Hierarchical Vertical Decompositions, Ray Shooting, and Circular Arc Queries in Simple Polygons

Siu-Wing Cheng* Otfried Cheong*
Hazel Everett† René van Oostrum‡

Abstract

A new hierarchical decomposition of a simple polygon is introduced. The hierarchy has depth $O(\log n)$, linear size, and its regions have maximum degree three. 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 $O(n)$. The decomposition is also applied to three other circular arc query problems: shortest directed arc, arc bending, and arc pushing. Using these queries, the largest empty lune determined by two query points in a simple polygon can be computed in $O(\log^3 n)$ time, while the circular visibility region of a query point in a simple polygon can be reported in time $O(m \log^3 n)$, where m is the output size.

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.

*Hong Kong University of Science & Technology, Dept. of Computer Science, Clear Water Bay, Kowloon, Hong Kong. Research of the first author is partly supported by RGC Competitive Earmarked Grant HKUST650/95E. Research of the second author is partly supported by HKUST Direct Allocation Grant 97/98.EG15 and RGC Competitive Earmarked Grant HKUST6144/98E.

†Dép. d'informatique, Université du Québec à Montréal, C.p. 8888, Succursale Centre-Ville, Montréal, Canada H3C 3P8. This research was done during a sabbatical spent at HKUST, and supported by NSERC and the Research Grants Council of Hong Kong.

‡Universiteit Utrecht, Vakgroep Informatica, Postbus 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 visiting HKUST.

SCG'99 Miami Beach Florida

Many such decompositions have been introduced to solve a variety of problems [7, 16]. Most data structures for geometric search problems are in fact hierarchical decompositions.

Guibas and Hershberger [13] 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 [12, 11]. For technical reasons, we prefer to base it on the vertical decomposition (or trapezoidal map) of the polygon instead of a triangulation, but our construction would work with a triangulation just as well.

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 circular arc queries and their applications.

A data structure for circular ray shooting problem in a simple polygon is 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 $O(\log^2 n)$ 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 [14].

We also show how to perform several circular arc queries, in which a given circular arc is deformed by moving its center along a line until the arc encounters a polygon edge or vertex. Our queries take time $O(\log^3 n)$. As an application, we show how to compute the largest empty lune defined by two points p and q inside a polygon within the same time bound.

Agarwal and Sharir [2] show how to compute the circular visibility region of a point in a simple polygon in time $O(n \log n)$. Our circular arc queries can be used to solve an on-line version of this problem: we preprocess the polygon into a data structure that allows us to retrieve the circular visibility region of a query point in time roughly linear in its size.

2 The hierarchical vertical decomposition

The *hierarchical vertical decomposition* of a simple polygon P is based on its *vertical decomposition* (or *trapezoidal map*) $\mathcal{T}(P)$ [8, 10]. Recall that $\mathcal{T}(P)$ 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, that no two distinct endpoints have the same x-coordinate, (which can easily be simulated, [4] ch. 6), and therefore a trapezoid has at most four doors. We prefer to deal with at most three doors, and therefore split all trapezoids that have four doors by an additional vertical door in the middle. The result is a vertical decomposition of $O(n)$ trapezoids. Its dual graph is a tree.

The hierarchy $\mathcal{H}$ is a rooted binary tree. The nodes of the tree are called *regions*, and are connected sets of trapezoids of $\mathcal{T}(P)$. 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 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 the trapezoids of $\mathcal{T}(P)$. 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.

Before we can prove properties of our decomposition, 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 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. □

Theorem 2 *Let P be a simple polygon with n vertices. Then there exists a hierarchical vertical decomposition $\mathcal{H}$ of P such that 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 $O(n \log n)$, and $\mathcal{H}$ 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 $\mathcal{T}(P)$, constructed in time $O(n \log n)$ from P [10, 8]. We create the leaves of $\mathcal{H}$ from $\mathcal{T}(P)$.

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 $\mathcal{T}(P)$, 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 $\mathcal{T}(P)$ is a tree of degree at most three. For every pair of trapezoids there is thus a unique sequence of trapezoides that connects them. We discuss below how to use $\mathcal{H}$ to retrieve this sequence in a compact form.

Lemma 3 *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 time $O(n \log n)$ that returns, in $O(\log n)$ time, the sequence of trapezoids connecting two query trapezoids r_1 and r_2 represented as a sequence of $O(\log n)$ precomputed subsequences.*

Proof: The data structure consists of the hierarchical vertical decomposition $\mathcal{H}$ for P. Furthermore, for every region $r \in \mathcal{H}$, we precompute the sequence of trapezoids connecting any pair of doors of r. All this can be precomputed in time $O(n \log n)$, and takes $O(n \log n)$ space.

