A Pedestrian Approach to Ray Shooting: Shoot a Ray, Take a Walk*

John Hershberger[†] Subhash Suri[‡]

August 16, 1993

Abstract

We propose a very simple ray shooting algorithm, whose only data structure is a triangulation. The query algorithm, after locating the triangle containing the origin of the ray, walks along the ray, advancing from one triangle to a neighboring one until the polygon boundary is reached. The key result of the paper is a Steiner triangulation of a simple polygon with the property that a ray can intersect at most $O(\log n)$ triangles before reaching the polygon boundary. We are able to compute such a triangulation in linear sequential time, or in $O(\log n)$ parallel time using $O(n/\log n)$ processors. This gives a simple, yet optimal, ray shooting algorithm for a simple polygon. Using a well-known technique, we can extend our triangulation procedure to a multi-connected polygon with k components and n vertices, so that a ray intersects at most $O(\sqrt{k}\log n)$ triangles.

*A preliminary version of this paper appears in the proceedings of the *4th ACM-SIAM Symposium on Discrete Algorithms,* 1993.

[†]DEC Systems Research Center, 130 Lytton Avenue, Palo Alto, California 94301

[‡]Bell Communications Research, 445 South Street, Morristown, New Jersey 07960

1 Introduction

The *ray shooting* problem in a simple polygon P asks for a data structure representing P that can efficiently answer queries of the form "find the first intersection between a ray and the boundary of P." A *ray* is a directed half-line, specified by a point of origin p and a direction vector. An algorithm for the ray shooting problem returns the label of the edge of P that meets the ray at a point closest to p. (The ray shooting problem makes sense regardless whether the ray-origin p lies inside or outside the polygon. We will assume for simplicity that p lies inside the polygon, although our solution works for the other case as well.) The ray shooting problem is a fundamental problem in plane computational geometry, with numerous applications in computer vision and motion planning problems.

We propose an extremely simple algorithm for the ray shooting problem. Our data structure consists of a Steiner triangulation of P such that any ray intersects at most $O(\log n)$ triangles before meeting the boundary of P. A query is processed by first determining the triangle containing the ray-origin, and then walking through the triangulation along the ray until the polygon boundary is reached. Our triangulation guarantees that the walk will terminate after $O(\log n)$ triangles. Because each triangle can be traversed in $O(1)$ time, the time to answer a ray shooting query is $O(\log n)$, exclusive of the time to locate the initial triangle. The first triangle can be found also in $O(\log n)$ time, using any of the well-known data structures for point location in a planar subdivision [8, 15].

Except for the point-location structure, our algorithm for ray shooting requires no other data structure besides an adjacency graph of the Steiner triangulation. In applications in which the starting triangle is always known, the point-location structure can be dispensed with. Such is the case, for instance, with ray-tracing algorithms in graphics, in which a ray is tracked as it reflects off polygon boundaries following the laws of optics; the terminus of each ray shooting query becomes the origin for the next query. Our Steiner triangulation has $O(n)$ points, and thus our data structure requires linear space. The triangulation can be computed in $O(n)$ preprocessing time. A parallel implementation of our preprocessing algorithm runs in optimal time $O(\log n)$ with $O(n/\log n)$ processors in the concurrent-read exclusive-write PRAM model.

Chazelle and Guibas first achieved an optimal query time for the ray-shooting problem [3]. Their technique is quite complicated and relies on several sophisticated data structures. The preprocessing step of Chazelle and Guibas's algorithm required $O(n \log n)$ time, but it was later improved to $O(n)$ for *pre-triangulated* polygons by Guibas et al. [12]. Another optimal algorithm for ray shooting was recently proposed by Chazelle et al. [2]; their algorithm is based on a balanced hierarchical decomposition of the input polygon into *geodesic triangles*. (A geodesic triangle is a region of a simple polygon bounded by shortest paths between three points.) A key property of this *geodesic triangulation* is that a ray intersects $O(\log n)$ geodesic triangles. A ray shooting query in polygon P can therefore be decomposed into $O(\log n)$ instances of ray shooting in geodesic triangles, and the latter can indeed be answered in logarithmic time each. This leads to an $O(\log^2 n)$ time algorithm for the original ray shooting problem. At this point, Chazelle et al. [2] apply weight-balanced trees and an advanced data structuring technique called *fractional cascading* [4, 5] to bring the query cost down to $O(\log n)$, albeit at the expense of complicating the algorithm. We should mention, though, that despite the use of fractional cascading, the algorithm of [2] is vastly simpler than the earlier algorithm [3, 12].

Our paper follows the geodesic triangulation approach, but develops it further to obtain a decomposition of the polygon into ordinary triangles—not geodesic triangles—that still has the property that a ray intersects only $O(\log n)$ triangles. This settles a key open problem posed in [2]. Our result is based on a careful triangulation of each geodesic triangle in the balanced geodesic triangulation of P. It is relatively easy to triangulate a geodesic triangle τ of size m so that a ray intersects $O(\log m)$ triangles in τ. This bound is also tight: for any triangulation of τ, there is a ray intersecting $\Omega(\log m)$ triangles. Since in general a ray hits $\Theta(\log n)$ geodesic triangles, the strategy of independently but optimally triangulating each geodesic triangle gives only a bound of $O(\log^2 n)$ for the total number of triangles intersected by the ray. Our algorithm controls the triangulation globally, so that even though a ray may intersect $\Omega(\log n)$ triangles in a single geodesic triangle, the amortized bound is $O(1)$ per geodesic triangle. Since a ray shooting problem in a plane triangle can obviously be solved in $O(1)$ time, we achieve an extremely simple, yet optimal, algorithm for ray shooting in a simple polygon.

Our paper is organized in eight sections. Section 2 introduces the balanced geodesic triangulation of the polygon and a construct known as a *boomerang*. Section 3 describes a method for triangulating a boomerang using a binary tree as a guide. Section 4 presents a method for assigning weights to the edges of a boomerang; these weights are used to build a weight-balanced tree, which in turn guides the triangulation of a boomerang. In Section 5, we prove that our triangulation indeed has the desired property: a ray intersects only a logarithmic number of triangles. Section 6 gives an algorithm for computing our triangulation, and Section 7 shows how to extend our result to a family of disjoint polygons. Finally, we conclude in Section 8.

2 The Balanced Geodesic Triangulation

Our starting point is a *balanced geodesic triangulation* of the polygon, which was proposed by Chazelle et al. [2], but also by Clarkson [6] in another context. This triangulation is obtained by computing a hierarchy of shortest (also called geodesic) paths inside the polygon between pairs of polygon vertices. We organize the triangulation process in $O(\log n)$ stages; the first stage differs slightly from all the subsequent ones. The first stage picks three equally-spaced vertices of P, say, x, y, and z, and computes geodesic paths between the pairs (x, y), (y, z), and (z, x). This divides up the boundary of P into three chains, namely, $P(x, y)$, $P(y, z)$, and $P(z, x)$. Subsequent stages pick the middle vertex of each chain and compute geodesic paths from it to the endpoints of the chain. In particular, the second stage computes geodesic paths from x and y to the middle vertex of the chain $P(x, y)$, from y and z to the middle vertex of $P(y, z)$, and from z and x to the middle vertex of $P(z, x)$. In general, at the start of the k'th stage, for $k > 1$, the boundary of P is divided into $3 \cdot 2^{k-2}$ subchains, and we compute two geodesic paths for each of them. The procedure terminates when each chain consists of a singleton edge.

Remark. The balanced geodesic triangulation of a *convex* polygon is the familiar hierarchical decomposition of a convex polygon used by Dobkin and Kirkpatrick [7] in their intersection-detection algorithms. In the case of the convex polygon, each geodesic path is a straight line segment and the regions they define are ordinary triangles.

Let us now consider the geometric structure of these geodesic paths. We initially compute three paths corresponding to (x, y), (y, z), and (z, x). These paths form a figure called a *kite*, which consists of a *geodesic triangle* and three *tails*: Let (x, x') be the maximal subpath common to the geodesic paths (x, y) and (x, z), and define subpaths (y, y') and (z, z') similarly. These three subpaths are the tails. The region bounded by the three concave chains joining x', y', and z' is the geodesic triangle (x', y', z'). The concave chains are subpaths of the original geodesic paths. The points x', y', and z' are called the *apexes* of the geodesic triangle. See Figure 1.

