# Hierarchical Decompositions and Circular Ray Shooting in Simple Polygons

Siu-Wing Cheng[1]  Otfried Cheong[2]  Hazel Everett[3]  René van Oostrum[2]

**Abstract**

A hierarchical decomposition of a simple polygon is introduced. The hierarchy has depth $O(\log n)$, linear size, and its regions have at most three neighbors. Using this hierarchy, circular ray shooting queries in a simple polygon can be answered in $O(\log^2 n)$ query time and $O(n \log n)$ space. If the radius of the circle is fixed, the query time can be improved to $O(\log n)$ and the space to $O(n)$.

## 1 Introduction

Geometric problems lend themselves naturally to solution by divide-and-conquer algorithms. Subproblems can be identified by partitioning space into regions, and the problem can be solved in each region separately. Recursively continuing the partition leads to a *hierarchical decomposition* of a geometric space or object. Many such decompositions have been introduced to solve a variety of problems [5, 13]. Most data structures for geometric search problems are in fact hierarchical decompositions.

Guibas and Hershberger [11] introduced a hierarchical decomposition of a simple polygon to efficiently answer shortest-path queries within the polygon. Their structure is a hierarchy of regions, the root corresponding to the whole polygon, the leaves corresponding to triangles of a fixed triangulation of the polygon, and every non-leaf region being split into two children using a diagonal. Every region is thus an area of the polygon, connected to the remainder of the polygon through a number of "doors." In Guibas and Hershberger's structure, a region can have $\Theta(\log n)$ doors, where $n$ is the number of edges of the polygon.

We give a new hierarchical decomposition of a simple polygon where regions have at most three doors. In other words, we make sure that when we split a region with three doors, both subregions contain at least one of the original three doors. Our technique for achieving this is inspired by the topology tree hierarchy of Frederickson [9, 10]. The decomposition can be based either on a fixed triangulation or on the vertical decomposition (trapezoidal map) of the polygon.

We believe that this decomposition will prove useful in a number of applications in computational geometry that deal with problems involving paths in simple polygons. In this paper, we concentrate on circular ray shooting, and use the decomposition based on the trapezoidal map.

A data structure for circular ray shooting problem in a simple polygon was given by Agarwal and Sharir [1], achieving $O(\log^4 n)$ query time with space $O(n \log^3 n)$. We improve this result to

[1]Hong Kong University of Science & Technology, Dept. of Computer Science, Clear Water Bay, Kowloon, Hong Kong. This research was supported by RGC Competitive Earmarked Grant HKUST650/95E.

[2]Institute of Information and Computing Sciences, Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, Netherlands. This research was supported by the Netherlands' Organization for Scientific Research (NWO). A portion of this research was done while O.C. and R.v.O. were at HKUST, and supported by the Research Grants Council of Hong Kong.

[3]LORIA, 615 rue du Jardin Botanique, B.P. 101, 54602 Villers-lès-Nancy, cedex, France. Part of this research was done during a sabbatical spent at HKUST, and was supported by NSERC and the Research Grants Council of Hong Kong.

$O(\log^2 n)$ query time and $O(n \log n)$ space. If the radius of the query arc is fixed, that is, given at preprocessing time, the query time can be improved to $O(\log n)$ and the space to $O(n)$. This matches the best known result for linear ray shooting in a simple polygon [12].

The hierarchical decomposition has other applications as well. Based on a triangulation, it can be used to replace the decomposition in Guibas and Hershberger's structure (this does not lead to an improvement of the asymptotic complexity). In forthcoming papers we will describe how to perform ray shooting along parabolic and hyperbolic arcs, as well as finding the largest empty lune determined by two points, or the on-line computation of a circular visibility region in a simple polygon.

## 2 The hierarchical vertical decomposition

The *hierarchical vertical decomposition* of a simple polygon $P$ is based on its *vertical decomposition* (or *trapezoidal map*) [6, 8]. Recall that the trapezoidal map is obtained by drawing a vertical line segment through every vertex of $P$, cutting the interior of $P$ into two parts. The result is a partition of $P$ into trapezoids (which can degenerate into triangles) separated by vertical sides. We call these vertical sides separating two trapezoids *doors*. We assume general position, that is no two distinct endpoints have the same $x$-coordinate. This can easily be simulated [2, Chapter 6]. Therefore a trapezoid has at most four doors. We split trapezoids with four doors into two trapezoids using a vertical door in the middle. The result is a vertical decomposition $\mathcal{T}$ of $O(n)$ trapezoids with at most three doors each. Its dual graph is a tree with maximum degree three.

The hierarchy $\mathcal{H}$ is a rooted binary tree. The nodes of the tree are called *regions*, and are connected sets of trapezoids of $\mathcal{T}$. The *doors* of a region are the doors of trapezoids of the region that are shared with trapezoids outside the region. A remarkable property of our decomposition will be that every region has at most three doors. Two regions consisting of disjoint sets of trapezoids are called *adjacent* if they share a door.