Let r_{12} be the lowest common ancestor of the two trapezoids r_1 and r_2 in $\mathcal{H}$. Since r_1 and r_2 are contained in different daughter regions of r_{12}, the dual tree path ρ from r_1 to r_2 intersects the door of r_{12} separating its daughter regions. Inductively, suppose that r_1 is contained in a region r descending from r_{12}, and ρ intersects the door d of r. Let r' and r'' be the daughter regions of r containing and not containing r_1 respectively. If r'' is adjacent to d (that is, if d is also a door of r''), then ρ must visit r'' from r' (crossing the door between them) to get to d. Thus, r'' should belong to the sequence. We then recursively deal with r'. Symmetrically, we can descend from r_{12} to process regions containing r_2.

This yields $O(\log n)$ regions connecting r_1 and r_2, as well as the precomputed sequence of trapezoids intersected by the dual tree path in each region. □

We have based our hierarchical decomposition on the trapezoidal map of P because every x-monotone curve lying entirely inside P between two points p, q in P traverses exactly the same sequence of trapezoids of $\mathcal{T}(P)$. (This is, for instance, not true for a triangulation of P.) Lemma 3 can be applied to retrieve this sequence as $O(\log n)$ precomputed subsequences.

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 γ with endpoints p and q, with p inside P, (semi-infinite rays can be simulated by chosing q far away outside P), and the goal is to find the first intersection of γ with the boundary of P, or to determine that there is no such intersection. The curve γ should be such that intersections between it 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.)

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. Given a door d, let r_d be the region whose daughters are separated by d. In other words, we have $d_{r_d} = d$. The *level* of a region is its distance from the root—the root has level zero, its daughters have level one, and so on. We denote the level of a region r by $\ell(r)$. The *level of a door* $\ell(d)$ is the level $\ell(r_d)$.

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 $\mathcal{T}(P)$ [16]. 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 Λ be the set of trapezoids connecting d_1 and d_2. We assume that we have constructed a data structure that allows us to test, in time $Q(n)$, whether a given query curve passes through all the trapezoids in Λ without intersecting any of its 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.

Our approach is to follow γ and for each region r entered, we test if γ passes through r or not. If yes, we continue by entering the next region. Otherwise, we zoom into r to find the first intersection point. Note that even if p and q both lie in P, the first intersection may occur in a trapezoid not in the sequence computed by Lemma 3, so a somewhat different strategy is needed, as described below.

We start by finding the trapezoid r_0 containing p, using the point location structure in time $O(\log n)$. Then we determine whether γ intersects an edge of r_0, or if q lies in r_0. In both cases, we are done. Otherwise, we determine the door d_1 of r_0 through which γ passes.

Let r_1 be the daughter of r_{d_1} *not* containing p. The curve enters r_1 through the door d_1. If we can verify that γ leaves r_1 through a door d_2 of r_1 and γ does not intersect the boundary of P between d_1 and d_2, then we enter another region r_2 which is the daughter of r_{d_2} not containing p. We continue in this fashion, passing through a sequence of regions and doors $r_0, d_1, r_1, d_2, r_2, \ldots, d_k, r_k$, until we end up in region r_k such that either r_k contains q or γ intersects the boundary of P in r_k.

In general, suppose γ enters r_i through door d_i. Testing whether γ intersects another door d_{i+1} of r_i takes constant time. If γ intersects no other door, then r_i either contains q or γ intersects the boundary of P in r_i. Otherwise, we query the data structure constructed for the pair of doors (d_i, d_{i+1}) of r_i and this takes $Q(n)$ time. It is not difficult to see that we have $\ell(d_{i+1}) < \ell(d_i)$. Since the depth of $\mathcal{H}$ is $O(\log n)$, this implies that $k = O(\log n)$, and the total query time spent so far is $O(Q(n) \log n)$.

To find the actual intersection point (or the absence thereof), we have to descend in $\mathcal{H}$ to the descendents of r_k as follows. Let $r = r_k$, and let $d = d_k$. Let r', r'' be the two daughters of r, where r' is adjacent to d. If $q \in r'$, set $r \leftarrow r'$, and continue. Otherwise, consider the pair of doors (d, d_r). We use the data structure for this pair of doors of r' to determine whether γ intersects the boundary of P inside r'. If so, set $r \leftarrow r'$, and continue. If not, the first intersection point or q must lie in r''. We set $d \leftarrow d_r$ and $r \leftarrow r''$, and continue.

This process continues until r is the trapezoid containing the first intersection point of γ with the boundary of P, or the point q. We can then answer the query in constant time.

Since the depth of $\mathcal{H}$ is $O(\log n)$, the total running time for this procedure is $O(Q(n) \log n)$.

4 Fixed radius circular ray shooting

Let's first study how to use our technique from the previous section to perform ray shooting along circular arcs with fixed radius (that is, the radius is known at preprocessing time). In fact, this is mostly 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.

