Optimal Shortest Path and Minimum-Link Path Queries Between Two Convex Polygons in the Presence of Obstacles

Yi-Jen Chiang and Roberto Tamassia

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-94-03
Febuary 1994

Optimal Shortest Path and Minimum-Link Path Queries Between Two Convex Polygons in the Presence of Obstacles *

Yi-Jen Chiang Roberto Tamassia

Department of Computer Science
Brown University
Providence, R. I. 02912–1910

Abstract

We present efficient algorithms for shortest-path and minimum-link-path queries between two convex polygons inside a simple polygon P, which acts as an obstacle to be avoided. Let n be the number of vertices of P, and h the total number of vertices of the query polygons. We show that shortest-path queries can be performed optimally in time $O(\log h + \log n)$ (plus $O(k)$ time for reporting the k edges of the path) using a data structure with $O(n)$ space and preprocessing time, and that minimum-link-path queries can be performed in optimal time $O(\log h + \log n)$ (plus $O(k)$ to report the k links), with $O(n^3)$ space and preprocessing time.

We also extend our results to the dynamic case, and give a unified data structure that supports both queries for convex polygons in the same region of a connected planar subdivision $\mathcal{S}$. The update operations consist of insertions and deletions of edges and vertices. Let n be the current number of vertices in $\mathcal{S}$. The data structure uses $O(n)$ space, supports updates in $O(\log^2 n)$ time, and performs shortest-path and minimum-link-path queries in times $O(\log h + \log^2 n)$ (plus $O(k)$ to report the k edges of the path) and $O(\log h + k \log^2 n)$, respectively. Performing shortest-path queries is a variation of the well-studied *separation* problem, which has not been efficiently solved before in the presence of obstacles. Also, it was not previously known how to perform minimum-link-path queries in a dynamic environment, even for two-point queries.

(February 1994)

*An extended abstract of this paper has been presented at the *Army Research Office and MSI Stony Brook Workshop on Computational Geometry*, Raleigh, North Carolina, October, 1993. Research supported in part by the National Science Foundation under grant CCR-9007851, by the U.S. Army Research Office under grants DAAL03-91-G-0035 and DAAH04-93-0134, and by the Office of Naval Research and the Defense Advanced Research Projects Agency under contract N00014-91-J-4052, ARPA order 8225.

1 Introduction

In this paper, we present efficient algorithms for shortest-path and minimum-link-path queries between two convex polygons inside a simple polygon, which acts as an obstacle to be avoided. We give efficient techniques for both the static and dynamic versions of the problem.

Let R_1 and R_2 be two convex polygons with a total of h vertices that lie inside a simple polygon P with n vertices. The (*geodesic*) shortest path $\pi_G(R_1, R_2)$ is the polygonal chain with the shortest length among all polygonal chains joining a point of R_1 and a point of R_2 without crossing edges of P. A minimum-link path $\pi_L(R_1, R_2)$ is a polygonal chain with the minimum number of edges (called *links*) among all polygonal chains joining a point of R_1 and a point of R_2 without crossing edges of P. The number of links in $\pi_L(R_1, R_2)$ is called the *link distance* $d_L(R_1, R_2)$.

The related problem of computing the length of the shortest path between two polygons R_1 and R_2 *without obstacle* P has been extensively studied; this problem is also known as finding the *separation* of the two polygons [11], denoted by $\sigma(R_1, R_2)$. If both R_1 and R_2 are convex their separation can be computed in $O(\log h)$ time [7, 12, 5, 11]; if only one of them is convex an $O(h)$-time algorithm is given in [7]; if neither is convex, an optimal algorithm is recently given by Amato [1], who improves the previous result of Kirkpatrick [18] from $O(h \log h)$ to $O(h)$.

Although there has been a lot of work on the separation problem, the more general shortest-path problem for two objects *in the presence of obstacle* P has been previously studied only for the simple case when the objects are points, for which there exist efficient static [16] and dynamic [6, 15] solutions. The static technique of [16] supports two-point shortest-path queries in optimal $O(\log n)$ time (plus $O(k)$ if the k edges of the path are reported), employing a data structure that uses $O(n)$ space and can be built in linear time. The dynamic technique of [6] performs shortest-path queries between two points in the same region of a connected planar subdivision $\mathcal{S}$ with n vertices in $O(\log^3 n)$ time (plus $O(k)$ to report the k edges of the path), using a data structure with $O(n \log n)$ space that can support updates (insertions and deletions of edges and vertices) of $\mathcal{S}$ each in $O(\log^3 n)$ time. The very recent result of [15] improves the query and update times to $O(\log^2 n)$, with space complexity also improved to $O(n)$.

The minimum-link path problem between two points has been extensively studied. In many applications, such as robotics, motion planning, VLSI and computer vision, the link distance often provides a more natural measure of path complexity than the Euclidean distance [17, 22, 27, 29, 31]. For example, in a robot system, a straight-line navigation is often much cheaper than rotation, thus it is desirable to minimize the number of turns in path planning [27, 31]. Also, in graph drawing, it is often desirable to minimize the number of bends [28, 32].

All previously known techniques for the minimum-link path problem are restricted to the static environment, where updates to the problem instance are not allowed. The method of [29] computes a minimum-link path between two *fixed* points inside a simple polygon in linear time. In [31], a scheme based on *window partition* can answer link distance queries from a *fixed* source in $O(\log n)$ time, after $O(n)$ time preprocessing. The best known results are due to Arkin, Mitchell and Suri [2]. Their data structure uses $O(n^3)$ space and preprocessing time, and supports minimum-link-path queries between two points and between two segments in optimal $O(\log n)$ time (plus $O(k)$ if the k links are reported). Their technique can also perform minimum-link-path queries between two convex polygons, however, in non-optimal $O(\log h \log n)$ time. Also, efficient parallel algorithms are given in [3].

There are other results on the variations of the minimum-link-path problem. Efficient algorithms for link diameter and link center are given in [10, 14, 19, 17, 24, 23, 30]. A minimum-link path between two fixed points in a multiply connected polygon can be computed efficiently [22]. Sequential and parallel algorithms for *rectilinear* link distance are respectively given by de Berg [8] and Lingas *et al.* [20]. De Berg *et al.* [9] study the problem of finding a shortest rectilinear path among rectilinear obstacles. Mitchell *et al.* [21] consider the problem of finding a shortest path with at most K links between two query points inside a simple polygon, where K is an input parameter.

Our main results are outlined as follows.

- Let P be a simple polygon with n vertices. There exists an optimal data structure that supports shortest-path queries between two convex polygons with a total of h vertices inside P in time $O(\log h + \log n)$ (plus $O(k)$ if the k links of the path are reported), using $O(n)$ space and preprocessing time; all bounds are worst-case.
- Let P be a simple polygon with n vertices. There exists a data structure that supports minimum-link-path queries between two convex polygons with a total of h vertices inside P in optimal time $O(\log h + \log n)$ (plus $O(k)$ if the k links of the path are reported), using $O(n^3)$ space and preprocessing time; all bounds are worst-case.
- Let $\mathcal{S}$ be a connected planar subdivision whose current number of vertices is n. Shortest-path and minimum-link-path queries between two convex polygons with a total of h vertices that lie in the same region of $\mathcal{S}$ can be performed in times $O(\log h + \log^2 n)$ (plus $O(k)$ to report the k links of the path) and $O(\log h + k \log^2 n)$, respectively, using a fully dynamic data structure that uses $O(n)$ space and supports insertions and deletions of vertices and edges of $\mathcal{S}$ each in $O(\log^2 n)$ time; all bounds are worst-case.

The contributions of this work can be summarized as follows:

- We provide the first optimal data structure for shortest-path queries between two convex polygons inside a simple polygon P that acts as an obstacle. No efficient data structure was known before to support such queries. All previous techniques either consider the case where P is not present or the case where the query objects are points.
- We provide the first data structure for minimum-link-path queries between two convex polygons inside a simple polygon P in optimal $O(\log h + \log n)$ time. The previous best result [2] has query time $O(\log h \log n)$ (and the same space and preprocessing time as ours).
- We provide the first fully dynamic data structure for shortest-path queries between two convex polygons in the same region of a connected planar subdivision $\mathcal{S}$. No such data structure was known before even for the static version.
- We provide the first fully dynamic data structure for minimum-link-path queries between two convex polygons in the same region of a connected planar subdivision $\mathcal{S}$. No such data structure was known before even for two-point queries.

We summarize the comparisions of our results with the previous ones in Tables 1-4.

Static Shortest Paths	Query Type	Query	Space	Preprocessing
Guibas-Hershberger [16]	two query points	$\log n$ *	n *	n *
This paper	two query convex polygons	$\log h + \log n$ *	n *	n *

* optimal

Table 1 Results for static shortest-path queries.

We briefly outline our techniques. Given the available static techniques with optimal query time for shortest paths and minimum-link paths between *two points*, our main task in performing the *two-polygon* queries is to find two points $p \in R_1$ and $q \in R_2$ such that their shortest path or minimum-link path gives the desired path between R_1 and R_2. As we shall see later, the notion of

Dynamic Shortest Paths	Query Type	Query	Space	Update
Chiang-Preparata-Tamassia [6]	two query points	$\log^3 n$	$n \log n$	$\log^3 n$
Goodrich-Tamassia [15]	two query points	$\log^2 n$	n *	$\log^2 n$
This paper	two query convex polygons	$\log h + \log^2 n$	n *	$\log^2 n$

* optimal

Table 2 Results for dynamic shortest-path queries.

Static Min-Link Paths	Query Type	Query	Space	Preprocessing
Suri [31]	one fixed point and one query point	$\log n$ *	n *	n *
Arkin-Mitchell-Suri [2]	two query points/segments	$\log n$ *	n^3	n^3
	two query convex polygons	$\log h \log n$	n^3	n^3
This paper	two query convex polygons	$\log h + \log n$ *	n^3	n^3

* optimal

Table 3 Results for static minimum-link-path queries.

geodesic hourglass between R_1 and R_2 is central to our method. The geodesic hourglass is *open* if R_1 and R_2 are mutually visible, and *closed* otherwise. As for shortest-path queries, the case where R_1 and R_2 are mutually visible is a basic case that, surprisingly, turns out to be nontrivial (the complication comes from the fact that the shortest path in this case may still consist of more than one link), and our solution makes use of interesting geometric properties. If R_1 and R_2 are not visible, then the geodesic hourglass gives two points p_1 and p_2 that are respectively visible from R_1 and R_2 such that the shortest path between any point of R_1 and any point of R_2 must go through p_1 and p_2. Then the shortest path between R_1 and R_2 is the union of the shortest paths between R_1 and p_1, between p_2 and R_2 (both are basic cases), and between two points p_1 and p_2. The geodesic hourglass also gives useful information for minimum-link-path queries. When it is open, a minimum-link path is just a single segment; if it is closed, then it gives two edges such that extending them to intersect R_1 and R_2 gives the desired points p and q whose minimum-link path is a minimum-link path $\pi_L(R_1, R_2)$. However, it seems difficult to compute the geodesic hourglass in optimal time. Interestingly, we can get around this difficulty by computing a *pseudo hourglass* that gives all the information we need about the geodesic hourglass. We also extend these results to the dynamic case, by giving the first dynamic method for minimum-link-path queries between two points.

The rest of this paper is organized as follows. In Section 2 we briefly review the basic geometric notions used by our method. Section 3 shows how to perform shortest-path queries in the static environment, in particular how to compute the pseudo hourglass and how to handle the nontrivial

Dynamic Min-Link Paths	Query Type	Query	Space	Update
This paper	two query convex polygons	$\log h + k \log^2 n$	n *	$\log^2 n$

* optimal

Table 4 Results for dynamic minimum-link-path queries.

basic case where two query polygons are mutually visible. Sections 4, 5 and 6 are devoted to dynamic shortest-path, static minimum-link-path, and dynamic minimum-link-path queries, respectively.

2 Preliminaries

For the geometric terminology used in this paper, see [26]. A *connected planar subdivision* $\mathcal{S}$ is a subdivision of the plane into polygonal regions whose underlying planar graph is connected. Thus each region of $\mathcal{S}$ is a simple polygon P. A polygonal chain γ is *monotone* if any horizontal line intersects it in a single point or in a single interval or not at all. A simple polygon P is *monotone* if its boundary consists of two monotone chains. A *cusp* of a polygon P is a vertex v whose interior angle is greater than π and whose adjacent vertices are both strictly above (lower cusp) or strictly below (upper cusp) v. If we draw from a cusp v of P two horizontal rays that terminate when they first meet the edges of P, the resulting segments to the left and right of v are called *left lid* and *right lid* of v, respectively. A polygon is monotone if and only if it has no cusps.

The notion of *window partition* was introduced in [31]. Given a point or a line segment s in region P, let $WP(s)$ denote the partition of P into maximally-connected subregions with the same link distance from s; $WP(s)$ is called the *window partition* of P with respect to s. Associated with $WP(s)$ is a set of *windows*, which are chords of P that serve as boundaries between adjacent subregions of the partition.

Given two points p and q that lie in the same region P of $\mathcal{S}$ (or in the same simple polygon P), it is well known that their shortest path $\pi_G(p,q)$ is unique and only turns at the vertices of P. On the contrary, a minimum-link path is not unique and may turn at any point inside P. Adopting the terminology of [31], we define the (unique) *greedy minimum-link path* $\pi_L(p,q)$ to be the minimum-link path whose first and last links are respectively the extensions of the first and last links of $\pi_G(p,q)$, and whose other links are the extensions of the windows of $WP(p)$. The number of links in $\pi_L(p,q)$ is then the link distance $d_L(p,q)$. In the following we use the term "window" to refer to both a window and its extension.

Given a shortest path $\pi_G(p,q)$, an edge $e \in \pi_G(p,q)$ is an *inflection edge* if its predecessor and its successor lie on opposite sides of e. It is easily seen that an edge $e \in \pi_G(p,q)$ is an inflection edge if and only if it is an internal common tangent of the boundaries of P.

Given two convex polygons R_1 and R_2 inside P, we say that R_1 and R_2 are *mutually visible* if there exists a line l connecting R_1 and R_2 without crossing any edge of P; we call such line l a *visibility link* between R_1 and R_2. Now we define the *left and right boundaries* B_L and B_R of P with respect to R_1 and R_2 when they are not mutually visible through a horizontal line. For $i = 1,2$, let u_i and d_i be the highest and lowest vertices of R_i, respectively. Without loss of generality, we assume that $y(u_1) \geq y(u_2)$ (otherwise we exchange the roles of R_1 and R_2). We choose $q_1 \in \{u_1, d_1\}$ and $q_2 \in \{u_2, d_2\}$ such that (i) the subpolygon P' of P delimited by both e_1 and e_2 contains both R_1 and R_2, where e_i is a horizontal chord of P going through q_i, $i = 1,2$, and (ii) among the four shortest paths $\pi_G(u_1,u_2)$, $\pi_G(u_1,d_2)$, $\pi_G(d_1,u_2)$ and $\pi_G(d_1,d_2)$, $\pi_G(q_1,q_2)$ has the largest number of cusps (see Fig. 1). Now P' is bounded by e_1, e_2 and two polygonal chains. We define B_L and B_R as these two polygonal chains of P': B_L is the one to the left of $\pi_G(q_1,q_2)$ when we walk along $\pi_G(q_1,q_2)$ from q_2 to q_1, and B_R is the one to the right (see Fig. 1). Clearly, any shortest path π between a point in R_1 and a point in R_2 can only touch the vertices of P on B_L and B_R, and the inflection edges of π are those edges that have one endpoint on B_L and the other endpoint on B_R.