The root of $\mathcal{H}$ is the region consisting of the whole polygon $P$. The leaves of $\mathcal{H}$ are regions consisting of a single trapezoid of $\mathcal{T}$. All non-leaf regions $r$ have two *daughter regions*, obtained by splitting the set of trapezoids of $r$ into two connected subsets. We can visualize this as splitting the region along an interior door.

The regions on each level of $\mathcal{H}$ form a decomposition of $P$. The adjacency relationship induces a tree of maximum degree three on these regions. Figure 1 shows an example of a hierarchical vertical decomposition for a polygon with 15 vertices.

Before we can prove that such a decomposition exists, we need a small lemma.

**Lemma 1** *Let $T$ be a tree of $m$ nodes of maximum degree three. There is a matching $M$ of $T$ of size at least $m/4$ that does not match two nodes of degree three.*

**Proof:** The statement is clearly true if $T$ is a path, so assume there is at least one node of degree three. Let this be the root of $T$, inducing a parent-child relationship on $T$.

Consider now a maximal chain of nodes of degree less than three. Its bottom node (that is, the node furthest from the root) is either a node of degree one (a leaf) or degree two (whose child is a node of degree three); the remaining nodes on the chain are of degree two. The parent of the highest node in the chain is of degree three. If the total number of nodes on the chain is even, we can match them to each other in pairs. If their total number is odd, we match them starting from the bottom, and match the highest node with its parent, unless this parent (a node of degree three) has already been matched. We do this for every such maximal chain.

We analyze the size of the matching. Let $h$ be the number of nodes of degree three. Then the sum of node degrees is at least $3h + (m - h) = 2h + m$, and since $T$ is a tree it follows that $2h + m \leq 2m - 2$ and therefore $h \leq m/2 - 1$. We now charge every unmatched node of $T$ to a node of degree three in the following way. An unmatched node of degree three is charged to itself. An unmatched node of degree less than three is charged to its parent, necessarily a matched node of

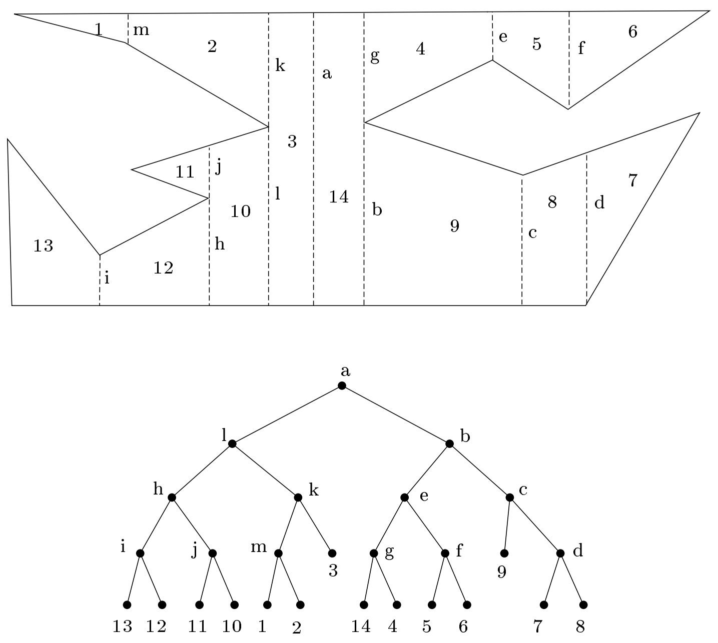

Figure 1: A hierarchical decomposition $\mathcal{H}$ for a simple polygon $P$. Inner nodes of $\mathcal{H}$ are annotated with the door splitting them, leaves with the trapezoid.

degree three. Except for the root, every node of degree three has two children, and can therefore be charged at most once. The root has three children and can be charged at most twice. The total charge is therefore at most $h + 1 \leq m/2$, and the bound follows. $\square$

**Theorem 2** *Let $P$ be a simple polygon with $n$ vertices. Then there exists a hierarchical vertical decomposition $\mathcal{H}$ of $P$ with the following properties.*

- *Every region has at most three doors.*
- *The depth of the hierarchy is $O(\log n)$.*
- *The number of regions in $\mathcal{H}$ is $O(n)$.*
- *The total size of all regions in $\mathcal{H}$ is $O(n \log n)$.*
- *$\mathcal{H}$ can be computed in time $O(n \log n)$.*

**Proof:** We give an algorithm to construct $\mathcal{H}$ in time $O(n \log n)$. The properties will follow from the construction. We start out with $\mathcal{T}$, constructed in time $O(n \log n)$ from $P$ [6, 8]. We create the leaves of $\mathcal{H}$ from $\mathcal{T}$.

The construction now proceeds in phases. In every phase, we merge adjacent regions in pairs. Note that such a merge is admissible unless both regions have three doors (because that would result in a region with four doors).

Assume we have $m$ regions at the beginning of a phase. They partition $P$, and the adjacency relationship between them induces a tree $T$ with $m$ nodes and maximum degree three. By Lemma 1, there is a matching $M$ of size at least $m/4$. We merge regions according to the matching $M$, resulting in at most $3m/4$ regions at the end of the phase.

