Trajectory Visibility

Patrick Eades
University of Sydney, Australia
patrick.eades@sydney.edu.au

Ivor van der Hoog
Utrecht University, the Netherlands
i.d.vanderhoog@uu.nl

Maarten Löffler
Utrecht University, the Netherlands
m.loffler@uu.nl

Frank Staals
Utrecht University, the Netherlands
f.staals@uu.nl

Abstract

We study the problem of testing whether there exists a time at which two entities moving along different piece-wise linear trajectories among polygonal obstacles are mutually visible. We study several variants, depending on whether or not the obstacles form a simple polygon, trajectories may intersect the polygon edges, and both or only one of the entities are moving.

For constant complexity trajectories contained in a simple polygon with n vertices, we provide an $\mathcal{O}(n)$ time algorithm to test if there is a time at which the entities can see each other. If the polygon contains holes, we present an $\mathcal{O}(n \log n)$ algorithm. We show that this is tight.

We then consider storing the obstacles in a data structure, such that queries consisting of two line segments can be efficiently answered. We show that for all variants it is possible to answer queries in sublinear time using polynomial space and preprocessing time.

As a critical intermediate step, we provide an efficient solution to a problem of independent interest: preprocess a convex polygon such that we can efficiently test intersection with a quadratic curve segment. If the obstacles form a simple polygon, this allows us to answer visibility queries in $\mathcal{O}(n^{\frac{3}{4}} \log^3 n)$ time using $\mathcal{O}(n \log^5 n)$ space. For more general obstacles the query time is $\mathcal{O}(\log^k n)$, for a constant but large value k, using $\mathcal{O}(n^{3k})$ space. We provide more efficient solutions when one of the entities remains stationary.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases trajectories, visibility, data structures, semi-algebraic range searching

Digital Object Identifier 10.4230/LIPIcs.SWAT.2020.23

Funding All authors are partially supported through the University of Sydney – Utrecht University Partnership Collaboration Awards: Algorithms and data structures to support computational movement analysis.
Ivor van der Hoog: Supported by the Netherlands Organisation for Scientific Research (NWO); 614.001.504.
Maarten Löffler: Partially supported by the Netherlands Organisation for Scientific Research (NWO) under project numbers 614.001.504 and 628.011.005.

1 Introduction

We consider the following question: two entities q and r follow two different trajectories, with (possibly different) constant speed. Their trajectories lie in an environment with obstacles that block visibility. Can the two entities, at any time, see each other? This question combines two key concepts from computational geometry, namely *trajectories* and *visibility*.

Trajectories. A recent increase in the availability of low-cost, internet connected GPS tracking devices has driven considerable interest in spatio-temporal data (commonly called *trajectories*) across fields including GIScience, databases, and computational geometry. Problems studied recently in computational geometry include detecting and describing flocks [5, 31], detecting hotspots, clustering and categorising trajectories, map construction and others [9, 25, 32]. Formally, a trajectory is a sequence of time-stamped locations in the plane, or more generally in $\mathbb{R}^d$, which models the movement of an entity. Trajectory data is obtained by tracking the movements of animals [7, 30], hurricanes [41], traffic [33], or other moving entities [20] over time. For a more extensive overview of trajectory analysis we refer the reader to the excellent survey by Gudmundsson et al. [27].

Visibility. Two points, amidst a number of obstacles, are mutually *visible* if the line segment between them does not intersect any obstacle. Visibility is one of the most studied topics in computational geometry [36, 42] and in adjacent fields such as computer graphics [22], GIScience [24], and robotics [36]. Within computational geometry, Gosh and Giswani compiled a survey of *unsolved* problems in this area [26]. Core visibility problems in computational geometry include *ray shooting* [16, 13], guarding [15, 23] and visibility graph recognition [10]. For more information about visibility problems, refer to O'Rourke's book [39] ch. 8, and the surveys by Durant [22] and Gosh [26].

Trajectory visibility. In this paper, we study the following fundamental question, which we refer to as *trajectory visibility* testing. Given a simple polygon, or a polygonal domain, P, and the trajectories of two moving entities q and r, is there a time t at which q and r can see each other? We assume that q and r move linearly with constant (but possibly different) speeds between trajectory vertices, and cannot see though the edges of P. We distinguish several variants depending on whether P is a simple polygon or a polygonal domain, and whether the trajectories are allowed to intersect P (e.g. vehicles moving through fog, animals moving through foliage) or not (e.g. pedestrians moving among buildings, ships moving on water bodies). These variants are illustrated in Figure 1. We further consider the same variants in the simpler scenario in which one of the entities is a point (e.g. a stationary guard and a moving intruder). Note that we are interested only whether there *exists* a time at which the two entities see each other. This implies we can temporally decompose the problem: the answer is *no* if and only if the answer is no between all two consecutive time stamps. When considering this question, two fundamentally different approaches come to mind. On the one hand, when the number τ of trajectory vertices is small compared to the number n of polygon vertices, the best approach may be to simply solve the problem for each time interval separately. On the other hand, when τ is large compared to n, it may be more efficient to spend some time on preprocessing P first, if this allows us to spend less time per trajectory edge. We therefore distinguish between the *algorithmic* question and the *data structure* question. Our results are discussed below and summarized in Table 1.

Related work on visibility for moving entities. There is a vast amount of research on both trajectories and visibility, but surprisingly not much previous work exists on their combination. One reason is that the already developed tools for visibility and trajectory analysis cannot be combined in a straightforward manner. Consider two trajectories q and r within a simple polygon P: existing visibility tools allow us to easily check if there are subtrajectories of q and r which are mutually visible. However, the two moving entities see each other only if there is a time at which the two entities are simultaneously within two

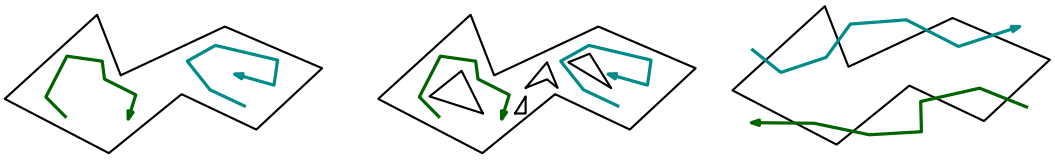

Figure 1 Different variations of the problem: two trajectories inside a simple polygon (left), two trajectories in a polygonal domain (middle), or two trajectories intersecting a simple polygon (right). Our approach for the middle variant is independent of whether the trajectories intersect the domain.

mutually visible subtrajectories. There could be quadratically many pairs of subtrajectories which are mutually visible; yet, it could be that the two entities are never simultaneously within such a pair. To determine visibility between moving entities, one needs to incorporate the concept of time into pre-existing tools for visibility. An early result in this direction is by Bern et al. [6] and Mulmuley [37], who study maintaining the visibility polygon of a point that moves over a straight path. Aronov et al. [3] demonstrate a kinetic data structure that tracks the visibility polygon of a moving query point q. The most recent result on visibility and motion is by Diez et al. [18] who show how to maintain the shortest path between two moving entities using a kinetic data structure. Here q and r are mutually visible if and only if their shortest path is a line segment.