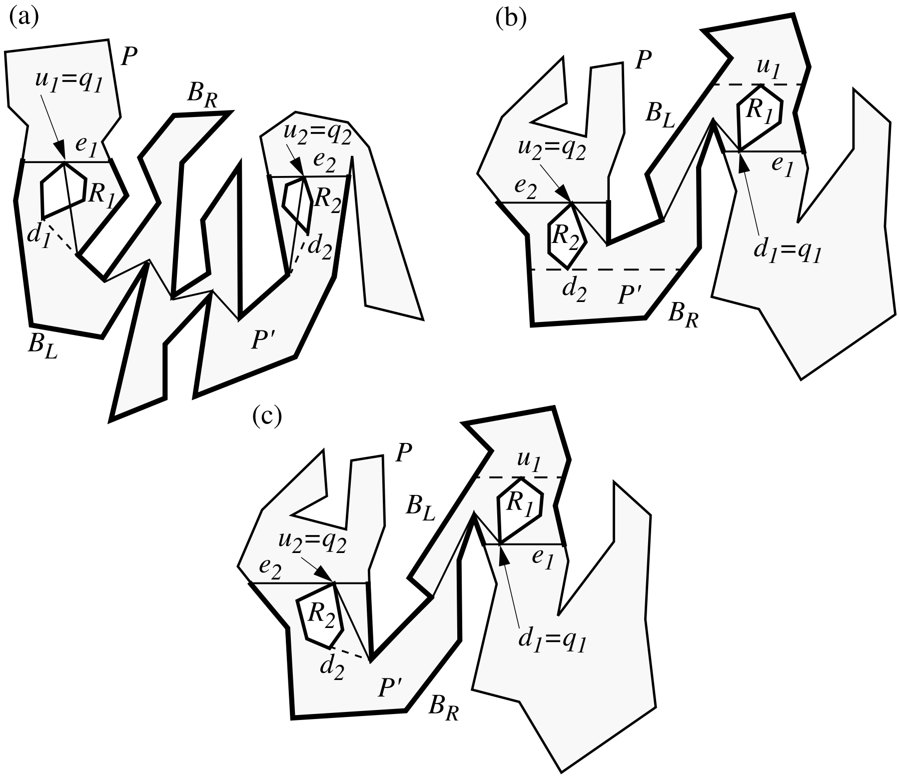

Figure 1: Left and right boundaries B_L and B_R of P: (a) several choices of (q_1, q_2) satisfy condition (ii) but only one satisfies (i); (b) several choices of (q_1, q_2) satisfy condition (i) (e.g., (u_1, d_2) is also valid) but only one satisfies (ii); (c) neither (i) nor (ii) alone enforces a unique choice of (q_1, q_2), but their conjunction does.

3 Static Shortest Path Queries

In this section we show how to compute the shortest path $\pi_G(R_1, R_2)$ between two convex polygons R_1 and R_2 with a total of h vertices inside an n-vertex simple polygon P. The data structure of Guibas and Hershberger [16] computes the shortest path $\pi_G(p, q)$ between any two points p and q inside P in $O(\log n)$ time, where in $O(\log n)$ time we get an implicit representation (a balanced binary tree) and the length of $\pi_G(p, q)$, and using additional $O(k)$ time to retrieve the k links we get the actual path. Point-location queries can also be performed in $O(\log n)$ time. The data structure uses $O(n)$ space and can be built in $O(n)$ time after triangulating P (again in $O(n)$ time by Chazelle's linear-time triangulation algorithm [4]). We modify this data structure so that associated with the implicit representation of a shortest path π_G, there are two balanced binary trees respectively maintaining the inflection edges and the cusps on π_G in their path order. The balanced binary tree representing π_G and the two associated binary trees support split and splice operations, so that we can extract a portion of π_G in logarithmic time.

With this data structure, our task is to find points $p \in R_1$ and $q \in R_2$ such that $\pi_G(p, q) = \pi_G(R_1, R_2)$. We say that p and q *realize* $\pi_G(R_1, R_2)$. Note that p and q lie on the boundaries of R_1 and R_2 but are not necessarily vertices.

To obtain a better intuition, let us imagine surrounding R_1 and R_2 with a rubber band inside P. The resulting shape is called the *relative convex hull* of R_1 and R_2. It is formed by four pieces: shortest paths $\pi_1 = \pi_G(a_1, a_2)$, $\pi_2 = \pi_G(b_1, b_2)$ $(a_1, b_1 \in R_1$ and $a_2, b_2 \in R_2)$, and the boundaries of R_1 and R_2 farther away from each other. We call a_1, b_1, a_2, and b_2 the *geodesic tangent points*, and π_1 and π_2 the *geodesic external tangents* of R_1 and R_2. Note that if π_1 consists of more than one link, then the first (resp. last) link of π_1 is a common tangent between R_1 (resp. R_2) and the convex hull inside P of a portion of the boundary of P (see Fig. 2), and similarly for π_2. Let $s_1 = (a_1, b_1)$ and $s_2 = (a_2, b_2)$. If we replace R_1 and R_2 with s_1 and s_2, then the relative convex hull of s_1 and s_2 is the *hourglass* $H(s_1, s_2)$ bounded by s_1, s_2, π_1, and π_2. Note that π_1 and π_2 stay unchanged. We call $H(s_1, s_2)$ the *geodesic hourglass* between R_1 and R_2. We say that $H(s_1, s_2)$ is *open* if π_1 and π_2 do not intersect, and *closed* otherwise. When $H(s_1, s_2)$ is closed, there is a vertex p_1 at which π_1 and π_2 join together, and a vertex p_2 at which the two paths separate (possibly $p_1 = p_2$); we call p_1 and p_2 the *apices* of $H(s_1, s_2)$ (see Fig. 2(b)). Also, we say that $\pi_G(a_1, p_1)$ and $\pi_G(b_1, p_1)$ form a *funnel* $F(s_1)$. The only internal common tangent ρ_1 of P among all edges of $F(s_1)$ is called the *penetration* of $F(s_1)$, and similarly for ρ_2 in funnel $F(s_2)$ (see Fig. 2(b)). Hereafter we use H_G to denote the geodesic hourglass, and a_1, b_1 ($\in R_1$), a_2, b_2 ($\in R_2$) to denote the geodesic tangent points.

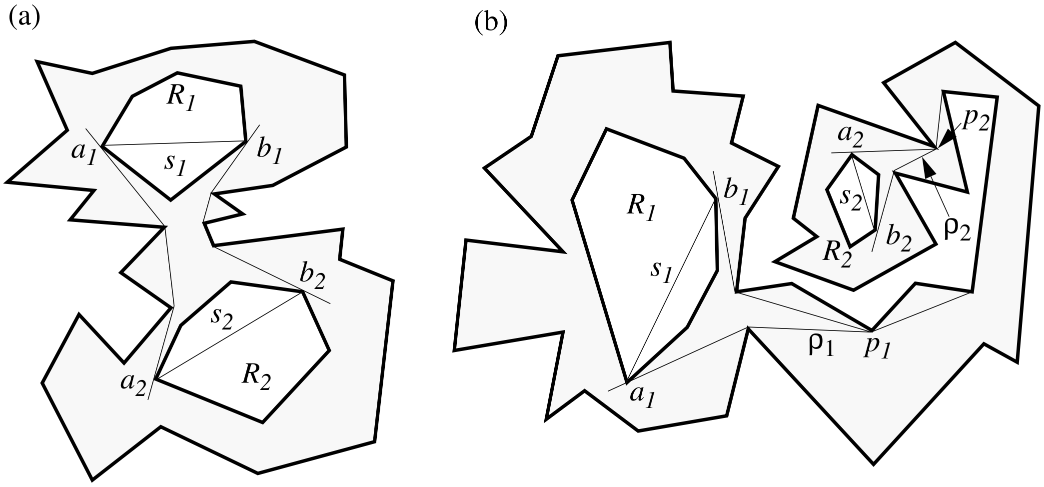

Figure 2: Geodesic hourglass H_G and geodesic external tangents: (a) H_G is open; (b) H_G is closed.

Observe that H_G is open if and only if R_1 and R_2 are mutually visible (see Fig. 2(a)). If H_G is closed, then $\pi_G(p', q')$ between any point $p' \in R_1$ and any point $q' \in R_2$ must go through p_1 and p_2 (see Fig. 2(b)). Thus $\pi_G(R_1, R_2)$ must go through p_1 and p_2, i.e., $\pi_G(R_1, R_2) = \pi_G(R_1, p_1) \cup \pi_G(p_1, p_2) \cup \pi_G(p_2, R_2)$. Since R_1 and p_1 are mutually visible, the algorithm for computing $\pi_G(R_1, R_2)$ when R_1 and R_2 are mutually visible can be used to compute $\pi_G(R_1, p_1)$ as well, and similarly for $\pi_G(p_2, R_2)$. In summary, we need to handle the following two main tasks: (i) deciding whether H_G is open or closed, and finding apices p_1 and p_2 when H_G is closed, and (ii) computing $\pi_G(R_1, R_2)$ when R_1 and R_2 are mutually visible.

3.1 The Pseudo Geodesic Hourglass

We first discuss how to compute the information about geodesic hourglass H_G in optimal $O(\log h + \log n)$ time. A straightforward method is to compute H_G directly. As shown in [2], we can compute the geodesic external tangents between R_1 and R_2 (and hence H_G) by a binary search mimicking the algorithm [25] for finding ordinary common tangents, where in each iteration we compute the shortest path between two chosen points rather than the segment joining them. However, this results in a computation of $O(\log h \log n)$ time. Also, it seems difficult to compute H_G in optimal time.

To overcome the difficulty, we notice that it is not necessary to compute H_G exactly. As for shortest-path queries, we only need to know whether H_G is open or closed, and the apices p_1 and p_2 of H_G when it is closed; as for minimum-link path queries (see Section 5), we only need to know a visibility link between R_1 and R_2 when H_G is open, and the penetrations ρ_1 and ρ_2 of H_G when it is closed. Interestingly, we can obtain the above information by computing a *pseudo hourglass* H'' with the property that if H'' is open then H_G is open, and if H'' is closed then H_G is closed with the same penetrations and apices. We first describe the algorithm and then justify its correctness.

Algorithm Pseudo-Hourglass

1. Ignore P and compute the ordinary external common tangents (a'_1, a'_2) and (b'_1, b'_2) between R_1 and R_2, using the algorithm of Overmars and van Leeuwen [25], where $a'_1, b'_1 \in R_1$ and $a'_2, b'_2 \in R_2$. Let $s'_1 = (a'_1, b'_1)$ and $s'_2 = (a'_2, b'_2)$. Compute shortest paths $\pi_1 = \pi_G(a'_1, a'_2)$ and $\pi_2 = \pi_G(b'_1, b'_2)$. If they are disjoint (i.e., neither has an inflection edge) then the hourglass $H' = H(s'_1, s'_2)$ is open. In this case s'_1 and s'_2 are mutually visible, implying that R_1 and R_2 are mutually visible. Use algorithm [25] to compute an internal common tangent l between π_1 and π_2, report {open with visibility link l} and stop.
2. Else (π_1 and π_2 are not disjoint) H' is closed. Now the geodesic external tangents (which constitute H_G) must go through vertices of P, and it is still possible that H_G is open. Let u_1 and d_1 be the highest and lowest vertices of R_1, respectively, and similarly for u_2 and d_2 in R_2. Assume that $y(u_1) \geq y(u_2)$ (otherwise exchange the roles of R_1 and R_2). Compute shortest paths $\pi_G(u_1, u_2)$, $\pi_G(u_1, d_2)$, $\pi_G(d_1, u_2)$ and $\pi_G(d_1, d_2)$. Take π as the one with the largest number of cusps (break ties arbitrarily). Consider π as *oriented from R_2 to R_1*.
3. From $R_i, i = 1, 2$, compute horizontal projection points l_i and r_i respectively on the left and right boundaries B_L and B_R of P, by discriminating the following cases.
   (a) π has no cusp at all.
   There are two subcases.
      i. $y(d_1) \leq y(u_2)$, i.e., there is a vertical overlap between horizontal projections of R_1 and R_2.
      In this case the line $l : y = y(u_2)$ connects R_1 and R_2 without being blocked (to be proved in Lemma 1). Report {open with visibility link l} and stop.
      ii. There is no vertical overlap (see Fig. 3).
      Project u_1 horizontally to the left and right on the boundaries B_L and B_R of P to get points l_1 and r_1, respectively (via point location), and similarly project d_2 to the left and right to get l_2 and r_2.

   (b) π has cusps.
   Consider R_1 (and symmetrically for R_2). Look at cusp c_1 of π closest to R_1, and denote π' the portion of π from c_1 to the point on R_1. Without loss of generality, assume that c_1 is a lower cusp. There are two cases.
      i. c_1 is lower than or as low as d_1 ($y(c_1) \leq y(d_1)$).
      This means that R_1 is entirely blocked by c_1. Project u_1 horizontally to the left and right to get l_1 and r_1, respectively.
      ii. c_1 is higher than d_1 ($y(c_1) > y(d_1)$).
      Then R_1 "stretches" beyond c_1. Consider the following subcases.
         A. The first link of π' (oriented toward R_1) goes toward left (see Fig. 5(a)(b)).
         Project both u_1 and d_1 to the right to get r_1 and l_1, respectively. Also a special-

case checking is needed: if segment (d_1, l_1) intersects R_2 at v, then report {open with visibility link $l = (d_1, v)$} and stop.

B. The first link of π' goes toward right.
Project both u_1 and d_1 to the left to get l_1 and r_1, respectively. Again perform a special-case checking: if segment (d_1, r_1) intersects R_2 at v, then report {open with visibility link $l = (d_1, v)$} and stop.

4. Compute shortest paths $\pi_l = \pi_G(l_1, l_2)$ and $\pi_r = \pi_G(r_1, r_2)$. Extract the "left bounding convex chain" C_{L1} for R_1 as the portion of π_l from l_1 to x, where x is the first vertex v_1 on B_R or the first point c with $y(c) = y(l_1)$ or the second cusp c_2, whichever is closest to R_1, or $x = l_2$ if none of v_1, c and c_2 exists. Note that C_{L1} includes the first inflection edge if $x = v_1$. Similarly extract the "right bounding convex chain" C_{R1} of R_1 from π_r. The left and right bounding convex chains C_{L2} and C_{R2} of R_2 are computed analogously (see Fig. 4).
5. Compute *pseudo tangent points* $a_1'', b_1'' \in R_1$ and $a_2'', b_2'' \in R_2$ such that the *pseudo hourglass* H'' formed by $\pi_G(a_1'', a_2''), \pi_G(b_1'', b_2''), s_1'' = (a_1'', b_1'')$ and $s_2'' = (a_2'', b_2'')$ has the desired property. Point a_1'' is computed from R_1 and C_{L1} by the following steps (and analogously b_1'', a_2'' and b_2'' are computed from R_1 and C_{R1}, from R_2 and C_{L2}, and from R_2 and C_{R2}, respectively).
   (a) Check whether R_1 intersects C_{L1} (viewing $C_{L1} = \pi_G(l_1, x)$ as a convex polygon with edge (l_1, x) added) using the algorithm [5], which runs in logarithmic time and also reports a common point g inside both R_1 and C_{L1} if they intersect. If $R_1 \cap C_{L1} = \emptyset$, then find the internal common tangent $t = (v, w)$ between R_1 and C_{L1}, $v \in R_1, w \in C_{L1}$, such that R_1 lies on the right side of t if t is directed from w to v (see Fig. 3). Note that only one of the two internal common tangents between R_1 and C_{L1} satisfies the criterion for t. Now check whether t intersects C_{R1} via a binary search on C_{R1}.
      i. $t \cap C_{R1} = \emptyset$. Set $a_1'' := v$.
      ii. $t \cap C_{R1} = \{y_1, y_2\}$. Let C'_{R1} be the portion of C_{R1} between points y_1 and y_2. Find the external common tangent $t' = (v', w')$ between R_1 and C'_{R1}, $v' \in R_1, w' \in C'_{R1}$, such that both R_1 and C'_{R1} lie on the right side of t' if t' is directed from w' to v'. Set $a_1'' := v'$. (See Fig. 3.)
   (b) Else ($R_1 \cap C_{L1} \neq \emptyset$, with a common point g inside both R_1 and C_{L1}), then there is only one edge of C_{L1} intersecting R_1 (to be proved in Lemma 3). Compute this edge (u, b) by applying Lemma 3. Suppose b is closer to R_2 than u; call b the *blocking point*. Consider the following two cases.
      i. The blocking point b is on the left boundary B_L.
      Compute a_1'' as the tangent point from b to R_1 such that R_1 is on the right side of (b, a_1'') when (b, a_1'') is directed toward a_1''. (See Fig. 7(a)(b).)
      ii. The blocking point b is on the right boundary B_R.
      Take C as the convex portion of π_l (oriented from R_1 to R_2) from b to z, where z is the first vertex v_1' on B_L again or the first point c' with $y(c') = y(b)$ or the second cusp c_2' after b, whichever is closest to R_1. Note that such v_1' always exists since $\pi_l = \pi_G(l_1, l_2)$ finally goes to $l_2 \in B_L$, and that C includes the first inflection edge after b if $z = v_1'$. Find the external common tangent $t'' = (v'', w'')$ between R_1 and C, $v'' \in R_1, w'' \in C$, such that both R_1 and C lie on the right side of t'' if t'' is directed from w'' to v''. Set $a_1'' := v''$. (See Fig. 7(c)–(f).)
6. Compute shortest paths $\pi_1 = \pi_G(a_1'', a_2'')$ and $\pi_2 = \pi_G(b_1'', b_2'')$ to form pseudo hourglass H''. Check whether H'' is open or closed.
   (a) H'' is open (neither π_1 nor π_2 has an inflection edge).