Finding the matching and merging the regions can be done in time $O(m)$. Since the number of regions decreases geometrically, the total merging time is $O(n)$. The number of regions generated is $O(n)$, and the depth of the hierarchy is $O(\log n)$. Clearly every region created has at most three doors.

Since the regions on a level of $\mathcal{H}$ form a partition of the trapezoids of $\mathcal{T}$, the total size of all regions on this level is $O(n)$. Since the depth of the hierarchy is $O(\log n)$, the total size of all regions is bounded by $O(n \log n)$. ◻

The dual graph of $\mathcal{T}$ is a tree of degree at most three. For every pair of trapezoids there is a unique sequence of trapezoids that connects them. The hierarchical decomposition $\mathcal{H}$ can be used to retrieve this sequence in a compact form, as the following lemma shows.

We first introduce a bit of notation. A non-leaf region $r \in \mathcal{H}$ has two daughter regions separated by a door that we denote as $d_r$. The *level* of a region is its distance from the root—the root has level zero, its daughters have level one, and so on.

**Lemma 3** *Let $P$ be a simple polygon with $n$ vertices with hierarchical vertical decomposition $\mathcal{H}$. Given two query trapezoids $\Delta_1$ and $\Delta_2$, Algorithm FindSequence computes in time $O(\log n)$ a sequence $\pi = r_1, \dots, r_k$ of regions of $\mathcal{H}$ such that $r_1 = \Delta_1$, $r_k = \Delta_2$, $r_i$ and $r_{i+1}$ are adjacent for $1 \leq i < k$, and $k = O(\log n)$.*

**Proof:** We observe that the loop in line 5 maintains the following invariant: region $r$ is adjacent to the first region in $\pi$, and the common door is $d$. Furthermore, the level of $r$ increases by one in each iteration. Similarly, the loop of line 15 maintains the invariant that region $r$ is adjacent to the last region of $\pi$, the common door is $d$, and again the level of $r$ increases in each iteration. It follows that consecutive regions in $\pi$ are disjoint and adjacent along a common door, implying the correctness of the result. The running time and the length of the returned sequence is linear in the depth of $\mathcal{H}$, and therefore $O(\log n)$. ◻

**Algorithm** *FindSequence*$(\Delta_1, \Delta_2)$
**Input:** Two trapezoids $\Delta_1$, $\Delta_2$ of $\mathcal{T}$
**Output:** A sequence $\pi = r_1, \dots, r_k$ of regions of $\mathcal{H}$.

```
1.  Let $r^*$ be the the lowest common ancestor of $\Delta_1, \Delta_2$ in $\mathcal{H}$.
2.  Let $r$ be the daughter of $r^*$ containing $\Delta_1$.
3.  $d \leftarrow d_{r^*}$.
4.  Let $\pi$ be the empty sequence.
5.  while $r \neq \Delta_1$
6.      do Let $r'$ be the daughter of $r$ containing $\Delta_1$.
7.         Let $r''$ be the other daughter of $r$.
8.         if $d$ is a door of $r''$
9.           then prepend $r''$ to $\pi$.
10.               $d \leftarrow d_r$.
11.        $r \leftarrow r'$.
12. Prepend $\Delta_1$ to $\pi$.
13. Let $r$ be the daughter of $r^*$ containing $\Delta_2$.
14. $d \leftarrow d_{r^*}$.
15. while $r \neq \Delta_2$
16.     do Let $r'$ be the daughter of $r$ containing $\Delta_2$.
17.        Let $r''$ be the other daughter of $r$.
18.        if $d$ is a door of $r''$
19.          then append $r''$ to $\pi$.
20.               $d \leftarrow d_r$.
21.        $r \leftarrow r'$.
22. Append $\Delta_2$ to $\pi$.
23. return $\pi$
```

As an immediate application, we can precompute a set of $O(n)$ subsequences of trapezoids, and then express the sequence connecting two given trapezoids using $O(\log n)$ precomputed subsequences.

**Corollary 4** *Let $P$ be a simple polygon with $n$ vertices. There is a data structure of size $O(n \log n)$ that can be computed in $O(n \log n)$ time and that allows to return, in $O(\log n)$ time, the sequence of trapezoids connecting two query trapezoids $\Delta_1$ and $\Delta_2$ as a sequence of $O(\log n)$ precomputed subsequences.*

**Proof:** The data structure consists of the hierarchical vertical decomposition $\mathcal{H}$. A region $r$ of $\mathcal{H}$ has at most three doors. For each of the at most three pairs $(d_1, d_2)$ of doors of $r$, we precompute the sequence of trapezoids of $r$ connecting $d_1$ and $d_2$. There are $O(n)$ such precomputed sequences, and by Theorem 2 their total size is $O(n \log n)$. To answer a query, we apply Algorithm *FindSequence* to obtain a sequence of $O(\log n)$ regions $\Delta_1 = r_1, \ldots, r_k = \Delta_2$. For each region $r_i$, $1 < i < k$, we have a precomputed sequence connecting the doors $r_i$ shares with $r_{i-1}$ and $r_{i+1}$. $\Box$

We have based our description of the hierarchical decomposition on the trapezoidal map of $P$ because this version will be needed for our application to ray shooting in the following sections. However, the decomposition can be built based on a fixed triangulation of $P$ in exactly the same way.