From event based modelling to algebraic range searching. The above attempts for visibility testing model the passage of time using a sequence of *events*. However, as q and r traverse their trajectories there may be a linear number of such events and thus far, there does not exist a data structure that can answer if there is visibility between trajectories in sub-linear time. In this paper, we diverge from the classical event-based approach and model the problem in an algebraic way. Such an algebraic reformulation is not rare in computational geometry: for example, circular range queries get solved by reformulating them into higher-dimensional halfspace range queries. Yet we are unaware of any algebraic approaches used for visibility testing. The challenge with such an algebraic approach is that the more complicated the algebraic expression, the worse the complexity of both the paper and the runtimes involved. However, our transformation in Section 2 that transforms visibility testing into testing for an intersection between a convex polygon and an algebraic curve, uses only degree two planar polynomials. This transformation allows us to introduce tools and concepts from algebraic range searching to obtain the first sublinear query times for visibility testing for a variety of geometric settings.

Our Results. We focus on trajectories of at most two vertices; any set of trajectories of τ vertices can be handled by applying our algorithms or queries τ times. Our results are summarized in Table 1. In Section 2, we discuss our algorithmic results; we build on the structural geometric properties established in this section in the remainder of the paper. In Section 3 we consider the sub-problem of preprocessing a convex polygon P' for intersection queries with quadratic curve segments. We then extend the solution in Section 4 to a data structure for visibility testing in a simple polygon P using multi-level data structures. In Section 5 we discuss the case in which one of the entities is stationary. In Section 6 we briefly outline our results for polygonal domains, more detailed descriptions of this data structures can be found in Appendix C. Due to space constraints, several proofs and the full description of all results are deferred to the appendix.

Table 1 The two leftmost columns specify if the query entity is a point (•) or line segment (/). The third column specifies if the domain P is a simple polygon where the query segments may not intersect P (S), a simple polygon where the query segments may intersect P (I), or a polygonal domain with n vertices where the query segments may intersect P (D). The value k is an unspecified constant.

q	r	P	algorithm	data structure			source
				space	preprocessing	query	
•	•	S or I	$\Theta(n)$	$\mathcal{O}(n)$	$\Theta(n)$	$\Theta(\log n)$	[28]
		D	$\Theta(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\Theta(\log n)$	[40]
•	/	S	$\Theta(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n \log n)$	$\Theta(\log n)$	Section 5
		I	$\mathcal{O}(n \log n)$	$\mathcal{O}(n^2 \log^3 n)$	$\mathcal{O}(n^2 \log^3 n)$	$\mathcal{O}(n^{\frac{3}{4}+\varepsilon})$	Section 5
		D	$\Theta(n \log n)$	$\mathcal{O}(n^4 \log^3 n)$	$\mathcal{O}(n^4 \log^3 n)$	$\mathcal{O}(n^{\frac{3}{4}+\varepsilon})$	Section 5
/	/	S	$\Theta(n)$	$\mathcal{O}(n \log^5 n)$	$\mathcal{O}(n^1 \log^5 n)$	$\mathcal{O}(n^{\frac{3}{4}} \log^3 n)$	Section 4
		I	$\mathcal{O}(n \log n)$	$\mathcal{O}(n^{3k})$	$\mathcal{O}(n^{3k})$	$\mathcal{O}(\log^k n)$	Appendix C
		D	$\Theta(n \log n)$	$\mathcal{O}(n^{3k})$	$\mathcal{O}(n^{3k})$	$\mathcal{O}(\log^k n)$	Appendix C

2 Algorithms for testing visibility

Let P be a polygonal domain with n edges and let q and r each be a line segment or a point in the plane. We first present an $\mathcal{O}(n \log n)$ time algorithm to solve the visibility problem for q, r and P and we show this algorithm is tight in the worst case. Then we show how to solve the visibility problem in linear time in the case where P is a simple polygon and q and r are contained in P. All other sections depend on the notion of *hourglass* that is presented here.

An $\mathcal{O}(n \log n)$ time algorithm. The entities q and r each move along a line segment with constant speed (depending on the length of the line segment) during the time interval $[0, 1]$.[1] Consider the line $g(t)$ through both entities at time t. We dualize this line to a point using classical point-line dualization (i.e. we map the line $y = ax + b$ to the point $(a, -b)$); this point $\gamma(t)$ now traces a segment of a curve $\gamma : [0, 1] \to \mathbb{R}^2$ in the dual space.[2]

▶ **Lemma 1.** *The segment γ is a segment of a quadratic curve with 5 degrees of freedom.*

Let e be an edge of P and denote by L_e the set of lines intersecting e. The dual of L_e is a *wedge* Λ_e [17]. If the segment between q and r is blocked by e at time t then $g(t)$ must lie in L_e. In the dual, this means that the curve segment γ must intersect Λ_e. There are at most two connected time intervals where a quadratic curve segment γ can intersect a wedge Λ_e; it follows that each edge e has at most two connected time intervals where it blocks the visibility between q and r. This leads to a straightforward general algorithm to test if there is a time at which q can see r: for each edge $e \in P$, we compute the at most two time intervals where it blocks visibility between q and r in constant time. We sort these time intervals in $\mathcal{O}(n \log n)$ time and check if their union covers the time interval $[0, 1]$.

▶ **Theorem 2.** *Given a polygonal domain P with n vertices and moving entities q and r, we can test trajectory visibility in $\mathcal{O}(n \log n)$ time.*

[1] With slight abuse of notation, we use q and r to refer to both the (moving) entities and their trajectory.
[2] Throughout this paper we follow the convention of using latin letters for objects in the primal space and greek letters for their duals.

Figure 2 The reduction when P is a polygonal domain with $\Omega(n)$ holes.