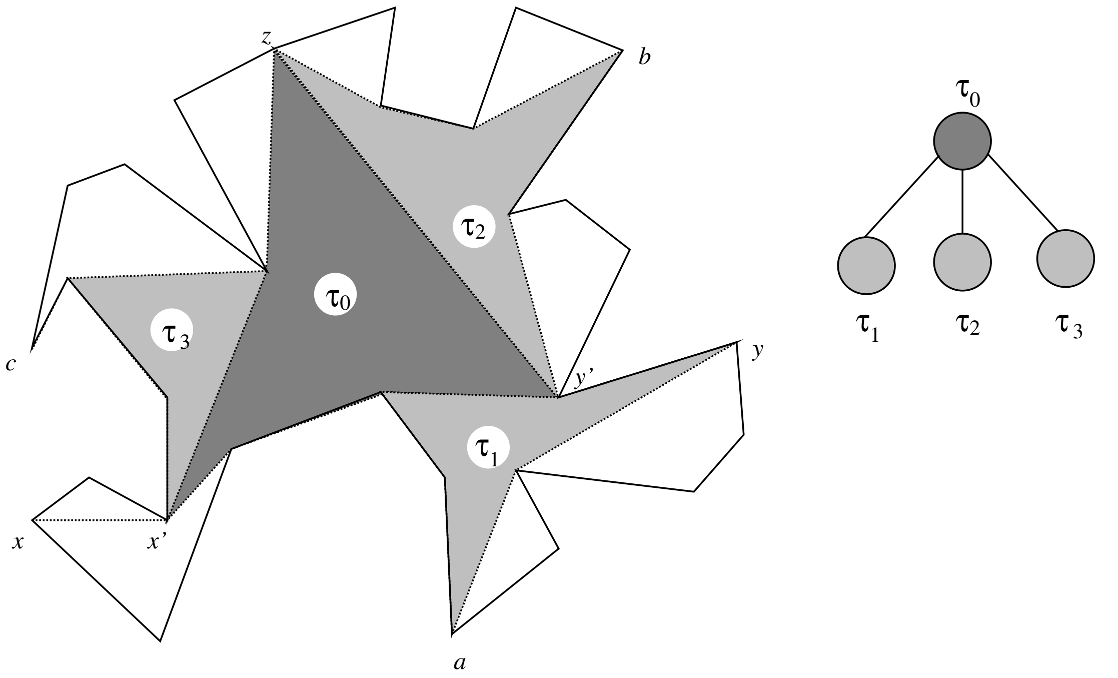

Figure 1: First two stages of a balanced geodesic triangulation. (x, y, z) is the first kite and $\tau_0 = (x', y', z)$ is the first geodesic triangle. The second stage produces three kites $(x, a, y), (y, b, z), (z, c, x)$, and three geodesic triangles τ_1, τ_2, τ_3. On the right, the top two levels of the corresponding GDT are shown.

Remark. A kite may have an empty geodesic triangle. This happens when one of the three vertices defining the kite lies on the geodesic path between the other two.

2.1 The Geodesic Decomposition Tree

Our balanced geodesic triangulation produces a kite in the first stage, and then $3 \cdot 2^{k-2}$ additional kites in stage k, for $k > 1$. We can model this decomposition process by a free tree of degree three, called the *geodesic decomposition tree* (GDT). The root of GDT corresponds to the kite produced in the first stage, and the nodes at depth i correspond to kites generated in stage $i+1$, with the obvious parent-child relationship. The height of GDT is no more than $\log n$.[1] Observe that, in general, there is a substantial overlap of edges among the various kites. To maintain linear storage, we store the geodesic triangles at the nodes of GDT, not the full kites. In particular, if a node w in GDT corresponds to a kite κ, we store κ's geodesic triangle at w, but not the edges belonging to its tails. In this scheme, no edge is stored more than twice. The geodesic triangles also inherit the ancestor/descendant relationships of their corresponding kites: geodesic triangles τ_1 and τ_2 have the same ancestor-descendant relationship in GDT as do their kites κ_1 and κ_2.

Lemma 2.1 *GDT has linear space complexity.*

> **Proof:** In a balanced geodesic triangulation, no new vertices are added and no two edges cross. Thus, the number of new edges introduced by all geodesic paths is at most $n-3$, the number required for triangulating the polygon. Since each edge is stored at most twice in the decomposition tree, the total space complexity is $O(n)$. ∎

Lemma 2.2 *Let s be a line segment contained in P, and let X be the set of geodesic triangles intersected by s in a balanced geodesic triangulation of P. The nodes corresponding to triangles of X lie on a single path in the corresponding GDT, and hence $|X| \le 2\log n$.*

> **Proof:** Extend s until it hits the boundary of P at both ends. The endpoints of the extended segment s partition the boundary of P into two subchains. A kite intersects s if and only if its three defining vertices do not all lie in the same subchain. As in the Dobkin-Kirkpatrick polygon hierarchy, there are at most $2\log n$ kites with this property. The geodesic triangles that s intersects are a subset of those whose kites are intersected. ∎

2.2 Boomerangs

We next partition each geodesic triangle into a constant number of triangles and *boomerangs*. A boomerang is a geodesic triangle with two concave chains degenerated to single edges. We call these single edges the *sides* of the boomerang. Figure 2 illustrates how to decompose a geodesic triangle into boomerangs and triangles.

To decompose a geodesic triangle τ with apexes x, y, and z, we pick two rays incident to each apex and extending into the geodesic triangle. We extend and connect these rays in order, as shown in Figure 2 (i). These six edges generate three Steiner points a, b, and c, which are connected to form the triangle $\triangle abc$. This construction partitions the interior of τ into four triangles $\triangle xac$, $\triangle zcb$, $\triangle yab$, and $\triangle abc$, and three boomerangs (x, a, y), (y, b, z), and (z, c, x).

There are obvious modifications to this construction if some of τ's concave chains are degenerate. If two or three chains of τ are degenerate, then we leave τ alone: it's already either a boomerang or a triangle. If one side is degenerate, then we partition the interior into three triangles and two boomerangs, as shown in Figure 2 (ii).

We now come to the main step of our algorithm: triangulating a boomerang. Our algorithm triangulates each boomerang separately. There are many ways of triangulating

[1] All logarithms in this paper are to the base 2.

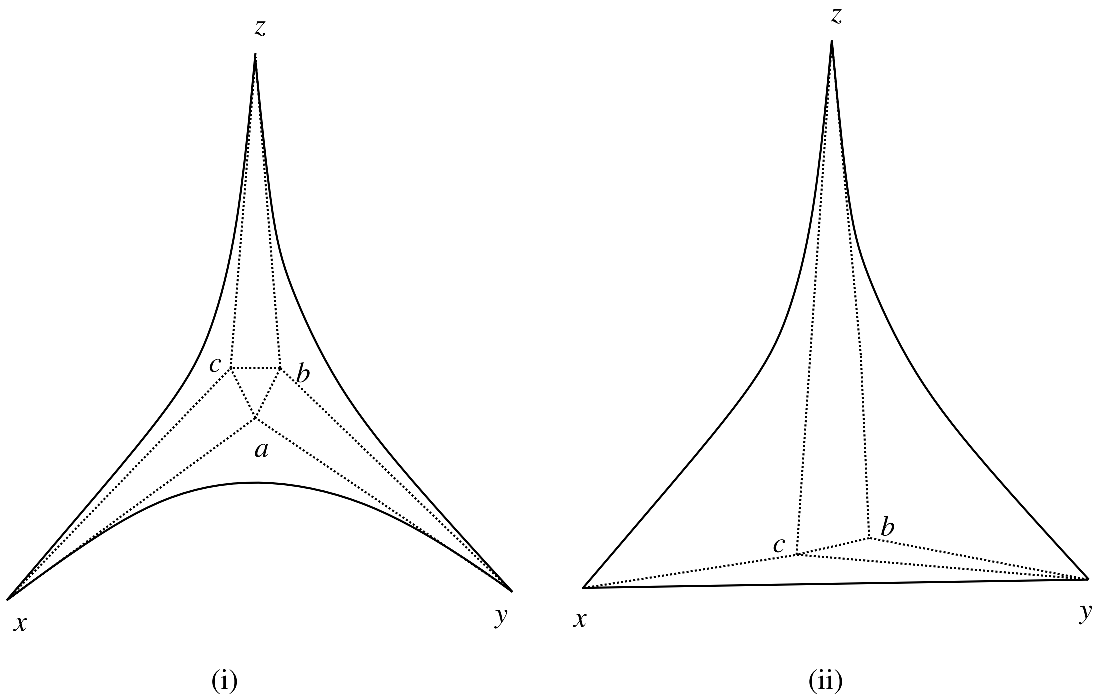

Figure 2: Decomposing a geodesic triangle into boomerangs. Figure (i) shows the construction for the general case, while figure (ii) shows it for a geodesic triangle with only two concave chains.

a boomerang; we are interested in one that minimizes the number of triangles intersected by a ray. In the next section, we discuss a specific algorithm for triangulating a boomerang that is guided by a binary tree.

3 A Tree-guided Triangulation of a Boomerang

Our scheme for triangulating a boomerang is based on rooted binary trees. We link the number of triangles intersected by a ray s in a triangulation S to the depth of a certain node in a binary tree associated with S. In the following, we often use the term *intersection number* of a segment s in S to denote the number of triangles intersected by s in the triangulation S.

Let us consider a boomerang B, with apexes x, y, and z and sides xz and yz. B has a concave chain C between x and y, whose edges are labeled $e_1, e_2, \ldots, e_k$ in counterclockwise sequence. Now consider a rooted binary tree $\mathcal{T}$, whose nodes in symmetric order correspond to the edges $e_1, e_2, \ldots, e_k$. We use the tree structure of $\mathcal{T}$ to guide a divide-and-conquer triangulation of B.

Suppose the root of $\mathcal{T}$ is the edge e_i, with endpoints u_i and v_i. We extend the edge e_i inside B until it intersects the sides xz and yz at points a and b, respectively (Figure 3 (i)). We form the triangles $\triangle xaz$, $\triangle au_iz$, $\triangle u_iv_iz$, $\triangle v_ibz$, and $\triangle byz$. Observe that triangles $\triangle xaz$ and $\triangle byz$ are degenerate (zero-area). These zero-area triangles have no ill effects for our ray-shooting application. Topologically, we treat a and b as interior points of B, leading to the conceptual triangulation shown in Figure 3 (ii). (Of course, we can also turn the conceptual triangulation of Figure 3 (ii) into a real triangulation by perturbing the points a and b, thus ensuring that all triangles have nonzero area.)

This construction partitions B into five triangles and two boomerangs. The boomerang B_1 has sides xa, au_i, and the concave chain $e_1, e_2, \ldots, e_{i-1}$, while the boomerang B_2 has sides bv_i, yb, and the concave chain $e_{i+1}, e_{i+2}, \ldots, e_k$. We recursively triangulate the boomerangs B_1 and B_2 using $\mathcal{T}$'s left and right subtrees $\mathcal{T}_l$ and $\mathcal{T}_r$, respectively. Observe that $\mathcal{T}_l$ and $\mathcal{T}_r$ satisfy the required invariant that they are rooted binary trees whose nodes in symmetric order correspond to the edges of the concave chains in B_1 and B_2, respectively.