Compute an internal common tangent l between π_1 and π_2, report {open with visibility link l} and stop.

(b) H'' is closed.
Penetration $\rho_1 = (w_1, p_1)$ is chosen from the first inflection edges of π_1 and of π_2 (one of such edges might be missing) as the one that is closer to R_1, and the endpoint p_1 of ρ_1 farther away from R_1 is an apex. The other penetration ρ_2 and apex p_2 are found similarly. Recall that an inflection edge has one endpoint on B_L and the other on B_R. To decide whether the first and last links of π_1 and π_2 are inflection edges, points a_1'' and a_2'' are viewed as on B_L, and b_1'' and b_2'' as on B_R. After computing ρ_1, ρ_2, p_1 and p_2, report {closed with penetrations ρ_1 and ρ_2 and apices p_1 and p_2}, and stop.

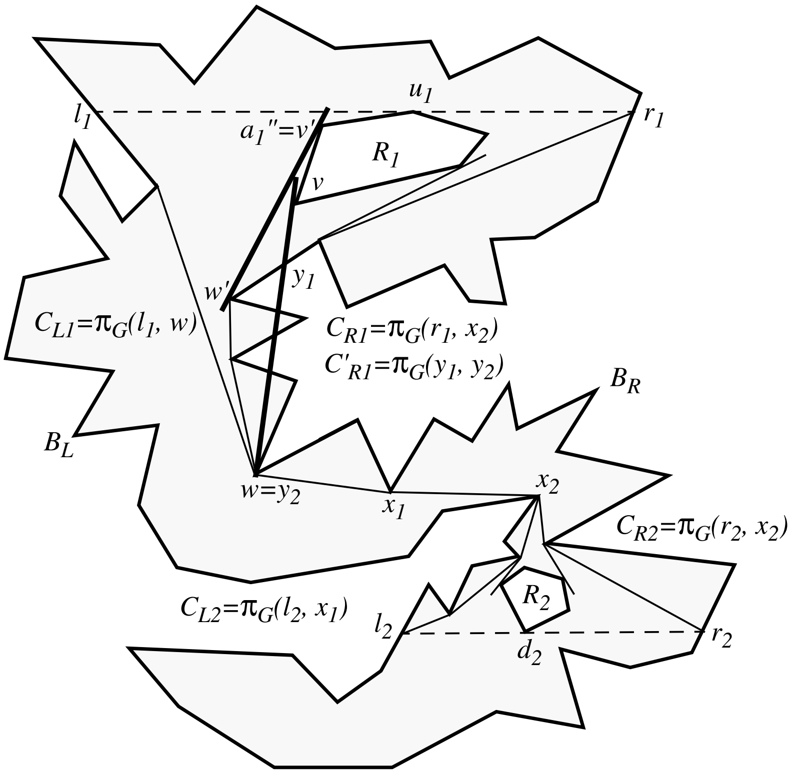

Figure 3: A running example for Algorithm *Pseudo-Hourglass* in the case where π has no cusps and $C_{L1} \cap R_1 = \emptyset$.

The correctness of the algorithm is justified by the following lemmas.

Lemma 1 *In step 3(a)i of Algorithm* Pseudo-Hourglass*, the line* $l : y = y(u_2)$ *connects* R_1 *and* R_2 *without being blocked.*

Proof: Recall that there is a vertical overlap between the horizontal projections of R_1 and R_2, i.e., $y(u_1) \geq y(u_2) \geq y(d_1)$. By the definition of π and the fact that π has no cusp, the shortest path between u_1 and u_2 must have no cusp. Thus any lower cusp c' of P in between R_1 and R_2 has $y(c') \geq y(u_2)$. Similarly, any upper cusp c'' of P in between R_1 and R_2 has $y(c'') \leq \max\{y(d_1), y(d_2)\}$. Note that $y(u_2) \geq \max\{y(d_1), y(d_2)\}$, therefore $y(c'') \leq y(u_2) \leq y(c')$, i.e., the line $l : y = y(u_2)$ connects R_1 and R_2 without being blocked. □

Lemma 2 *The projection points* l_i *and* r_i, $i = 1, 2$ *obtained in step 3 of Algorithm* Pseudo-Hourglass *lie on distinct boundaries of* P, *i.e.,* $l_i \in B_L$ *and* $r_i \in B_R$.

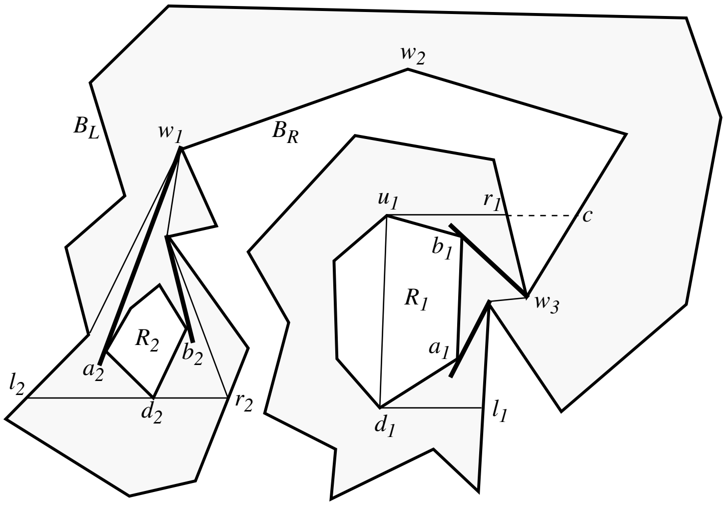

Figure 4: Step 4 of Algorithm *Pseudo-Hourglass* and proof of Lemma 4. As for step 4, notice how we get the bounding convex chains C_{L1}, C_{R1}, C_{L2} and C_{R2}, especially C_{R1} and C_{R2}; as for Lemma 4, note that R_1 and R_2 do not intersect any of the bounding convex chains, $S_1 = (u_1, r_1) \cup (u_1, d_1) \cup (d_1, l_1)$, $S_2 = (l_2, r_2)$, and $H_G = (a_1, b_1) \cup (a_2, b_2) \cup \pi_G(a_1, a_2) \cup \pi_G(b_1, b_2)$ is properly contained in $S_1 \cup S_2 \cup \pi_l \cup \pi_r$.

Proof: The claim is obvious for steps 3(a)ii and 3(b)i since l_i and r_i are obtained by projecting the same point to the left and right. Now consider step 3(b)iiA (step 3(b)iiB is similar). It is clear that r_1 is on B_R, so we look at l_1. If all points on π are higher than d_1, then the horizontal line $y = y(d_1)$ is not blocked by π and thus is to the left of π (recall that π is oriented from R_2 to R_1). So l_1 is on B_L (see Fig. 5(a)). On the other hand, if π contains some point c' lower than d_1, then for c_1 to be a lower cusp, there must be an upper cusp on π between c_1 and c' that is higher than c_1 (see Fig. 5(b)). Let c'' be such upper cusp closest to c_1, then the line $y = y(d_1)$ is blocked by c'' and the projection point l_1 is on B_L. □

Lemma 3 *In step 5b of Algorithm* Pseudo-Hourglass, *where $R_1 \cap C_{L1} \neq \emptyset$ with a common point g inside both R_1 and C_{L1}, there is only one edge of C_{L1} intersecting R_1. Furthermore, this edge (u, b) can be computed in $O(\log n)$ time.*

Proof: We prove the first part by contradiction. If there were more than one edge of C_{L1} intersecting R_1, say (v_1, v_2) and (v_2, v_3) (see Fig. 6(a)), then v_2 would be inside R_1 and would also be a vertex of P, contradicting the fact that R_1 is in a free space of P.

Now we show how to compute (u, b) in $O(\log n)$ time. Assume that l_1 is obtained in step 3 of Algorithm *Pseudo-Hourglass* by projecting u_1. Then u_1 is inside R_1 but outside C_{L1}, thus segment $(g, u_1) \in R_1$ intersects the boundary of C_{L1} (see Fig. 6(b)). By the first part of this lemma, there is only one edge (u, b) of C_{L1} that can be intersected by a segment inside R_1. Performing a binary search on C_{L1} to identify the edge intersected by (g, u_1), (u, b) can be found in $O(\log n)$ time. □

Lemma 4 *The pseudo hourglass H'' computed from steps 5 and 6 of Algorithm* Pseudo-Hourglass *has the property that if H'' is open then the geodesic hourglass H_G is open, and if H'' is closed with penetrations ρ_1 and ρ_2 and apices p_1 and p_2 then H_G is closed with the same penetrations and apices.*

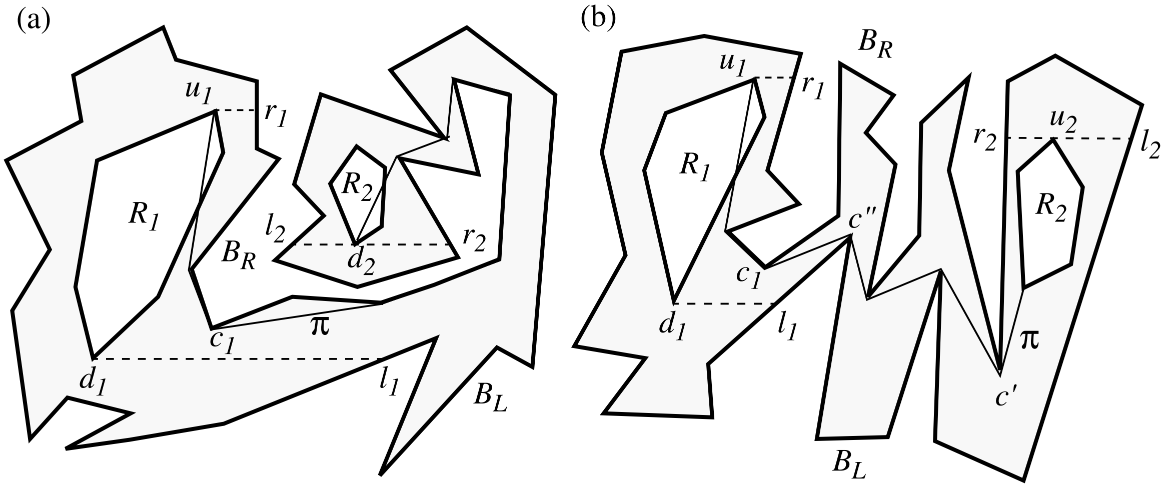

Figure 5: Step 3(b)iiA of Algorithm *Pseudo-Hourglass* and proof of Lemma 2: r_1 and l_1 are obtained by projecting u_1 and d_1 horizontally to the right; r_1 is on B_R and l_1 is on B_L.

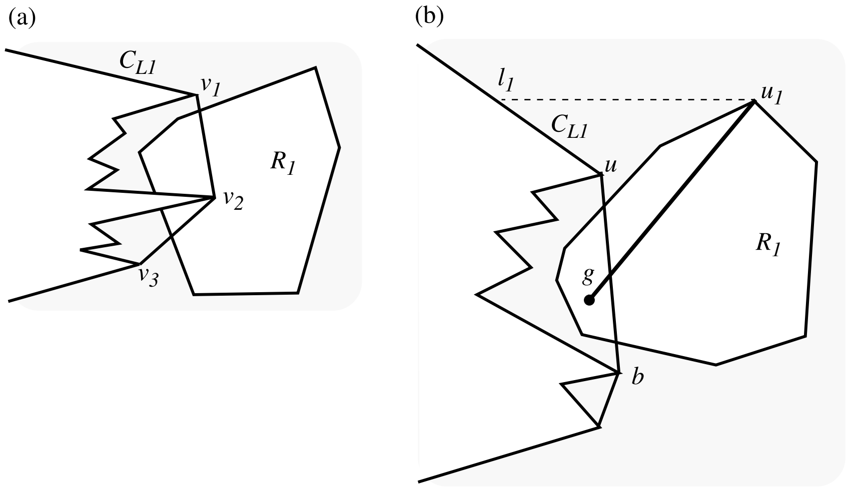

Figure 6: Proof of Lemma 3: (a) impossibility for C_{L1} to have more than one edge intersecting R_1; (b) finding edge (u, b).

Proof: Recall that $a_1, b_1 \in R_1$ and $a_2, b_2 \in R_2$ are the geodesic tangent points. We first consider the case in which the bounding convex chains C_{L1} and C_{R1} do no intersect R_1, and C_{L2} and C_{R2} do not intersect R_2 either (see Fig. 4). Define $S_i, i = 1, 2$, as follows. If l_i and r_i are obtained by projecting the same point of R_i then $S_i = (l_i, r_i)$; otherwise assuming without loss of generality that l_i is obtained from projecting d_i and r_i from u_i, then $S_i = (u_i, r_i) \cup (u_i, d_i) \cup (d_i, l_i)$. We observe that the area bounded by $S_1, S_2, \pi_l = \pi_G(l_1, l_2)$ and $\pi_r = \pi_G(r_1, r_2)$ properly contains H_G, therefore a_1 and b_1 are computed from the common tangents between R_1 and C_{L1}/C_{R1}, and similarly for a_2 and b_2 (see Fig. 4, and also Fig. 3 for one more example). These are exactly what we compute in steps 5a–5(a)ii, i.e., $H'' = H_G$, and the lemma follows.

Next, look at the case where at least one of the bounding convex chains intersects R_1 or R_2. Since a''_1, a''_2, b''_1 and b''_2 are computed independently, we consider only a''_1; the same argument applies for the others. As we have already seen, $a''_1 = a_1$ when C_{L1} does not intersect R_1, so we consider a''_1 when C_{L1} intersects R_1.

We claim that in this case either $a_1'' = a_1$, or $\pi_G(a_1'', a_2'')$ and $\pi_G(a_1, a_2)$ join together at a point before their first inflection edge (if any) closest to R_1. This implies that if $\pi_G(a_1, a_2)$ has no inflection edge (a case where whether H_G is open or closed is decided by $\pi_G(b_1, b_2)$) then $\pi_G(a_1'', a_2'')$ has no inflection edge either, and if ρ_1' is the first inflection edge of $\pi_G(a_1, a_2)$ (a case where H_G is closed with ρ_1' a candidate for ρ_1) then ρ_1' is also the first inflection edge of $\pi_G(a_1'', a_2'')$, and thus the lemma follows.

We now give the details for proving the above claim. Note that $\pi_G(a_1, a_2)$ joins π_l at some point then leaves π_l later, and similarly for $\pi_G(a_1'', a_2'')$. First, look at the case where the blocking point b is on B_L (step 5(b)i) and refer to Fig. 7(a)(b) to visualize the proof. By the definition of C_{L1} and the fact that b is on B_L, $\pi_G(l_1, b) \subseteq C_{L1}$ is the convex hull inside P of the boundary of B_L from l_1 to b and it does not touch B_R, so no vertex of B_R lies to the left of $(u, b) \in \pi_G(l_1, b)$. But a_1'' is to the left of (u, b), thus $(a_1'', b) \cap B_R = \emptyset$. We classify two subcases: (i) $(a_1'', b) \cap (B_L - \{b\}) = \emptyset$ and (ii) $(a_1'', b) \cap (B_L - \{b\}) \neq \emptyset$. For (i), let q be the vertex on $C_{L1} = \pi_G(l_1, x)$ immediately after b. Such q always exists since $b \neq x$: for (u, b) to intersect R_1, b cannot be l_2 or the first point c with $y(c) = y(l_1)$ or the second cusp c_2, and b cannot be the first vertex v_1 on B_R either since $b \in B_L$. Because C_{L1} is convex toward right, the chain $(u, b, q) \subseteq C_{L1}$ is convex toward right, but then (a_1'', b, q) is also convex toward right (see Fig. 7(a)). This means that the shortest path $\pi_G(a_1, p') \subseteq \pi_G(a_1, a_2)$ from a_1 to any point p' on π_l beyond b must go through b. Then the first link of $\pi_G(a_1, a_2)$ is (a_1'', b) since (a_1'', b) is tangent to R_1 and does not cross any boundary of P. Therefore $a_1'' = a_1$. For (ii), let CH be the convex hull inside P of the boundary of B_L between b and b', where b' is the intersection of B_L and (a_1'', b) such that CH is as large as possible while not intersecting R_1. Clearly $\pi_G(a_1, a_2)$ goes through b, starting with a common tangent between R_1 and CH then following CH up to b; likewise, $\pi_G(a_1'', a_2'')$ goes through b starting with a tangent from a_1'' to CH then following CH up to b (see Fig. 7(b)). Observe that $\pi_G(a_1'', a_2'')$ and $\pi_G(a_1, a_2)$ join together at a point on CH that is before b, and neither path has an inflection edge before reaching b, so the claim holds.