An $\Omega(n \log n)$ lower bound. If P is a polygonal domain with $\Omega(n)$ holes, this result is tight: suppose we are given a set A of n numbers and we want to test if $A = B$ for a given arbitrary sorted set $B = \{x_1, x_2, \ldots, x_n\}$. Ben-Or [4] shows that this problem has an $\Omega(n \log n)$ lower bound in the algebraic decision tree model. This leads to the following reduction (illustrated in Figure 2): we construct a set of n horizontal edges whose y-coordinates are 0 and whose x-coordinates are $\{(x_i + \varepsilon, x_{i+1} - \varepsilon) \mid i \in [1, n-1]\}$ where ε is smaller than half of the minimal difference between two consecutive numbers in B; a value for ε can be found in linear time since B is sorted. For each of the n numbers $x \in A$ we construct an axis-aligned rectangle from the point $(x, 1)$ to $(x, 2)$ with a width of 2ε. The entity q walks from the point $(x_1, -1)$ to $(x_n, -1)$ and entity r walks from $(x_1, 3)$ to $(x_n, 3)$. Suppose the number x_j from B is not in A, then q can see r at the x-coordinate x_j. Note that this construction also extends to the case where one of the two entities is stationary: consider the cone between the stationary entity q and a horizontal line segment trajectory r: we can transform the set B into a set of n horizontal edges that cut in the cone between q and r. Each rectangle modelling a number $a \in A$ gets stretched such that it intersects a ray from q to r.

▶ **Theorem 3.** *There exists a polygonal domain P with n vertices, and entities q and r moving inside P for which testing trajectory visibility requires $\Omega(n \log n)$ time.*

A linear-time algorithm for when P is a simple polygon and $q, r \subset P$. Any segment between q and r that is contained in P is a geodesic shortest path in P between a point on q and a point on r. Guibas and Hershberger [28] define, for any two segments q and r in a simple polygon P, the *hourglass* $H(q, r)$ to be the union of all shortest paths between points on q and r. The hourglass $H(q, r)$ is a subset of P and bounded by the segments q and r and by two shortest paths. The *"upper" chain*[3] is the shortest path between the upper end points p_q^+ p_r^+ of q and r, and the *"lower" chain* is the shortest path between p_q^- and p_r^- (refer to Figure 3). We define the *visibility glass* $L(q, r)$ as the (possibly empty) union of *line segments* between q and r that are contained in P. Notice that $L(q, r)$ is a subset of $H(q, r)$.

▶ **Observation 1.** *For any two segments $q, r \subset P$ either $L(q, r)$ is empty or there exist segments $q' \subset q, r' \subset r$ such that $L(q, r) = H(q', r')$. Moreover, q' and r' are bounded by two bitangents on the shortest paths between the endpoints of q and r.*

Proof. Suppose that the interior of the upper and lower chains of $H(q, r)$ intersect. Then the visibility glass $L(q, r)$ is either a single segment or empty. Thus we can either find two points q' and r' on q and r whose line segment forms $H(q', r') = L(q, r)$ or $L(q, r)$ is empty. If the interior of the upper and lower chains are disjoint then they are semi-convex [28].

[3] We use the names "upper" and "lower" since they intuitively correspond to our figures. If q and r are not vertical but their endpoints are in convex position, we rotate the plane until one of them is vertical. If the endpoints of q and r do not lie in convex position, the two chains share an endpoint, which is a simpler case.

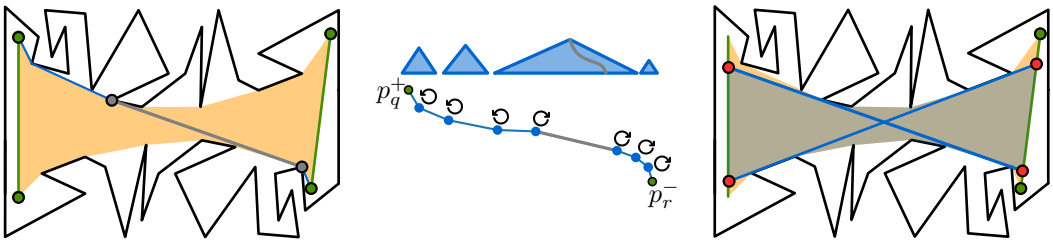

Figure 3 (left) $H(q,r)$ in orange. The path from p_q^+ to r_q^- shown as a dotted path. (middle) The path from p_q^+ to r_q^- can be represented as a collection of binary trees on the vertices. The bitangent (in grey) is incident to the only vertex at which the path switches from making left turns to making right turns (or vice versa) (right) Using the bitangents, we identify the endpoints of q' and r' and obtain $L(q,r)$ in grey.

Consider the shortest path from p_q^+ to p_r^-, it has one edge (u,v) connecting the upper and lower chain. This is the unique edge for which the path makes a clockwise turn at u and a counterclockwise turn at v or vice versa. There exists an edge with similar properties on the shortest path between p_q^- and p_r^+. The extension of these two edges bounds q' and r' [14]. ◀

Chazelle and Guibas [14] note that (the supporting lines of) all line segments in $L(q,r)$ can be dualized into a convex polygon of linear complexity which we denote by $\Lambda(q,r)$. The shortest path between two points in P can be computed in linear time [29]. Finding the bitangents also takes linear time. It follows that we can compute $L(q,r)$ and its dual $\Lambda(q,r)$ in linear time. Suppose that we are given two entities q and r contained in a simple polygon P. Recall that the line $g(t)$ through q and r traces a quadratic segment γ in the dual.

▶ **Observation 2.** *Entities q and r are mutually visible at time t if $\gamma(t)$ lies in $\Lambda(q,r)$.*

Proof. The entities can see one another at time t if and only if $g(t) \in L(q,r)$. ◀

We can derive γ in constant time, construct $\Lambda(q,r)$ in linear time, and we can check if a quadratic curve intersects a convex polygon in linear time. Thus we conclude:

▶ **Theorem 4.** *Given a simple polygon P with n vertices and two entities q and r moving linearly inside P, we can test trajectory visibility in $\Theta(n)$ time.*

3 Intersecting a convex polygon with an algebraic curve

We now turn our attention to the data structure question: can we preprocess P such that trajectory visibility may be tested efficiently (i.e. in sublinear time) for a pair of *query* segments q, r? By Observation 2, we can phrase such a query as an intersection between a quadratic curve segment and a convex polygon. Note, however, that both the curve segment and the convex polygon depend on the query segments q and r. As an intermediate step, we study a simplified problem in which the convex polygon is independent of q and r. In particular, the question that we study is: let P' be a convex polygon with n edges. Is it possible to preprocess P' such that for any quadratic curve segment γ, we can quickly test if γ intersects P'? We believe this problem to be of independent interest. We then use our solution to this question as a subroutine in Section 4.