In the following discussion, let us fix a binary tree $\mathcal{T}$ and the corresponding triangulation S for the boomerang B. Consider a line segment s contained in the boomerang B; in our case, the segment s is typically the portion of a ray contained in a boomerang. Since making the segment longer only increases its intersection number, we may assume that both endpoints of s are on the boundary of B. In the next two lemmas, we derive an upper bound on the intersection number of s in S in terms of the depth of a particular edge in $\mathcal{T}$. Lemma 3.1 treats the case in which s intersects the concave chain of B, and Lemma 3.2 considers the other case.

Lemma 3.1 *Let s be a segment in B that intersects an edge e of the concave chain C. The intersection number of s in S is at most $4d + 3$, where d is the depth of e in $\mathcal{T}$.*

Proof: Our proof is by induction on the depth d. The base case ($d = 0$) is easy: e is the root edge, and s intersects exactly three of the five triangles generated at the top level of triangulation (Figure 4 (i)).

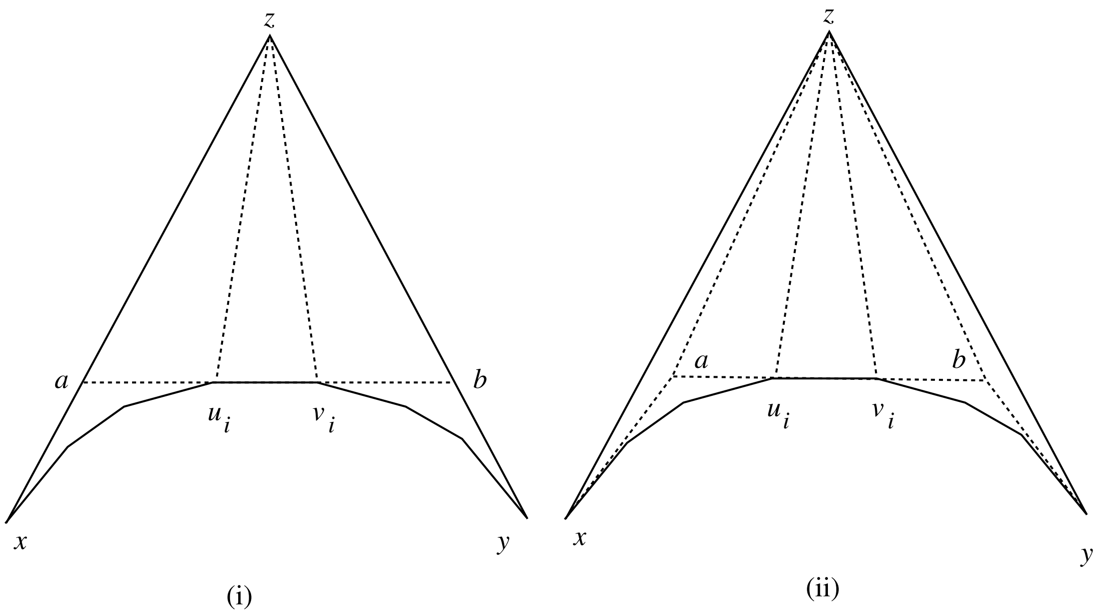

Figure 3: Triangulating a boomerang.

In the general case, we inductively assume that the lemma holds for all depths up to $d-1$, and that s intersects an edge of depth d. Our triangulation splits B into two boomerangs B_1 and B_2 by extending the edge stored at the root of $\mathcal{T}$, and then recursively triangulates B_1 and B_2 according to the subtrees $\mathcal{T}_l$ and $\mathcal{T}_r$, respectively. Without loss of generality, assume that e lies in B_1, and therefore it is stored in $\mathcal{T}_l$ at depth $d-1$. Obviously, s does not intersect B_2 at all, and intersects at most four triangles at the top level (Figure 4 (ii)). Finally, by induction, the number of triangles intersected by s in B_1 is at most $4(d-1)+3$. Thus, the intersection number of s is at most $4+4(d-1)+3 \ = \ 4d+3$. ▮

Next, we consider the case in which s does not intersect the concave chain C. In this case, we focus on a vertex of C that admits a tangent separating s from C inside B; the tangent need not be parallel to s, just disjoint from it inside B. Given a vertex $v \in C$, we define its *depth* in $\mathcal{T}$ to be the larger of the depths of the two edges incident to v.

Lemma 3.2 *Let s be a segment that intersects the two sides of B but not the concave chain C, and let $v \in C$ be the vertex with minimum depth that supports a tangent t separating C from s inside B. The intersection number of s in S is at most $5d+5$, where d is depth of v in $\mathcal{T}$.*

Proof: We again use induction on d. The base case ($d=1$) occurs when one of the edges incident to v, say e_i, is the root, and the other, say e_j, is a child of the root, where $j \in \{i-1, i+1\}$. Without loss of generality, assume that $j = i-1$, so e_j lies in the left child boomerang B_1. Since e_{i-1} is the root edge of the left subtree, its extension is used to split B_1 into an *empty* right-child boomerang B_{12} and a non-empty left-child

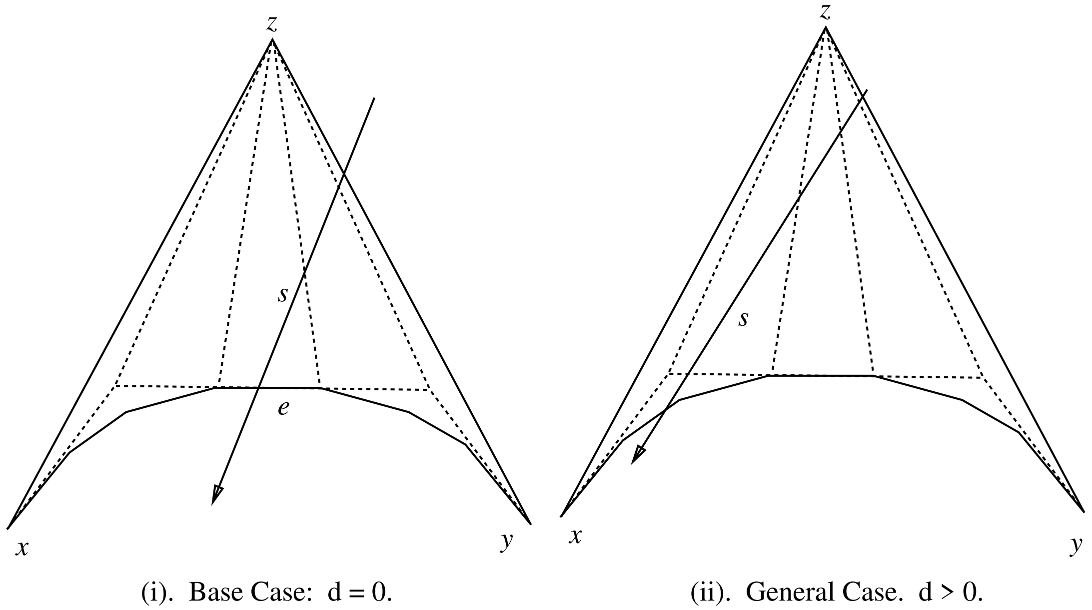

Figure 4: Proof of Lemma 3.1.

boomerang B_{11} with concave chain $(e_1, e_2, \ldots, e_{i-2})$. Because v supports the separating tangent t, the left extension of e_{i-1} separates s from B_{11}, and the right extension of e_i separates s from B_2. Thus s intersects neither B_2 nor B_{11}. The only triangles intersected by s are at the top levels of B and B_1. There are at most $5+3$ such triangles, which is less than $5d+5$, for $d=1$.

In the general case, $d>1$, assume without loss of generality that v lies in the left child B_1. See Figure 5. The same reasoning as in the base case of the induction shows that all triangles intersected by s lie either at the top level or in B_1. The top level has 5 triangles, and, by induction, the number of triangles intersected in B_1 is at most $5(d-1)+5$, which matches the claimed bound. This completes the proof. ▮

Remark. By Lemma 2.2, an internal segment in the polygon intersects a logarithmic number of geodesic triangles, each of which can be partitioned into a constant number of boomerangs and triangles. If we use a balanced binary tree to guide the triangulation of a boomerang, then Lemmas 3.1 and 3.2 guarantee that a segment intersects at most $O(\log n)$ triangles in each boomerang, and thus at most $O(\log^2 n)$ triangles in the complete triangulation.