Now look at the case where b is on B_R (step 5(b)ii). There are four subcases: (1) $w'' \in B_L$ and $(w'', a_1'') \cap (B_L - \{w''\}) = \emptyset$; (2) $w'' \in B_L$ and $(w'', a_1'') \cap (B_L - \{w''\}) \neq \emptyset$; (3) $w'' \in B_R$ and $(w'', a_1'') \cap B_L = \emptyset$; and (4) $w'' \in B_R$ and $(w'', a_1'') \cap B_L \neq \emptyset$. For (1), let x_1 and x_2 be the vertices on π_l immediately before and after w'' (see Fig. 7(c)). Note that (x_1, w'') is an inflection edge, so the chain (x_1, w'', x_2) is convex toward right (although $\pi_G(b, w'')$ is convex toward left). But the slope of (w'', a_1'') is even bigger than the slope of (w'', x_1), thus (a_1'', w'', x_2) is also convex toward right. Similar to case (i), this means that $\pi_G(a_1, p') \subseteq \pi_G(a_1, a_2)$ from a_1 to any point p' on π_l beyond w'' must go through w'', but (a_1'', w'') is a tangent to R_1 not blocked by P and hence the first link of $\pi_G(a_1, a_2)$, i.e., $a_1'' = a_1$. Case (2) is similar to case (ii) as w'' plays the role of b, i.e., both $\pi_G(a_1, a_2)$ and $\pi_G(a_1'', a_2'')$ go through a convex hull CH inside P of some portion of B_L then reach w'', with no inflection edge up to w'' (see Fig. 7(d)). For (3), it is clear that $\pi_G(a_1, p') \subseteq \pi_G(a_1, a_2)$ from a_1 to any point p' on π_l beyond w'' must go through w'', but (a_1'', w'') is a tangent to R_1 not blocked by P, so (a_1'', w'') is the first link of $\pi_G(a_1, a_2)$ and $a_1'' = a_1$ (see Fig. 7(e)). For (4), let CH' be the convex hull inside P of the boundary of B_L from q_1 to q_2, where q_1 is the intersection of (w'', a_1'') and B_L closest to w'', and q_2 is the intersection of (w'', a_1'') and B_L such that CH' is as large as possible while not intersecting R_1. Then $\pi_G(a_1, a_2)$ goes through w'', starting with a common tangent between R_1 and CH', followed by a portion of CH', a common tangent s between CH' and $C = \pi_G(b, z)$, then a portion C' of C up to w''; likewise, $\pi_G(a_1'', a_2'')$ goes through w'' starting with a tangent from a_1'' to CH', followed by a portion of CH' then s then C' up to w'' (see Fig. 7(f)). Clearly, the paths $\pi_G(a_1, a_2)$ and $\pi_G(a_1'', a_2'')$ join together at some point on CH' before reaching their first inflection edge s. This completes our proof of the claim. □

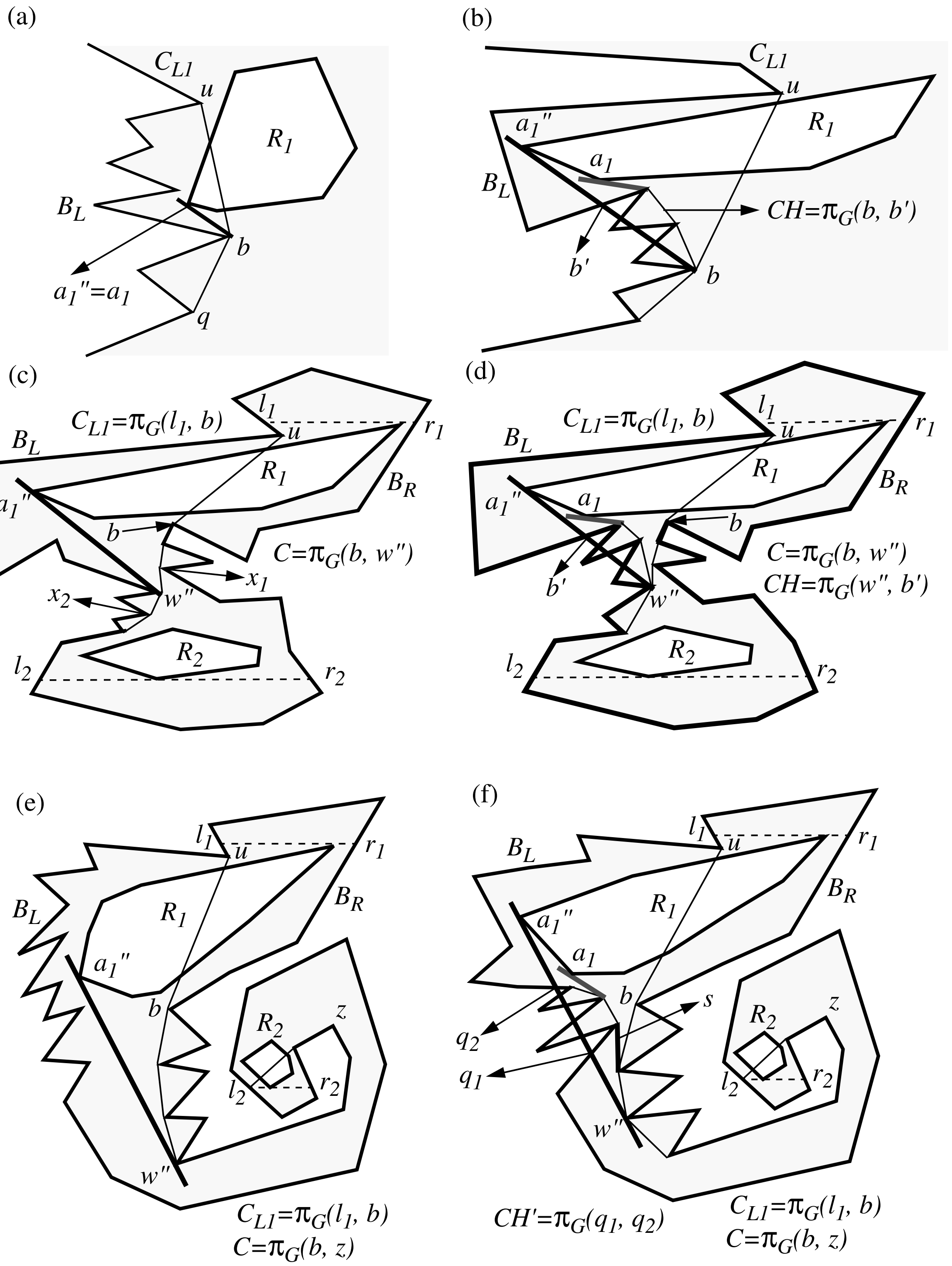

Figure 7: Steps 5(b)i–5(b)ii of Algorithm *Pseudo-Hourglass* and proof of Lemma 4: (a) $b \in B_L$ and $(b, a_1'') \cap (B_L - \{b\}) = \emptyset$; (b) $b \in B_L$ and $(b, a_1'') \cap (B_L - \{b\}) \neq \emptyset$; (c) $b \in B_R, w'' \in B_L$ and $(w'', a_1'') \cap (B_L - \{w''\}) = \emptyset$; (d) $b \in B_R, w'' \in B_L$ and $(w'', a_1'') \cap (B_L - \{w''\}) \neq \emptyset$; (e) $b \in B_R, w'' \in B_R$ and $(w'', a_1'') \cap B_L = \emptyset$; and (f) $b \in B_R, w'' \in B_R$ and $(w'', a_1'') \cap B_L \neq \emptyset$.

We conclude with the following lemma.

Lemma 5 *Algorithm* Pseudo-Hourglass *correctly decides whether the geodesic hourglass H_G is open or closed, giving a visibility link when it is open or giving the penetrations and apices of H_G when it is closed, in $O(\log h + \log n)$ time, which is optimal.*

Proof: The correctness follows from Lemmas 1–4. As for time complexity, recall from our data structure (described at the beginning of Section 3) that we can extract a portion of a shortest path (*path extraction* for short) via split/splice operations in logarithmic time. Step 1 performs $O(1)$ tangent computations and shortest-path queries. Step 2 performs four shortest-path queries. Step 3(a)i can be done in $O(1)$ time, and step 3(a)ii involves $O(1)$ point-location queries to find projection points. In Step 3b, we perform a path extraction; in steps 3(b)i and 3(b)ii, we perform $O(1)$ point-location queries to project points and also binary searches for special-case checkings. We compute two shortest-path queries and extract four bounding convex chains in step 4. Step 5a invloves $O(1)$ calls to algorithm [5], and $O(1)$ tangent computations and binary searches. Step 5(a)i can be done in $O(1)$ time, and step 5(a)ii performs $O(1)$ path extractions and tangent computations. Step 5b applies the computation of Lemma 3, which is a binary search. Steps 5(b)i–5(b)ii invlove $O(1)$ tangent computations (5(b)i and 5(b)ii) and path extractions (5(b)ii). Finally, we perform $O(1)$ shortest-path queries, tangent computations and binary searches in step 6. In summary, we perform a constant number of logarithmic-time computations, and the time complexity follows. □

3.2 The Case of Mutually Visible Query Polygons

We now discuss how to compute $\pi_G(R_1, R_2)$ when R_1 and R_2 are mutually visible, i.e., when the geodesic hourglass H_G is open. Surprisingly, this case turns out to be nontrivial, and its solution makes use of interesting geometric properties. Note that $\pi_G(R_1, R_2)$ in this case may still consist of more than one link (see, e.g., Fig. 8, where $\pi_G(R_1, R_2) = \pi_G(p, q)$).

Ignoring P and using any one of the methods for computing the separation of two convex polygons [7, 11, 12], we can find $p' \in R_1$ and $q' \in R_2$ with $length(p', q') = \sigma(R_1, R_2)$ in $O(\log h)$ time. Now we compute $\pi_G(p', q')$. If $\pi_G(p', q')$ has only one link, then (p', q') is not blocked by P and thus is the desired shortest path $\pi_G(R_1, R_2)$. Otherwise $\pi_G(p', q')$ must touch the boundary of P, and there are two cases: (1) $\pi_G(p', q')$ touches only one of the two geodesic external tangents $\pi_G(a_1, a_2)$ and $\pi_G(b_1, b_2)$; or (2) $\pi_G(p', q')$ touches both $\pi_G(a_1, a_2)$ and $\pi_G(b_1, b_2)$.

Lemma 6 *Let the geodesic hourglass H_G be open and (p', q') with $p' \in R_1$ and $q' \in R_2$ be the shortest path between R_1 and R_2 without obstacle P. If $\pi_G(p', q')$ touches only one of $\pi_G(a_1, a_2)$ and $\pi_G(b_1, b_2)$, say $\pi_G(a_1, a_2)$, then $\pi_G(R_1, R_2)$ touches $\pi_G(a_1, a_2)$ but does not touch $\pi_G(b_1, b_2)$.*

Proof: We refer to Fig. 8 to visualize the proof. Let (w, z) be any segment tangent to the convex chain $\pi_G(p', q')$, where $w \in R_1$ and $z \in R_2$. *Without obstacles* C and D, the distance between a point on the boundary of R_1 and a point on the boundary of R_2 is a *bimodal function*, i.e., it decreases and then increases, with the minimum occurring at p' and q'. In particular, moving w downward along the boundary of R_1 to any point w' and/or moving z downward along the boundary of R_2 to any point z' will cause $(w', z') > (w, z)$, and $\pi_G(w', z') \geq (w', z')$ since $\pi_G(w', z')$ may have to avoid obstacles. Thus if $p \in R_1$ and $q \in R_2$ satisfy $\pi_G(p, q) = \pi_G(R_1, R_2)$, then p must lie on the boundary $(w, ..., p')$ of R_1 counterclockwise from w to p', and q must lie on the clockwise boundary $(z, ..., q')$ of R_2. It follows that $\pi_G(p, q)$ touches $\pi_G(a_1, a_2)$ but does not touch $\pi_G(b_1, b_2)$. □

Therefore in the above situation (see Fig. 8), if t'_1 and t'_2 are the points of obstacle C where $\pi_G(p', q')$ first touches C and finally leaves C, respectively, and t_1 and t_2 are the points of C where $\pi_G(p, q)$ first touches C and finally leaves C (recall that $\pi_G(p, q) = \pi_G(R_1, R_2)$), then t_2 is the point where the shortest path $\pi_G(t'_1, R_2)$ from t'_1 to R_2 finally leaves C, and similarly for t_1.

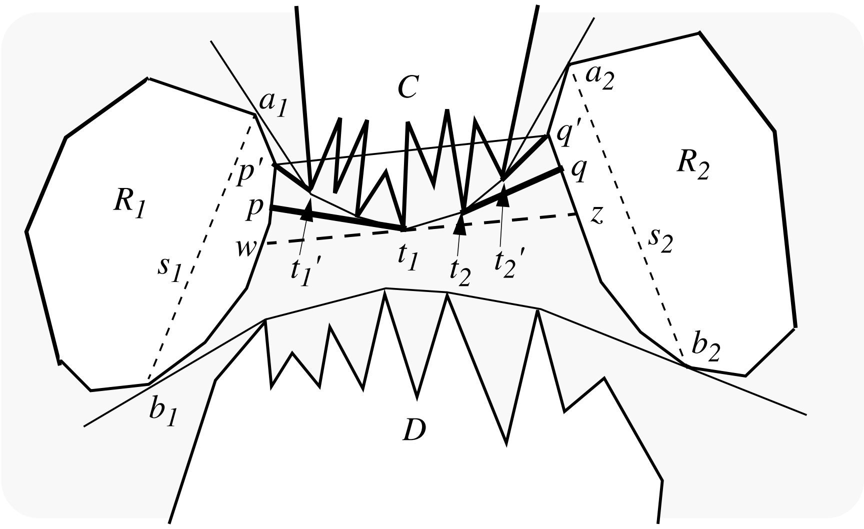

Figure 8: Lemma 6

We say that $t_2 \in C$ and $q \in R_2$ *realize* $\pi_G(t_1', R_2)$, and similarly for the other side. It is clear that $\pi_G(R_1, R_2)$ consists of $(p, t_1), \pi_G(t_1, t_2)$ (which is a portion of $\pi_G(p', q')$), and (t_2, q). So we only need to independently compute $t_2 \in C$ and $q \in R_2$ that realize $\pi_G(t_1', R_2)$, and by a similar algorithm to compute t_1 and p that realize $\pi_G(t_2', R_1)$.

Before describing how to compute t_2 and q (and similarly for t_1 and p), we first argue that the other case where $\pi_G(p', q')$ touches both $\pi_G(a_1, a_2)$ and $\pi_G(b_1, b_2)$ can be handled in the same way.

Lemma 7 *Let the geodesic hourglass H_G be open and (p', q') with $p' \in R_1$ and $q' \in R_2$ be the shortest path between R_1 and R_2 without obstacle P. If $\pi_G(p', q')$ touches both $\pi_G(a_1, a_2)$ and $\pi_G(b_1, b_2)$, say first $\pi_G(a_1, a_2)$ (entering at point t_1 and leaving at point t_3) and then $\pi_G(b_1, b_2)$ (entering at t_4 and leaving at t_2), then $\pi_G(R_1, R_2) = \pi_G(R_1, t_3) \cup (t_3, t_4) \cup \pi_G(t_4, R_2)$. (See Fig. 9.)*

Proof: We refer to Fig. 9. We extend (t_3, t_4) on both directions to intersect R_1 and R_2 at w and z, respectively. Notice that (w, z) is an internal common tangent of two convex chains $\pi_G(a_1, a_2)$ and $\pi_G(b_1, b_2)$. Again, without obstacles the distance between a point on R_1 and a point on R_2 is a bimodal function. In particular, moving z upward along the boundary of R_2 to any point z' and/or moving w downward along the boundary of R_1 to any point w' will make $(w', z') > (w, z)$. Observe that $\pi_G(w', z') \geq (w', z')$ since it may have to avoid the obstacles. Therefore the desired points $p \in R_1$ and $q \in R_2$ with $\pi_G(p, q) = \pi_G(R_1, R_2)$ must lie on the clockwise boundary $(p', ..., w)$ of R_1 and on the clockwise boundary $(q', ..., z)$ of R_2, respectively. It follows that $\pi_G(p, q)$ must be first tangent to $\pi_G(a_1, a_2)$ at some point, coincide with $\pi_G(a_1, a_2)$ from there to t_3, follow (t_3, t_4) to enter $\pi_G(b_1, b_2)$, join $\pi_G(b_1, b_2)$ from t_4 to some tangent point, which together with q are the two endpoints of the last link. Therefore $\pi_G(R_1, R_2) = \pi_G(R_1, t_3) \cup (t_3, t_4) \cup \pi_G(t_4, R_2)$. □

It is clear that for the above situation, what we need to do is to independently compute the two points that realize $\pi_G(R_1, t_3)$ and two points that realize $\pi_G(t_4, R_2)$.

