$ time.

Proof. We begin by computing $EVP(e, P)$ in $O(n)$ time using the algorithm of Heffernan and Mitchell [12]. This yields a set of windows $W$ and their associated pockets. For each window $w_i \in W$ that is not a diagonal of $P$, we do the following.

Let $x$ be the endpoint of $w_i$ closer to $e$, and let $y$ be the endpoint farther from $e$. Then $x$ is a vertex of $P$, and $y$ is an interior point on some edge $\overline{pq}$ of $P$. Without loss of generality let $p$ be the endpoint of $\overline{pq}$ that is inside the pocket of $w_i$, as illustrated in Figure 5 (b). Since $x$ sees $y$, we can use Lemma 4(b) to compute the geodesic from $x$ to $p$. Let $P(w_i)$ denote the polygon enclosed by the geodesic from $x$ to $p$, $\overline{pq}$ and $w_i$. It is simple to verify that the extended edge-visibility polygon is $EEVP(e, P) = EVP(e, P) \cup (\bigcup_{w_i \in W} P(w_i))$.

By Lemma 4 (b), each pocket $P(w_i)$ can be computed in time linear in the size of the pocket of $w_i$. Since pockets are disjoint and can be processed in order, $\bigcup_{w_i \in W} P(w_i)$, and thus $EEVP(e, P)$, can be computed in $O(n)$ time.

We now proceed to triangulate $EEVP(e, P)$. Consider $P(w_i)$. Let $T$ be the triangle determined by the points $x$, $y$ and $q$. If $e$ sees $q$, then $q$ sees each vertex in $P(w_i) \cup T$ by Lemma 4 (a). Therefore, $P(w_i) \cup T$ is a weakly edge-visible pure subpolygon of $P$. By Lemma 5, we can triangulate $P(w_i) \cup T$ in $O(|P(w_i)|)$ time.

If $e$ does not see $q$ then $q \in P(w_j)$ for some $w_j \in W$ with $j \neq i$. Let $Q$ be the quadrilateral determined by the endpoints of $w_i$ and $w_j$. The polygon $Y = P(w_i) \cup P(w_j) \cup Q$ is a pure subpolygon of $P$ that is weakly edge-visible from $\overline{pq}$. This implies that $Y$ can be triangulated using a simple method as before.

It is straightforward to verify that all of the pure subpolygons of $EEVP(e, P)$ triangulated thus far are pairwise non-overlapping. If $T$ is the union of these subpolygons then the closure of $EEVP(e, P) \setminus T$ is a weakly edge-visible pure subpolygon of $EEVP(e, P)$ and thus can also be triangulated in linear time. This results in a triangulation of $EEVP(e, P)$, as required. □
When $EEVP(e, P_i)$ is triangulated, diagonals of $P$ that are on $\partial EEVP(e, P_i)$ become windows of new pockets. Each such window serves as the edge from which a new visibility polygon will be computed and triangulated, within its respective pocket. In this recursive manner we break pockets into smaller components until all of $P$ is triangulated. The procedure, although straightforward, is outlined below in more detail. This is followed by the analysis of the time complexity, where we show that the recursion depth is of the order of the number of guards that suffice to guard $\partial P$.

We will maintain a queue $S$ of non-overlapping polygons such that each $P_i \in S$ has one edge $w_i$ labelled as a window. Thus elements of $S$ are pairs $(w_i, P_i)$. We start with $S := (w, P)$, where $w$ is an arbitrary boundary edge of $P$. We process the elements of $S$ in the order in which they were inserted. The main loop of our algorithm is as follows:

**TriangulateWithoutGuards($P$)**

1. $S := (w, P)$, where $w$ is an arbitrary edge of $P$.
2. while $S \neq \emptyset$
   3. do Choose the next element from $S$ and call it $(w_i, P_i)$.
   4. Remove $(w_i, P_i)$ from $S$.
   5. Compute and triangulate $EEVP(w_i, P_i)$.
   6. Add the edges of the triangulation to $P$.
   7. for each boundary edge $w_j$ of $EEVP(w_i, P_i)$ that is a diagonal of $P$,
      do Identify $Q_j$ as the untriangulated portion of $P$ whose boundary is enclosed by $w_j$ and $\partial P$.
      Add the remaining untriangulated portion $(w_j, Q_j)$ to $S$.