Our data structure relies on the curve γ being x-monotone. We therefore need to 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 can be handled similarly. By scaling, we can always assume that the query arc lies on a unit circle.

From Section 3, it follows that we only have to provide a data structure that can store a sequence of m trapezoids Λ 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 or intersects any of their bounding polygon edges.

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

We precompute the set L of all possible points x such that a unit circle centered at x touches Λ^* from below. First compute the Minkowski sum of a unit circle with each segment in Λ^*. Then L is the lower envelope of these Minkowski sums. Its complexity is proportional to the size of Λ^*, and it can be constructed in linear time by scanning the line segments in Λ^* from left to right. The query arc γ lies completely below Λ^* if and only if the center of γ lies below L, and this can be tested by binary search in time $O(\log |L|)$. Similarly we can construct the upper envelope of lower semi-circles to test whether γ lies completely above Λ_*.

By Theorem 2, the total size of all regions of $\mathcal{H}$ is $O(n \log n)$. Since we precompute lower and upper envelopes for at most two pairs of doors of every region, the total size of all the auxiliary data structures is $O(n \log n)$, and they can be constructed in the same time bound. The data structure can decide whether a given query arc passes through one of the regions, given that it intersects two of its doors, in time $Q(n) = O(\log n)$. We can now use the strategy from Section 3 to perform ray shooting queries using arcs on a unit circle in time $O(\log^2 n)$.

The space requirement can be improved to $O(n)$ by observing that a trapezoid participates in each region on a path from the root to a leaf. The lower envelope for a region therefore consists of a prefix of the envelope for its left daughter, a single new breakpoint, and a suffix of the envelope of its right daughter. The total number of different breakpoints is only $O(n)$, and we can store them in linear space by storing each breakpoint in the highest region where it occurs, with minor modifications in the search algorithm. Details can be found in the full paper.

Furthermore, the query time can be improved to $O(\log n)$ as a standard application of fractional cascading in a tree [5, 6].

Theorem 4 *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 $O(\log n)$.*

5 Circular ray shooting

We now turn to the general circular ray shooting problem, where the radius is part of the query input. We split a query arc into at most five pieces, each of which lies on a quarter-circle. See Figure 1. 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).

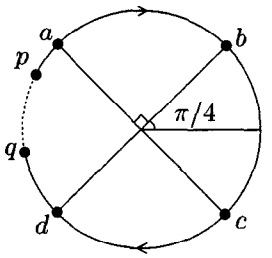

Figure 1: The query arc pq is split into five pieces pa, ab, bc, cd and dq so that each lies on a quarter-circle.

Again we use the basic approach from Section 3. Given a sequence Λ 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 γ (known to lie on an upper quarter-circle and known to enter at d_1 and leave at d_2) passes through all the trapezoids. As we saw in the last section, this is equivalent to testing whether the set Λ^* lies completely above γ, and whether Λ_* lies completely below γ.

To test whether γ lies below Λ^*, we build a point-location data structure for the Voronoi diagram of the line segments in Λ^*, and we also store the point $m(\Lambda^*) \in \Lambda^*$ with the smallest y-coordinate. To perform the query, we first test whether $m(\Lambda^*)$ lies below γ. If that is not 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 $m(\Lambda^*)$ lies above γ and the distance of s to x is larger than the radius of the query circle, then γ lies below Λ^*.

To test whether γ lies above Λ_*, we make use of a furthest point Voronoi diagram of the vertices in Λ_*. Let w be the width of the vertical strip bounding Λ_*. Let $v(\Lambda_*)$ be the highest vertex of Λ_*. Pass a horizontal line through $v(\Lambda_*)$ and insert another horizontal line at distance w below. See Figure 2. Let S be the square enclosed between these two horizontal lines within the vertical strip bounding Λ_*. We construct a furthest point Voronoi diagram for the vertices of Λ_* inside S and a point-location data structure. To perform the query, we first test whether γ lies below $v(\Lambda_*)$. If that is the case, then γ clearly intersects Λ_*. If not and γ does not intersect S, then γ clearly lies above Λ_*. Otherwise, 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 γ lies completely above Λ_* if the distance between s and x is smaller than the radius of γ. This follows from the fact that, since γ is on the upper quarter-circle, if γ intersects S and lies above $v(\Lambda_*)$, then the part of S lying below γ is the same as the part of S lying inside the disc supporting the query arc.

We have thus shown how to implement the auxiliary data structure for a region of size m using $O(m)$ space and $Q(m) = O(\log m)$ query time. Since the auxiliary data structures for each region can be computed in $O(m \log m)$ time, the preprocessing time is $O(n \log^2 n)$. We have the following theorem.