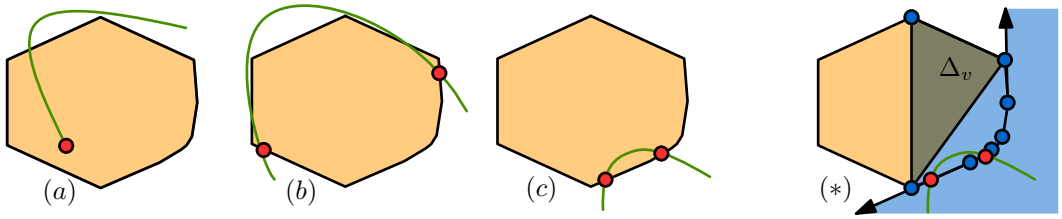

Figure 4 The cases (a), (b) and (c) of intersection between γ and P'. (*) the right-subspace bounded by three halfspaces. The red points are the points of intersection which we try to identify.

Semi-algebraic range searching. Let X be a set of n geometric objects, where each object is parametrized by a vector $\vec{x}$ in $\mathbb{R}^d$ (e.g. a point is parametrized by a vector of its coordinates). Let Γ be a family of geometric regions (called semi-algebraic ranges) in $\mathbb{R}^d$ where each region $G \in \Gamma$ is bounded by an algebraic surface γ which is parametrized by a vector $\vec{a}$. Agarwal et al. [2] describe how to store X, such that for any query range $G \in \Gamma$, we can efficiently count the number of objects of X whose parameter vector lies in G. Let F be a function mapping (the parameterizations of) $x \in X$ and $G \in \Gamma$ to $\mathbb{R}$ such that $F(\vec{x}, \vec{a}) \leq 0$ if and only if $\vec{x} \in G$. Agarwal et al. show that if F can be written in the form $F(\vec{x}, \vec{a}) = g_0(\vec{a}) + \sum_{i=1}^{k} g_i(\vec{a}) f_i(\vec{x})$ then there is a function f that maps the objects in X to points in $\mathbb{R}^k$, and a function g that maps the ranges in Γ to halfspaces in $\mathbb{R}^k$, such that $f(x) \in g(G)$ if and only if $\vec{x} \in G$. This so-called *linearization* process transforms a d-dimensional semi-algebraic range searching problem into a halfspace range searching problem in $\mathbb{R}^k$. Refer to Appendix B for examples.

The resulting set of n points in $\mathbb{R}^k$ can be stored in a data structure of linear size such that the points in a query halfspace can be counted in $\mathcal{O}(n^{1-\frac{1}{k}})$ time (with high probability) [11]. Testing if a query halfspace is empty can be done in expected $\mathcal{O}(n^{1-\frac{1}{k/2}})$ time. Data structures with a slightly slower deterministic query time are also known [11]. If we are willing to use (much) more space, faster query times are also possible [11, 35].

Semi-algebraic range searching and our intersection query. The n edges of a convex polygon P' are geometric objects. A natural parametrization for an edge $e \in P'$ is a 4-dimensional vector $\vec{x}_e$ specifying its start and end points. A quadratic curve segment γ per definition is an semi-algebraic range parametrized by its own parameters $\vec{a}_\gamma$. If we want to apply semi-algebraic range searching, we need to design a predicate function $F(\vec{x}_e, \vec{a}_\gamma)$ that outputs a negative real number whenever the edge e is intersected by γ.

It is tempting to immediately construct such a predicate function using the parameters $\vec{x}_e$ and $\vec{a}_\gamma$ only (ignoring geometric facts such as the convexity or connectedness of P'). However, we have to linearize the resulting predicate function $F(\vec{x}_e, \vec{a}_\gamma)$ into k terms. The more complex the description of the objects, the query, and their intersection, the more complex the predicate will be and therefore the higher this number k will be.

Geometry of our intersection query. Let P' be a convex polygon with n edges. Let γ be a quadratic curve segment ending in the points s and z and let Γ denote the unique degree-2 curve $\Gamma \supset \gamma$ given by $a_1x^2 + a_2x + a_3xy + a_4y + a_5y^2 + a_6 = 0$. We say $\vec{a} = (a_1, \ldots, a_6)$. Observe (Figure 4) that if γ intersects P', then either (a) an endpoint s or z lies in P', or (b) γ cuts off a vertex or (c) γ intersects only a single edge of P' twice and has no endpoint in P' (we call this *dipping*). Intersections of type (a) and (c) can be identified with a regular binary search on P'. An intersection of type (b) is detected using algebraic range searching.

Since P' is convex, we can test if an endpoint of γ lies inside P' in $\mathcal{O}(\log n)$ time. To detect an intersection of case (c), we store a Dobkin-Kirkpatrick hierarchy [19] of P'. This takes $\mathcal{O}(n)$ space and requires $\mathcal{O}(n)$ preprocessing time. Given γ, we detect an intersection of type (c) as follows (Figure 4 (*)): any node v in this decomposition represents a sub-polygon P'' of P' and a triangle Δ_v which splits P'' in a left and right part. Consider the border of the right part $R = (e_1, e_2, \ldots, e_m)$. Note that if γ does not intersect Δ_v, then γ can only dip an edge in R if it is contained in the union of the halfspaces that lie to the right of the lines supporting: (1) one edge of Δ_v and (2) e_1 and e_m (refer to the blue area in the figure). Given the node v we do three constant time checks. First we check if γ intersects Δ_v. If not then we check for both the left and right sub-polygon if γ is contained in the specified area. If that is the case for both or neither sub-polygons then γ can never dip an edge of P', else we recurse in the approprate subtree. It follows that we can detect case (c) in $\mathcal{O}(\log n)$ time.