We now discuss how to compute two points $t_2 \in C$ and $q \in R_2$ that realize $\pi_G(t_1', R_2)$ in the situation of Fig. 8; the other case (Fig. 9) can be handled analogously. Note that we only need to consider the two convex chains $\pi_G(u, t_2')$ (denoted by C_1) and the clockwise boundary $(v, ..., q')$ of R_2 (denoted by C_2), where (u, v) is the external common tangent between the convex hull of C and R_2 with $u \in C$ and $v \in R_2$. Our algorithm is based on the following useful properties.

Lemma 8 *Let $v_1, v_2, ..., v_k$ be a sequence of points on C_2 in clockwise order, and e_i' and e_i'' be the*

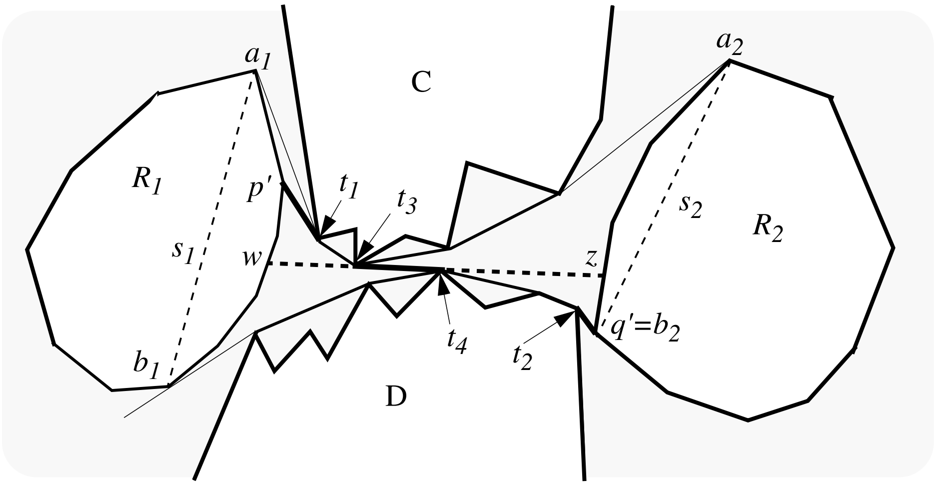

Figure 9: Lemma 7

two segments of C_2 incident on v_i with e_i' following e_i'' in clockwise order (e_i' and e_i'' are on the same straight line if v_i is not a vertex). From each v_i draw a line l_i tangent to C_1. Let θ_i be the angle formed by l_i and e_i' and measured from l_i clockwise to e_i', and ϕ_i be the angle formed by e_i'' and l_i and measured from e_i'' clockwise to l_i (see Fig. 10). Then $\theta_1 < \theta_2 < ... < \theta_k$ and $\phi_1 > \phi_2 > ... > \phi_k$. Also, if $\theta_i \geq \frac{\pi}{2}$ then $\phi_{i+1} < \frac{\pi}{2}$, and similarly if $\phi_{i+1} \geq \frac{\pi}{2}$ then $\theta_i < \frac{\pi}{2}$.

Proof: We extend tangent l_{i+1} to intersect l_i at some point r, and also extend e_i' on both directions so that θ_{i+1}' and ϕ_i' are both exterior angles of $\triangle r v_i v_{i+1}$ (see Fig. 10). It follows that $\theta_{i+1} \geq \theta_{i+1}' > \theta_i$ (the equality holds if v_{i+1} is not a vertex), and $\phi_i \geq \phi_i' > \phi_{i+1}$ (the equality holds if v_i is not a vertex). For the last statement, consider $\triangle r v_i v_{i+1}$. It is clear that at most one of θ_i and ϕ_{i+1} can be larger than or equal to $\frac{\pi}{2}$. □

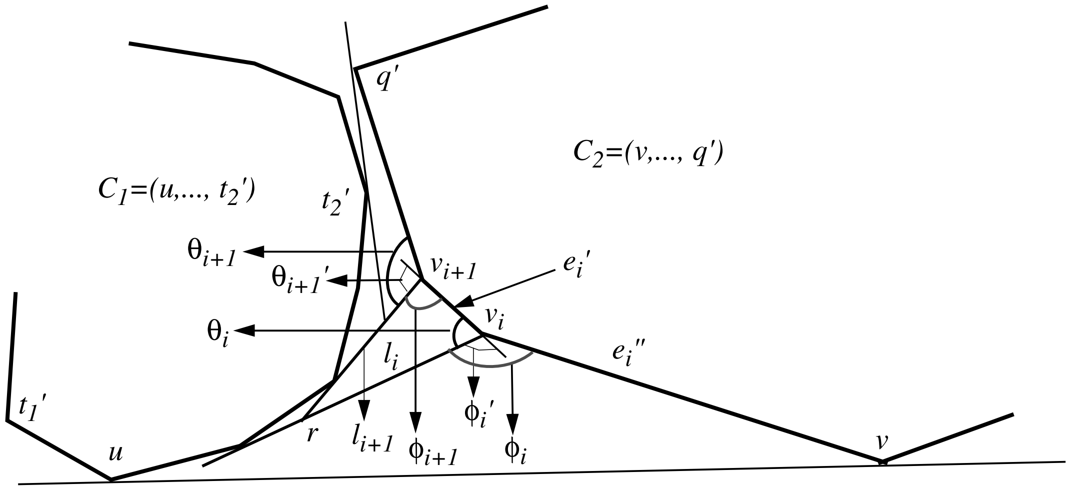

Figure 10: Lemma 8

Lemma 9 *Let $v_1, v_2, ..., v_k$ and each θ_i and ϕ_i be as defined in Lemma 8. If $\phi_i \geq \frac{\pi}{2}$, then $\pi_G(v_i, t_1') < \pi_G(v_{i-1}, t_1')$. Similarly, if $\theta_i \geq \frac{\pi}{2}$, then $\pi_G(v_i, t_1') < \pi_G(v_{i+1}, t_1')$.*

Proof: We refer to Fig. 11 to visualize the proof. Let the tangent points on C_1 of l_i and of l_{i-1}

be u_j and u_m, respectively, where $u_1, u_2, ...$ are the vertices of C_1 in counterclockwise order. We extend each of (u_s, u_{s+1}) to the right to intersect C_2 at some point u'_s, $s = m, m+1, ..., j-1$. In $\triangle v_i u_j u'_{j-1}$, $(u_j, u'_{j-1}) > (u_j, v_i)$ since $\phi_i \geq \frac{\pi}{2}$ is the biggest angle. Adding (u_j, u_{j-1}) to both sides of the inequality, we have $\pi_G(v_i, u_{j-1}) = (v_i, u_j) + (u_j, u_{j-1}) < (u'_{j-1}, u_j) + (u_j, u_{j-1}) = (u'_{j-1}, u_{j-1})$, thus $\pi_G(v_i, t'_1) = \pi_G(v_i, u_{j-1}) + \pi_G(u_{j-1}, t'_1) < (u'_{j-1}, u_{j-1}) + \pi_G(u_{j-1}, t'_1) = \pi_G(u'_{j-1}, t'_1)$, i.e., $\pi_G(v_i, t'_1) < \pi_G(u'_{j-1}, t'_1)$. Now, $\phi'_i = \angle u'_{j-2} u'_{j-1} u_j$ is an exterior angle of $\triangle u'_{j-1} v_i u_j$, so $\phi'_i > \phi_i \geq \frac{\pi}{2}$. By the previous argument, $\pi_G(u'_{j-1}, t'_1) < \pi_G(u'_{j-2}, t'_1)$. Applying this process repeatedly, we have $\pi_G(v_i, t'_1) < \pi_G(u'_{j-1}, t'_1) < \pi_G(u'_{j-2}, t'_1) < ... < \pi_G(v_{i-1}, t'_1)$. The other statement can be proved in the same way. □

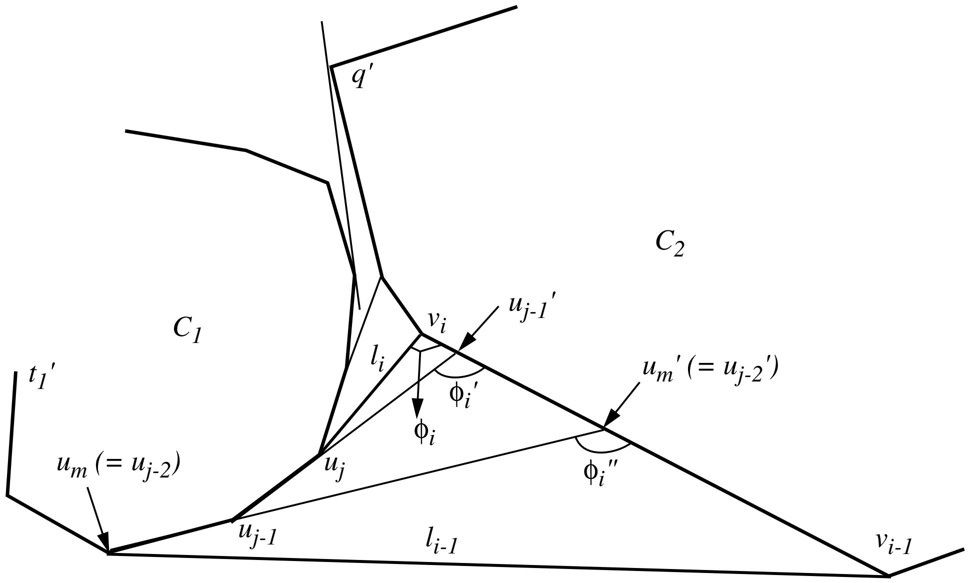

Figure 11: Lemma 9

Notice that for each $v_i \in C_2$, $\theta_i + \phi_i \geq \frac{\pi}{2}$ since C_2 is a convex chain (the equality holds when v_i is not a vertex), thus either $\phi_i \geq \frac{\pi}{2}$ and $\pi_G(v_i, t'_1) < \pi_G(v_{i-1}, t'_1) < \pi_G(v_{i-2}, t'_1) < ...$, or $\theta_i \geq \frac{\pi}{2}$ and $\pi_G(v_i, t'_1) < \pi_G(v_{i+1}, t'_1) < \pi_G(v_{i+2}, t'_1) < ...$, by Lemmas 8 and 9. If both $\phi_i \geq \frac{\pi}{2}$ and $\theta_i \geq \frac{\pi}{2}$, then $v_i = q$, i.e., $\pi_G(v_i, t'_1) = \pi_G(C_2, t'_1)$. We summarize this result in the following lemma.

Lemma 10 *Let w be a point on C_2. Moving w along C_2, the length of $\pi_G(w, t'_1)$ is a bimodal function, i.e., it decreases and then increases. In particular, the minimum value occurs at $w = v_i$ with $\phi_i \geq \frac{\pi}{2}$ and $\theta_i \geq \frac{\pi}{2}$. If this v_i is not a vertex, then $\phi_i = \theta_i = \frac{\pi}{2}$, namely, the line issuing from v_i and tangent to C_1 is perpendicular to the edge of C_2 containing v_i.*

Up to now we can compute $t_2 \in C_1$ and $q \in C_2$ that realize $\pi_G(t'_1, C_2)$ by a binary search on the vertices of C_2, where at each step we compute a tangent of C_1 from the current vertex of C_2, check for angles θ and ϕ and then reduce the search space. Finally, we also have to take care of the case where q is not a vertex. Since tangent computation takes logarithmic time, this method has time complexity $O(\log h \log n)$. To speed up the algorithm, we appeal to the properties from C_1.

Lemma 11 *Let $u_1 = u, u_2, ..., u_k = t'_2$ be the vertices of C_1 in counterclockwise order. The extension of each edge (u_{i-1}, u_i) intersects C_2 at some point v_i, $i = 2, ...k$. Let v'_i and v''_i be the two vertices of C_2 adjacent to v_i, with v'_i following v''_i in clockwise order. Let $\theta_i = \angle u_i v_i v'_i$ and $\phi_i = \angle u_i v_i v''_i$ (see Fig. 12). Then $\theta_2 < \theta_3 < ... < \theta_k$, and $\phi_2 > \phi_3 > ... > \phi_k$.*

Proof: Since (q', t'_2) is a tangent to C_1 (recall this from Fig. 8), its slope is larger than the slope of

(u_{k-1}, t_2'), which shows that the extension of (u_{k-1}, t_2') is below q' and thus intersects C_2. Similar argument applies to the extension of (u_1, u_2), so all such extensions intersect C_2. We now prove that $\theta_i < \theta_{i+1}$; the proof of $\phi_i > \phi_{i+1}$ is similar. Let $w_1, ..., w_l$ be the vertices of C_2 between v_i and v_{i+1} in clockwise order. Draw a segment to connect u_i with each of $w_1, ..., w_l$ and define $\theta_i' = \angle u_i w_i w_{i+1}$ ($\theta_l' = \angle u_i w_l v_{i+1}$). Then $\theta_i < \theta_1' < ... < \theta_l' < \theta_{i+1}$ by the argument that an exterior angle of a triangle is larger than each of the two far interior angles. $\square$

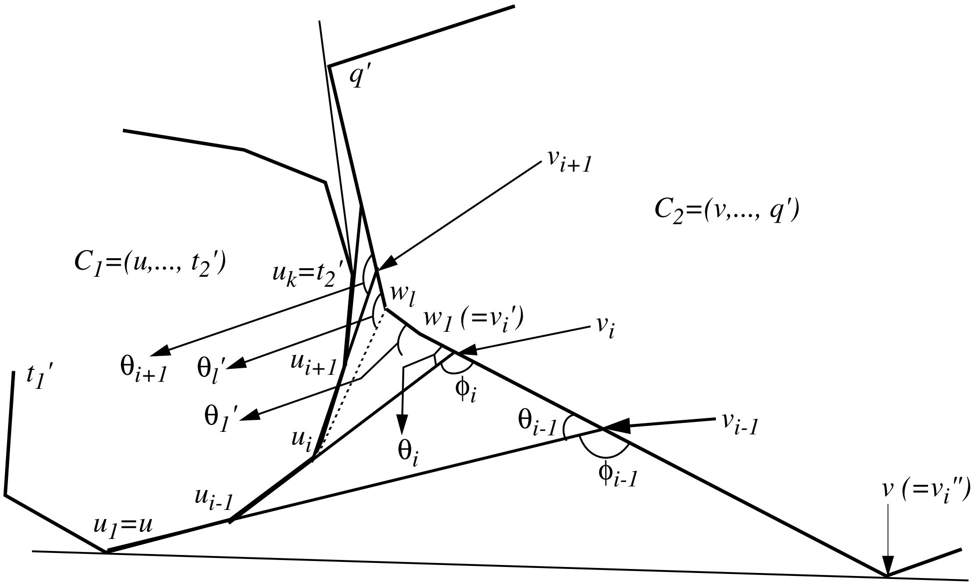

Figure 12: Lemma 11

Lemma 12 *Let $t_2 \in C_1$ and $q \in C_2$ realize $\pi_G(t_1', C_2)$, where t_2 is some vertex u_j. Let each θ_i be defined as in Lemma 11. Then $\theta_j < \frac{\pi}{2}$ and $\theta_{j+1} > \frac{\pi}{2}$.*

Proof: We refer to Fig. 13. Let v' and v'' be the two vertices of C_2 adjacent to point q, with v' following v'' in clockwise order. There are two cases. If q is not a vertex, then by Lemma 10, (u_j, q) is perpendicular to (v', v'') (see Fig. 13(a)). We extend (v', v'') to intersect rays (u_{j-1}, u_j) and (u_j, u_{j+1}) respectively at r'' and r', and make angles θ'' and θ' as shown. We see that $\theta' > \angle u_j q r' = \frac{\pi}{2}$ since it is an exterior angle of $\triangle u_j q r'$, and $\theta_{j+1} \geq \theta'$ (the equality holds when ray (u_j, u_{j+1}) intersects *edge* (v', v'')), so $\theta_{j+1} > \frac{\pi}{2}$. Similarly $\theta'' < \frac{\pi}{2}$ (since in $\triangle u_j q r''$, $\angle u_j q r'' = \frac{\pi}{2}$) and $\theta_j \leq \theta''$ (again, the equality holds when ray (u_{j-1}, u_j) intersects (v', v'')), so $\theta_j < \frac{\pi}{2}$. In the other case where q is a vertex, by Lemma 10 $\angle u_j q v', \angle u_j q v'' \geq \frac{\pi}{2}$ (see Fig. 13(b)). Again we extend (q, v') to intersect ray (u_j, u_{j+1}) and make angle θ', and extend (q, v'') to intersect ray (u_{j-1}, u_j) and make angle θ'' as shown. By the same argument, we have that $\theta_{j+1} \geq \theta' > \angle u_j q v' \geq \frac{\pi}{2}$ and $\theta_j \leq \theta'' < \frac{\pi}{2}$. $\square$

Now we are ready to state the algorithm for computing $t_2 \in C_1$ and $q \in C_2$ that realize $\pi_G(t_1', C_2)$. This is actually a double-binary search.