9. return $P$.

**Theorem 7** The algorithm TriangulateWithoutGuards triangulates an $n$-vertex $k$-guardable polygon in $O(kn)$ time.

**Proof.** We first note that the $EEVP$s created by our algorithm define a tree structure $T$, as follows. At the root of $T$ is $EEVP(w, P)$. For every window $w_j$ of $EEVP(w_i, P_i)$, we have that $EEVP(w_j, P_j)$ is a child of $EEVP(w_i, P_i)$. The construction of the child nodes from their parents ensures that no $EEVP$ overlaps with any other and that the triangulation covers the entire polygon $P$.

We now show that $T$ has at most $3k$ levels (a level is a set of nodes at the same distance from the root) which implies that the main loop of the algorithm performs at most $3k$ iterations. Let $\ell_i$, $\ell_{i+1}$, and $\ell_{i+2}$ be three successive levels of $T$, in which all the nodes in $\ell_{i+1}$ are descendants of the nodes in $\ell_i$, and where all the nodes in $\ell_{i+2}$ are descendants of the nodes in $\ell_{i+1}$. Further, let $G$ be a point set of size $k$ such that every point $p \in \partial P$ sees at least one guard of $G$. Assume, for the purpose of obtaining a contradiction, that there are no guards from $G$ in the $EEVP$s corresponding to the nodes in levels $\ell_i$, $\ell_{i+1}$, or $\ell_{i+2}$.

Let $g$ be a guard which sees into a node $n_i$ at level $\ell_i$ through window $w_i$. There are two cases: either $g$ is at a higher level than $\ell_i$ or it is at a lower level. If $g$ is in a higher level and is visible from a window of $n_i$, then $g$ can be in only one level: $\ell_{i+1}$ (because $\ell_{i+1}$ contains the union of all the edge-visibility polygons of the windows of the nodes in $\ell_i$). We have assumed that this can not happen. Otherwise, if $g$ is in a lower level, $g$ can not see into any level higher than $\ell_i$, because $w_i$ must be the window which created $n_i$.

The combination of these two facts implies that no guard from $G$ can see into $\ell_{i+1}$. This is a contradiction to $G$ being a guarding set. Therefore, $G$ must have at least one guard in $\ell_i$, $\ell_{i+1}$, or $\ell_{i+2}$. This implies that there is at least one guard for every three levels, or at most three levels per guard.
Each level of the tree can be processed in $O(n)$ time by Lemma 6, since all nodes of a level are disjoint. Therefore, the algorithm terminates in $O(kn)$ time.

As is apparent from the proof of Theorem 7, our algorithm runs in $O(tn)$ time, where $t$ is the number of iterations of the while-loop. The above argument also implies a stronger result. The number of iterations, $t$, of the while loop is proportional to the link diameter, $d$, of the polygon, since any minimum link path between two points must have at least one bend for every three levels. This leads to the following corollary:

**Corollary 8** The algorithm TriangulateWithoutGuards triangulates an $n$-vertex polygon with link diameter $d$ in $O(dn)$ time.

### 4.2 Triangulating with a given set of guards

Let $G = \{g_1, \ldots, g_k\}$ be a given set of $k$ guards in $P$ that jointly see $\partial P$. In this section we describe a simple algorithm that triangulates $P$ in $O(kn)$ time. One advantage of this algorithm is that it does not use any subroutines other than visibility-polygon computation. In particular, it does not compute the edge-visibility polygon, which is more complicated.