▶ **Lemma 5.** *We can preprocess a convex polygon P' consisting of n edges in $\mathcal{O}(n)$ time and using $\mathcal{O}(n)$ space, such that for any degree-2 curve segment γ we can detect an intersection of type (a) or (c) in $\mathcal{O}(\log n)$ time.*

The curve Γ of which γ is a segment divides the plane into two areas, $\Gamma^- := \{a_1x^2 + a_2x + a_3xy + a_4y + a_5y^2 + a_6 \leq 0\}$ and its complement Γ^+. An edge $((x_1, x_2), (x_3, x_4))$ of P' is intersected by γ with an intersection of type (b) only if one endpoint of the edge lies in Γ^- and the other in Γ^+. If Γ is a curve with $k + 1 \leq 6$ degrees of freedom then the formulation of Γ^- and Γ^+ is a predicate that specifies whenever a point $\vec{x} = (x_1, x_2)$ lies in Γ^- or Γ^+ with k linearized terms. Thus we can detect if an edge has two endpoints on opposite sides of Γ with two consecutive halfspace range queries in $\mathbb{R}^k$. We build a three-level data structure where the first two levels are 5-dimensional partition trees [34, 11].[4] On top of each node in the second level we build a binary tree on the clockwise ordering (with respect to P') of the edges in that node.

During query time, we transform the degree-2 curve Γ into two k-dimensional halfspaces $g(\Gamma^+)$ and $g(\Gamma^-)$. With two consecutive halfspace range queries we obtain the collection $E_\Gamma(P)$ of edges which have one endpoint on each side of Γ in $\mathcal{O}(n^{1-\frac{1}{k}})$ time. Note that the set $E_\Gamma(P)$ does not have to be a connected set of edges of P (refer to Figure 4 (b)). However, the subset of $E_\Gamma(P)$ that is intersected by the curve *segment* γ is consecutive in the clockwise ordering of $E_\Gamma(P)$. The set $E_\Gamma(P)$ is returned as $\mathcal{O}(n^{1-\frac{1}{k}})$ subtrees $\{T_1, T_2, \ldots, T_m\}$ of the secondary trees. Consider a subtree T_i and the associated binary search tree on its edges. Because of the earlier discussed property, the subset of $E_\Gamma(P)$ that is intersected by the segment γ must be a consecutive subset of the leaves of T_i. Thus using T_i we can obtain these consecutive leaves in $\mathcal{O}(\log n)$ time by testing if the segment γ lies before or after the point of intersection between Γ and an edge in $E_\Gamma(P)$.

The time and space needed for detecting case (b) dominates the time and space needed for case (a) and (c) and we conclude:

▶ **Theorem 6.** *Let P' be a convex polygon with n vertices. In $\mathcal{O}(n \log^2 n)$ time we can build a data structure of size $\mathcal{O}(n \log^2 n)$ with which we can test if an arbitrary degree-2 query curve segment γ with $k + 1 \leq 6$ degrees of freedom intersects P' in $\mathcal{O}(n^{1-\frac{1}{k}} \log n)$ time.*

[4] Alternatively, cutting trees [12] can be applied to obtain faster query time at a larger space cost.

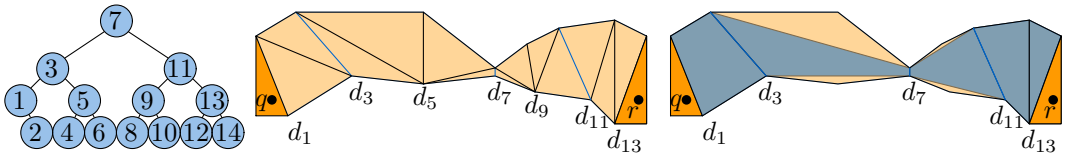

Figure 5 A triangulated polygon P with the diagonals labelled d_1 to d_{14}. There are 14 diagonals between q and r. However, we have pre-stored hourglasses $H(d_1, d_3)$, $H(d_3, d_7)$, $H(d_7, d_{11})$ and $H(d_{11}, d_{14})$. At query time, we only have to concatenate these $\mathcal{O}(\log n)$ hourglasses to get $H(d_1, d_{14})$.

4 A data structure for two entities moving inside a simple polygon

In this section we build a data structure to answer trajectory visibility queries when both the entities q and r move linearly, possibly at different speeds, inside a simple polygon P. Our main approach is the same as in our algorithm from Theorem 4: we obtain the convex polygon $\Lambda(q, r)$ that is the dual of the visibility glass $L(q, r)$, and test if the curve segment γ tracing the line through q and r in the dual space intersects $\Lambda(q, r)$. By Observation 2 this allows us to answer trajectory visibility queries. The main challenge is that we cannot afford to construct $\Lambda(q, r)$ explicitly. Instead, our data structure will allow us to obtain a compact representation of $\Lambda(q, r)$ that we can query for intersections with γ.

To obtain $\Lambda(q, r)$ we use a variation of two-point shortest-path query data structure of Guibas and Hershberger [28]. Their data structure (Figure 5) compactly stores a collection of hourglasses that can be concatenated to obtain a shortest path between two arbitrary points $p, p' \in P$. All shortest paths, in particular the boundary of the hourglasses, are represented using balanced binary search trees storing the vertices on the path. By reusing shared subtrees these $\mathcal{O}(n)$ hourglasses can be stored using only $\mathcal{O}(n)$ space. To report the shortest path between two query points their data structure concatenates $\mathcal{O}(\log n)$ of these hourglasses. The result is again represented by a balanced binary tree.

We now present a short overview of our data structure (refer to Figure 6). Unlike the Guibas and Hershberger structure, we store the hourglasses explicitly. In particular, the vertices on the boundary of an hourglass are stored in the leaves of a balanced binary search tree. The internal nodes of these trees correspond to semi-convex subchains. Let T denote the collection of all these nodes. Each node $v \in T$ stores its subchain C_v in an associated data structure. Specifically, we dualize the supporting-lines of the edges in C_v to points (refer to Figure 7). Two consecutive edges produce two points in the dual, which we again connect into semi-convex polygonal chains. So for every vertex in the sub-chain C_v the associated data structure Δ_v actually stores a line-segment; together these segments again form a polygonal chain Ψ_v. The associated data structure will support intersection queries with a quadratic query segment γ; i.e. it will allow us to report the segments of Ψ_v intersected by γ. We implement Δ_v using the data structure from Theorem 6.

▶ **Lemma 7.** *The total size of all chains Ψ_v over all nodes v is $\mathcal{O}(n \log^3 n)$.*

Proof. The Guibas and Hershberger data structure is essentially a balanced hierarchical subdivision that recursively partitions the polygon into two roughly equal size subpolygons. Every subpolygon has $\mathcal{O}(\log n)$ diagonals [28], and thus stores at most $\mathcal{O}(\log^2 n)$ hourglasses.[5]

[5] We use the version of Guibas and Hersberger's structure that achieves only $\mathcal{O}(\log^2 n)$ query time.