Algorithm Double-Search

1. If either $| C_1 | = 1$ or $| C_2 | \leq 2$ then go to step 3.
2. Else, pick the median vertices v and w of current C_1 and C_2. Let v' be the vertex of C_1 that precedes v in counterclockwise order, and w' be the vertex of C_2 that follows w in clockwise order. Intersect the ray $r = (v', v)$ with the line extension l' of edge (w, w'). Let θ be the

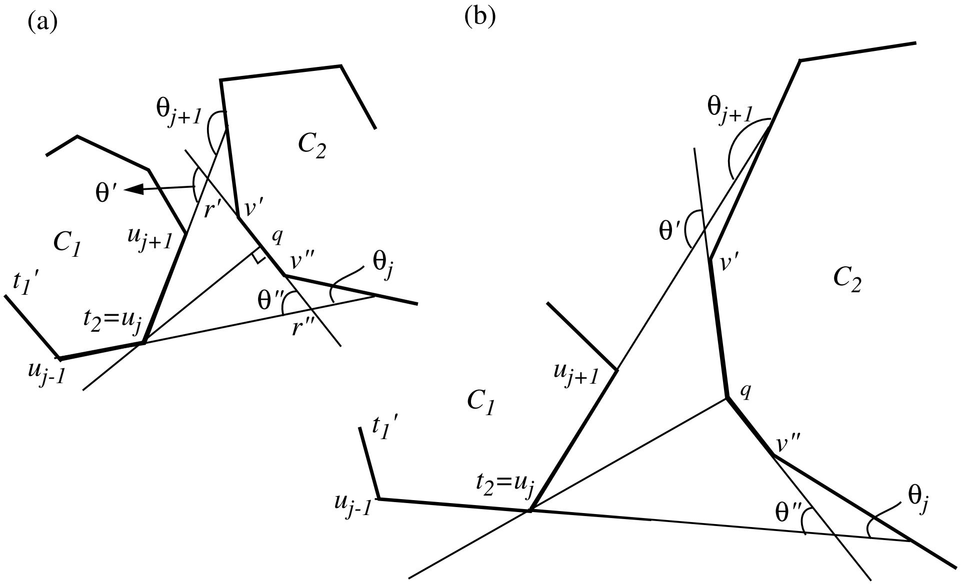

Figure 13: Lemma 12

angle made by r and l' by measuring clockwise from r to l'. The actions (and the verification) depend on the following cases (see Fig. 14):

(a) The intersection is below (w, w') and $\theta \geq \frac{\pi}{2}$ (Fig. 14(a)): prune the wiggly portion (not including w).
Verification: Draw a line l from w parallel to (v', v). Since l is above (v', v), the tangent t from w to C_1 must make an angle $\theta' > \theta \geq \frac{\pi}{2}$. Thus the tangent of C_1 from any point in the wiggly portion will make an angle even bigger, so this portion can be pruned away by Lemma 10.

(b) The intersection is below (w, w') and $\theta < \frac{\pi}{2}$ (Fig. 14(b)): prune the wiggly portion (including v').
Verification: The real intersection between ray (v', v) and C_2 makes an angle $\theta' < \theta < \frac{\pi}{2}$. By Lemmas 11 and 12, any edge in the wiggly portion will make an angle even smaller and thus this portion can be pruned away.

(c) The intersection is above (w, w') and $\theta \geq \frac{\pi}{2}$ (Fig. 14(c)): prune the wiggly portion (including v). This is symmetric to case (b).

(d) The intersection is above (w, w') and $\theta < \frac{\pi}{2}$ (Fig. 14(d)): prune the wiggly portion. This is symmetric to case (a). Note that w itself is not a candidate for q but w is not pruned away here, since q may still lie on (w, w') and thus w must be kept to retain (w, w').

(e) The intersection is on (w, w') and $\theta \geq \frac{\pi}{2}$ (Fig. 14(e)): prune the two wiggly portions (including v but not w' so that (w, w') is kept). This is a situation combining cases (a) and (c).

(f) The intersection is on (w, w') and $\theta < \frac{\pi}{2}$ (Fig. 14(f)): prune the two wiggly portions (including v' but not w so that (w, w') is kept). Again this is a situation combining cases (b) and (d).

After pruning the appropriate portions, go to step 1.

3. Now $\mid C_1 \mid = 1$ or $\mid C_2 \mid \leq 2$, a situation where the double-binary search in step 2 cannot proceed (either $\mid C_1 \mid = 1$ and $\mid C_2 \mid \neq$ *constant* or $\mid C_2 \mid = 1$ and $\mid C_1 \mid \neq$ *constant*) or may not

make any progress (case (d) with $\mid C_2 \mid= 2$ and $\mid C_1 \mid\neq constant$). The operations depend on the following cases:

(a) $\mid C_2 \mid= 1$. The only vertex of C_2 is q. Compute the tangent from q to C_1 and take t_2 as the tangent point. Report q and t_2, and stop.

(b) $\mid C_2 \mid= 2$. Let $C_2 = \{w_1, w_2\}$ such that walking from w_1 to w_2 the interior of R_2 is to the right of (w_1, w_2). From w_1 and w_2 compute tangents (w_1, v_1) and (w_2, v_2) of C_1, where $v_1, v_2 \in C_1$. Let $\theta_1 = \angle v_1 w_1 w_2$ and $\phi_2 = \angle v_2 w_2 w_1$. There are three subcases.

   i. $\theta_1 \geq \frac{\pi}{2}$. By Lemma 10, $q = w_1$ (and $t_2 = v_1$). Report q and t_2, and stop. Note that $\phi_2 < \frac{\pi}{2}$ by Lemma 8.

   ii. $\phi_2 \geq \frac{\pi}{2}$. Report $q = w_2$, $t_2 = v_2$, and stop. This is symmetric to case i.

   iii. $\theta_1 < \frac{\pi}{2}$ and $\phi_2 < \frac{\pi}{2}$ (and $q \neq w_1, w_2$). By Lemma 10, (t_2, q) is perpendicular to (w_1, w_2) and is tangent to C_1. Perform a binary search on subchain $(v_1, ..., v_2)$ of C_1 to find such vertex t_2: At each iteration with current vertex v, compute its projection point v' on (w_1, w_2), check whether vertex v on C_1 is concave, reflex or supporting with respect to (v, v') and branch appropriately. When v is supporting, report $t_2 = v$, $q = v'$ and stop.

(c) $\mid C_1 \mid= 1$. The only vertex of C_1 is t_2. Now perform a binary search on C_2. Let $w_1, ..., w_k$ be the vertices of C_2 in clockwise order. At each step with current vertex w_i, let $\theta_i = \angle t_2 w_i w_{i+1}$ and $\phi_i = \angle t_2 w_i w_{i-1}$. Recall that $\theta_1 < \theta_2 < ... < \theta_k$ by Lemma 8, and if $\theta_i \geq \frac{\pi}{2}$ and $\phi_i \geq \frac{\pi}{2}$ then $w_i = q$ by Lemma 10. The binary search proceeds to find the smallest index i such that $\theta_i \geq \frac{\pi}{2}$. If also $\phi_i \geq \frac{\pi}{2}$, then $q = w_i$; report q and t_2, and stop. Else, both ϕ_i and θ_{i-1} are less than $\frac{\pi}{2}$, and thus t_2 has a projection q on (w_i, w_{i-1}). Report t_2 and q, and stop.

Note that the loop formed by steps 1 and 2 eventually makes either $\mid C_1 \mid= 1$ or $\mid C_2 \mid\leq 2$, and thus we finally exit the loop and go to step 3. Indeed, when C_1 is reduced (cases (b), (c), (e) and (f) of step 2), either v or v' is also pruned away, so that C_1 with $\mid C_1 \mid= 2$ is further reduced to $\mid C_1 \mid= 1$; when only C_2 is reduced (cases (a) and (d) of step 2), one of the two portions preceding and following w is pruned away, so that C_2 with $\mid C_2 \mid= 3$ is further reduced to $\mid C_2 \mid= 2$.

Lemma 13 *The time complexity of Algorithm* Double-Search *is* $O(\log h + \log n)$.

Proof: In each case of step 2, we always discard half of C_1 and/or half of C_2, so the loop formed by steps 1 and 2 takes $O(\log h + \log n)$ time. Step 3 also takes logarithmic time, since either $\mid C_1 \mid$ or $\mid C_2 \mid$ is a constant and a constant number of simple binary searches are performed on the other chain. □

We now give an algorithm for computing the shortest path $\pi_G(R_1, R_2)$ between R_1 and R_2 when they are mutually visible.

Algorithm Visible-Path

1. Ignore P and compute the separation $\sigma(R_1, R_2)$ of R_1 and R_2 by any one of the methods [7, 11, 12], which gives two points $p' \in R_1$ and $q' \in R_2$ such that $length(p', q') = \sigma(R_1, R_2)$.
2. Compute $\pi_G(p', q')$. If $\pi_G(p', q')$ has only one link, then (p', q') is not blocked by P; report $\pi_G(R_1, R_2) = (p', q')$ and stop.
3. Otherwise, $\pi_G(p', q')$ must touch the boundary of P. Let (p', t_1') and (t_2', q') be the first and last links of $\pi_G(p', q')$. Discriminate the two cases below:

   (a) There is no inflection edge in $\pi_G(p', q')$: this is the case of Lemma 6 (Fig. 8). Let $C = \pi_G(t_1', t_2')$. Find the external common tangent (u, v) between C and R_2, where

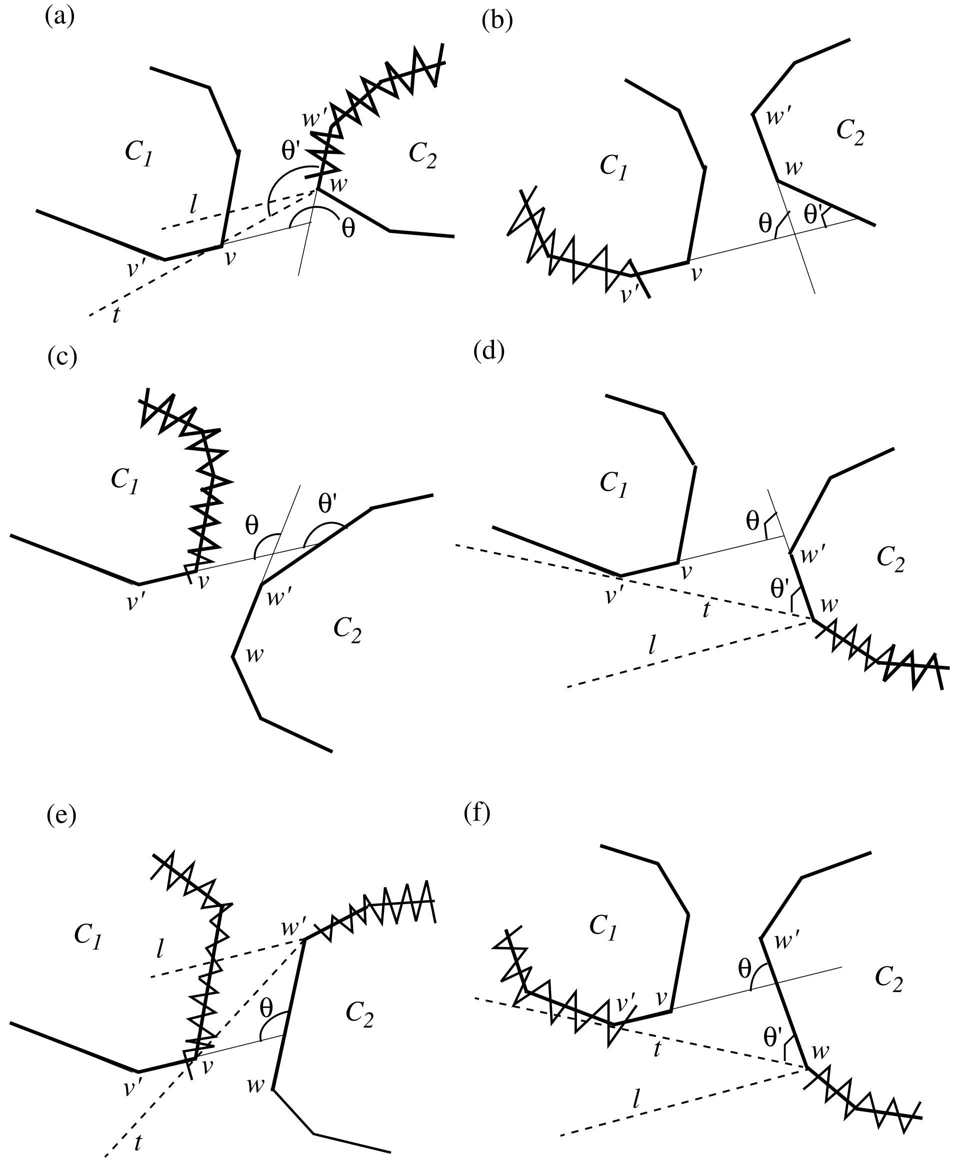

Figure 14: The cases (a)–(f) in step 2 of Algorithm *Double-Search*.

$u \in C$ and $v \in R_2$; let C_1 be $\pi_G(u, t_2')$ and C_2 be the clockwise boundary $(v, ..., q')$ of R_2. Compute $t_2 \in C$ and $q \in R_2$ that realize $\pi_G(t_1', R_2)$ by performing Algorithm *Double-Search* on C_1 and C_2, and similarly compute $t_1 \in C$ and $p \in R_1$ that realize $\pi_G(t_2', R_1)$. Report $\pi_G(R_1, R_2) = (p, t_1) \cup \pi_G(t_1, t_2) \cup (t_2, q)$ and stop.

(b) There is an inflection edge (t_3, t_4) in $\pi_G(p', q')$: this is the case of Lemma 7 (Fig. 9). Use Algorithm *Double-Search* to compute two pairs of points that respectively realize $\pi_G(R_1, t_3)$ and $\pi_G(t_4, R_2)$. Report $\pi_G(R_1, R_2) = \pi_G(R_1, t_3) \cup (t_3, t_4) \cup \pi_G(t_4, R_2)$ and stop.

Lemma 14 *The time complexity of Algorithm* Visible-Path *is* $O(\log h + \log n)$ *(plus* $O(k)$ *if the* k *links are reported).*

Proof: The separation computation in step 1 can be done in logarithmic time. Other computations involve a shortest-path query (step 2), two tangent computations and two calls of Algorithm *Double-Search* (step 3(a) or 3(b)), each taking logarithmic time. □

3.3 The Overall Algorithm

The overall algorithm for computing the shortest path $\pi_G(R_1, R_2)$ between R_1 and R_2 is as follows.