Lemmas 3.1 and 3.2 show the close connection between the intersection number of s in a boomerang triangulation and the depth of the edge e that intersects s or the depth of a vertex v that supports a tangent separating s from the concave chain. If we knew e or v *a priori*, we could reduce the intersection number of s by placing e or the edges incident to v near the top of the tree. Of course, we don't know *a priori* where the query segment s

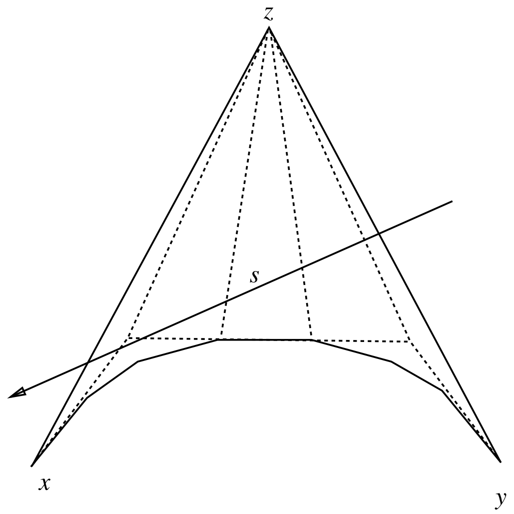

Figure 5: Proof of Lemma 3.2.

is going to intersect. Nevertheless, this idea suggests the use of a weight-balanced tree to guide the triangulation of our boomerangs. We will associate real-valued weights with the edges of geodesic triangles and triangulate each boomerang according to a weight-balanced binary tree. We show that, with a proper set of weights,

$$\sum_e depth(e) + \sum_v depth(v) \quad = \quad O(\log n),$$

where the first sum runs over all boomerang edges e intersected by s and the second sum runs over all minimum-depth boomerang vertices v that support tangents separating their concave chains from s. Of course, the segment s may still hit $\Omega(\log n)$ triangles in a single boomerang, but we prove an amortized bound of $O(1)$ triangles intersected per boomerang.

The next two sections present the main contribution of our paper. Section 4 describes our choice of weights, and Section 5 presents a proof that a triangulation with these weights indeed has intersection number $O(\log n)$ for any segment internal to the polygon. Section 6 then shows that our triangulation can be built in worst-case time $O(n)$.

4 Weight Assignment

4.1 Ordered weight-balanced trees

We assign a non-negative, real-valued weight to each edge of the concave chain C of a boomerang B. If the chain C consists of edges e_i, for $i = 1, 2, \ldots, k$, then a weight w_i is assigned to e_i subject to the following two conditions:

$$w_i \quad \geq \quad 1/(3n). \tag{1}$$

$$\sum_i w_i \quad = \quad 1. \tag{2}$$

The first condition puts a lower bound on individual weights so that the height of our tree is logarithmic, and the second condition is just for normalization. We can now build a binary tree $\mathcal{W}$ whose nodes in symmetric order correspond to the edges of C and that is *weight-balanced* in the following sense: *the weight of a subtree rooted at a node x is at most half the weight of the subtree rooted at the parent of x.* We call $\mathcal{W}$ an *ordered weight-balanced tree* for C with weights w_i.

Such a tree can be built easily in a top-down manner, as follows. Pick the unique i, for $1 \le i \le k$, such that

$$\sum_{j=1}^{i-1} w_j \le \frac{1}{2}\sum_{j=1}^{k} w_j \qquad \text{and} \qquad \sum_{j=1}^{i} w_j > \frac{1}{2}\sum_{j=1}^{k} w_j.$$

We make e_i the root of $\mathcal{W}$, and then recursively build the left and right subtrees corresponding to the chains $(e_1, e_2, \ldots, e_{i-1})$, and $(e_{i+1}, e_{i+2}, \ldots, e_k)$, respectively. We can build $\mathcal{W}$ in $O(k)$ time by storing the prefix sums of the weights in an array, then finding the root of each subtree by a dovetailed doubling search on the corresponding subarray. For example, at the top level we find the splitting weight w_i as follows:

> Initialize $j := 0$;
> **while** $2^j < i < k - 2^j$ **do** $j := j + 1$;
> Find i by binary search in $[1, 2^j]$ or $[k - 2^j, k]$, as appropriate.

The construction time obeys the recurrence $f(k) = f(i-1) + f(k-i) + \log(\min(i, k-i))$, which solves to $f(k) = O(k)$. The following property of these weight-balanced trees is easily proved.

Lemma 4.1 *An edge with weight w_i is stored at depth at most $\lceil -\log w_i \rceil$ in $\mathcal{W}$.*

Proof: The lemma follows readily from the condition that the total weight of a subtree rooted at a node of depth d is at most 2^{-d}. ∎

4.2 An overview of the weights

Consider a geodesic triangle τ in the geodesic triangulation of P. In general, τ has a parent $p(\tau)$ and two children τ_1 and τ_2. (The exceptional cases are the triangle corresponding to the root of GDT, which has no parent and has three children, and the nodes at the bottom two levels of the tree, which have zero, one, or two children.)

Of the three concave chains of τ, the one sharing an edge with $p(\tau)$ is called the *parent chain* of τ; it is possible that τ shares an edge only with the kite corresponding to $p(\tau)$, and not the parent geodesic triangle itself, but we won't need to make a distinction between these two cases. The remaining (at most) two chains of τ are called the *child chains* of τ.

Our weight scheme assigns three kinds of weights to the edges of a geodesic triangle:

1. w_b: the bay-size weight,
2. w_u: the up-tangent weight, and
3. w_d: the down-tangent weight.

Roughly speaking, the bay-size refers to the number of edges in the subpolygon (bay) cut off by the edge, and the up-tangent (down-tangent) refers to a tangent between a chain and a chain of an ancestor (descendant) triangle.

Within each geodesic triangle, edges of the parent chain receive only the bay-size weights, while edges of a child chain may receive all three kinds of weights. The chains of the root triangle also receive only bay-size weights. For each child chain of a non-root triangle, we normalize each of the three weights independently, so that the bay-size, up-tangent, and down-tangent weights assigned to a chain individually sum up to one.

With this normalization, the weight $w(e)$ of a parent-chain edge e equals its bay-size weight:

$$w(e) \;=\; w_b(e).$$

The same holds for all three chains of the root triangle. The weight $w(e)$ of a child chain edge e equals the average of its three weights:

$$w(e) \;=\; \frac{w_b(e) + w_u(e) + w_d(e)}{3}.$$

The next few subsections explain the precise details of these weight assignments.

4.3 The bay-size weights

Let C be a chain of the geodesic triangle τ, and let $e_1, e_2, \ldots, e_k$ be the counterclockwise enumeration of its edges. Each edge e_i partitions the polygon P into two subpolygons, one containing τ and the other excluding τ. The subpolygon that *excludes* τ is called the *bay of* e_i, and its size (the number of polygon edges) is denoted by b_i. Figure 6 illustrates these definitions. (Note that the bay-size is meaningful only in the context of an accompanying geodesic triangle. In general, an edge has two bays, one for each of the two geodesic triangles containing it; in fact, the two bays are complements of each other.)

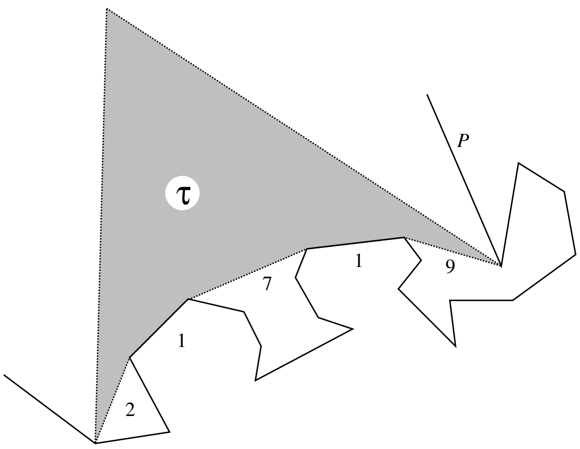

Figure 6: The bay-size weights. Numbers next to the edges of τ's concave chain indicate their bay sizes.

The bay-size weight of e_i is defined as

$$w_b(e_i) \;=\; b_i/\sigma,$$

where $\sigma = \sum_{i=1}^{k} b_i$ is the sum of all the bay-sizes for C.

We observe that since each bay has at least one edge, $b_i \geq 1$. Furthermore, since the bays corresponding to the edges of C are pairwise disjoint, $\sigma \leq n$. Thus the bay-size weight is never less than $1/n$.

Before we can describe the assignment of up-tangent and down-tangent weights, we need to explain the concept of an admissible outer common tangent between two chains.

4.4 Admissible outer common tangents

Consider a geodesic triangle τ and one of its child chains C_τ. Let α be another geodesic triangle that is an ancestor of τ in GDT. Let $C(\alpha, \tau)$ denote the child chain of α that is *not* an ancestor of τ; since τ descends from exactly one of the two child chains of α, the chain $C(\alpha, \tau)$ is well-defined. We call an edge $\ell = (u, v)$ the *admissible outer common tangent* (ACT) of τ and α if

1. the edge ℓ is locally tangent to the chains $C(\alpha, \tau)$ and C_τ, respectively, at vertices u and v,
2. the chains $C(\alpha, \tau)$ and C_τ lie on the same side of ℓ, and
3. the edge ℓ lies completely inside the polygon P.

In the definition of ACT, conditions (1), (2), and (3) imply tangency, *outer* tangency, and admissibility (visibility), in that order. Notice that two nonintersecting concave chains always admit a tangent. An ACT, however, needs to satisfy two additional constraints: it must be an outer tangent (supporting the two chains on the same side), and it must be contained entirely inside the polygon.

4.5 The up-tangent weights

Pick a geodesic triangle τ and one of its child chains C. We compute admissible outer common tangents from τ to all of its (at most $\log n$) ancestors, and focus on the tangents incident to the chain C. We label these tangents consecutively in the ancestor-order: if $\tau_1, \tau_2, \ldots, \tau_m$ are the ancestors of τ that share an admissible outer common tangent with C (ordered from lowest to highest, with $m \leq \log n$), then the ACT from C to τ_j is labeled ℓ_j, for $j = 1, 2, \ldots, m$. It is easy to see that the ℓ_j's appear in sorted order around C. See Figure 7. We assign up-tangent weights to the edges of C based on these tangent labels.

The admissible common tangent ℓ_j, $1 \leq j \leq \log n$, carries a weight of 2^{-j}, which it assigns to the vertex of C incident to ℓ_j. This vertex in turn distributes the weight evenly among the (up to two) edges of C incident to it. More precisely, if ℓ_j is tangent to C at vertex v, then v receives a weight of 2^{-j}. If v is an interior vertex of C, each of the two edges of C incident to v receives a weight of 2^{-j-1}; otherwise, v is an endpoint of C and the only edge incident to it receives a weight of 2^{-j}.