Theorem 5 *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 $O(\log^2 n)$.*

6 Circular arc queries

We present three circular arc query problems and propose efficient data structures for answering them. In the next section, we will present applications of these queries in solving the empty lune problem and the circular visibility region problem for simple polygons. These query problems or the techniques for solving them may find other applications.

Let β be the bisector of two points p and q oriented such that p and q lie on the left and right side of β respectively. For any real number λ, define $D_{pq}(\lambda)$ to be the disk touching p and q whose center is on β at distance λ from $pq \cap \beta$ (where the distance is counted positive in the direction of β, negative in the opposite direction). Let $\gamma_{pq}(\lambda)$ denote the clockwise directed arc from p to q on the boundary of $D_{pq}(\lambda)$. Let H be the halfplane bounded by pq that contains the negative side of β.

6.1 Shortest directed arc

Given two query points p and q inside P, we want to find the minimum λ such that $\gamma_{pq}(\lambda)$ lies inside P. We call this the *shortest clockwise directed arc query*. The length of $\gamma_{pq}(\lambda)$ decreases with λ. To answer this query, we first study the same problem for a set S of line segments. We want to find the smallest λ such that $D_{pq}(\lambda) \cup H$ contains S. We denote this smallest value of λ by $\lambda_*(p, q, S)$.

Lemma 6 *Given a set S of n line segments. There is a data structure of size $O(n \log n)$ that can be computed in time $O(n \log n)$ that allows to determine $\lambda_*(p, q, S)$ in $O(\log^2 n)$ time for any two query points p and q.*

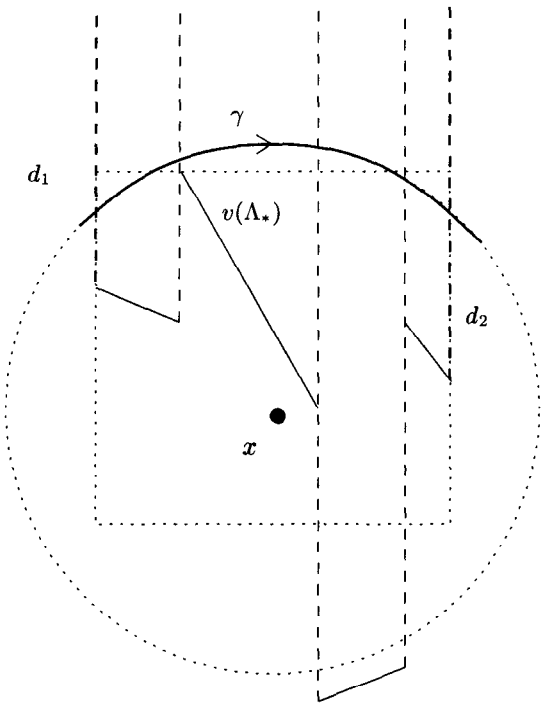

Figure 2: Λ_* is shown in solid segments. The curve γ is shown in bold. The part of the square S lying below γ is the same as the part of S lying inside the circle supporting the query arc.

Proof: $\gamma_{pq}(\lambda_*(p,q,S))$ must be pq or in contact with a convex hull vertex of S. We construct the convex hull of S, and build a range tree on the convex hull vertices, so that we can retrieve any interval of the convex hull as a disjoint union of $O(\log n)$ canonical subsets. The range tree has size $O(n \log n)$.

The query procedure works as follows. We first compute the two intersection points between the convex hull and the line through pq in $O(\log n)$ time. The range tree now allows us to retrieve the convex hull vertices on the positive side of β. For each canonical subset W, we will determine in $O(\log n)$ time the smallest λ_W such that $D_{pq}(\lambda_W) \cup H$ contains W. Then the maximum of λ_W, over all canonical subsets W, is $\lambda_*(p,q,S)$.

Given a canonical subset W, λ_W is the unique value where the center u of $D_{pq}(\lambda_W)$ has the same distance from the furthest point in W as from p (and q). We transform this problem to a ray shooting problem in three-dimensional space as follows: Let $\mathcal{U}$ be the unit paraboloid $z = x^2 + y^2$. For a point $w = (x,y) \in \mathbb{R}^2$, consider the point $w' = (x,y,x^2+y^2) \in \mathcal{U}$, and let $\pi(w)$ be the plane tangent to $\mathcal{U}$ in w'. Let $\mathcal{L}$ be the lower envelope of the planes $\pi(w)$ for all $w \in W$ and let ℓ be the line $\pi(p) \cap \pi(q)$. Then λ_W corresponds to the unique point u' where ℓ intersects $\mathcal{L}$.

We can find this intersection point by shooting a linear ray along ℓ from a point below $\mathcal{L}$. This query can be answered in $O(\log |W|)$ time using a data structure of $O(|W|)$ space and $O(|W|)$ preprocessing time. [9].