Algorithm Shortest-Path

1. Perform Algorithm *Pseudo-Hourglass* to decide whether the geodesic hourglass H_G is open or closed (with apices p_1 (closer to R_1) and p_2 (closer to R_2)).
2. If H_G is open, then apply Algorithm *Visible-Path* to report $\pi_G(R_1, R_2)$ and stop.
3. Otherwise (H_G is closed), apply Algorithm *Visible-Path* to find shortest paths $\pi_G(R_1, p_1)$ and $\pi_G(p_2, R_2)$ by treating p_1 and p_2 as "convex polygons" consisting of only one vertex. Compute shortest path $\pi_G(p_1, p_2)$, report $\pi_G(R_1, R_2) = \pi_G(R_1, p_1) \cup \pi_G(p_1, p_2) \cup \pi_G(p_2, R_2)$ and stop.

Lemma 15 *Algorithm* Shortest-Path *has time complexity $O(\log h + \log n)$ (plus $O(k)$ if the k links of the path are reported), which is optimal.*

Theorem 1 *Let P be a simple polygon with n vertices. There exists an optimal data structure that supports shortest-path queries between two convex polygons with a total of h vertices inside P in time $O(\log h + \log n)$ (plus $O(k)$ if the k links of the path are reported), using $O(n)$ space and preprocessing time; all bounds are worst-case.*

Remark. Although the case of mutually visible R_1 and R_2 is nontrivial, our algorithms (*Double-Search* and *Visible-Path*) turn out to involve only simple computations, by applying useful geometric properties. The other key technique, Algorithm *Pseudo-Hourglass*, to decide whether H_G between R_1 and R_2 is open (and compute a visibility link) or closed (and compute apices and penetrations), however, is more involved. We pose as an open problem whether there exist simpler techniques to perform the same operations in the same (optimal) time bound. Also, whether we can directly compute H_G in optimal time is an open problem, and may be of independent interest.

4 Dynamic Shortest Path Queries

In this section, we consider the shortest-path problem in a connected planar subdivision $\mathcal{S}$ in a dynamic environment. The query operation is to compute the shortest path $\pi_G(R_1, R_2)$, where the two query convex polygons R_1 and R_2 are given in the same region P of $\mathcal{S}$. In addition, we support edge/vertex insertions and deletions on $\mathcal{S}$ in our data structure. Specifically, we define the following update operations on $\mathcal{S}$:

InsertEdge$(e, v, w, P; P_1, P_2)$: Insert edge $e = (v, w)$ into region P such that P is partitioned into two regions P_1 and P_2.

RemoveEdge$(e, v, w, P_1, P_2; P)$: Remove edge $e = (v, w)$ and merge the regions P_1 and P_2 formerly on the two sides of e into a new region P.

InsertVertex$(v, e; e_1, e_2)$: Split the edge $e = (u, w)$ into two edges $e_1 = (u, v)$ and $e_2 = (v, w)$ by inserting vertex v along e.

RemoveVertex$(v, e_1, e_2; e)$: Let v be a vertex with degree two such that its incident edges $e_1 = (u, v)$ and $e_2 = (v, w)$ are on the same straight line. Remove v and merge e_1 and e_2 into a single edge $e = (u, w)$.

AttachVertex$(v, e; w)$: Insert edge $e = (v, w)$ and degree-one vertex w inside some region P, where v is a vertex of P.

DetachVertex(v, e): Remove a degree-one vertex v and edge e incident on v.

The above repertory of operations is complete for connected subdivisions. That is, any connected subdivision $\mathcal{S}$ can be constructed "from scratch" using only the above operations.

We make use of the dynamic data structure of Goodrich and Tamassia [15]. Their technique supports two-point shortest-path queries and *ray-shooting* queries, which consist of finding the first edge or vertex of $\mathcal{S}$ hit by a query ray. Their data structure is based on *geodesic triangulation* of each region of $\mathcal{S}$. Given three vertices u, v, and w of a region P (a simple polygon), which occur in that order, the *geodesic triangle* they determine is the union of the shortest paths $\pi_G(u, v)$, $\pi_G(v, w)$ and $\pi_G(w, u)$. A *geodesic triangulation* of P is a decomposition of P's interior into geodesic triangles whose boundaries do not cross. The technique [15] dynamically maintains such triangulations by viewing their dual trees as balanced trees. Also, rotations in these trees can be implemented via a simple "diagonal swapping" operation performed on the corresponding geodesic triangles, and edge insertion and deletion can be implemented on these trees using operations akin to the standard *split* and *splice* operations. Moreover, ray shooting queries are performed by first locating the ray's endpoint and then walking along the ray from geodesic triangle to geodesic triangle until hitting the boundary of some region of $\mathcal{S}$. The two-point shortest path is obtained by locating the two points and then walking from geodesic triangle to geodesic triangle either following a boundary or taking a shortcut through a common tangent [15].

Let n be the current number of vertices in $\mathcal{S}$. Using the data structure of [15], we can perform each of the above update operations as well as ray-shooting and two-point shortest-path queries in $O(\log^2 n)$ time, using $O(n)$ space, where in $O(\log^2 n)$ time we get an implicit representation (a balanced binary tree) and the length of the queried shortest path, and using additional $O(k)$ time to retrieve the k links we get the actual path [15]. Again we enhance this data structure so that associated with the implicit representation of a shortest path π_G, there are two balanced binary trees respectively maintaining the inflection edges and the cusps on π_G in their path order. Moreover, we can extract a portion of π_G via split/splice operations in logarithmic time. Using this data structure to support two-point shortest-path queries as needed by Algorithm *Shortest-Path*, we get a dynamic technique for shortest-path queries between two convex polygons in $\mathcal{S}$.

Theorem 2 *Let $\mathcal{S}$ be a connected planar subdivision whose current number of vertices is n. Shortest-path queries between two convex polygons with a total of h vertices that lie in the same region of $\mathcal{S}$ can be performed in time $O(\log h + \log^2 n)$ (plus $O(k)$ to report the k links of the path), using a fully dynamic data structure that uses $O(n)$ space and supports updates of $\mathcal{S}$ in $O(\log^2 n)$ time; all bounds are worst-case.*

Remark. Our update operations are, in the usual dynamic setting, allowed only on $\mathcal{S}$. If R_1 and/or R_2 are also updated, say, by inserting an edge (u, v) between vertices u and v of R_1 and removing the clockwise boundary of R_1 from u to v (or by an inverse operation while preserving the convexity of R_1), we can, of course, first update R_1 and/or R_2 and then re-compute $\pi_G(R_1, R_2)$ by our query algorithm, in $O(\log h + \log^2 n)$ time. An interesting open problem is whether we can support such updates on R_1 and R_2 while maintaining $\pi_G(R_1, R_2)$ in time $O(\text{polylog}(h))$.

5 Static Minimum-Link Path Queries

Given two convex polygons R_1 and R_2 with a total of h vertices inside an n-vertex simple polygon P, we want to compute their minimum-link path $\pi_L(R_1, R_2)$. The data structure given by Arkin, Mitchell and Suri [2] supports minimum-link-path queries between two points and between two segments inside P in optimal $O(\log n)$ time, and between two convex polygons R_1 and R_2 in time $O(\log h \log n)$ (plus $O(k)$ if the k links are reported), using $O(n^3)$ space and preprocessing time. We show in this section how to improve the two-polygon queries to optimal $O(\log h + \log n)$ time, using the same data structure.

Let H_G be the geodesic hourglass of R_1 and R_2, with geodesic tangent points $a_1, b_1 \in R_1$ and $a_2, b_2 \in R_2$. As shown in [2], a minimum-link path between the two segments $s_1 = (a_1, b_1)$ and $s_2 = (a_2, b_2)$ gives a desired minimum-link path between R_1 and R_2, i.e., $\pi_L(s_1, s_2) = \pi_L(R_1, R_2)$. Note that $H(s_1, s_2) = H_G$. Recall from Section 3.1 that when H_G is open (R_1 and R_2 are mutually visible) Algorithm *Pseudo-Hourglass* returns a visibility link l, which can serve as the desired link-one path $l = \pi_L(R_1, R_2)$. So we look at the case where H_G is closed. As we shall see in Lemma 20 (Section 6), if hourglass $H(s_1, s_2)$ is closed with penetrations ρ_1 (closer to s_1) and ρ_2 (closer to s_2), then there exists a minimum-link path $\pi_L(s_1, s_2)$ that uses ρ_1 and ρ_2 as the first and last links. This means that $\pi_L(p, q) = \pi_L(s_1, s_2) = \pi_L(R_1, R_2)$, where points p and q are obtained by extending ρ_1 and ρ_2 to intersect R_1 and R_2, respectively. Therefore the two-polygon queries can be reduced to the two-point queries. We summarize this result in the following lemma.

Lemma 16 *Let the geodesic hourglass H_G be closed with penetrations ρ_1 (closer to R_1) and ρ_2 (closer to R_2), and the line extensions of ρ_1 and ρ_2 intersect R_1 and R_2 at points p and q, respectively. Then $\pi_L(p, q)$ is a minimum-link path $\pi_L(R_1, R_2)$ between R_1 and R_2.*

We now give the algorithm for computing a minimum-link path $\pi_L(R_1, R_2)$ between R_1 and R_2.

Algorithm Min-Link-Path

1. Perform Algorithm *Pseudo-Hourglass* to decide whether the geodesic hourglass H_G is open (with a visibility link l) or closed (with penetrations ρ_1 (closer to R_1) and ρ_2 (closer to R_2)).
2. If H_G is open, then report $\pi_L(R_1, R_2) = l$, $d_L(R_1, R_2) = 1$ and stop.
3. Otherwise (H_G is closed), extend ρ_1 and ρ_2 to intersect R_1 and R_2 respectively at p and q via binary searches on R_1 and R_2. Compute $\pi_L(p, q)$ (and thus also $d_L(p, q)$) by the algorithm of [2]. Report $\pi_L(R_1, R_2) = \pi_L(p, q)$, $d_L(R_1, R_2) = d_L(p, q)$ and stop.

Lemma 17 *The time complexity of Algorithm* Min-Link-Path *is $O(\log h + \log n)$ (plus $O(k)$ if the k links are reported), which is optimal.*

Theorem 3 *Let P be a simple polygon with n vertices. There exists a data structure that supports minimum-link-path queries between two convex polygons with a total of h vertices inside P in optimal time $O(\log h + \log n)$ (plus $O(k)$ if the k links of the path are reported), using $O(n^3)$ space and preprocessing time; all bounds are worst-case.*

6 Dynamic Minimum-Link Path Queries

In this section we show that the dynamic data structure given in Section 4 can also support minimum-link-path queries between two convex polygons in the same region of a connected planar subdivision $\mathcal{S}$. As we have already seen from the last section, we only need to support two-point queries and justify the correctness of Lemma 16, which in turn establishes the correctness of Algorithm *Min-Link-Path*.

6.1 Basic Properties

Let p and q be two points that lie in the same region P of $\mathcal{S}$, and (p, p') and (q', q) be the first and last links of the shortest path $\pi_G(p, q)$, respectively (see Fig. 15). If $\pi_G(p', q')$ is not a monotone chain, there are some cusps $c_1, \cdots, c_i$ such that $\pi_G(p', c_1)$, $\pi_G(c_1, c_2), \cdots, \pi_G(c_i, q')$ are the maximal monotone subchains of $\pi_G(p', q')$. For c_1, we draw a left or right lid l such that l and $\pi_G(p', c_1)$ lie on opposite (left and right) sides of c_1. Let $w_1 = (p, u)$ be the extension of (p, p'), where u is obtained by ray shooting (see Fig. 15). We consider the subregion P' of P delimited by w_1 and l. For each cusp v of P', we draw both lids of v if they do not intersect with $\pi_G(p', c_1)$, otherwise we draw left or right lid of v that does not intersect with $\pi_G(p', c_1)$. Then P' is partitioned into a collection of monotone polygons, among which we denote by $sleeve(w_1)$ the monotone sleeve that uses w_1 as its boundary and contains $\pi_G(p', c_1)$ (see Fig. 15). Excluding segment w_1, the boundary of $sleeve(w_1)$ consists of left and right monotone chains C_1 and C_2. We say that a line t is an *internal common tangent* of $sleeve(w_1)$ if t is locally tangent to two vertices a and b respectively on C_1 and C_2 (if t goes through u, then u is also considered as a tangent point, and similarly for p'). If t intersects with w_1 and a is closer to w_1 than b, we call t a *left tangent* of $sleeve(w_1)$; a *right tangent* is defined similarly.

Suppose that t' and t'' are two left (or right) tangents of $sleeve(w_1)$. Let $\pi'_G(p, q)$ be the set of points on $\pi_G(p, q)$ each of which is visible from some point of t', and v' be the point of $\pi'_G(p, q)$ that is closest to q; v'' is defined similarly with respect to t''. We say that t' *extends farther* than t'' if v' is closer to q than v'' on $\pi_G(p, q)$. Among the left tangents of $sleeve(w_1)$, the one that extends the farthest is called the *maximal left tangent* of $sleeve(w_1)$; similarly for the definition of *maximal right tangent*. By the definitions of $\pi_L(p, q)$ and of window partition, we have the following preliminary algorithm for computing $\pi_L(p, q)$, when the shortest path $\pi_G(p, q)$ is given (see Fig. 15).

Algorithm Prelim

1. If $\pi_G(p, q)$ has only one link then report $\pi_L(p, q) = (p, q)$ and stop; else if the extensions of the first and last links of $\pi_G(p, q)$ meet at some point v, then report $\pi_L(p, q) = (p, v, q)$ and stop.
2. Otherwise, perform the following steps.
   (a) Perform ray shooting to extend the first link of $\pi_G(p, q)$; this gives the first window w_1.
   (b) From w_1, compute the monotone sleeve $sleeve(w_1)$ as described above, and compute the maximal left tangent t_1 of $sleeve(w_1)$ and the maximal right tangent t_2. Choose t from t_1 and t_2 as the one that extends farther. The second window w_2 is (p_1, v_2), where $p_1 = w_1 \cap t$, and v_2 is obtained by performing ray shooting from p_1 along t toward q.
   (c) Repeat step 2b to compute subsequent windows, until the current window intersects with the extension of the last link of $\pi_G(p, q)$, which is the last window w_k.
   (d) Let $p_i = w_i \cap w_{i+1}$. Report $\pi_L(p, q) = (p, p_1, \cdots, p_{k-1}, q)$ and stop.

Let $e_1, \cdots, e_j$ be the inflection edges of $\pi_G(p, q)$. Then $e_1, \cdots, e_j$ partition $\pi_G(p, q)$ into subchains that are always left-turning or always right-turning, namely, into *inward convex* subchains (see Fig. 16). It is shown that every inflection edge $e \in \pi_G(p, q)$ must be contained in $\pi_L(p, q)$ [2, 3, 13]. Hence, extending each inflection edge of $\pi_G(p, q)$ by ray shooting on both sides, together with the extensions of the first and last links of $\pi_G(p, q)$ (where the first link extends towards q and the last toward p), we have fixed windows $W_1, \cdots, W_{j+2}$ (see Fig. 16). Now the task is how to connect consecutive fixed windows. In particular, each W_i has a portion $(u, v) \in \pi_G(p, q)$, with u closer to p than v in $\pi_G(p, q)$. Let the endpoints of W_i be u' and v' such that $W_i = (u', u, v, v')$ (note that $u' = u = p$ if $i = 1$ and $v' = v = q$ if $i = j + 2$). We call (u', u) the *front* of W_i and (v, v') the *rear*

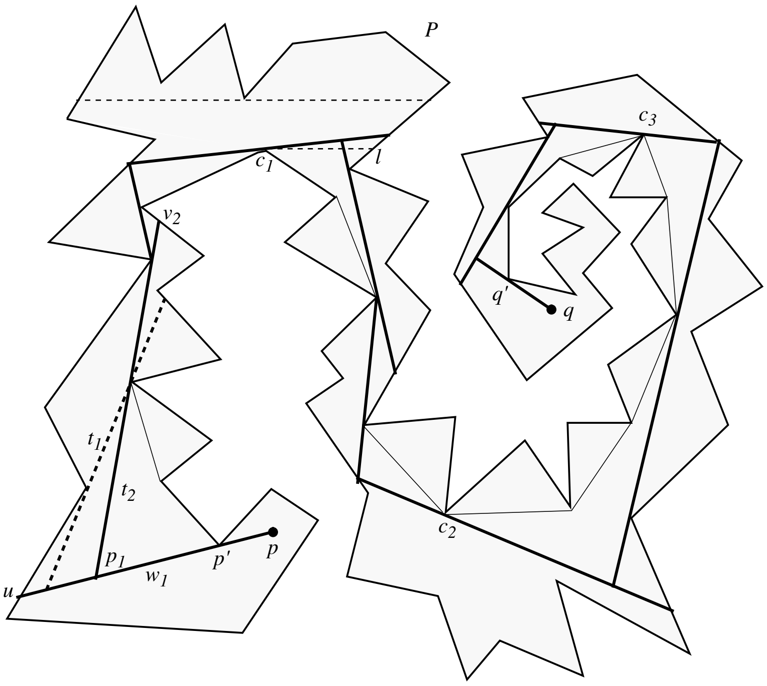

Figure 15: Computing $\pi_L(p,q)$ by Algorithm *Prelim*: the window w_2 following w_1 is chosen to be t_2 since it extends farther than t_1, and so on.

of W_i. We want to connect the rear of W_i with the front of W_{i+1} for each $i = 1, \cdots, j+1$.