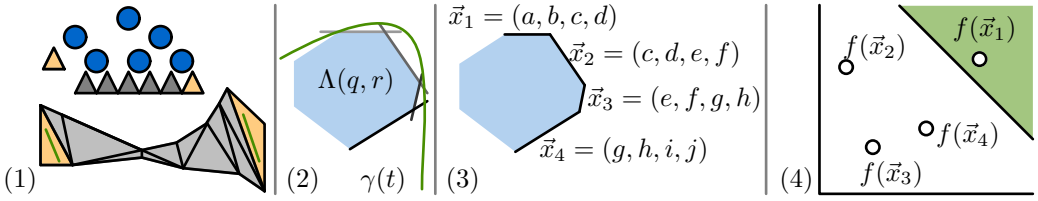

Figure 6 (1) The base level of our data structure is a hierarchical triangulation. (2) Given q and r, we compute $\Lambda(q,r)$ and the degree-2 curve segment γ. (3) We store the parameters of each edge of $\Lambda(q,r)$ (4) Each parameter vector gets mapped to a point in $\mathbb{R}^4$ and the query curve segment gets mapped to a 4-dimensional halfspace which is empty only if γ intersects no edge from $\Lambda(q,r)$.

It follows that all hourglasses of a subpolygon of size m use at most $\mathcal{O}(m \log^2 n)$ space. The height of the balanced hierarchical subdivision is $\mathcal{O}(\log n)$, and at every level the total size of the subpolygons is $\mathcal{O}(n)$. Therefore, the total size of all subpolygons is $\mathcal{O}(n \log n)$. ◀

For a chain of size $m = |\Psi_v|$ the data structure Δ_v has size $\mathcal{O}(m \log^2 m)$ and can be built in $\mathcal{O}(m \log^2 m)$ time. It follows that our data structure uses $\mathcal{O}(n \log^5 n)$ space in total, and can be built in $\mathcal{O}(n \log^5 n)$ time.

Querying the data structure. When we get a trajectory visibility query with entities q and r we have to test if the curve γ traced by the point dual to the line through q and r intersects $\Lambda(q,r)$. By Observation 1 the primal representation $L(q,r)$ of $\Lambda(q,r)$ is an hourglass $H(q',r')$. We now argue that: (i) we can find the subsegments q' and r' in $\mathcal{O}(\log n)$ time, (ii) that our data structure can report $\mathcal{O}(\log^2 n)$ nodes from T that together represent an hourglass $H(q',r')$, and (iii) that we can then test if γ intersects $\Lambda(q,r)$ by using the associated data structures of these reported nodes. This will result in $\mathcal{O}(n^{\frac{3}{4}} \log^3 n)$ query time.

▶ **Lemma 8.** *Given our data structure, we can detect if $L(q,r)$ is empty, or compute the subsegments $q' \subseteq q$ and $r' \subseteq r$ such that $L(q,r) = H(q',r')$ in $\mathcal{O}(\log n)$ time.*

Proof. By Observation 1 the visibility glass $L(q,r)$ is either empty or the hourglass $H(q',r')$ for two subsegments q' and r' and these two subsegments are bounded by the two bitangents of $H(q,r)$. These bitangents are the extension of two edges, from the shortest paths between the edges of q and r. We explained in the proof of Observation 1 that the hourglass $H(q,r)$ had an upper and lower semi-convex chain which may or may not share a point. The upper and lower chain are both a shortest path between endpoints of q and r. We can obtain them

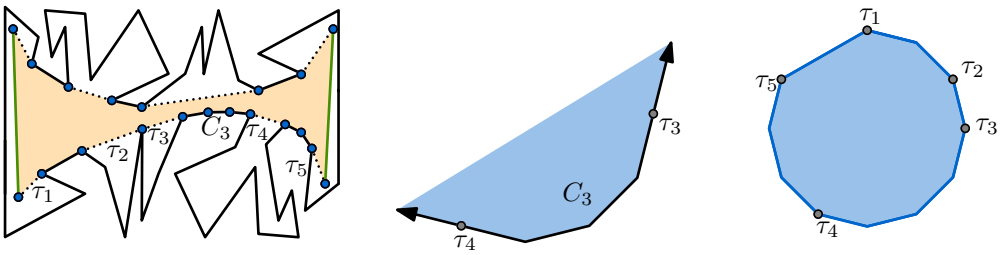

Figure 7 (left) An hourglass between q and r in orange. The lower chain consists of four chains that coincide with P, joined by outer tangents in dotted lines labelled $\tau_1 \ldots \tau_5$. (middle) The area bounded by the dualized chain C_3. Note that this chain has four edges since in the primal C_3 has four vertices. (right) A simplified version of $\Lambda(q,r)$. Outer tangents become vertices of $\Lambda(q,r)$.

using the data structure $\mathcal{D}$ from [28] as a balanced binary search tree and we can verify if they share a point using this tree. If that is the case then $L(q,r)$ is either empty or a single segment and we can verify this using an additional $\mathcal{O}(\log n)$ time.

If the upper and lower chain do not share a point then we want to identify the subsegments q' and r' for which $L(q,r) = H(q',r')$ and recall that q' and r' are bounded by the bitangents of $H(q,r)$. Such a bitangent is the extension of an edge (u,v) on the shortest path between two endpoints of q and r. The edge (u,v) is the unique edge on this path for which the path makes a clockwise turn at u and a counterclockwise turn at v or vice versa. Using $\mathcal{D}$ we can obtain any path as a balanced binary search tree. We perform a binary search on this tree to identify the edge (u,v) whose endpoints have this unique clockwise ordering. ◀

We use Lemma 8 to find the endpoints q_1, q_2 of q' and r_1, r_2 of r', respectively. We can obtain the shortest paths $\pi(r_1,q_1)$ and $\pi(r_2,q_2)$ bounding $L(q,r) = H(q',r')$ by concatenating $\mathcal{O}(\log n)$ of the pre-stored hourglasses. To concatenate two hourglasses we actually select two contiguous subchains in both hourglasses, and compute two bridge edges connecting them. Such a contiguous subchain can be represented by $\mathcal{O}(\log n)$ nodes in the binary search trees representing the hourglass boundary. It follows that $\pi(r_1,q_1)$ can be represented by $\mathcal{O}(\log^2 n)$ nodes; each representing a pre-stored subchain in the data structure, together with $\mathcal{O}(\log^2 n)$ line-segments (the bridge segments). We now observe that the chains stored in the associated data structures of these nodes together with $\mathcal{O}(\log^2 n)$ line segments Ξ (the duals of the bridge segments) actually represent the dual $\Lambda(q,r)$ of $L(q,r)$.