Since a vertex may be incident to several ACTs, an edge may receive multiple weights. The up-tangent weight of an edge equals the sum of all up-tangent weights contributed to it by its endpoints. Finally, we normalize the weights so that the up-tangent weights for all edges of C sum to one.

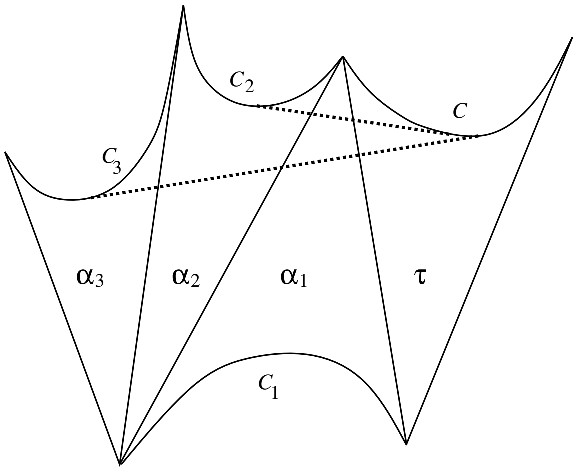

Figure 7: The up-tangent weights. The geodesic triangle τ has three ancestors α_1, α_2, and α_3; only the latter two admit ACTs with τ's chain C. Labeling by the ancestor-order, the tangent to C_2 is labeled ℓ_1, and the tangent to C_3 is labeled ℓ_2.

Lemma 4.2 *Let $\mathcal{W}$ be the ordered weight-balanced tree for a chain C with normalized up-tangent weights on its edges. If an edge $e \in C$ shares an endpoint with the j'th admissible up-tangent of C, then e has depth at most $j+1$ in $\mathcal{W}$.*

Proof: The pre-normalized up-tangent weight of e is at least 2^{-j-1}, and pre-normalized up-tangent weights for all the edges of C add up to at most one, because

$$\sum_{j \geq 1} 2^{-j} \;=\; 1.$$

Thus, the normalized up-tangent weight of e also is at least 2^{-j-1}, and the bound on the depth of e follows from Lemma 4.1. ∎

4.6 The down-tangent weights

The down-tangent weights are assigned according to admissible common tangents between a chain and its descendants. Let us consider a geodesic triangle τ and one of its child chains C. Suppose that the depth of τ in GDT is $i > 0$. Then τ has at most $n/2^i$ descendants (an overestimate). We compute admissible outer common tangents between C and all descendants of τ. Suppose τ' is a descendant of τ at depth j, for $i < j \leq \log n$, that admits an ACT $\ell = (u, v)$ from C, where u is the endpoint of ℓ in C. We assign a weight of 4^{i-j} to u, which then distributes it evenly among the (at most two) edges incident to it. Thus, if u is an interior vertex of C, each of the two edges incident to u receives a weight of $\frac{1}{2} \cdot 4^{i-j}$; otherwise, u is an endpoint of C and the only edge incident to it receives a weight of 4^{i-j}.

Again, since a vertex u may be an endpoint of several ACTs incident to C, an edge may receive more than one weight. The *down-tangent* weight of an edge is the sum of all down-tangent weights contributed to it by its endpoints. In the end, we normalize the down-tangent weights so that the down-tangent weights for all edges of C sum to one.

Lemma 4.3 *Let C be a chain of a geodesic triangle τ that does not correspond to the root of GDT, and let $\mathcal{W}$ be the ordered weight-balanced tree for C with normalized down-tangent weights on its edges. If an edge $e \in C$ shares an endpoint with an admissible common tangent reaching a descendant j levels lower, then e has depth at most $2j$ in $\mathcal{W}$.*

Proof: The pre-normalized down-tangent weight of e is at least $\frac{1}{2} \cdot 4^{-j}$. The pre-normalized down-tangent weights for all the edges of C sum up to at most $1/2$: There are at most 2^j descendants of τ that are j levels below it, but only at most 2^{j-1} of these are descendants of the child chain that is not C. Each such descendant has at most one child chain that may form an outer tangent with C, contributing a weight of 4^{-j}. The total weight is at most

$$\sum_{j>0} 2^{j-1} \cdot 4^{-j} = \frac{1}{2}.$$

Thus, the normalized down-tangent weight of e is at least 4^{-j}, and so e's depth in $\mathcal{W}$ is at most $\log(4^j) = 2j$ (Lemma 4.1). ∎

This completes our discussion of the weight assignments. At this point, we have computed edge weights for the concave chains of all boomerangs. Recall again that an edge in the parent chain receives only a bay-size weight, while edges in child chains receive the average of their bay-size, up-tangent, and down-tangent weights. Consider a boomerang B, and suppose that $C = (e_1, e_2, \ldots, e_k)$ is its concave chain. Let $\mathcal{W}$ denote the ordered weight-balanced tree corresponding to C, with weights $w(e_i)$.

Lemma 4.4 *$\mathcal{W}$ has height $O(\log n)$.*

Proof: The key observation is that $w(e) \geq 1/(3n)$, for any $e \in C$. This follows from the facts that $w_b(e) \geq 1/n$, and that $w(e) \geq w_b(e)/3$. Thus, the depth of any edge e in $\mathcal{W}$ is at most $\lceil \log(3n) \rceil$. ∎

We use the ordered weight-balanced tree associated with each boomerang to guide its triangulation; each boomerang is triangulated independently. Let S denote the resulting (Steiner) triangulation of the polygon P. We show in the following section that a ray can intersect at most $O(\log n)$ triangles in S.

5 Analysis of Ray Shooting

We consider an arbitrary segment s internal to the polygon P, and show that s intersects at most $O(\log n)$ triangles of our triangulation S. Let us first recall from Lemma 2.2 that s intersects at most $2 \log n$ geodesic triangles in the balanced geodesic triangulation of P. The nodes corresponding to these geodesic triangles lie on a single path in GDT. We split this path at the topmost node to obtain two descending subpaths, each with at most $\log n$ nodes. We analyze the intersection number of s with respect to one of these subpaths. Let $\Gamma = (\tau_0, \tau_1, \ldots, \tau_k)$, where $k < \log n$, denote the sequence of geodesic triangles corresponding to one subpath; assume that τ_0 is the ancestor of all other τ_i's in this sequence. We start our analysis with some preliminary facts about these geodesic triangles.

5.1 Starting up

Without loss of generality, let us assume that s is a horizontal ray, directed from left to right; the leftmost (resp. rightmost) geodesic triangle intersected by s is τ_0 (resp. τ_k). Let e_i denote the edge of the *parent* chain of τ_i through which s enters τ_i, for $i = 1, 2, \ldots, k$. (Notice that e_i is also the edge through which s exits τ_{i-1}; however, e_i belongs to a child chain of τ_{i-1}.) Let b_i denote the bay-size of e_i in τ_i. The following lemma shows that the b_i's are monotonically increasing, and almost all of them are large.

Lemma 5.1 *Let τ_i and b_i be defined as above, for $i = 0, 1, \ldots, k$. Then (i) $b_1 < b_2 < \cdots < b_k$, and (ii) $b_2 > n/3$.*

Proof: To prove the first part, we observe that the edges $e_1, e_2, \ldots, e_k$ occur in left to right order along s. Since P is a simple polygon, the e_i's must form a nested sequence within P, and since $bay(e_i)$ lies to the left of e_i, we have

$$bay(e_1) \subset bay(e_2) \subset \cdots bay(e_k).$$

This clearly implies that $b_1 < b_2 < \cdots < b_k$.

To prove the second part, let us assume that τ_0 is the root of GDT; this results in no loss of generality since the first part has shown that bay-sizes increase monotonically. Let $\kappa_0 = (x, y, z)$ denote the kite corresponding to the root geodesic triangle τ_0. See Figure 8. This kite partitions the boundary of P into three equal-sized pieces $P(x, y)$, $P(y, z)$, and $P(z, x)$. Let P_{xy} denote the region enclosed by $P(x, y)$ and the geodesic path between x and y. Similarly, define P_{yz} and P_{zx}. Notice that the interiors of P_{xy}, P_{yz}, and P_{zx} are pairwise disjoint. Without loss of generality, assume that $\tau_1, \tau_2, \ldots, \tau_k$ lie in P_{yz}, and that τ_1 is produced by the kite $\kappa_1 = (u, v, w)$, where all three vertices belong to the chain $P(y, z)$, and $v \in P(u, w)$. Assume that s leaves τ_1 by crossing the geodesic path between u and v.

We now observe that the region P_{uv} is disjoint from at least one of P_{xy} and P_{zx}. Either P_{xy} or P_{zx} is contained in $bay(e_i)$, for $i > 1$. The claim now follows since $|P_{xy}| = |P_{zx}| = n/3$. ∎

With these preliminaries in place, we are now ready to start our analysis. We first bound the contribution of certain miscellaneous triangles, namely, the triangles in the top two geodesic triangles τ_0, τ_1, and the triangles introduced in the decomposition of geodesic triangles into boomerangs. Our algorithm partitions each geodesic triangle into a constant number of boomerangs and triangles. Since s intersects only $O(\log n)$ geodesic triangles, the total number of these triangles intersected by s is at most $O(\log n)$. Next, since the depth of each weight-balanced tree is $O(\log n)$ by Lemma 4.4, the maximum number of triangles intersected by s in any geodesic triangle is $O(\log n)$: there are at most three boomerangs per geodesic triangle and, by Lemmas 3.1 and 3.2, the maximum number of triangles intersected by s in a boomerang is linearly related to the height of the weight-balanced tree. Thus, the total contribution of the top two geodesic triangles τ_0 and τ_1 is $O(\log n)$.

