A Simpler Circular Ray Shooting Algorithm

Ralph P. Boland*† Jorge Urrutia‡

Abstract

The circular ray shooting problem is to preprocess a simple polygon P so that, for a query directed circular arc ζ, one can quickly compute the first intersection point (if any) of ζ on P. We present a solution to this problem with the same preprocessing time, space, and query time complexities as the best known algorithm but our algorithm is simpler. Our algorithm is also more modular and uses tools for which we expect to find further applications. In the electronic version of this paper we also provide a modification that allows the worst case preprocessing time of both our circular ray shooting algorithm and the best known algorithm to be reduced by an $O(\log n)$ factor.

1 Introduction

Let P be an n vertex simple polygon. Given a directed circular arc ζ, we denote the starting point of ζ by ζ_o and the endpoint of ζ by ζ_e. We use $\hbar_\zeta(P)$ to indicate the first point on $\zeta - \zeta_o$ to intersect the boundary of P in a directed traversal of ζ; if there is no such point then we give $\hbar_\zeta(P)$ the value ζ_e. We denote by $\zeta(P)$ the portion of ζ between ζ_o and $\hbar_\zeta(P)$, by $\text{circle}(\zeta)$ the circle of which ζ is an arc, and by ζ_c the center of $\text{circle}(\zeta)$. Consider the following problem:

Problem 1.1. Preprocess P so that, for a query arc ζ, with ζ_o interior to P, one can quickly determine $\hbar_\zeta(P)$.

We call this problem the circular ray shooting problem. For reasons that are explained in [4] and [2], we shall assume (without loss of generality) that the arcs used in this paper are x-monotone and directed from left to right with ζ_c below ζ, and that the angle of any tangent to the arc is between -$\pi/4$ and $\pi/4$.

The first solution to Problem 1.1 was given in [1] where it was shown that in $O(n\log^3 n)$ time a data structure requiring $O(n\log^3 n)$ space can be constructed with which a circular ray shooting query (henceforth called a query) can be performed on P in $O(\log^4 n)$ time. This result was improved upon in [4] with an algorithm in which P is preprocessed in $O(n\log^2 n)$ time to construct a data structure requiring $O(n\log n)$ space so that a query can be performed in $O(\log^2 n)$ time; this paper also provides a number of applications of circular ray shooting. We reported in [3] that [4] contains an error but this claim is based upon a misinterpretation and this error does not exist.

A major part of the results in [4] is a data structure, called a Hierarchical Vertical Decomposition, which is used to solve Problem 1.1. The main contribution of this paper is an alternate data structure that can be also be used to solve Problem 1.1 in the same complexities as in [4]. The main advantage of our data structure it that it is much simpler. It also has the advantage of being built from simpler components that we believe will find other applications. In the electronic version of this paper we also provide a modification to one of the sub-algorithms in [4] that allows the preprocessing time of both the circular ray shooting algorithm in [4] and ours to be reduced by an $O(\log n)$ factor. Although we review some of the material in [1] and [4] we assume the reader is familiar with these papers.

In solving Problem 1.1 we use a much simpler version of the Hierarchical Vertical Decomposition data structure that is designed for monotone polygons. To apply our algorithm to P we use 2 new polygon partitioning algorithms that together partition P into x-monotone subpolygons such that any x-monotone curve interior to P, intersects at most $2\lfloor\log n\rfloor - 1$ subpolygons. This results in a circular ray shooting algorithm that is slower than the complexity we require by an $O(\log n)$ factor.

We remove this $O(\log n)$ factor using a new structure for applying range queries to trees; here we exploit that our partitioning algorithm produces a partition whose dual is a tree. The key reason that our algorithm is simpler despite needing these tools is that these tools are especially simple; alas the proofs that these tools work is more difficult and beyond the scope of this paper. The claims we make for our data structures and partitioning algorithms are established in [2].