To check if the quadratic query segment γ intersects $\Lambda(q,r)$ we check if one of the endpoints of Q lies in $\Lambda(q,r)$; in this case one of the paths $\pi(r_1,q_1)$ or $\pi(r_2,q_2)$ is actually a single segment, or if γ intersects the boundary of $\Lambda(q,r)$. To this end, we query each of these associated data structures. Since γ has $k+1=5$ degrees of freedom (Lemma 1) this takes $\mathcal{O}(n^{\frac{3}{4}}\log^3 n)$ time. We test for intersection with the segments in Ξ separately. We thus obtain the following result.

▶ **Theorem 9.** *Let P be a simple polygon with n vertices. We can store P in a data structure of size $O(n\log^5 n)$ that allows us to answer trajectory visibility queries in $\mathcal{O}(n^{\frac{3}{4}}\log^3 n)$ time. Building the data structure takes $\mathcal{O}(n\log^5 n)$ time.*

5 A data structure for queries with one moving entity

In this section we develop data structures that can efficiently answer trajectory visibility queries in case one of the entities q is stationary, while r travels along a line segment. The appraoch described above also applies here; but we present a simpler solution using linear space which gives $O(\log n)$ query time.

We consider three variants of this setting: (i) P is a simple polygon and r is contained in P, (ii) P is a simple polygon but the trajectory of r may intersect edges of P, and (iii) P is a polygonal domain and r may intersect edges of P.

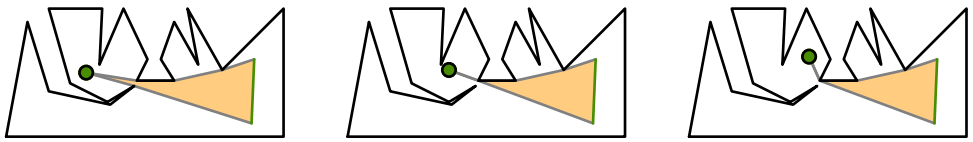

Figure 8 Three times a query pair (q,r) in a simple polygon. In the middle case, the paths π_1, π_2 share their first line segment but there still is a point on r which is visible from q.

Entity r is contained in a simple polygon. Consider the shortest paths π_1, π_2 from q to the end points of r. Observe that if edges of π_1 and π_2 coincide, they coincide in a connected chain from q [28]. Moreover (Figure 8) if more than one line segment of π_1 and π_2 coincide, then any shortest path from q to a point on r cannot be a single line segment. If no edges of π_1 and π_2 coincide then there is at least one point on r, whose shortest path to q is a line segment. If exactly one line segment of π_1 coincides with a segment of π_2, then that segment must be connected to q and if there is a line-of-sight between q and r, it has to follow that line segment. This observation allows us to answer a visibility query by considering only the first three vertices of π_1 and π_2. These vertices can be found in $\mathcal{O}(\log n)$ time using the two-point shortest path data structure of Guibas and Hershberger [28]. We conclude:

▶ **Theorem 10.** *Let P be a simple polygon with n vertices. We can store P in a data structure of size $O(n)$ that allows us to answer trajectory visibility queries between a static and a linearly moving entity in $\mathcal{O}(\log n)$ time. Building the data structure takes $\mathcal{O}(n)$ time.*

Entity r can cross a simple polygon. If r is able to move through edges of P then its trajectory may intersect the boundary of P linearly often. Inspecting each of the resulting subsegments explicitly would thus require at least $\Omega(n)$ time. Hence, we use a different approach. Let V_q denote the *visibility polygon* of point q: the set of all points visible from q. There is a time at which q can see r if and only if the trajectory of r intersects (the boundary of) V_q. We build a data structure to find such a point (if one exists).

Aronov et al. [3] actually developed an $\mathcal{O}(n^2)$ size data structure that can be built in $\mathcal{O}(n^2 \log n)$ time and can report the visibility polygon of an arbitrary query point $q \in P$ in $\mathcal{O}(\log^2 n)$ time. The visibility polygon V_q is returned in its combinatorial representation, that is, as a (pointer to a) balanced binary search tree, storing the vertices of V_q in order along the boundary. It is important to note that this combinatorial representation does not explicitly store the locations of all vertices of V_q. Instead, a vertex v of V_q may be represented by a pair (e, w), indicating that v is the intersection point of polygon edge e and the line through vertex $w \in P$ and the query point q. Computing the explicit location of all vertices of V_q thus takes $\mathcal{O}(|V_q|)$ time, if so desired, by traversing the tree. We now extend the results of Aronov et al. in such a way that we can efficiently test if a line segment intersects V_q *without* spending the $\mathcal{O}(|V_q|)$ time to compute the explicit locations.

We briefly review the results of Aronov et al. first. They build a balanced hierarchical decomposition of P [14]. Each node v in the balanced hierarchical decomposition represents a subpolygon P_v of P (the root corresponds to P itself) and a diagonal of P_v that splits P_v into two roughly equal size subpolygons P_ℓ and P_r. For subpolygon P_ℓ the data structure stores a planar subdivision $\mathcal{S}_\ell$ (of the area outside P_ℓ) such that for all points in a cell of $\mathcal{S}_\ell$ the part of the visibility polygon inside P_ℓ has the same combinatorial representation. Moreover, for each cell it stores the corresponding combinatorial representation. These representations can be stored compactly by traversing $\mathcal{S}_\ell$ while maintaining the (representation of the) visibility polygon in P_ℓ in a partially persistent red black tree [3]. The data structure stores an analogous subdivision for P_r. The complete visibility polygon of q can be obtained by concatenating $\mathcal{O}(\log n)$ subchains of these pre-stored combinatorial representations (one from every level of the hierarchical decomposition).