Summing over all nodes in the range tree, we obtain a total space of $O(n \log n)$ and a preprocessing time of $O(n \log n)$. The query time is $O(\log^2 n)$. □

To find the shortest $\gamma_{pq}(\lambda)$ inside P, we first retrieve the sequence of $O(\log n)$ regions connecting the trapezoids containing p and q by Lemma 3. For each region r identified, we retrieve the sequence Λ of trapezoids intersected by the path in the dual tree from the trapezoid containing p to the trapezoid containing q. Let Λ_r be the parts of the polygon edges that bound Λ and lie on the right of this dual tree path. We query the precomputed data structure of Lemma 6 for Λ_r in $O(\log^2 n)$ time to find the smallest λ such that $D_{pq}(\lambda) \cup H$ contains Λ_r. The maximum λ among the $O(\log n)$ regions would yield the shortest clockwise directed arc from p to q if this arc is not intercepted by the boundary of P. We test this in $O(\log^2 n)$ time using the data structure for circular ray shooting. The total query time is $O(\log^3 n)$.

The correctness of the query algorithm hinges on the fact that the shortest clockwise directed arc from p to q must pass through a vertex in Λ_r for some region r identified by Lemma 3. Such a vertex must lie on the shortest directed geodesic path from p to q. This path turns right at every vertex if a circular arc from p to q exists. Now, it suffices to observe that the dual tree

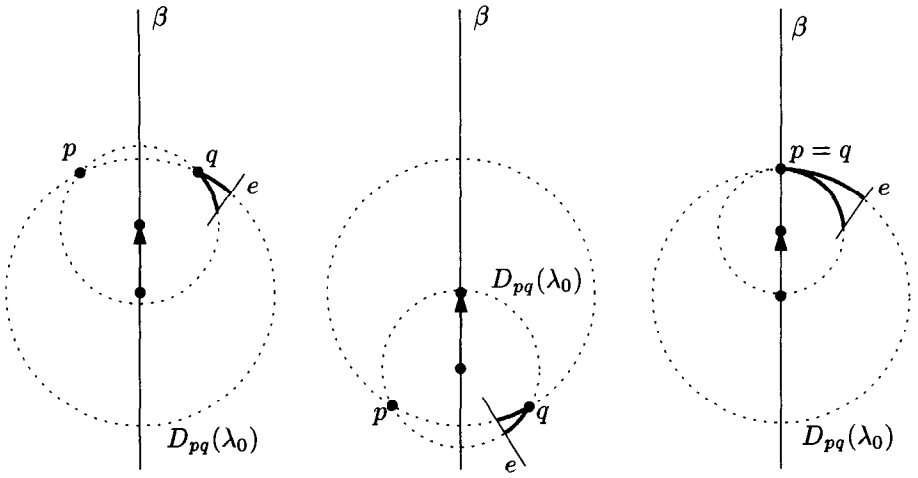

Figure 3: Arcs bending to the right.

path between the trapezoids containing p and q visits the same sequence of trapezoids as this geodesic path, and that the vertices of the geodesic path lie to the right of the dual tree path.

We have thus proven the following result.

Theorem 7 *Given a simple polygon P of size n, we can compute in time $O(n \log^2 n)$ a data structure of size $O(n \log^2 n)$ that, given query points p and q in P, returns the shortest clockwise directed arc inside P from p to q. The query time is $O(\log^3 n)$.*

6.2 Arc bending

We are given points p and q inside P, and a clockwise directed query arc γ on $D_{pq}(\lambda_0)$ for some λ_0 such that γ starts from q and ends on an edge e of P. We want to bend γ and stop when γ hits a vertex of P (which may be an endpoint of e) or when γ becomes tangent to e. The arc γ is bent by increasing λ from λ_0. Since γ is parameterized by p, q and λ, we denote it by $\alpha_{pq}(\lambda)$, see Figure 3. In other words, we want to find the smallest $\lambda \geq \lambda_0$ such that $\alpha_{pq}(\lambda)$ passes through a vertex of P or is tangent to e. The vertex hit or the tangential point on e will also be returned.

The query procedure is as follows. We first determine the endpoint w of e that $\alpha_{pq}(\lambda)$ would encounter when λ increases. Let λ_1 be the value so that $\alpha_{pq}(\lambda_1)$ reaches w. We also compute the value λ_2 such that $\alpha_{pq}(\lambda_2)$ is tangent to e. Apply Lemma 3 to find a sequence of $O(\log n)$ regions connecting the trapezoids containing q and w. For each such region, we also have a pair of doors d_1 and d_2 through which we pass. Let Λ be the sequence of trapezoids connecting d_1 and d_2. There is a directed path ρ in the dual tree that passes through Λ from d_1 to d_2. Let Λ_r be the parts of polygon edges that bound Λ and lie on the right of ρ. We precompute the data structure of Lemma 6 for Λ_r to find $\lambda_*(q, p, \Lambda_r)$. Our solution is the minimum of λ_1, λ_2 and $\lambda_*(q, p, \Lambda_r)$ among the $O(\log n)$ regions identified by Lemma 3. The correctness can be argued as for Theorem 7.