For the remainder of this paper we will need some terminology. Let P be a simple polygon. We denote the first vertex of P by $\text{start}(P)$; in this paper $\text{start}(P)$ will always be a leftmost or rightmost vertex of P. We denote by $\Pi_\square(P)$ the trapezoids of a vertical trapezoidization of P and by $\Gamma_\square(P)$ the set of chords of P that induces $\Pi_\square(P)$. For any element γ of $\Gamma_\square(P)$, we denote by P_γ the subpolygon of P, of the 2 determined by γ, that does not contain $\text{start}(P)$. Let Q be any subpolygon of P determined by a connected subset of $\Pi_\square(P)$. We denote by $\text{door}(Q)$ the chord of P

*University of New Brunswick, Canada
email: rboland@unb.ca

†Funded in part by NSERC and the University of New Brunswick

‡Instituto de Mathematicas Universidad Nacional Autonoma de Mexico email: urrutia@math.unam.mx

on the boundary of Q such that $P_{\text{door}(Q)}$ contains Q; if start(P) is a vertex of Q then Q has no door. We denote by windows(Q) the chords of P on the boundary of Q other than door(Q). Let R and S be 2 subpolygons of the partition of P by any subset of the elements of $\Gamma_\Box(P)$ such that door$(S) \in$ windows(R). Then we say that R is the *parent polygon* of S and S is a *child polygon* of R.

The remainder of this paper is organized as follows. In Section 2 we discuss a method of solving Problem 1.1 on an x-monotone polygon and its extension to general polygons. In Section 3 we discuss a method of partitioning P into x-monotone subpolygons such that any x-monotone curve interior to the polygon, intersects at most $2\lfloor \log n \rfloor - 1$ of the subpolygons. In Section 4 we discuss a method of solving Problem 1.1 where the query arc intersects as many as $2\lfloor \log n \rfloor - 1$ x-monotone subpolygons yet the cost is only a small constant factor higher than for an x-monotone polygon of the same size. This method uses the centroid path partition of a tree [5] and a new weighted binary tree structure [2] which is applied to each centroid path of the partition.

2 Circular Ray Shooting in an X-Monotone Polygon

The data structures and algorithms in [4] and here for solving Problem 1.1 on an x-monotone polygon are nearly the same. The main differences between the method in [4] and our method are in how the x-monotone polygon case is generalized to handle simple polygons. In [4] the data structure itself is modified to deal with general polygons. We, instead, partition the polygon into x-monotone subpolygons and apply the data structure/algorithm multiple times.

Let Q be an h vertex x-monotone polygon whose leftmost points are on a vertical edge. We denoted this edge by $\ell(Q)$. If the rightmost points of Q are on a vertical edge then we denote the interior of this edge by $r(Q)$; otherwise $r(Q) = \emptyset$. We call a circular arc ζ a *canonical* arc of Q if ζ_o belongs to the interior of $\ell(Q)$. For any canonical arc ζ of Q we call a test to determine if $\hbar_\zeta(Q) \in r(Q)$ a *canonical arc intersection test* on Q. If $\hbar_\zeta(Q) \notin r(Q)$ then the result of the test is *positive;* otherwise it is *negative*. The following result gives us a canonical arc intersection testing tool.

Lemma 2.1. *[4] In $\Theta(h \log h)$ time we can construct a data structure of size $\Theta(h)$ that allows a canonical arc intersection test on Q to be done in $\Theta(\log h)$ time.*

With this tool and using divide and conquer, we can answer a query for Problem 1.1 on an m vertex x-monotone polygon in $O(\log^2 m)$ time. Let U be an m vertex x-monotone polygon.

The preprocessing stage involves the construction of a balanced rooted binary tree whose nodes are regions of U where each region is a connected subset of $\Pi_\Box(U)$. The leaves of the tree, from left to right, are the elements of $\Pi_\Box(U)$ ordered from left to right. Each nonleaf region R of the tree is the union of its 2 interior disjoint child regions. The root node is the region consisting of U. If 2 regions R_i and R_j with disjoint interiors share a portion of their boundaries then we say that R_i is a neighbor of R_j. Each region has a link to an ancestor node of at most one of its neighbor regions.[1] We call these links *neighbor ancestor links.* The tree has $\lceil \log m \rceil$ levels and each level requires $O(m)$ space so $O(m \log m)$ space is required in total.