A *vertical decomposition* of $P$—also known as a trapezoidal decomposition of $P$, leading to the notation $T(P)$—is obtained by adding a *vertical extension* to each vertex of $P$. A *vertical extension* of $v$, denoted vert($v$), is the maximal vertical line segment which is contained in $\text{int}(P)$ and intersects $v$. We sometimes refer to an *upward* (resp. *downward*) *vertical extension* of $v$. This is the (possibly empty) part of vert($v$) that is above (resp. below) $v$.

Let $g$ be a guard and $w$ be a window of $VP(g, P)$. $P_w$ denotes the pocket of $w$ with respect to $VP(g, P)$. The *vertical projection onto* $w$ is the ordered list of intersection points of $w$ with the vertical extensions of the vertices of $P_w$ (see Figure 7).

![Figure 7: The vertical projection onto $w$ is $(x_1, x_2, x_3)$.](image)

Our algorithm finds the vertical decomposition $T(P)$ of $P$ in $O(kn)$ time. In particular, we show how to compute all vertical extensions of $T(P)$ that are contained in or cross the visibility polygon of a guard in $O(n)$ time. Since each vertex of $P$ is seen by at least one guard, every vertical extension is computed by our algorithm. It is well known that finding a triangulation of a polygon $P$ is simple given the vertical decomposition of $P$ [5]. The most complicated procedure used in our algorithm has the difficulty level of computing the visibility polygon of a point.

Below is a high-level description of our algorithm. The details of the various steps will be discussed later.
**TriangulateWithGuards**\((P, G)\)

1. for each guard \( g \in G \)
2. do find the visibility polygon \( VP_1(g, P) \).
3. for each window \( w \) in \( VP_1(g, P) \)
4. do compute the vertical projection onto \( w \) and add the resulting Steiner points to \( w \).
5. \( \triangleright \) After all windows of \( VP_1(g, P) \) have been processed, we have a simple polygon \( VP_2(g, P) \) that includes the points in the vertical projections as Steiner points on the windows.
6. Compute the vertical decomposition of \( VP_2(g, P) \). For every vertex \( v \) of \( VP_2(g, P) \) that is not a Steiner point created in Step 2, add the vertical extension of \( v \) to \( \partial VP_2(g, P) \), creating \( VP_3(g, P) \).
7. \( \triangleright \) We have now computed the restriction of \( T(P) \) to \( VP_2(g, P) \); every vertical extension that is part of \( T(\partial VP_3(g, P)) \) is contained in a vertical extension of \( T(P) \) and every vertical extension of \( T(P) \) that crosses \( VP_2(g, P) \) is represented in \( T(\partial VP_3(g, P)) \).
8. for each vertex \( v \) of \( VP_3(g, P) \),
9. do Determine the endpoints of \( \text{vert}(v) \) on \( \partial P \).

Figure 8 shows a sample execution of our algorithm.

Figure 8: Sample execution of algorithm **TriangulateWithGuards**. The line of code is shown at the top-left of each sub-figure. The guard is depicted by a box, unfilled circles are new Steiner points, and filled circles are points from which a vertical extension is computed.

By Lemma 1, Step 2 takes \( O(n) \) time. We now discuss the other steps of the algorithm.

**Step 4: Computing a vertical projection onto a window.** Without loss of generality, we assume that \( w \) is not vertical and that \( \text{int}(VP(g, P)) \) is above \( w \) (see Figure 9). Let \( v \) be a vertex of \( P_w \) such that \( \text{vert}(v) \) intersects \( w \). Furthermore, let \( z \) be a point at infinite distance vertically above some point on \( w \). Observe that if we remove the parts of \( P \) above \( w \) so that \( z \) can see all of \( w \), then \( z \) can see \( v \). This implies that we should remove all parts of \( P_w \) that are inside the “vertical slab” above \( w \), so that vertices whose vertical extensions intersect \( w \) are precisely those that form the visibility polygon of \( z \). The technique of computing a visibility polygon of a point at infinity was first used by Toussaint and El Gindy [25].