Theorem 8 *Given a simple polygon P of size n, we can compute in time $O(n \log^2 n)$ a data structure of size $O(n \log^2 n)$ that, given an arc $\alpha_{pq}(\lambda_0)$ ending on an edge e of P, returns the smallest $\lambda \geq \lambda_0$ such that $\alpha_{pq}(\lambda)$ hits a vertex of P or is tangent to e. The vertex hit or the tangential point on e is also returned. The query time is $O(\log^3 n)$.*

6.3 Arc pushing

Given query points p and q in P and an arc $\gamma_{pq}(\lambda_*)$ inside P, we want to report the largest $\lambda \geq \lambda_*$ such that $\gamma_{pq}(\lambda)$ lies inside P. We call this the *arc pushing query*.

To this end, we first solve a subproblem. We want to preprocess a set S of line segments to report, given query points p, q and λ_0 such that S lies outside $D_{pq}(\lambda_0)$, the largest $\lambda \geq \lambda_0$ such that $D_{pq}(\lambda)$ does not intersect S. We use $\lambda^*(p, q, \lambda_0, S)$ to denote this solution. If $\lambda^*(p, q, \lambda_0, S)$ is finite, then $\gamma_{pq}(\lambda^*(p, q, \lambda_0, S))$ touches a line segment or an endpoint of a line segment in S, because $\gamma_{qp}(\lambda^*(p, q, \lambda_0, S))$ lies inside $D_{pq}(\lambda_0)$ and so cannot touch any line segment in S.

Lemma 9 *Given a set S of n line segments, we can compute a data structure that, given query points p, q and λ_0 such that S lies outside $D_{pq}(\lambda_0)$, decides in $O(\log n)$ time whether $\lambda^*(p, q, \lambda_0, S) > \lambda$ for any $\lambda \geq \lambda_0$. The data structure can also return $\lambda^*(p, q, \lambda_0, S)$ and the corresponding contact point with S in $O(\log^2 n)$ time. The preprocessing time is $O(n \log n)$ and the storage is $O(n)$.*

Proof: The data structure is simply the Voronoi diagram $\mathcal{V}$ of S, preprocessed for point location [15, 16]. Point-location yields the line segment in S closest to the center of $D_{pq}(\lambda)$ and so we can check whether $D_{pq}(\lambda)$ intersects S. We have $\lambda^*(p, q, \lambda_0, S) < \lambda$ if and only if the intersection is not empty. This takes $O(\log n)$ time. $\lambda^*(p, q, \lambda_0, S)$ can be obtained by applying parametric searching [3], adding another $O(\log n)$ factor to the running time. □

Note that the smaller endpoint of the interval can of course be found by simply reversing the orientation of the line β. As is often the case, the parametric search in this data structure can be replaced by randomization.

The data structure used in the following theorem consists of two hierarchical decompositions $\mathcal{H}_v$ and $\mathcal{H}_h$ that are constructed from vertical and horizontal trapezoidal maps of P, and uses the structure of Lemma 9 as its auxiliary structure. Details can be found in the full paper.

Theorem 10 *Given a simple polygon P of size n, we can compute a data structure that, given a clockwise directed arc $\gamma_{pq}(\lambda_*)$ inside P, returns the largest $\lambda \geq \lambda_*$ such that $\gamma_{pq}(\lambda)$ lies inside P and the corresponding contact point with P. The query time is $O(\log^3 n)$. The preprocessing time is $O(n \log^2 n)$ and the storage is $O(n \log n)$.*

7 Applications

7.1 Empty Lune in Polygon

The *empty lune* of p and q is defined by the range of λ such that $\gamma_{pq}(\lambda)$ or $\gamma_{qp}(\lambda)$ lies inside P. The lune itself is the union of these clockwise directed arcs from p to q or from q to p. Given P, we want to preprocess it so that the empty lune can be reported efficiently for any query points p and q. Let's focus on the subrange of λ such that $\gamma_{pq}(\lambda)$ lies inside P. The smaller endpoint λ_* of the subrange is specified by the shortest clockwise directed arc from p to q which can be found in $O(\log^3 n)$ by Theorem 7. With $\gamma_{pq}(\lambda_*)$, we can apply Theorem 10 to find the larger endpoint of the subrange in $O(\log^3 n)$ time. The subrange of λ such that $\gamma_{pq}(\lambda)$ lies inside P can be found symmetrically.