## 3 Ray shooting with the hierarchical vertical decomposition

Given a simple polygon $P$, we would like to build a data structure that stores $P$ in such a way that certain ray shooting queries can be answered quickly. A query consists of an $x$-monotone curve $\gamma$ with endpoints $p$ and $q$, with $p$ inside $P$, and the goal is to find the first intersection of $\gamma$ with the boundary of $P$, or to determine that there is no such intersection. Semi-infinite rays can be simulated by choosing $q$ far away outside $P$. The curve $\gamma$ should be such that intersections between $\gamma$ and a line segment can be computed in constant time. (A data structure for queries with starting point *outside* the polygon can be built in the same way by interpreting the exterior of $P$ as a "generalized" polygon.)

Our data structure is based on the hierarchical vertical decomposition $\mathcal{H}$ of $P$. We first compute $\mathcal{H}$ and a point location structure for $\mathcal{T}$ [7]. A region $r \in \mathcal{H}$ has at most three doors, and therefore at most two pairs of doors through which an $x$-monotone path could traverse $r$. Let $d_1, d_2$ be such a pair of doors. Let $\Lambda$ be the set of trapezoids connecting $d_1$ and $d_2$. We suppose for the moment that we have constructed a data structure that allows us to test, in time $Q(|\Lambda|)$, whether a given query curve known to intersect the doors $d_1$ and $d_2$ passes through all the trapezoids in $\Lambda$ without intersecting any of their bounding edges. That is equivalent to testing whether the curve, if it passes through $d_1$ also passes through $d_2$ without intersecting the boundary of $P$ in between.

Our approach is to walk along $\gamma$ through regions of $\mathcal{H}$. When $\gamma$ enters a region $r$, we use the auxiliary data structure to test if it passes through $r$ without intersecting the boundary of $P$. If so, we continue the walking in the neighboring region. Otherwise, we know that the first point of intersection between $\gamma$ and $P$ can be found in $r$.

Note that even if $p$ and $q$ both lie in $P$, but $\gamma$ does not completely lie in $P$, then the first intersection may occur in a trapezoid not in the sequence computed by Algorithm *FindSequence*. This makes the search somewhat more complex.

**Lemma 5** *Let $P$ be a simple polygon with $n$ vertices with hierarchical vertical decomposition $\mathcal{H}$. Suppose that for each pair of doors of a region in $\mathcal{H}$ consisting of $m$ trapezoids, a data structure of size $S(m)$ can be constructed in time $P(m)$ that tests, in time $Q(m)$, whether an $x$-monotone query curve passes through the two doors without intersecting any edge of $P$ in between. Then a data structure of size $O(S(n) \log n)$ can be constructed in time $O(P(n) \log n)$ that answers ray-shooting queries along $x$-monotone curves in time $O(Q(n) \log n)$.*

**Proof:** The data structure consists of the hierarchical decomposition $\mathcal{H}$, where each region $r$ has been augmented with at most two auxiliary structures to test the pairs of doors that can be connected by an $x$-monotone path. Since we can assume $P(m)$ and $S(m)$ to be at least linear, the total size is $O(S(n) \log n)$, and the total preprocessing time is $O(P(n) \log n)$.

The query procedure is given by Algorithm *ShootRay*. The loop in line 6 has the following invariant: the curve $\gamma$ does not intersect $P$ before it reaches the door $d$, which is a door of $r$. If $flag$ = **true**, then the part of $\gamma$ between $p$ and $d$ lies in $r$, otherwise $\gamma$ enters $r$ through $d$ and either ends in $r$ or has an intersection point with $P$ in $r$.

The level of the door $d$ decreases in each iteration of the loop, and therefore it is executed at most $O(\log n)$ times. The test in line 10 takes $Q(m) = O(Q(n))$ time, and so the loop takes time $O(Q(n) \log n)$.

The loop in line 14 maintains the invariant that $d$ is a door of $r$ and $\gamma$ either ends or has an intersection with $P$ in $r$. The level of $r$ increases by one in each iteration, so the loop is executed $O(\log n)$ times. Its running time is dominated by the query of the auxiliary data structure in line 17, and so the total time for this loop is $O(Q(n) \log n)$.