We remove all the parts of \( P_w \) that are outside the vertical slab directly below \( w \), as follows. Imagine shooting two rays downward from the left and right endpoints of \( w \). We call the rays \( r_1 \) and \( r_2 \) where \( r_1 \) is to the left of \( r_2 \). We keep two counters called \( c_1 \) and \( c_2 \) that are initialized to 0, and are associated to \( r_1 \) and \( r_2 \), respectively. We begin scanning \( \partial P_w \) at one of the endpoints of \( w \) and proceed toward the other endpoint. If an edge of \( \partial P_w \) intersects \( r_1 \) from the right, we increment \( c_1 \) and proceed as follows until \( c_1 \) is again 0. We continue scanning \( \partial P_w \),
Figure 9: Computing a vertical projection. (a) A polygon that does not wrap around \( w \). (b) Its vertical projection. (c) A polygon that wraps around \( w \). The counter \( c_2 \) is incremented at \( i_1 \) and \( i_2 \), decremented at \( i_3 \), incremented again at \( i_4 \), and decremented two more times at \( i_5 \) and \( i_6 \), at which time it is once again 0.

throwing away edges as we go. If an edge intersects \( r_1 \) from the right, we increment \( c_1 \) and if an edge intersects \( r_1 \) from the left, we decrement \( c_1 \). When \( c_1 \) is 0, we connect the first and last intersections of \( \partial P_w \) by a segment. The procedure is essentially the same when an edge intersects \( r_2 \) except that we interchange “right” and “left”. Note that if \( P_w \) winds around \( w \) many times, \( c_1 \) or \( c_2 \) might be much larger than 1. Finally, once \( \partial P_w \) has been traced back to \( w \), we remove potential intersections between newly constructed line segments along \( r_1 \) by shifting them to the left by a small amount proportional to their length. We shift the new segments along \( r_2 \) to the right by a small amount proportional to their length. The simplicity of \( P \) implies that the new segments are either nested or disjoint, so we obtain a simple polygon that does not cross the vertical slab above \( w \). Finally, we remove \( w \) and attach its endpoints to \( z \), thus obtaining polygon \( \hat{P}_w \). The vertices of \( VP(z, \hat{P}_w) \) are precisely those vertices of \( P_w \) whose vertical extensions intersect \( w \) and appear as output in sorted order.

**Lemma 9** The vertical projection onto \( w \) can be computed in \( O(|P_w|) \) time.

**Proof.** The algorithm described in the text consists of a scan of \( \partial P_w \) and a visibility polygon calculation, which has complexity \( O(|P_w|) \). Therefore, it remains to show that a point \( x \) is added to \( w \) if and only if there is a corresponding vertex \( v_x \) in \( P_w \) whose vertical extension intersects \( w \).

Suppose there is a vertex \( v_x \) whose vertical extension intersects \( w \). Then \( v_x \) is visible from \( z \), so \( v_x \) is included in \( VP(z, \hat{P}_w) \) and thus \( x \) is added to \( w \). On the other hand, suppose there is a point \( x \) added to \( w \). This occurs if there is a vertex \( v_x \) which is visible to \( z \) through \( w \). Since this is the case, the vertical extension of \( v_x \) intersects \( w \).

**Step 5: Computing a vertical decomposition of a star-shaped polygon.** Let \( S \) be a given star-shaped polygon and \( g \) be a point inside the kernel of \( S \). We assume that the vertices of \( S \) are given in counterclockwise order around \( S \). To simplify the algorithm, we describe only the computation of the upward vertical decomposition (that is, for each vertex \( v \), we find the upper endpoint of \( \text{vert}(v) \)) of the part of \( S \) that is to the left of the vertical line through \( g \). See Figure 10. We say that a vertex \( v \) supports a vertical line \( \ell \) if the two edges adjacent to \( v \) are both on the same side of \( \ell \).