We process be a query arc ζ of U where some trapezoid R_0 of $\Pi_\Box(R)$ contains ζ_o as follows. First, $\hbar_\zeta(R_0)$ is determined and if $\hbar_\zeta(U) = \hbar_\zeta(R_0)$ we are done. Otherwise a link at R_0 to a neighbor ancestor region, say R_1, of R_0 is followed and the maximal sub-arc, say ζ_1, of ζ that is canonical with respect to R_1 is determined. No part of ζ_1 is to the left of $\ell(R_0)$. A canonical arc intersection test is now performed with ζ_1 on R_1. If this test is negative then this process is repeated with R_1 replacing R_0 and in general R_{i+1} replacing R_i until a region, say R_k, of the tree is found such that the canonical arc intersection test on R_k is positive. Now $\hbar_\zeta(U)$ is found by recursively following the link to the child region of R_k containing $\hbar_\zeta(U)$, as determined by a canonical arc intersection test of the left child, until the trapezoid of $\Pi_\Box(U)$ containing $\hbar_\zeta(U)$ is found. From this trapezoid $\hbar_\zeta(U)$ is determined. Since the tree has at most $\lceil \log m \rceil$ levels, at most $2\lceil \log m \rceil - 1$ canonical arc intersection tests are required to find $\hbar_\zeta(U)$ and thus the query time is $O(\log^2 m)$.

In [4] this tree structure is modified to work for general polygons. Among the changes needed, a region may be represented by as many as 2 x-monotone polygons. The percentage of regions of polygon P represented by 2 x-monotone polygons is roughly the percentage of elements of $\Pi_\Box(P)$ with more than 2 adjacent trapezoids. Our algorithm, instead, partitions P into x-monotone subpolygons. There is then a tree for each x-monotone polygon in the partition.

3 X-Monotone Subpolygon Partitions

In this section we show how to partition P into x-monotone subpolygons such that any x-monotone curve interior to P intersects at most $2\lfloor \log n \rfloor - 1$ of the subpolygons. This is done in 2 stages. We call a polygon T

[1] In practice each region would have a link to an ancestor node of at most two of its neighbor regions since this would allow circular ray shooting with counterclockwise oriented arcs.

tree $\pm x$-monotone if, for any point q interior to T, there is an x-monotone curve contained in T from start(T) to q; if start(T) is a leftmost vertex of T then we call T *tree $+x$-monotone* and if start(T) is a rightmost vertex of T then we call T *tree $-x$-monotone*. In the first stage we partition P into $\pm x$-monotone subpolygons using an algorithm in [2]; we denote this partition by $\Pi_{\mathcal{T}}(P)$. The algorithm runs in $O(n)$ time and is surprisingly simple, given $\Pi_{\square}(P)$. A longer but clearer variation of this algorithm is as follows.

Construct the dual tree of $\Pi_{\square}(P)$ such that the vertices are contained in their corresponding trapezoids and the edges are x-monotone curves interior to P. Set the root node of the tree to the node whose dual trapezoid contains start(P) and direct the edges of the tree towards the root. Color each edge of the tree blue if it is directed to the left and red if it is directed to the right. Give each vertex of the tree the same color as its only outward directed incident edge except the root vertex which is given the color of its only incident edge. Let ξ be the set of edges of the tree whose endpoints are colored differently. Let $\Gamma_{\mathcal{T}}(P)$ be the subset of $\Gamma_{\square}(P)$ whose elements are crossed by some element of ξ. Note that $|\xi| = |\Gamma_{\mathcal{T}}(P)|$. Then $\Pi_{\mathcal{T}}(P)$ is the partition of P induced by $\Gamma_{\mathcal{T}}(P)$.