We now come to the main component of our analysis, which is to bound the number of triangles intersected by s in the $O(\log n)$ boomerangs associated with the sequence of

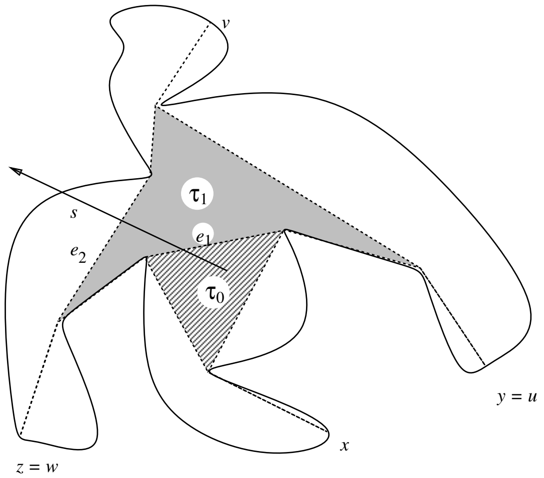

Figure 8: The proof of Lemma 5.1.

geodesic triangles Γ. We introduce one more piece of notation to facilitate this part of the analysis. We call a boomerang B a *home-in* boomerang if s intersects its concave chain, and a *fly-by* boomerang if s intersects B but does not intersect its concave chain. Each geodesic triangle of Γ is decomposed into three boomerangs, two of which are home-in and at most one of which is fly-by. (This classification of boomerangs is dependent on the segment s under consideration.) We count separately the number of triangles that s intersects in home-in and fly-by boomerangs.

5.2 Analysis for the home-in boomerangs

Consider a geodesic triangle $\tau_i \in \Gamma$ intersected by s, where $i \geq 2$. Suppose that s enters τ_i through an edge e_i and leaves it through an edge e_{i+1} (Figure 9). Let B_1 and B_2 denote the boomerangs in τ_i that contain e_i and e_{i+1}, resp., and let b_i and $\overline{b}_{i+1}$ denote the bay sizes of e_i and e_{i+1} in τ_i, resp. (Notice that e_i belongs to the parent chain of τ_i, while e_{i+1} belongs to a child chain. For a single edge e_i, b_i denotes the weight when s enters τ_i through e_i; $\overline{b}_i$ denotes the weight when s leaves τ_{i-1} through e_i. It follows that $b_i + \overline{b}_i = n$.)

We first note that e_i has a large weight. This follows from Lemma 5.1, which says that $b_i > n/3$, and since for an edge of the parent chain the weight equals its bay-size weight, we have

$$w(e_i) \; > \; 1/3. \tag{3}$$

Eq. 3 implies that the depth of e_i in B_1's weight-balanced tree is at most $O(1)$, and therefore s intersects only a constant number of triangles in B_1 (by Lemma 3.1).

Next, let σ denote the sum of all bay-sizes in the concave chain of B_2; thus the bay-size weight of e_{i+1} is $w_b(e_{i+1}) = \overline{b}_{i+1}/\sigma$. Since the bays that contribute to σ are contained in the bay that defines $\overline{b}_i$ in τ_{i-1}, we have the following upper bound on σ:

$$\sigma \leq \overline{b}_i.$$

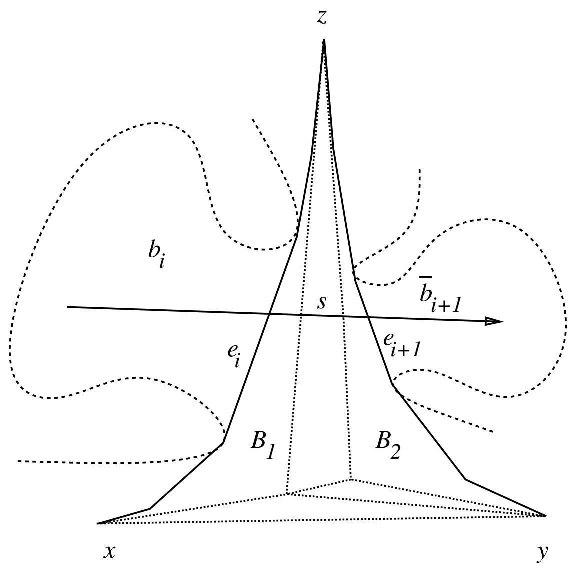

Figure 9: The analysis for a home-in boomerang.

It follows that $w_b(e_{i+1}) \geq \overline{b}_{i+1}/\overline{b}_i$, or that the total weight of e_{i+1} satisfies:

$$w(e_{i+1}) \;\; \geq \;\; \frac{\overline{b}_{i+1}}{3\overline{b}_i}. \tag{4}$$

Eq. 4, Lemma 3.1, and Lemma 4.1 together imply that s intersects at most $5\log(\overline{b}_i/\overline{b}_{i+1}) + O(1)$ triangles in B_2, and this holds for all geodesic triangles τ_i, $i > 1$. The total number of triangles that s intersects in the home-in boomerangs of all geodesic triangles τ_i, $i = 2, 3 \ldots, k$, is given by the telescoping sum

$$5\sum_{i=2}^{k}(\log \overline{b}_i - \log \overline{b}_{i+1} + O(1)) \;=\; 5(\log \overline{b}_2 \;-\; \log \overline{b}_{k+1}) \;+\; O(\log n).$$

We conclude that s intersects at most $O(\log n)$ triangles in all the home-in boomerangs. Next we analyze the fly-by boomerangs.

5.3 Analysis for the fly-by boomerangs

Imagine again that the segment s is drawn horizontally, with the geodesic triangles $\tau_0, \tau_1, \ldots, \tau_k$ numbered left-to-right. There are two kinds of fly-by chains to consider: those lying above s and those lying below s. We give the analysis for the fly-by chains lying below s; the other case is symmetric.

Label these fly-by chains $C_0, C_1, \ldots, C_k$ in left-to-right order. Now consider the *upper convex hull* CH of the union of these chains (Figure 10). We call a chain C_i a *hull* chain if C_i has a vertex on CH. Otherwise, C_i is called a *hidden* chain. Let $D_0, D_1, \ldots, D_m$, where $m \leq k$, be the left-to-right labeling of the hull chains of CH. We first count the number of triangles intersected in the boomerangs associated with the hull chains D_i, $i = 0, 1, \ldots, m$.

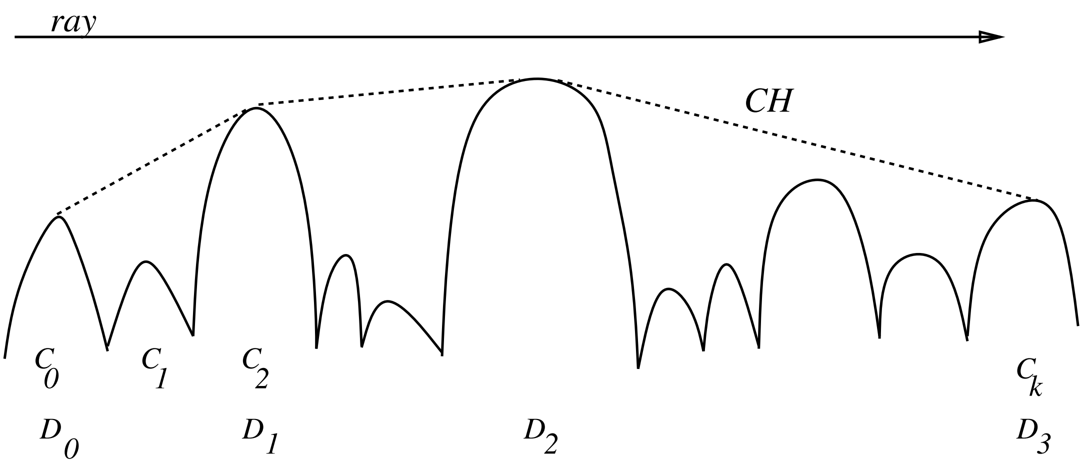

Figure 10: The upper convex hull CH (shown by dotted lines) of the fly-by chains. The hull chains are $D_0 = C_0$, $D_1 = C_2$, $D_2 = C_5$, and $D_3 = C_k$.

5.3.1 The hull chains

Let D_t be the hull chain closest to s; if two chains are closest to s, pick one arbitrarily. We call D_t the *top* chain of CH. The number of triangles intersected by s in D_t's boomerang is clearly $O(\log n)$, by Lemmas 4.4 and 3.2.

To bound the number of triangles intersected in the boomerangs of the remaining hull chains, we define the *neighbor* of a hull chain $D_i \neq D_t$ to be D_{i+1} if $i < t$, and D_{i-1} if $i > t$. Further, let d_i be the depth in GDT of the geodesic triangle containing D_i. Since the τ_i's are ordered left-to-right by depth, we have $d_0 < d_1 < \cdots < d_m$.