The algorithm for finding the upward vertical decomposition of \( S \) consists of a sequence of alternating leftward and rightward walks: a leftward walk which moves a pointer to a vertex
which supports a vertical line (locally) outside \( S \), and a rightward walk which adds vertical decomposition edges. The algorithm begins with the leftward walk which starts from the point directly above \( g \). It ends when the rightward walk passes under \( g \).

The leftward walk simply moves a pointer forward along \( \partial S \) until a vertex \( v_s \) which supports a vertical line outside \( S \) is encountered—so we concentrate on describing the rightward walk. The rightward walk begins with two pointers, \( p_u \) and \( p_d \), which initially point to \( v_s \), the last point encountered in the leftward walk. The pointers are moved simultaneously so that they always have the same \( x \)-coordinate, with \( p_d \) being moved forward along \( \partial S \)—that is, counterclockwise—while \( p_u \) is moved backward along \( \partial S \) (imagine sweeping rightward with a vertical line from \( v_s \)). If \( p_d \) encounters a vertex, then a vertical decomposition edge is created between \( p_d \) and \( p_u \). If \( p_u \) encounters a vertex \( v \) to which a vertical decomposition edge \( \text{vert}(v) \) is already attached (which implies that \( v \) supports a vertical line), then \( p_u \) moves to the top of \( \text{vert}(v) \) and continues from there. When \( p_d \) encounters a vertex \( v \) that supports a vertical line, the rightward walk ends and the leftward walk begins anew at \( v \).

Lemma 10  The vertical decomposition of a star-shaped polygon \( P \) is correctly computed by the above algorithm in \( O(n) \) time.

Proof.  The algorithm outlined in the text maintains the following extension invariant: the correct upward vertical extension has been found for every vertex to which \( p_d \) has pointed. Initially, the invariant is trivially true.

By construction, \( p_d \) visits all vertices of \( S \) that are the endpoints of the edges of the upward vertical decomposition of \( S \) in counterclockwise order. Hence the algorithm constructs a vertical extension for each of these vertices. It remains to show that the upper endpoint of the vertical extension is correctly identified. Denote the current position of \( p_d \) by \( v_d \). Again by construction, \( p_u \) lies vertically above \( v_d \) at position \( v_u \). We need to show that \( \overline{v_d v_u} \) is not intersected by an edge of \( S \).

Consider the triangle \( g v_d v_u \). Since \( g \) sees all of \( S \), \( \overline{g v_d} \) and \( \overline{g v_u} \) can not be intersected by an edge of \( S \). This implies that any edge \( e \) that intersects \( g v_d v_u \) must intersect \( \overline{v_d v_u} \). Furthermore, \( e \) must be an edge in the chain \( C_L \), which is the chain from \( v_u \) to \( v_d \) in counterclockwise order. To show that no edge from \( C_L \) intersects \( \overline{v_d v_u} \), we establish the order invariant: \( C_L \) is always to the left of \( \overline{p_u p_d} \). The invariant is trivially true whenever \( p_u \) and \( p_d \) point to \( v_s \), that is, whenever we begin a rightward walk. Suppose that the invariant has been true until step \( k \) and we will show that it is still true at step \( k + 1 \). Let \( C'_L \) be the chain from \( p_u \) to \( p_d \) at step \( k \) and \( C_L \) be the chain from \( p_u \) to \( p_d \) at step \( k + 1 \). There are three cases in step \( k \): (a) \( p_d \) is pointing to a vertex of \( S \), (b) \( p_u \) is pointing to a vertex of \( S \) without a vertical extension, or (c) \( p_u \) is pointing to a vertex \( v \) of \( S \) with a vertical extension. See Figure 11. In the first two cases, the invariant is maintained since \( C_L \) only differs from \( C'_L \) by two segments that by definition both lie to the left of \( \overline{p_u p_d} \). Since the vertices in \( C_L \) come before \( v_d \), the correct vertical extension of each vertex in \( C_L \) has been computed by the assumption of the extension invariant. This

Figure 10: Upward vertical decomposition of the part of \( S \) to the left of the guard \( g \).
implies that the order invariant is also maintained in the case where \( p_u \) is pointing to a vertex \( v \) of \( S \) with a vertical extension and is moved to the top of \( \text{vert}(v) \). This is because \( C_L' \) differs from \( C_L \) by a segment which is to the left of \( p_d \) and a chain which must be to the left of \( \overline{p_u p_d} \) since \( \text{vert}(v) \) is a valid vertical extension.