Lemma 18 *Let W_i and W_{i+1} be consecutive fixed windows, W the front of W_{i+1}, and w the rear of W_i or a window between the rear of W_i and the front of W_{i+1} as computed by Algorithm* Prelim. *If the hourglass $H(w, W)$ is closed, then the window w' following w is the penetration of funnel $F(w)$.*

Proof: For any local portion of P, the boundary of P consists of two bounding chains C_1 and C_2. Let $w = (a_1, b_1)$ and $W = (a_2, b_2)$, where a_1 and a_2 are on $\pi_G(p,q)$ (see Fig. 17). Then $\pi_G(a_1, a_2)$ is a convex hull inside P of a bounding chain, say C_1, of P. By Algorithm *Prelim*, there are two possible candidates for window w': the penetration of $F(w)$ and some internal common tangent t intersecting with w. Let p_1 be the apex of funnel $F(w)$. Note that $p_1 \in C_1$ and thus the other tangent point of the penetration lies on C_2. Then t must be tangent to two vertices v_1 and v_2 with $v_1 \in \pi_G(a_1, p_1)$ and $v_2 \in C_2$, where v_1 is closer to w than v_2 when walking along t. While extending towards q, the penetration has a slope closer to $\pi_G(a_1, a_2)$ than t, i.e., anything blocking the penetration certainly blocks t (see Fig. 17). Thus the penetration extends farther than t towards q and is chosen as the next window w'. $\square$

6.2 Two Point Queries

The algorithm for computing $\pi_L(p,q)$ between two query points p and q is as follows.

Algorithm Point-Query

1. Compute the shortest path $\pi_G(p,q)$. If $\pi_G(p,q)$ has only one link, then report $\pi_L(p,q) = (p,q)$, $d_L(p,q) = 1$ and stop.

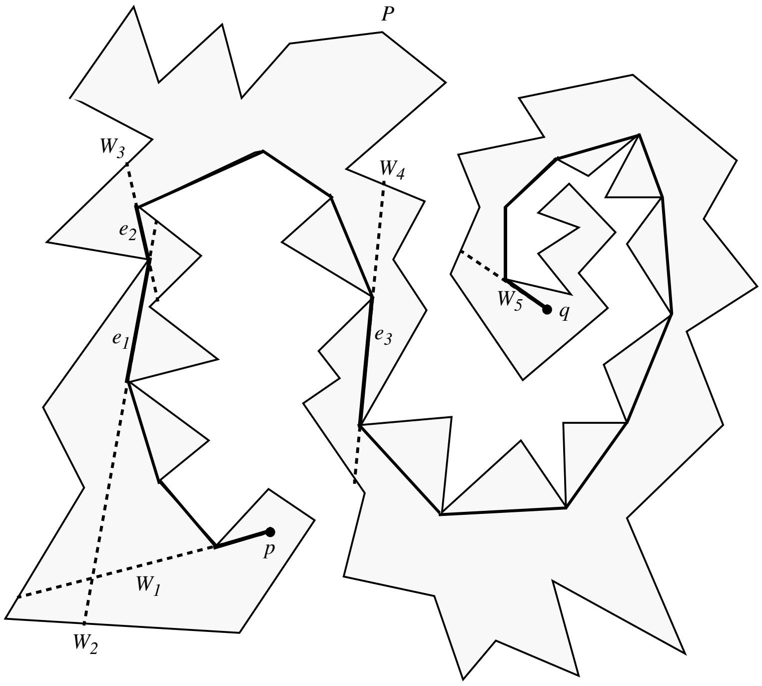

Figure 16: The shortest path $\pi_G(p, q)$ is partitioned by inflection edges e_1, e_2 and e_3. The fixed windows $W_1, \cdots, W_5$ are obtained by extending the inflection edges as well as the first and last links of $\pi_G(p, q)$.

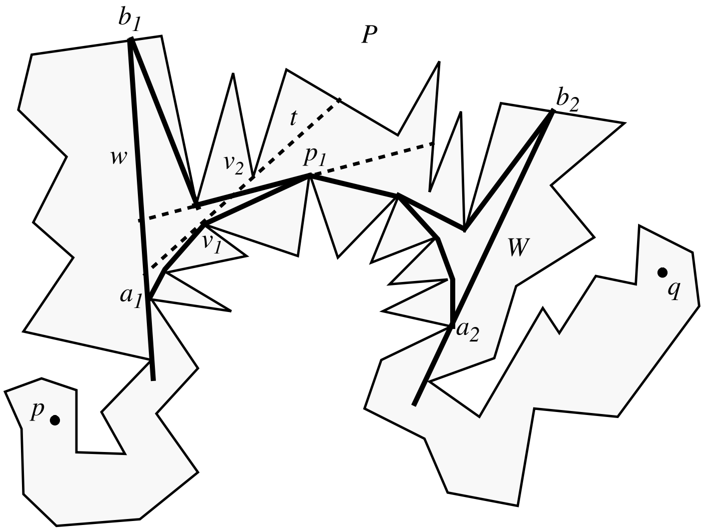

Figure 17: Proof of Lemma 18.

2. Else, perform ray-shooting queries to extend the first link of $\pi_G(p, q)$ in the direction toward q, and the last link of $\pi_G(p, q)$ in the direction toward p. If they intersect with each other at some point v, then report $\pi_L(p, q) = (p, v, q)$, $d_L(p, q) = 2$ (if p, v and q are collinear then $d_L(p, q) = 1$) and stop. Otherwise, also extend each inflection edge of $\pi_G(p, q)$ in both directions; together with the extensions of the first and last links of $\pi_G(p, q)$, this gives the fixed windows $W_1, \cdots, W_j$.

3. For each pair of consecutive fixed windows W_i and W_{i+1} that do not intersect with each other, repeat step 4 to compute the intermediate windows connecting the rear of W_i and the front of W_{i+1}.
4. Initially, let w be the rear of W_i. Let $W = (a_2, b_2)$ be the front of W_{i+1} with $a_2 \in \pi_G(p, q)$.
   (a) Assume that $w = (a_1, b_1)$ with a_1 on $\pi_G(p, q)$. Compute the shortest path $\pi_G(b_1, b_2)$.
   (b) If there is no inflection edge in $\pi_G(b_1, b_2)$, then $H(w, W)$ is an open hourglass. Compute an internal common tangent t of the two inward convex chains $\pi_G(a_1, a_2)$ and $\pi_G(b_1, b_2)$. Note that t connects w and W. Set t to be the window following w and exit step 4.
   (c) Else (there are inflection edges in $\pi_G(b_1, b_2)$) let ρ be the first inflection edge of $\pi_G(b_1, b_2)$, then $H(w, W)$ is a closed hourglass: one endpoint p_1 of ρ is an apex and ρ is the penetration of funnel $F(w)$. Extend ρ in the direction toward b_2 by ray shooting, which hits the boundary of P at some point u; also intersect line ρ with w at some point v. Set (v, u) to be the window following w. Note that p_1 is in (v, u) and is a vertex of P on $\pi_G(p, q)$. Set $w := (p_1, u)$ and go to step 4(a).
5. Now there are windows $w_1, \cdots, w_k$ connecting p and q. Let $v_i = w_i \cap w_{i+1}$, $i = 1, \cdots, k-1$. Report $\pi_L(p, q) = (p, v_1, \cdots, v_{k-1}, q)$, $d_L(p, q) = k$ and stop.

It is easily seen that we perform $O(1)$ ray-shooting and shortest-path queries to compute each link of $\pi_L(p, q)$. Therefore, we have:

Lemma 19 *The time complexity of Algorithm* Point-Query *is* $O(k \log^2 n)$*, where* k *is the number of links in the reported path.*

Now we are ready to give the following lemma, which justifies the correctness of Lemma 16 and thus also Algorithm *Min-Link-Path* given in Section 5.

Lemma 20 *Suppose that two segments* s_1 *and* s_2 *inside a polygonal region* P *are not mutually visible, i.e., the hourglass* $H(s_1, s_2)$ *(containing funnels* $F(s_1)$ *and* $F(s_2)$*) is closed. Let* ρ_1 *be the penetration of* $F(s_1)$ *and* ρ_2 *the penetration of* $F(s_2)$*. Then there exists a minimum-link path* $\pi_L(s_1, s_2)$ *between* s_1 *and* s_2 *that uses* ρ_1 *and* ρ_2 *as the first and last links.*

Proof: To compute $\pi_L(s_1, s_2)$, we can view s_1 and s_2 as "fictitious windows" and apply the method for two-point queries. Let p_1 and p_2 be the apices of $F(s_1)$ and $F(s_2)$, respectively. If $p_1 = p_2$ then the lemma holds trivially. Otherwise, let t be the first internal common tangent in $\pi_G(p_1, p_2)$, and W be the extension of t. If there is no such t, then let $W = s_2$. Since the shortest path from any point of s_1 to any point of s_2 must go through p_1 and p_2, s_1 and W serve as *consecutive fixed windows* in $\pi_L(s_1, s_2)$. If $H(s_1, W)$ is an open hourglass, then the penetration ρ_1 is an internal common tangent connecting fixed windows s_1 and W, and thus is chosen as the window following s_1. If $H(s_1, W)$ is closed, then as computed by Lemma 18, ρ_1 is the window following s_1. In either case, ρ_1 is chosen as the first link of $\pi_L(s_1, s_2)$. Similarly ρ_2 "extends the farthest" from s_2 towards s_1. Suppose that $\pi_L(s_1, s_2)$ so computed does not use ρ_2 as the last link, and w and w' are the last two windows of $\pi_L(s_1, s_2)$. Since ρ_2 extends no worse than the last link w', ρ_2 can also catch w, i.e., replacing w' with ρ_2 still gives a minimum-link path between s_1 and s_2. □

Using Algorithm *Point-Query* to support two-point queries as needed by Algorithm *Min-Link-Path*, we are now able to perform two-polygon queries.

Theorem 4 *Let* $\mathcal{S}$ *be a connected planar subdivision whose current number of vertices is* n*. Minimum-link-path queries between two convex polygons with a total of* h *vertices that lie in the same region of* $\mathcal{S}$ *can be performed in time* $O(\log h + k \log^2 n)$ *(where* k *is the number of links in the reported path), using a fully dynamic data structure that uses* $O(n)$ *space and supports updates of* $\mathcal{S}$ *in* $O(\log^2 n)$ *time; all bounds are worst-case.*

References

[1] Nancy M. Amato. An optimal algorithm for finding the separation of simple polygons. In *Proc. 3rd Workshop Algorithms Data Struct.*, volume 709 of *Lecture Notes in Computer Science*, pages 48–59. Springer-Verlag, 1993.

[2] E. M. Arkin, J. S. B. Mitchell, and S. Suri. Optimal link path queries in a simple polygon. In *Proc. 3rd ACM-SIAM Sympos. Discrete Algorithms*, pages 269–279, 1992.

[3] V. Chandru, S. K. Ghosh, A. Maheshwari, V. T. Rajan, and S. Saluja. NC-algorithms for minimum link path and related problems. Technical Report CS-90/3, TATA inst., Bombay, India, 1990.

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

[5] B. Chazelle and D. P. Dobkin. Intersection of convex objects in two and three dimensions. *J. ACM*, 34:1–27, 1987.

[6] Y.-J. Chiang, F. P. Preparata, and R. Tamassia. A unified approach to dynamic point location, ray shooting, and shortest paths in planar maps. In *Proc. 4th ACM-SIAM Sympos. Discrete Algorithms*, pages 44–53, 1993.

[7] F. Chin and C. A. Wang. Optimal algorithms for the intersection and the minimum distance problems between planar polygons. *IEEE Trans. Comput.*, C-32(12):1203–1207, 1983.

[8] M. de Berg. On rectilinear link distance. *Comput. Geom. Theory Appl.*, 1(1):13–34, July 1991.

[9] M. de Berg, M. van Kreveld, B. J. Nilsson, and M. H. Overmars. Finding shortest paths in the presence of orthogonal obstacles using a combined L_1 and link metric. In *Proc. 2nd Scand. Workshop Algorithm Theory*, volume 447 of *Lecture Notes in Computer Science*, pages 213–224. Springer-Verlag, 1990.

[10] H. N. Djidjev, A. Lingas, and J.-R. Sack. An $O(n \log n)$ algorithm for computing the link center of a simple polygon. *Discrete Comput. Geom.*, 8:131–152, 1992.

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

[12] H. Edelsbrunner. Computing the extreme distances between two convex polygons. *J. Algorithms*, 6:213–224, 1985.

[13] S. K. Ghosh. Computing visibility polygon from a convex set and related problems. *J. Algorithms*, 12:75–95, 1991.

[14] S. K. Ghosh and A. Maheshwari. Parallel algorithms for all minimum link paths and link center problems. In *Proc. 3rd Scand. Workshop Algorithm Theory*, volume 621 of *Lecture Notes in Computer Science*, pages 106–117. Springer-Verlag, 1992.

[15] M. T. Goodrich and R. Tamassia. Dynamic ray shooting and shortest paths via balanced geodesic triangulations. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 318–327, 1993.

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

[17] Y. Ke. An efficient algorithm for link-distance problems. In *Proc. 5th Annu. ACM Sympos. Comput. Geom.*, pages 69–78, 1989.

[18] D. G. Kirkpatrick. Efficient computation of continuous skeletons. In *Proc. 20th Annu. IEEE*

Sympos. Found. Comput. Sci., pages 18–27, 1979.

[19] W. Lenhart, R. Pollack, J.-R. Sack, R. Seidel, M. Sharir, S. Suri, G. T. Toussaint, S. Whitesides, and C. K. Yap. Computing the link center of a simple polygon. *Discrete Comput. Geom.*, 3:281–293, 1988.

[20] A. Lingas, A. Maheshwari, and J.-R. Sack. Parallel algorithms for rectilinear link distance problems. In *Proc. 7th IEEE Internat. Parallel Process. Sympos.* IEEE Computer Society, 1993.

[21] J. S. B. Mitchell, C. Piatko, and E. M. Arkin. Computing a shortest k-link path in a polygon. In *Proc. 33rd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 573–582, 1992.

[22] J. S. B. Mitchell, G. Rote, and G. Woeginger. Minimum-link paths among obstacles in the plane. *Algorithmica*, 8:431–459, 1992.

[23] B. J. Nilsson and S. Schuierer. An optimal algorithm for the rectilinear link center of a rectilinear polygon. In *Proc. 2nd Workshop Algorithms Data Struct.*, volume 519 of *Lecture Notes in Computer Science*, pages 249–260. Springer-Verlag, 1991.

[24] Bengt J. Nilsson and Sven Schuierer. Computing the rectilinear link diameter of a polygon. In *Computational Geometry — Methods, Algorithms and Applications: Proc. Internat. Workshop Comput. Geom. CG '91*, volume 553 of *Lecture Notes in Computer Science*, pages 203–215. Springer-Verlag, 1991.

[25] M. H. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Syst. Sci.*, 23:166–204, 1981.

[26] F. P. Preparata and M. I. Shamos. *Computational Geometry: an Introduction.* Springer-Verlag, New York, NY, 1985.

[27] J. H. Reif and J. A. Storer. Minimizing turns for discrete movement in the interior of a polygon. *IEEE J. Robot. Autom.*, pages 182–193, 1987.

[28] J. A. Storer. On minimal node-cost planar embeddings. *Networks*, 14:181–212, 1984.

[29] S. Suri. A linear time algorithm for minimum link paths inside a simple polygon. *Comput. Vision Graph. Image Process.*, 35:99–110, 1986.

[30] S. Suri. *Minimum link paths in polygons and related problems.* Ph.D. thesis, Dept. Comput. Sci., Johns Hopkins Univ., Baltimore, MD, 1987.

[31] S. Suri. On some link distance problems in a simple polygon. *IEEE Trans. Robot. Autom.*, 6:108–113, 1990.

[32] R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comput.*, 16(3):421–444, 1987.