The loop invariant guarantees that in line 20 the region $r$ is in fact a trapezoid containing either $q$ or the intersection point we are searching for, and so the algorithm is correct. $\square$

```
Algorithm ShootRay($\gamma$)
Input: An $x$-monotone curve $\gamma$ with endpoints $p$ and $q$.
Output: The first intersection between $\gamma$ and $P$.
1.  Let $\Delta$ be the trapezoid containing $p$.
2.  if $\gamma$ intersects the top or bottom edge of $\Delta$ then return the intersection point.
3.  if $q \in \Delta$ then return nil.
4.  Let $d$ be the door through which $\gamma$ exits $\Delta$.
5.  $r \leftarrow \Delta, flag \leftarrow$ true.
6.  while $flag$
7.      do Let $r^*$ be the region with separating door $d$, that is $d_{r^*} = d$.
8.          Let $r'$ be the daughter of $r^*$ not containing $\Delta$.
9.          Let $d'$ be the second door of $r'$ intersected by $\gamma$.
10.         if $d'$ exists and $\gamma$ passes through $r'$ from $d$ to $d'$
11.             then $d \leftarrow d'$
12.             else $flag \leftarrow$ false.
13.         $r \leftarrow r'$
14. while $r$ is not a trapezoid
15.     do $d' \leftarrow d_r$
16.         Let $r'$, $r''$ be the daughters of $r$, such that $r'$ has the door $d$.
17.         if $\gamma$ intersects $d'$ and $\gamma$ passes through $r'$ from $d$ to $d'$
18.             then $d \leftarrow d', r \leftarrow r''$.
19.             else $r \leftarrow r'$
20. if $\gamma$ intersects the top or bottom edge of $r$ then return the intersection point.
21. return nil
```

## 4 Circular ray shooting

We will now see how to use our technique from the previous section to perform ray shooting along circular arcs. Our data structure relies on the curve $\gamma$ being $x$-monotone. We therefore partition a query arc into at most five pieces, each of which lies on a quarter-circle. See Figure 2. In the following, we explain how to perform a query with a query arc on the upper quarter-circle (where the slope of the arc is between 1 and $-1$). The other cases are handled symmetrically.

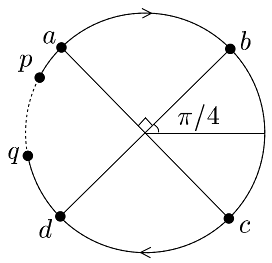

Figure 2: The query arc $pq$ is split into five pieces $pa$, $ab$, $bc$, $cd$ and $dq$ so that each lies on a quarter-circle.

By Lemma 5 we only have to provide a data structure that can store a sequence of $m$ trapezoids $\Lambda$ connecting two doors $d_1, d_2$ in such a way that we can quickly decide whether a query arc that intersects $d_1$ and $d_2$ passes through the trapezoids.

Let $\Lambda^*$ be the parts of polygon edges bounding $\Lambda$ from above. Let $\Lambda_*$ be the parts of polygon edges bounding $\Lambda$ from below. $\Lambda^*$ and $\Lambda_*$ lie in the vertical strip bounded by the vertical lines through $d_1$ and $d_2$. Since $\gamma$ is $x$-monotone, $\gamma$ passes through $\Lambda$ if and only if $\gamma$ lies completely below $\Lambda^*$, and completely above $\Lambda_*$.

To test whether $\gamma$ lies below $\Lambda^*$, we build a point-location data structure for the Voronoi diagram of the line segments in $\Lambda^*$, and we also store the point $m(\Lambda^*) \in \Lambda^*$ with the smallest $y$-coordinate. The Voronoi diagram of $\Lambda^*$ can be computed in $O(|\Lambda^*| \log |\Lambda^*|)$ time [14]. One can then break up the curved edges into $x$-monotone pieces and compute a vertical decomposition using plane-sweep in $O(|\Lambda^*| \log |\Lambda^*|)$ time. Afterwards, we build a point-location structure for monotone subdivisions with curved edges [7]. To perform the query, we first test whether $\gamma$ lies below $m(\Lambda^*)$. If that is the case, we then locate the center $x$ of our query arc in the Voronoi diagram, and thus obtain the segment $s \in \Lambda^*$ closest to $x$. If and only if $\gamma$ lies below $m(\Lambda^*)$ and the distance of $s$ to $x$ is larger than the radius of the query circle, then $\gamma$ lies below $\Lambda^*$.

To test whether $\gamma$ lies above $\Lambda_*$, we make use of the furthest point Voronoi diagram of the vertices in $\Lambda_*$. Let $w$ be the width of the vertical strip bounding $\Lambda_*$. Let $v(\Lambda_*)$ be the highest vertex of $\Lambda_*$. Pass a horizontal line through $v(\Lambda_*)$ and insert another horizontal line at distance $w$ below. See Figure 3. Let $S$ be the square enclosed between these two horizontal lines within the vertical strip bounding $\Lambda_*$. We construct the furthest point Voronoi diagram for the vertices of $\Lambda_*$ inside $S$ and a point-location data structure for the diagram. To perform the query, we first test whether $\gamma$ lies above $v(\Lambda_*)$. If that is not the case, then $\gamma$ intersects $\Lambda_*$. Otherwise, we check whether $\gamma$ lies completely above the square $S$. If so, then $\gamma$ lies above $\Lambda_*$. In the remaining case, we locate the center $x$ of our query arc in the furthest point Voronoi diagram. Let $s$ be the furthest vertex reported. We claim that $\gamma$ lies completely above $\Lambda_*$ if and only if the distance between $s$ and $x$ is smaller than the radius of $\gamma$. This follows from the fact that, since $\gamma$ is on the upper quarter-circle and contains points both in and above $S$, a point of $\Lambda_*$ lies above $\gamma$ if and only if it lies in $S$ outside the disc supporting the query arc.