Both \( p_d \) and \( p_u \) visit every vertex of \( S \) at most once, hence the running time is \( O(n) \).

---

**Figure 11:** Establishing correctness of the order invariant: three cases.

---

**Step 6: Computing the endpoints of vertical extensions.** The final step of the algorithm is to find and connect the endpoints of the vertical extensions of every vertex of \( VP_3(g, P) \). Let \( v \) be an arbitrary vertex of \( VP_3(g, P) \). If both endpoints of \( \text{vert}(v) \) are on the boundary of \( VP(g, P) \), we have already found and connected them in the previous step. Thus, let us assume that at least one of the endpoints of \( \text{vert}(v) \) is not on the boundary of \( VP(g, P) \). That is, \( \text{vert}(v) \) intersects at least one window of \( VP(g, P) \). Since we have already connected the endpoints of \( \text{vert}(v) \cap VP(g, P) \) in the previous step, it is sufficient to find the endpoints of \( \text{vert}(v) \) that are outside of \( VP(g, P) \). Thus, it suffices to examine vertices that are Steiner points on windows.

Let \( v_1, \ldots, v_j \) be vertices on window \( w \), in sorted order. Again without loss of generality, we assume that \( \text{int}(VP(g, P)) \) is above \( w \). To find the endpoint of \( \text{vert}(v) \) that is below \( w \) for all \( v \in \{v_1, \ldots, v_j\} \), we use the visibility polygon \( VP(z, \hat{P}_w) \) computed in Step 4 of the algorithm. Note that the vertices of \( VP(z, \hat{P}_w) \) as well \( \{v_1, \ldots, v_j\} \) are sorted by \( x \)-coordinate. Thus we find the endpoints of \( \{\text{vert}(v) | v \in \{v_1, \ldots, v_j\}\} \) by simultaneously scanning in \( VP(z, \hat{P}_w) \) and \( \{v_1, \ldots, v_j\} \) (as though performing a merge operation in merge-sort). Since \( \sum_w |P_w| \leq n \) and the number of Steiner points added to windows is at most \( n \), we find the endpoints of the vertical extensions of all Steiner points on windows in \( O(n) \) time.

Since each guard is processed in linear time, we obtain the following.

**Theorem 11** The algorithm \( \text{TriangulateWithGuards} \) computes the vertical decomposition of an \( n \)-vertex \( k \)-guardable polygon in \( O(kn) \) time, if the \( k \) guards are given.

---

**5 Open problems**

Several known classes of realistic polygons are in fact \( k \)-guardable. In particular, we have shown that the boundary of an \((\alpha, \beta)\)-covered polygon can be guarded by a constant number of guards depending on \( \alpha \) and \( \beta \). In other words, \((\alpha, \beta)\)-covered polygons are \( O(1) \)-guardable.

We also gave two simple algorithms that triangulate \( k \)-guardable polygons in linear time, if \( k \) is a constant. The first algorithm is slightly simpler, but does require the guards as input, while the second algorithm does not need the guards.

Our work leaves some open problems. First, can the techniques shown here be used to design a triangulation algorithm which does not depend on the number of guards? Second, are there other problems that can be solved efficiently for \( k \)-guardable polygons? Finally, are there more general classes of polygons that can be triangulated in linear time with simple algorithms?
Acknowledgments

This research was initiated at the Carleton-Eindhoven Workshop on Computational Geometry, July 18–22, 2005, organized by Mark de Berg and Prosenjit Bose. The authors are grateful to Mark de Berg for suggesting the problems studied in this paper and Boris Aronov for many discussions.

References