A key property of $\Pi_{\mathcal{T}}(P)$ is the following; see Fig. 1.

Lemma 3.1. *Any x-monotone curve interior to P intersects at most 2 elements of $\Pi_{\mathcal{T}}(P)$.*

Lemma 3.1 essentially reduces Problem 1.1 to the case that the polygon is tree $\pm x$-monotone polygon; we can then apply the data structure we construct for tree $\pm x$-monotone polygons to each element of $\Pi_{\mathcal{T}}(P)$. For example, say, for a query arc ζ, that ζ_o is contained in a tree $\pm x$-monotone polygon, say Q, of $\Pi_{\mathcal{T}}(P)$. First, we find $\hbar_{\zeta}(Q)$. Now if $\hbar_{\zeta}(Q)$ belongs to the interior of an element γ of $\Gamma_{\mathcal{T}}(P)$ then we use the sub-arc of ζ that has $\hbar_{\zeta}(Q)$ and ζ_e as its endpoints for a new query on the element of $\Pi_{\mathcal{T}}(P)$, other than Q, for which γ is either a door or a window. By Lemma 3.1, the second query must find $\hbar_{\zeta}(P)$.

Now we apply the second stage of our polygon partitioning algorithm. Let T be an m vertex element of $\Pi_{\mathcal{T}}(P)$; we partition T into x-monotone subpolygons such that any x-monotone curve interior to T intersects at most $\lfloor \log m \rfloor$ of the subpolygons. We use a method analogous to the construction of the centroid path partition of a tree [5]; see also [2]. For each trapezoid t in $\Pi_{\square}(T)$ with a window we mark all the windows of t except the window, say w_t, of t for which $|\Pi_{\square}(T_{w_t})|$ is maximum with ties broken arbitrarily; we denote this set of marked chords of $\Pi_{\square}(T)$ by $\Gamma_{\natural}(T)$. We further denote by $\Pi_{\natural}(T)$ the partition of T induced by $\Gamma_{\natural}(T)$. The following 2 properties of $\Pi_{\natural}(T)$ and $\Gamma_{\natural}(T)$ are used in this paper and are easily established; see [2]:

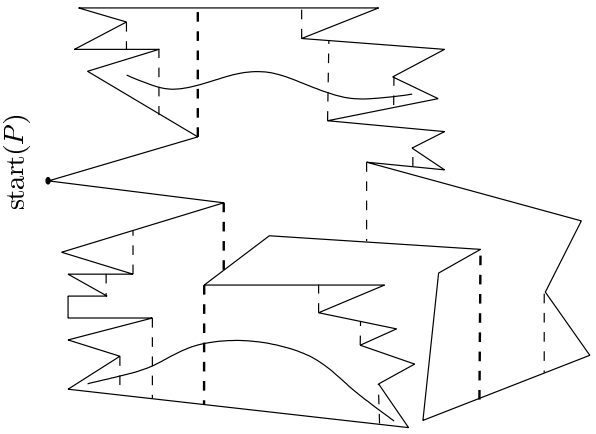

Figure 1: Any x-monotone curve interior to a polygon P intersects at most 2 elements of $\Pi_{\mathcal{T}}(P)$ (separated by thick dashed lines) and $2\lfloor \log n \rfloor - 1$ elements of $\Pi_{\natural}(P)$ (separated by all dashed lines) where $|P| = n$.