We have thus shown how to implement the auxiliary data structure for a region of size $m$ using $S(m) = O(m)$ space, $P(m) = O(m \log m)$ preprocessing tim, and $Q(m) = O(\log m)$ query time. We have the following theorem.

**Theorem 6** *Given a simple polygon $P$ with $n$ vertices. There is a data structure of size $O(n \log n)$ that can be computed in time $O(n \log^2 n)$ that allows to perform circular ray shooting queries with origin inside $P$ in time $O(\log^2 n)$.*

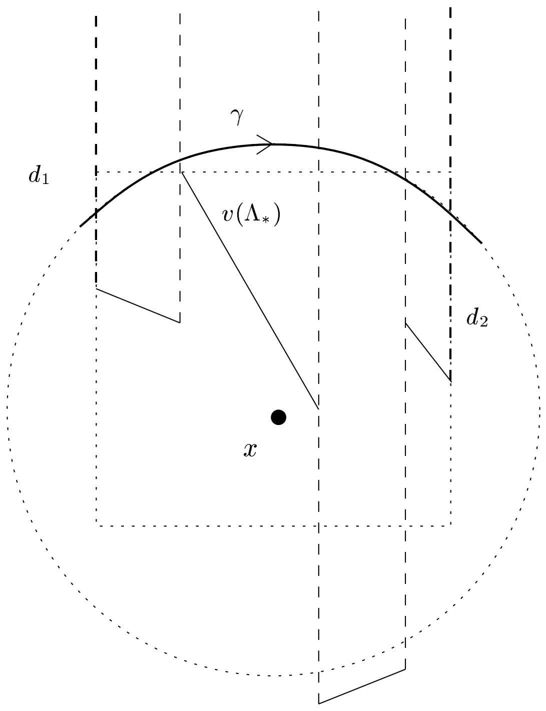

Figure 3: $\Lambda_*$ is shown in solid segments. The curve $\gamma$ is shown in bold. The part of the square $S$ lying below $\gamma$ is the same as the part of $S$ lying inside the circle supporting the query arc.

## 5 Fixed-radius circular ray shooting

We now consider the problem of ray shooting along circular arcs with fixed radius (that is, the radius is known at preprocessing time). This is also a case study: the same technique works for linear ray shooting, and ray shooting along parabolic or otherwise algebraic arcs, as long as the actual arcs are parts of translates of a curve given at preprocessing time. The data structure we obtain in this section does not improve on the more general structure of Theorem 6. It does, however, replace the use of a two-dimensional point location structure with a simple binary search, and that will allow us to obtain a data structure with optimal query time and storage in the following section.

We partition a circular ray shooting query into at most three queries, each of which lies on an $x$-monotone semi-circle. In the following, we describe how to perform a query when the query arc lies on an upper semi-circle. The other case is handled symmetrically. By scaling, we also assume that the query arc lies on a unit circle.

Again we use the basic approach from Section 3. Given a sequence $\Lambda$ of $m$ trapezoids connecting two doors $d_1$ and $d_2$, we want to build a data structure that allows to test whether a circular arc $\gamma$ (known to lie on an upper quarter-circle, to enter at $d_1$, and to leave at $d_2$) passes through all the trapezoids. As we saw in the last section, this is equivalent to testing whether $\gamma$ lies completely below $\Lambda^*$ and completely above $\Lambda_*$.

We precompute the set $L$ of points $x$ such that an upper semi-circle of unit radius and center $x$ touches $\Lambda^*$ from below. This is done by computing the Minkowski sum of a unit circle with each segment in $\Lambda^*$, and taking $L$ as the lower envelope of these Minkowski sums. See Figure 4.

**Lemma 7** *Let $s_1, s_2, \ldots, s_k$ be the segments of $\Lambda^*$ in order from left to right. When walking along $L$ from left to right, we encounter pieces induced by the segments in this order.*

**Proof:** Let $p_i$ and $p_j$ be points on $L$ induced by $s_i$ and $s_j$, with $p_i$ left of $p_j$. We need to show

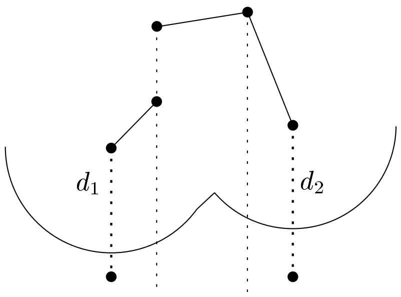

Figure 4: The dashed line segments show the sequence $\Lambda$ of trapezoids connecting the doors $d_1$ and $d_2$.