Consider a hull chain D_i. The upper hull construction guarantees that the tangent ℓ_i between D_i and its neighbor separates D_i from s inside the boomerang associated with D_i. Observe that ℓ_i is an admissible outer common tangent, and hence each edge of D_i incident to an endpoint of ℓ_i receives a weight of at least $\frac{1}{3} \cdot 4^{d_i - d_{i+1}}$ if $i < t$ (down-tangent weight) and at least $\frac{1}{3} \cdot 2^{d_{i-1} - d_i - 1}$ otherwise (up-tangent weight). Thus, in the weight-balanced tree of D_i, the shallowest vertex that supports a tangent separating D_i from s has depth at most $2|d_i - d_{i+1}| + O(1)$ if $i < t$ and at most $|d_i - d_{i-1}| + O(1)$ otherwise. By Lemma 3.2, therefore, s intersects $O(|d_i - d_{i-1}|)$ triangles in the boomerang associated with D_i. The total number of triangles intersected in all hull-chain boomerangs is bounded by the following telescoping sum:

$$O\left(\sum_{i=1}^{\log n} (2|d_i - d_{i-1}| + O(1))\right) = O(\log n).$$

The only remaining part of the analysis is to bound the number of intersections in the fly-by hidden chains.

5.3.2 The hidden chains

Every hidden chain lies between two consecutive hull chains. Let us consider two consecutive hull chains C_a and C_b, with C_a an ancestor of C_b. We draw in all the ACTs (up-tangents) incident to C_b that end at the chains strictly between C_a and C_b. Let $\ell_1, \ell_2, \ldots, \ell_h$ be the resulting sequence of up-tangents, with ℓ_i incident to the chain $C_{\alpha(i)}$. ($C_{\alpha(1)}, C_{\alpha(2)}, \ldots, C_{\alpha(h)}$ is a subsequence of the hidden chains between C_a and C_b; define $C_{\alpha(0)} = C_a$.) We repeat the process recursively for every pair of consecutive chains $\{C_{\alpha(i)}, C_{\alpha(i+1)}\}$, $i = 0, 1, \ldots, h-1$, that is, we recursively draw in the admissible common tangents incident to $C_{\alpha(i+1)}$ and some chain lying strictly between $C_{\alpha(i)}$ and $C_{\alpha(i+1)}$. Finally, we also add in the convex hull ACTs, namely, the tangents between adjacent hull chains. See Figure 11.

The resulting graph structure is a tree whose nodes are the chains C_i and whose root is $C_k = D_m$. If a node C_i is a child of C_j, then the ACT between C_i and C_j is an up-tangent of C_j. We call this tree the *tree of tangents*. Define $\Delta(C)$ to be the number of children of a node C in the tree of tangents.

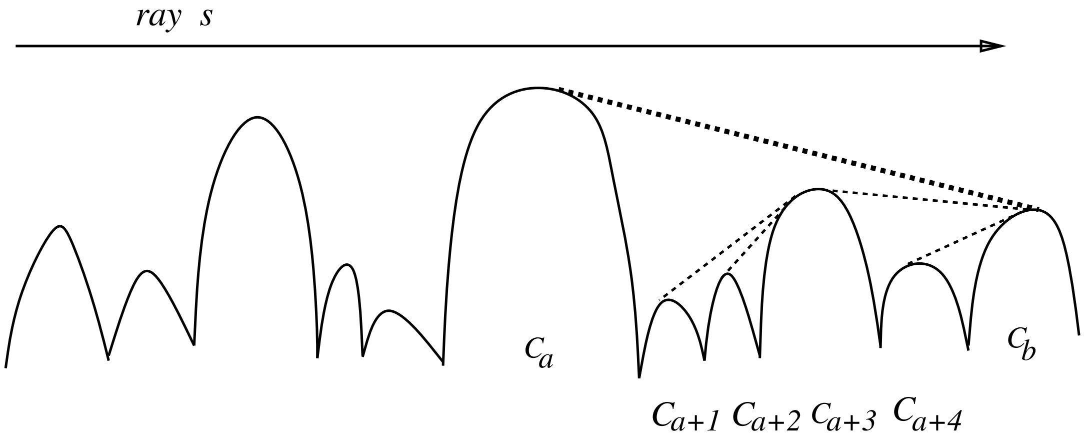

Figure 11: The tree of tangents. The node C_b has three children, C_a, C_{a+3}, C_{a+4}, and the node C_{a+3} has two children, C_{a+1} and C_{a+2}. Thus, $\Delta(C_b) = 3$ and $\Delta(C_{a+3}) = 2$. C_a is the left sibling of C_{a+3}.

Lemma 5.2 *Let C be a hidden chain, and let C' be the first fly-by chain to the left of C that is not a descendant of C in the tree of tangents. Then there is an ACT ℓ between C and C', and ℓ is the $(1 + \Delta(C))$'th up-tangent incident to C.*

Proof: Let $lca(C, C')$ be the lowest common ancestor of C and C' in the tree of tangents; $lca(C, C')$ is the parent of C'. Define $X(C)$ to be the set of nodes strictly above C in the tree of tangents and lying on the tree path from C to $lca(C, C')$. No node in $X(C)$ has an ACT to a chain left of C and right of C'. It follows that the common tangent ℓ between C and C' is not obstructed by any chain, since that would imply the existence of another ACT from a node in $X(C)$ to a chain between C and C'. Finally, by construction, if a chain C'' lying left of C and right of C' admits a common tangent from C, then C'' is a child of C in the tree of tangents. Thus C has exactly $\Delta(C)$ up-tangents leading to chains between C and C', and so ℓ is the $(1+\Delta(C))$'th up-tangent incident to C. In Figure 11, for instance, C_{a+3} and its left sibling C_a admit an ACT, which is the third up-tangent of C_{a+3}. This completes the proof. ▮

Lemma 5.2 is the key to analyzing the number of intersections in the boomerangs associated with the hidden chains. We observe that the tangent ℓ found by Lemma 5.2 separates the chain C from s in the boomerang associated with C. Now, if ℓ is tangent to C at vertex v, then the up-tangent weight of v is at least $2^{-\Delta(C)-1}$, implying that the edges incident to v have depth at most $\Delta(C)+O(1)$ in the weight-balanced tree of C. By Lemma 3.2, therefore, s intersects at most $O(1+\Delta(C))$ triangles in the boomerang associated with the hidden chain C. The total number of triangles intersected by s in all the hidden-chain boomerangs is given by the sum

$$\sum_{C}(\Delta(C)+O(1)),$$

where the sum is over all nodes C in the tree of tangents. Since the sum of all out-degrees in a rooted tree is at most the number of nodes, and since the tree of tangents has at most $\log n$ nodes (one per fly-by chain), this sum is at most $O(\log n)$.

This completes our analysis. We have established the following theorem.

Theorem 5.3 *A simple polygon with n vertices can be triangulated, using $O(n)$ Steiner points, such that a ray intersects at most $O(\log n)$ triangles in the polygon before meeting the boundary of the polygon.*

In the next section, we give a linear time algorithm for constructing our triangulation.

6 Building the Triangulation

Our Steiner triangulation can be constructed in $O(n)$ sequential time. The key to this construction is a linear time algorithm for building a balanced geodesic triangulation of the polygon, which was presented by Chazelle et al. [2]. We sketch only the main idea behind this algorithm.

We start with an arbitrary triangulation of the polygon, without Steiner points; such a triangulation can be computed in $O(n)$ time using a recent algorithm of Chazelle [1]. We then preprocess the polygon in linear additional time to build the shortest path data structure of Guibas and Hershberger [11, 13]. This data structure assigns a *height* to each diagonal of the triangulation, and lets us compute the shortest path between two vertices of the polygon in $O(h^2)$ time, where h is the height of the highest diagonal intersecting the shortest path; the data structure computes a compact representation of the shortest path, from which the actual path can be derived in time proportional to the number of edges in the path. It turns out that the sum of $O(h^2)$ over the $O(n)$ shortest paths needed for the balanced geodesic decomposition is also $O(n)$, which leads to a linear time construction of a balanced geodesic triangulation of the polygon.

The only remaining nontrivial task for building our Steiner triangulation is the computation of weights. The bay-size weights are easily computed in linear time. The up-tangent and down-tangent weights require the computation of admissible outer common tangents. Using the shortest path data structure, we can easily find a tangent between two concave chains, and check if it is an ACT, in $O(\log n)$ time. However, there are $O(n \log n)$ such tangent computations, corresponding to $O(\log n)$ ancestors for each of the $O(n)$ geodesic triangles, making the total cost $O(n \log^2 n)$. We use a standard trick to reduce this time to $O(n)$: compute the tangents for only the top $(\log n - 2 \log\log n)$ levels of the GDT. For the remaining geodesic triangles, we assign equal weights to all the edges.

The total cost of computing all the tangents between ancestor-descendant pairs of geodesic triangles is

$$O\left(\frac{n}{\log^2 n} \cdot \log n \cdot \log n\right).$$

The first term denotes the total number of geodesic triangles in the top $(\log n - 2 \log\log n)$ levels of the geodesic decomposition tree GDT, the second term is the number of ancestors, and the third term is the cost of computing an ACT.

For a given ray s, the total number of triangles intersected by s in the top $\log n - 2 \log\log n$ levels of geodesic triangles is still $O(\log n)$. But what about the geodesic triangles at the bottom $2 \log\log n$ levels? The key observation is that a geodesic triangle in these bottom levels has size $O(2^{2 \log\log n}) = O(\log^2 n)$. Thus, the ray can hit $O(\log(\log^2 n)) = O(\log\log n)$ triangles in each of these geodesic triangles. Recalling that all the geodesic triangles intersected by s lie on a path in GDT, we conclude that the total number of triangles intersected by s is bounded by

$$O(\log n) \; + \; O((\log\log n)^2) \; = \; O(\log n).$$

Theorem 6.1 *Given a simple polygon P on n vertices, in linear time we can compute a (Steiner) triangulation of P with the property that a ray intersects at most $O(\log n)$ triangles before meeting the boundary of P.*

The sequential algorithm can be extended to run in $O(\log n)$ parallel time using $O(n/\log n)$ processors. We compute an initial triangulation of the polygon with Goodrich's algorithm [9], which runs in $O(\log n)$ time with $O(n/\log n)$ processors in the CRCW (concurrent-read, concurrent-write) PRAM model. The rest of our algorithm runs under the less powerful CREW (concurrent-read, exclusive-write) model.