Lemma 3.2. *Each element of $\Pi_{\natural}(T)$ is x-monotone.*

Lemma 3.3. *Let S and γ be such that $S \in \Pi_{\natural}(T)$ and $\gamma \in$ windows(S). Then $2|\Pi_{\square}(T_{\gamma})| < |\Pi_{\square}(T_{\text{door}(S)})|$.*

Using Lemma 3.2, we can now solve Problem 1.1 on T. First, we apply the preprocessing stage of the algorithm from Section 2 to each x-monotone subpolygon in $\Pi_{\natural}(T)$. Then, given a query arc ζ, we apply the query stage of the algorithm from Section 2 to each x-monotone subpolygon in $\Pi_{\natural}(T)$ intersected by ζ in succession (as described above for arcs intersecting 2 elements of $\Pi_{\mathcal{T}}(P)$). Using Lemma 3.3, it can be shown that at most $\lfloor \log m \rfloor$ elements of $\Pi_{\natural}(T)$ are intersected by $\zeta(T)$. Using Lemma 2.1 and divide and conquer, we can answer a query for Problem 1.1 on an h vertex x-monotone polygon in $O(\log^2 h)$ time. Thus we can answer a query on T in $O(\log^3 m)$ time. This is slower than the complexity we require by an $O(\log m)$ factor.

4 Circular Ray Shooting in a Tree $\pm X$-Monotone Polygon

We reduce the cost of a circular ray shooting query on T to $O(\log^2 m)$ time by using an algorithm for performing a kind of range query on trees; see [2].

Our first step is to assign weights to the elements of $\Pi_{\square}(T)$. For each element t_i of $\Pi_{\square}(T)$ we assign a weight $w(t_i) = \max(1, \sum_{j=1}^{k} |\Pi_{\square}(P_{\gamma_j})|)$ where $\gamma_1, \gamma_2, ..., \gamma_k$ are the windows of the parent of t_i other than door(t_i).

The next step is to construct a version of the tree data structure described in Section 2 for each element of $\Pi_{\natural}(T)$. Each tree is constructed based on the augmented weighted binary tree structure described in [2]. For any x-monotone subpolygon Q of $\Pi_{\natural}(T)$ we use

$\mathbb{T}(Q)$ to denote the data structure we construct for Q and $W(Q)$ to denote the sum of the weights of the elements of $\Pi_{\square}(Q)$. Each node of $\mathbb{T}(Q)$ is assigned a weight which is the sum of the weights of the leaves of the subtree rooted at the node. The augmented part of the augmented weighted binary tree structure corresponds to the set of neighbor ancestor links of $\mathbb{T}(Q)$.

We now assume that T is tree $+x$-monotone; the alternate case is handled only slightly differently; see [2]. For a given query arc ζ of Q, let $w_\hbar$ be the weight of the trapezoid in $\Pi_{\square}(Q)$ immediately following the one containing $\hbar_\zeta(Q)$ or the rightmost trapezoid in $\Pi_{\square}(Q)$ if it contains $\hbar_\zeta(Q)$. Let w_ζ denote the total of the weights of the trapezoids of $\Pi_{\square}(Q)$ intersected by $\zeta(Q)$. In [2], it is shown that, using $\mathbb{T}(Q)$, we can find $\hbar_\zeta(Q)$ in $O(\max(1, \log w_\zeta - \log w_\hbar))$ canonical arc intersection tests. Consider that $\hbar_\zeta(Q)$ belongs to the interior of a window of Q. Let U be the element of $\Pi_\ell(T)$ whose door is this window, $\zeta' = \zeta - \zeta(Q)$, and $w'_\hbar$ be the weight of the trapezoid in $\Pi_{\square}(U)$ immediately following the one containing $\hbar_{\zeta'}(U)$ or the rightmost trapezoid in $\Pi_{\square}(U)$ if it contains $\hbar_{\zeta'}(U)$. Then, in [2], it is shown that $\hbar_\zeta(Q)$ and then $\hbar_{\zeta'}(U)$ can be found in

$$\Theta(\max(1, \log w_\zeta - \log w_\hbar) + (1, \log w_\hbar - \log w'_\hbar)) = \Theta(2, \log w_\zeta - \log w'_\hbar)$$

canonical arc intersection tests. Note that if the dual of $\Gamma_\ell(T)$ is a binary tree then $\log w_\hbar = W(U)$. Similarly, no matter how many elements of $\Pi_\ell(T)$ are intersected by ζ before intersecting T, the total number of canonical arc intersection tests performed is never more than $O(\log W(Q)) = O(\log n)$. A key point here is that Lemma 3.3 guarantees that ζ intersects at most $\lfloor \log m \rfloor$ elements of $\Pi_\ell(T)$ so the overhead of switching between x-monotone subpolygons is not a cost factor.