that $i < j$. Let $C_i$ (resp. $C_j$) be the region consisting of the unit circle centered at $p_i$ (resp. $p_j$) and all points below it. If $C_i$ and $C_j$ do not intersect, then clearly $i < j$. If they do intersect, their boundaries intersect in a single point $q$. The part of the boundary of $C_i \cup C_j$ left of $q$ belongs to $C_i$, the part right of $q$ to $C_j$. Since $s_i$ touches $C_i \cup C_j$ outside of $C_j$, and $s_j$ touches $C_i \cup C_j$ outside of $C_i$, we have $i < j$. ◻

Lemma 7 implies that each segment of $\Lambda^*$ induces at most one piece of the lower envelope $L$, so $L$ can be stored as a sorted array of the at most $m - 1$ break points, with a pointer to the inducing segment of $\Lambda^*$ for each interval. If the segments $\Lambda^*$ are given in left-to-right order, $L$ can be constructed incrementally in linear time. To add the next segment $s$, we compute the lower envelope $l$ of the Minkowski sum of $s$ and a unit disk. There is at most one intersection point $x$ between the current envelope $L$ and $l$. We update $L$ by removing the part of $L$ lying to the right of $x$, and then append the break point $x$ and a pointer to the segment $s$.

To determine whether a query arc $\gamma$ lies completely below $\Lambda^*$, we look up the $x$-coordinate of the center $p$ of the query arc in $L$ using binary search in time $O(\log m)$. This returns the segment $s$ of $\Lambda^*$ inducing the part of $L$ intersected by the vertical line through $p$. We then test whether $\gamma$ lies below $s$ in $O(1)$ time. The query arc lies completely below $\Lambda^*$ if and only if this is the case.

We build a symmetric structure for $\Lambda_*$ that allows us to decide in $O(\log m)$ time whether $\gamma$ lies completely above $\Lambda_*$.

Applying Lemma 5 with $P(m) = S(m) = O(m)$ and $Q(m) = O(\log m)$ gives us the following result.

**Lemma 8** *Given a simple polygon $P$ with $n$ vertices, and a radius $r > 0$. There is a data structure of size $O(n \log n)$ that can be computed in time $O(n \log n)$ such that circular ray shooting queries of radius $r$ with origin inside $P$ can be performed in time $O(\log^2 n)$.*

## 6 An optimal data structure for fixed-radius ray shooting

The query time and the space needed by the data structure of Lemma 8 is sub-optimal by a $\log n$-factor. We can obtain an optimal data structure by avoiding the duplication in the storage of lower envelopes, and by using fractional cascading [3, 4].

Consider a region $r$. It has at most three doors $d_1$, $d_2$, and $d_3$. At most two of the three possible pairs of doors permit an $x$-monotone path, so there are at most two auxiliary data structures associated with $r$. Consider one of these structures, say, for $d_1$ and $d_2$. Either $d_1$ and $d_2$ are both doors of one child $r_1$ of $r$, or the door $d_r$ splits the sequence of trapezoids connecting $d_1$ and $d_2$. In the first case we can reuse the auxiliary data structure in $r_1$—in other words, we store the structure for the pair of doors $(d_1, d_2)$ only at the highest node in $\mathcal{H}$ that contains both doors. In the latter case, the child $r_1$ will contain an auxiliary data structure for the pair of doors $(d_1, d_r)$, and the child $r_2$ will contain the auxiliary structure for the pair $(d_r, d_2)$. Let $\Lambda_1^*$ be the segments bounding the trapezoids between $d_1$ and $d_r$ from above, and let $\Lambda_2^*$ be the segments for the trapezoids between $d_r$ and $d_2$. Let $\Lambda^* := \Lambda_1^* \cup \Lambda_2^*$, and let $L$, $L_1$, and $L_2$ be the lower envelopes for $\Lambda^*$, $\Lambda_1^*$, and $\Lambda_2^*$, respectively. Since $\Lambda_1^*$ and $\Lambda_2^*$ are separated by the vertical line through $d_r$, Lemma 7 implies that $L$ contains a breakpoint $x$ such that the part of $L$ left of $x$ is a prefix of $L_1$, and the part of $L$ right of $x$ is a suffix of $L_2$. We can therefore modify our original data structure as follows: in $r_1$ we only store $x$, the suffix of $L_2$ not part of $L$, and a pointer to the auxiliary structure for the pair $(d_1, d_2)$, which is stored at some ancestor further up in the hierarchy.

Sometimes there is a choice as to which parent structure to refer to. In Figure 5, for instance, the structure for the pair $(d_1, d_r)$ could refer to either the structure for $(d_1, d_2)$, or the one for $(d_1, d_3)$. In this case we make the choice so that the length of the suffix of $L_1$ that has to be stored is smaller.