Theorem 11 *Given a simple polygon P with n edges, a data structure of size $O(n \log n)$ can be computed in time $O(n \log^2 n)$ that allows to find the lune defined by two points p and q in P in $O(\log^3 n)$ time.*

7.2 Circular visibility regions

In this section we show how our arc bending and pushing queries can be used for computing the *circular visibility region* $CV(p)$ of a point p in a simple polygon P. This is the region of all points q such that there exists a circular arc inside P that connects p and q.

We will preprocess P into a data structure, so that given a point p, we can report $CV(p)$ in $O(m \log^3 n)$ time, where m is the output size.

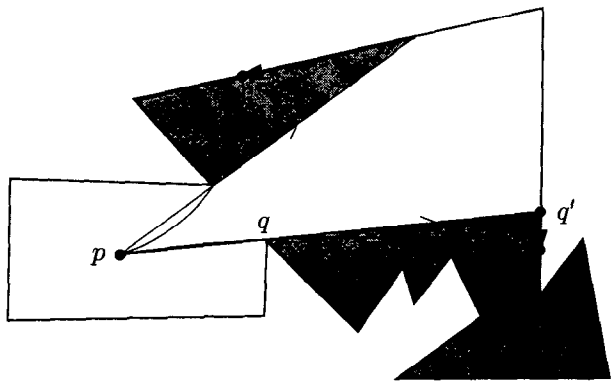

Figure 4: Starting with infinite-radius arcs

We start by computing the straight-line visibility polygon $V(p)$ of p in $O(k \log n)$ time, where $k \leq m$ is the complexity of $V(p)$. This can be done using linear ray shooting and shortest path queries, as follows. First, we shoot a linear ray from p in an arbitrary direction and record the edge e intersected. Then we sweep the ray in clockwise direction around p to identify the vertices of $V(p)$. The next vertex of $V(p)$ in clockwise direction can be obtained as follows: Let v be the endpoint of e in the clockwise direction, and find the shortest geodesic path from p to v in $O(\log n)$ time [13]. Let now w be the vertex on this shortest geodesic path closest to p. Extend pw using a linear ray shooting query to intersect e in a point w'. (If w is identical to v, then $w = w'$). The vertices w' and w are the next vertices of $V(p)$. We continue sweeping using the ray pw. Since finding a vertex of $V(p)$ can be done in time $O(\log n)$, the total running time for the sweep is $O(k \log n)$ time. This is $O(m \log n)$ as $k \leq m$.

For each reflex vertex q that lies on the boundary of $V(p)$ and for which only one of its incident edges is (partly) visible from p, there is a point q' on the boundary of P such that qq' separates the visibility region of p from the shadow areas (the connected regions of points in P that are not visible from p); see Figure 4).

We handle each shadow area separately as follows: we orient P such that p lies to the left of q, and q to the left of q'. The segments pq and qq' can be viewed as clockwise directed arcs $\gamma_{pq}(\lambda)$ and $\alpha_{pq}(\lambda)$ where $\lambda = -\infty$. When we increase λ, $\alpha_{pq}(\lambda)$ will sweep the shadow area. Notice that $\gamma_{pq}(\lambda)$ only sweeps what was already visible. The only interesting things happen when $\gamma_{pq}(\lambda)$ hits an edge or a vertex of P, or $\alpha_{pq}(\lambda)$ hits a vertex of P or becomes tangent to an edge.

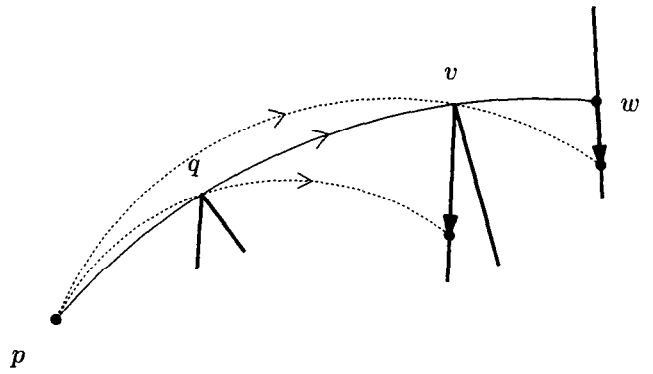

Figure 5: Splitting an arc and handling the two resulting shadow areas separately