The parallel algorithm uses the same building blocks as the sequential algorithm. After triangulating the polygon, it builds a geodesic triangulation with Hershberger's algorithm [14]. Once the edge weights are known, the algorithm computes weight-balanced trees [14], builds boomerangs, and triangulates each boomerang based on its weight-balanced tree. To triangulate a boomerang with k sides in $O(\log n)$ time with $O(k/\log n)$ processors, the algorithm walks down the tree in levels, performing the obvious parallel recursive construction. Each subtree with $j > \log n$ nodes is assigned $\lceil j/\log n \rceil$ processors; smaller subtrees are handled in $O(\log n)$ time by a single processor. Boomerangs with fewer than $\log n$ edges are combined into groups with a total of $\Theta(\log n)$ edges, and each group is handled by a single processor. Parallel prefix is used to assign processors.

As in the sequential case, computing edge weights is the only remaining nontrivial task.

The bay-size edge weights are easy to compute in parallel: they can be found in constant time by $O(n)$ processors, each associated with one edge of the geodesic triangulation.

To compute the up-tangent and down-tangent weights, the algorithm builds $O(n/\log n)$ shortest paths, as in the sequential case. It uses Goodrich, Shauck, and Guha's shortest path query structure to find the shortest paths in $O(\log n)$ time apiece [10]. Each processor determines in parallel whether the shortest path between two chains contains an admissible common tangent.

There are only $O(\log n)$ up-tangents per chain, so it is easy to add the up-tangent weights to the appropriate bay-size weights in $O(\log n)$ time. There may be many down-tangents contributing to the down-tangent weight of a single edge, and so we need to be more careful in summing the down-tangent weights. To compute the down-tangents, the algorithm finds shortest paths that connect one vertex of a source kite to the middle vertex of a destination kite. The down-tangents for a particular chain are incident to the chain in the same order as their destination kites appear along the polygon boundary. This makes it possible to compute the total down-tangent weight of each edge in $O(\log n)$ time by a doubling scheme.

We have established the following theorem.

Theorem 6.2 *Given a simple polygon with n vertices, we can compute the Steiner triangulation described in Theorem 6.1 in $O(\log n)$ time using $O(n/\log n)$ parallel processors under the CRCW PRAM model.*

7 Extension to Multiple Polygons

Our method can be extended to the case of k disjoint polygons. In this case, we are able to compute a triangulation of the plane such that a ray intersects at most $O(\sqrt{k}\log n)$ triangles before hitting a polygon boundary; n denotes the total number of vertices in all the polygons. This extension is based on the technique of using a spanning tree of small stabbing number; Chazelle et al. [2] utilized this technique to perform ray shooting among k polygons in $O(\sqrt{k}\log n)$ time. We achieve the same time complexity, but our query algorithm is much simpler: our result guarantees that a walk along the ray traverses only $O(\sqrt{k}\log n)$ triangles before hitting a polygon boundary.

The basic idea behind the extension to k polygons is to use $(k-1)$ auxiliary edges to connect up these polygons into a single polygon. To find these auxiliary edges, we first pick a representative vertex in each of the k polygons, and compute a spanning tree T on these k vertices, such that a line intersects at most $O(\sqrt{k})$ edges of T. A spanning tree with this property can be found in $O(k^{3/2}\log k)$ time, using an algorithm by Matoušek [16]. Although the edges of T do not cross each other, they may be intersected by the boundaries of input polygons, causing the edges of T to be subdivided into smaller segments. The total number of these segments is $O(n\sqrt{k})$; this follows from the property that no line intersects more than $O(\sqrt{k})$ edges of T and the fact that there are at most n polygon edges. Each of these segments either lies completely inside a polygon or it lies completely outside. The segments that lie outside the polygons either connect the boundary points of the same polygon or of two distinct polygons. We discard all segments except for the last type; that is, we keep only the segments that connect boundaries of two different polygons. By using the union-find data structure, we can choose $k-1$ such segments that together connect all k polygons.

We can now convert this planar structure into a simple polygon $\mathcal{P}$. (One commonly used method is to draw a suitably large rectangle around the structure and connect the rectangle to the structure with an edge. In this way, the common exterior of all the polygons minus the newly added edges becomes a simple polygon.) Finally, we compute a Steiner triangulation of $\mathcal{P}$, using the result of Section 5. We claim that a ray intersects $O(\sqrt{k}\log n)$ triangles of this triangulation before hitting the boundary of one of the original k polygons.

We conduct a walk through the triangulation of $\mathcal{P}$, starting from the triangle containing the origin of the ray. When we hit the boundary of $\mathcal{P}$, we check the type of this boundary edge. If it is an edge of one of the original polygons, we are done—the ray shooting query can be answered. Otherwise, the edge is one of the auxiliary edges, introduced to connect up the polygons; in this case, we resume our walk, counting it as a new ray shooting query. Since all the auxiliary edges are contributed by the spanning tree T, the second event can occur only as many times as a ray can intersect the edges of T, which is $O(\sqrt{k})$. Between two such events, the ray intersects only $O(\log n)$ triangles, from the simple polygon result. Thus, the total number of triangles intersected by a ray is $O(\sqrt{k}\log n)$.

Finally, we analyze the preprocessing time for the construction. We compute the spanning tree T in time $O(k^{3/2}\log k)$. We can find the $O(n\sqrt{k})$ intersections between the polygons and spanning tree edges in time $O(n\sqrt{k}+n\log n)$, by triangulating the interiors and the common exterior of all the polygons and tracing each edge of T through this triangulation. We can identify the segments along each tree edge and perform the union-find computation in $O(n\sqrt{k})$ time, then build the Steiner triangulation of $\mathcal{P}$ in $O(n)$ additional time. We conclude with the following theorem.

Theorem 7.1 *Given a family of k disjoint polygons in the plane, with a total of n vertices, we can find a triangulation of the plane such that a ray intersects at most $O(\sqrt{k}\log n)$ triangles before hitting the boundary of one of the polygons. The preprocessing time for building the triangulation is $O(n\sqrt{k}+n\log n+k^{3/2}\log k)$.*

8 Conclusions

We have presented a simple algorithm for ray shooting in polygons. Our algorithm is based on a (Steiner) triangulation of the polygon with the property that a ray can hit at most a logarithmic number of triangles before meeting the boundary of the polygon. This property permits a remarkably simple algorithm for answering ray shooting queries: locate the triangle containing the source of the ray and then walk through the triangulation in the direction of the ray until the boundary is met. We suspect that this property of our triangulation may find other applications, for example in mesh generation for the finite element method.

Our triangulation has already been applied by Mount to answer intersection queries for translates of preprocessed polygons [17]. The triangulation is also the basis of an output-sensitive construction of an arrangement clipped to the interior of a simple polygon. In particular, we can build the arrangement of d diagonals inside a simple polygon in $O(n+d\log n+k)$ time, where k is the output size (joint work with Leo Guibas).

References

[1] B. Chazelle. Triangulating a simple polygon in linear time. *Discr. Comput. Geom.*, 6:485–524, 1991.

[2] B. Chazelle, H. Edelsbrunner, M. Grigni, L. Guibas, J. Hershberger, M. Sharir, and J. Snoeyink. Ray shooting in polygons using geodesic triangulations. In *Proc. of the 18th ICALP*, pages 661–673. Springer-Verlag, 1991.

[3] B. Chazelle and L. Guibas. Visibility and intersection problems in plane geometry. *Discr. Comput. Geom.*, 4:551–581, 1989.

[4] B. Chazelle and L. J. Guibas. Fractional cascading: I. A data structuring technique. *Algorithmica*, 1:133–162, 1986.

[5] B. Chazelle and L. J. Guibas. Fractional cascading: II. Applications. *Algorithmica*, 1:163–191, 1986.

[6] K. Clarkson. Approximation algorithms for planar traveling salesman tours and minimum length triangulations. In *Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 17–23, 1991.

[7] D. P. Dobkin and D. G. Kirkpatrick. Fast detection of polyhedral intersections. *Theoretical Comput. Sci.*, 27:241–253, 1983.

[8] H. Edelsbrunner, L. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM J. Comput.*, 15:317–340, 1986.

[9] M. Goodrich. Planar separators and parallel polygon triangulation. In *Proceedings of the 24th ACM Symposium on Theory of Computing*, pages 507–516, 1992.

[10] M. T. Goodrich, S. B. Shauck, and S. Guha. Parallel methods for visibility and shortest path problems in simple polygons. In *Proceedings of the 6th ACM Symposium on Computational Geometry*, pages 73–82, 1990.

[11] L. Guibas and J. Hershberger. Optimal shortest path queries in a simple polygon. *J. Comput. Syst. Sci.*, 39(2):126–152, 1989.

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

[13] J. Hershberger. A new data structure for shortest path queries in a simple polygon. *Inf. Process. Lett.*, 38:231–235, 1991.

[14] J. Hershberger. Optimal parallel algorithms for triangulated simple polygons. In *Proceedings of the 8th ACM Symposium on Computational Geometry*, pages 33–42, 1992.

[15] D. Kirkpatrick. Optimal search in planar subdivisions. *SIAM J. Comput.*, 12:28–35, 1983.

[16] J. Matoušek. More on cutting arrangements and spanning trees with low stabbing number. Technical Report B-90-2, Freie Universität Berlin, 1990.

[17] D. M. Mount. Intersection detection and separators for simple polygons. In *Proceedings of the 8th ACM Symposium on Computational Geometry*, pages 303–311, 1992.