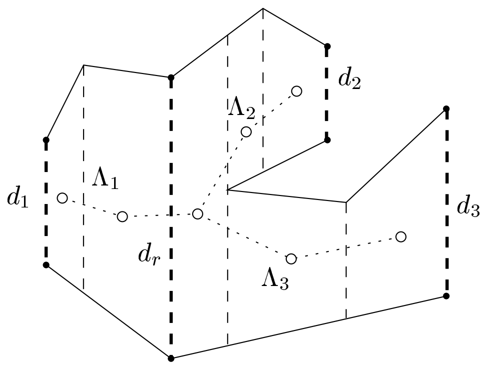

Figure 5: The region shown has two daughter regions separated by the door $d_r$. The doors are shown in bold dashed segments, the trapezoids are shown in dashed segments, and the polygon boundary is shown in solid line segments. The dotted lines indicate the sequences $\Lambda_1$, $\Lambda_2$, and $\Lambda_3$. Note that $\Lambda_2$ and $\Lambda_3$ share a trapezoid.

To analyse the space required by the modified data structure, consider a trapezoid $\Delta$. We say that $\Delta$ *appears weakly* in a region $r$ if there are doors $d_1$ and $d_2$ of $r$ such that there is an $x$-monotone path $\Lambda$ from $d_1$ to $d_2$ passing through $\Delta$. Furthermore, $\Delta$ *appears strongly* if the segment $s$ bounding $\Delta$ from above induces a piece of the lower envelope for $\Lambda^*$. Clearly all regions $r$ that $\Delta$ appears in are ancestors of $\Delta$. If $\Delta$ appears weakly in a region $r_1$, it also appears weakly in the child $r_2$ of $r_1$ containing $\Delta$. Furthermore, by the discussion above, if $\Delta$ appears strongly in $r_1$, it also appears strongly in $r_2$. This implies that the path from the root of $\mathcal{H}$ to $\Delta$ consists of three intervals: first $\Delta$ does not appear at all, then it appears weakly, and finally it appears strongly. The construction of our modified data structure implies that the segment $s$ is stored at the node for a region $r$ if and only if $\Delta$ appears strongly at $r$, but does not appear strongly at the parent of $r$. By the observation above, there is exactly one such node. It follows that our modified data structure takes $O(n)$ space.

We apply the same optimization to the auxiliary data structure for searching in $\Lambda_*$. All auxiliary data structures are based on lists of breakpoints, sorted by $x$-coordinate. All the lists being searched during a query are searched with the same search key, namely the $x$-coordinate of the query arc center. We can therefore apply fractional cascading [3, 4]. This adds additional keys to the lists, as well as cross pointers between the list stored in a node and the one stored in the parent node. As this is a standard application of fractional cascading on a tree, we do not discuss the details, and observe only that the space requirement increases by a constant. The effect of fractional cascading is the following: once we have performed a search in the list stored at a node, we can perform the search in a daughter node in constant time.

It remains to describe how to perform circular ray shooting queries in the modified structure. We first locate the trapezoid $\Delta$ containing the starting point of the ray. We then follow the path from the root of $\mathcal{H}$ to $\Delta$. In each node, we search all the lists stored at that node, using the $x$-coordinate of the query arc as the key. First, we test whether the key lies in the prefix or suffix stored higher up in the tree—if so, we reuse the search result obtained there earlier. Otherwise, we search the list stored at the node. Using fractional cascading, all this takes time $O(\log n)$.

We now proceed with the "ascending phase" of Algorithm *ShootRay*. The region $r'$ tested in line 10 is a daughter of $r^*$, which is an ancestor of $\Delta$. As we already have the search results for $r^*$, the auxiliary data structure for $r'$ can be queried in constant time.

Finally, we descend in $\mathcal{H}$ until we find the leaf containing the answer to the ray shooting query. As this descent follows a path in $\mathcal{H}$, we can perform each step in constant time.

It follows that the total query time is $O(\log n)$. We summarize the result in the following theorem.

**Theorem 9** *Given a simple polygon $P$ with $n$ vertices, and a radius $r > 0$. There is a data structure of size $O(n)$ that can be computed in time $O(n \log n)$ such that circular ray shooting queries of radius $r$ with origin inside $P$ can be performed in time $O(\log n)$.*

## Acknowledgments

We gladly acknowledge helpful discussions with Mordecai Golin and Giuseppe Italiano.

## References

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

[2] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, 1997.

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

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

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

[6] B. Chazelle and J. Incerpi. Triangulation and shape-complexity. *ACM Trans. Graph.*, 3:135–152, 1984.

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

[8] A. Fournier and D. Y. Montuno. Triangulating simple polygons and equivalent problems. *ACM Trans. Graph.*, 3:153–174, 1984.

[9] G. N. Frederickson. Ambivalent data structures for dynamic 2-edge-connectivity and $k$ smallest spanning trees. *SIAM J. Comput.*, 26:484–538, 1997.

[10] G. N. Frederickson. A data structure for dynamically maintaining rooted trees. *J. Algorithms*, 24:37–65, 1997.

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

[12] J. Hershberger and S. Suri. A pedestrian approach to ray shooting: Shoot a ray, take a walk. *J. Algorithms*, 18:403–431, 1995.

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

[14] C. K. Yap. An $O(n \log n)$ algorithm for the Voronoi diagram of a set of simple curve segments. *Discrete Comput. Geom.*, 2:365–393, 1987.