- If the interior of $\alpha_{pq}(\lambda)$ hits a reflex vertex v of P (see Figure 5), then the destination of $\alpha_{pq}(\lambda)$ will switch to the newly visible edge incident to v and a new sweep arc $\alpha_{pv}(\lambda)$ is generated. The value of λ for which $\alpha_{pq}(\lambda)$ hits v can be found by an arc bending query (see section 6.2).
- If $\gamma_{pq}(\lambda)$ hits a vertex v of P (Figure 6, top-left) or an edge e of P (Figure 6, bottom-left), then $\alpha_{pq}(\lambda)$ is an arc on the boundary of $CV(p)$, and we are done with this shadow area. The value of λ for this event can be determined with an arc pushing query (see section 7.1).
- If the destination of $\alpha_{pq}(\lambda)$ reaches a reflex vertex v of P as depicted in Figure 6 (top-right), then we can extend $\alpha_{pq}(\lambda)$ and determine its new destination w with a circular ray shooting query (see section 5). The extension between v and w is an arc on the boundary of $CV(p)$. The value of λ at which this event occurs is determined by an arc bending query.
- Likewise, if $\alpha_{pq}(\lambda)$ becomes tangent to an edge at a point v, we can also extend $\alpha_{pq}(\lambda)$ as in the last event. The value of λ at which tangency is reached is determined by an arc bending query.

Handling each event takes $O(\log^3 n)$ time, and the total number of events is proportional to the number of vertices of $CV(p)$.

Theorem 12 *Given a simple polygon P with n edges, a data structure of size $O(n \log^2 n)$ can be computed in time $O(n \log^2 n)$ that returns the circular visibility region of a query point p in P in time $O(m \log^3 n)$, where m is the number of vertices of the resulting region.*

Acknowledgments. The empty lunes problem was suggested by Leo Guibas. 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] P. K. Agarwal and M. Sharir. Circular visibility of a simple polygon from a fixed point. *Internat. J. Comput. Geom. Appl.*, 3:1–25, March 1993.

[3] P. K. Agarwal, M. Sharir, and S. Toledo. Applications of parametric searching in geometric optimization. *J. Algorithms*, 17:292–318, 1994.

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

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

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

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

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

[9] D. P. Dobkin and D. G. Kirkpatrick. Determining the separation of preprocessed polyhedra – a unified approach. In *Proc. 17th Internat. Colloq. Automata Lang. Program. Lecture Notes Comput. Sci.*, vol. 443, pages 400–413. Springer-Verlag, 1990.

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

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

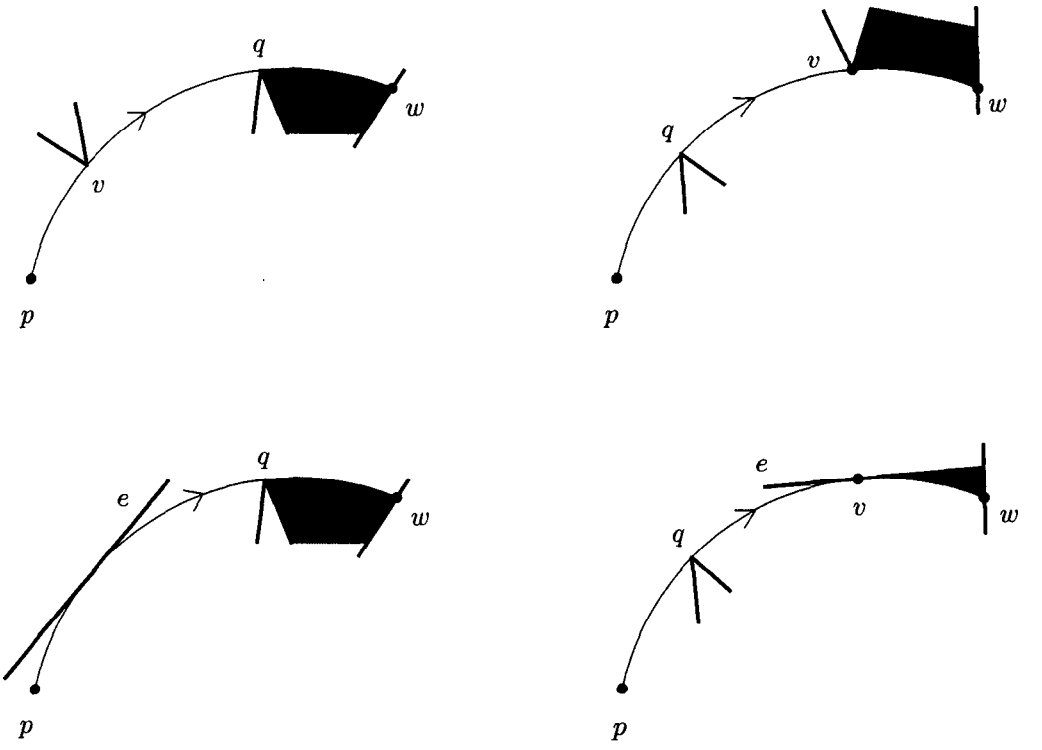

Figure 6: Events generating boundary arcs of $CV(p)$.

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

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

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

[15] D. G. Kirkpatrick. Efficient computation of continuous skeletons. In *Proc. 20th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 18–27, 1979.

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