5 Complexity Constants

Since both the solution to Problem 1.1 in [4] and our solution have the same asymptotic complexities, a comparison of the constants for these complexities is worthwhile. Let P be an n vertex simple polygon and k be the size of the largest element in $\Pi_{\mathcal{T}}(P)$ by vertex count. For our algorithm the preprocessing time and space complexity is $\Theta(n \log k)$ and the query complexity is $O(\log^2 k)$. The same asymptotic complexities can be shown for the circular ray shooting algorithm in [4]. Thus we restrict our analysis to the case that P is tree $\pm x$-monotone.

We can compare the query complexity constants by counting the number of canonical arc intersection tests performed which, in the worst case scenario, is determined by the height of the tree. In [4], the authors determined an upper bound on the height of the tree constructed by their algorithm of $2.41 \log n$.[2] We have determined an upper bound on the height of the tree constructed by our algorithm of $2.52 \log n + 2$.[3] We gratefully thank J. D. Horton for improving this bound from the original $3 \log n + 2$. Thus leads to an upper bound on the number of query arc intersection tests in a query for the algorithm in [4] of $2.48 \log n$ whereas the upper bound for our algorithm is $6.52 \log n$[4].

We can compare the preprocessing time and space complexity constants by counting the total number of trapezoids in the vertical trapezoidizations of the monotone subpolygons stored at the nodes of the trees described in Section 2. Using the height of the tree we get an upper bound of $4.82n \log n$ on the number of trapezoids used for the algorithm in [4] (we think the correct value is around $(3/2)n \log n$) and $2.52n \log n + 2n$ for our algorithm. However, in both cases these bounds are clearly not tight. The best case scenario for the algorithm in [4] is that P is x-monotone. Then, between $n \log n$ and $(n+1)\log(n+1)$ trapezoids are used. The same number of trapezoids are used in this case with our algorithm and we conjecture that it is the worst case scenario. The best case scenario for our algorithm is a complete perfectly balanced tree. Then our algorithm uses fewer than $5n/2$ trapezoids.

References

[1] Pankaj K. Agarwal and Micha Sharir. Circle shooting in a simple polygon. *J. Algorithms*, 14:69–87, 1993.

[2] R. P. Boland. *Polygon Visibility Decompositions with Applications*. Ph.D. thesis, submitted January, 2001, University of Ottawa, Ottawa, Ontario.

[3] R. P. Boland and J. Urrutia. Circular ray shooting: Slower but surer. In *Proc. 12th Canad. Conf. Comput. Geom.*, pages 37–42, 2000.

[4] Siu-Wing Cheng, Otfried Cheong, Hazel Everett, and René van Oostrum. Hierarchical vertical decompositions, ray shooting, and circular arc queries in simple polygons. In *Proc. 15th Annu. ACM Sympos. Comput. Geom.*, pages 227–236, June 1999.

[5] Richard Cole, Martin Farach-Colton, Ramesh Hariharan, Teresa Przytycka, and Mikkel Thorup. An $O(n \log n)$ algorithm for the maximum agreement subtree problem for binary trees. *SIAM J. Comput.*, 30(5):1385–1404 (electronic), 2000.

[2] We do not believe that this value is tight. We think it can be reduced to approximately $1.71 \log n$.

[3] Since an arc may intersect as many as 2 elements of $\Pi_{\mathcal{T}}(P)$ we should multiply this value by 2 (actually 1.5). However, reorganizing the weights assigned to nodes of the tree can eleminate this factor. This change may not be worthwhile since it may decrease the average case performance.

[4] This bound includes additional costs due to having to change between trees as many as $\log n$ times during a single query.