We use the same approach as Aronov et al. [3], but we use a different representation of V_q (refer to Figure 9). Our representation will be a weight balanced binary search tree ($BB[\alpha]$-tree [38]) whose leaves store the vertices of V_q in order along the boundary. An internal node of this tree corresponds to a subchain of vertices along V_q, which is stored in an associated data structure. We distinguish two types of vertices in such a chain: *fixed vertices*,

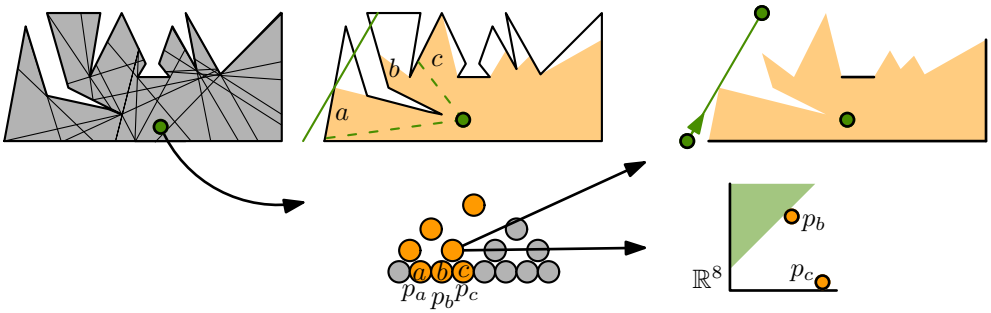

Figure 9 (left) A simple polygon split in $\mathcal{O}(n^2)$ cells. For each cell, there exists a red-black tree that represents a visibility polygon. (middle) Given V_q and r, r could intersect the explicit V_q depending on the location of q. We shoot two rays from q to r and find their intersection with V_q in the red-black tree. That gives us three leaves highlighted in orange. (right top) All the fixed edges in this node are stored in a ray-shooting data structure, (right bottom) all the variate edges have 8-dimensional points that are stored in an 8-dimensional partition tree.

for which we know the exact location, and *variate* vertices, which are represented by an polygon-edge, polygon-vertex pair (e, w). We store the fixed vertices in a linear size dynamic data structure that supports halfspace emptyness queries, that is, a dynamic convex hull data structure [8]. This data structure uses $\mathcal{O}(m)$ space, and supports $\mathcal{O}(\log m)$ time updates and queries, where m is the number of stored points. The variate vertices are mapped to a point in $\mathbb{R}^8$ using a function f that is independent of q. We give the precise definition later. We store the resulting points in a dynamic data structure that can answer halfspace emptyness queries [1]. This data structure uses $\mathcal{O}(m \log m)$ space, answers queries in $\mathcal{O}(m^{\frac{3}{4}+\varepsilon})$ time and supports updates in $\mathcal{O}(\log^2 m)$ time, where m is the number of points stored. It follows that our representation of V_q uses $\mathcal{O}(n \log^2 n)$ space, and supports updates in amortized $\mathcal{O}(\log^3 n)$ time.

Since all nodes in the data structure have constant in-degree we can make it partially persistent at the cost of $\mathcal{O}(\log^3 n)$ space per update [21]. It follows we can represent the visibility polygons for all cells in $\mathcal{S}_\ell$ in $\mathcal{O}(n^2 \log^3 n)$ space.

Querying. Given a query q, r we test if the segment r intersects V_q. The main idea is to query our data structure for the part of V_q in the wedge defined by q and r. We then extend r into a line ρ, and test if this line separates a vertex of V_q in this wedge from q. The segment r intersects V_q, and thus there is a time at which r is visible from q, if and only if this is the case.

We can obtain the part of V_q that lies in the wedge defined by q and r, represented by $\mathcal{O}(\log^2 n)$ $BB[\alpha]$-tree nodes. For each of these nodes we query the associated data structures to test if the halfspace $\rho^{\neg q}$ not containing q is empty. We can directly query the data structure storing the fixed vertices with $\rho^{\neg q}$. To test if there is a variate vertex that lies in $\rho^{\neg q}$ we map it to a halfspace in $\mathbb{R}^8$ using a function g.

▶ **Lemma 11.** *There are functions f and g such that f maps each variate vertex (e, w) to a point $f(e, w) \in \mathbb{R}^8$ and g maps each $\rho^{\neg q}$ to a halfspace $g(\rho^{\neg q})$ in $\mathbb{R}^8$ such that $f(e, w) \in g(\rho^{\neg q})$ if and only if the location of the variate vertex (e, w) in V_q lies in $\rho^{\neg q}$.*

Proof. Let $q = (a_3, a_4)$, and let $\rho = \{x, y \mid 0 = a_1 x - a_2 - y\}$ be the supporting line of the trajectory of r. We describe the construction for the case that q lies below ρ and ρ is non-vertical. The other cases can be handled analogously.

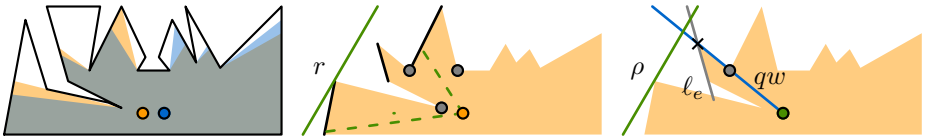

Figure 10 (left) Two points in orange and blue, which have the same implicit visibility polygon, however there could be several placement of r such that r only intersects one of the two explicit visibility polygons. (middle) Entity r in green, q in orange and the chain of uncertain edges. (right) An illustration of the geometric argument. We compute the intersection between ℓ_e and qv and check if that point lies below or above ρ.

Refer to Figure 10 (right) for an illustration of the proof. For each variate vertex (e, w) in a chain we know that the line qw intersects the line ℓ_e supporting e on the domain of e (this property is guaranteed since (e, w) is a vertex of V_q). Moreover, it is guaranteed that the intersection point between qv and ρ lies on the trajectory of r. It follows that q can see r if and only if, the intersection point (x, y) between qw and ℓ_e lies above ρ. Given ρ, q, w and ℓ_e, we can algebraically compute this intersection point (x, y). We then substitute the equation for (x, y) into the equation for ρ and the point (x, y) lies above this line if and only if the result is greater than 0:

$$wq := \left\{ x, y \mid 0 = \frac{x_4 - a_4}{x_3 - a_3}x - \frac{x_4 - a_4}{x_3 - a_3}x_3 + x_4 \right\}$$

The lines wq and ℓ_e intersect at the point where their y-coordinate is equal and therefore:

$$\begin{aligned} x_1x - x_2 &= \frac{x_4 - a_4}{x_3 - a_3}x - \frac{x_4 - a_4}{x_3 - a_3}x_3 + x_4 \\ (x_3 - a_3)(x_1x - x_2) &= (x_4 - a_4)x - (x_4 - a_4)x_3 + (x_3 - a_3)x_4 \\ (x_3 - a_3)x_1x - (x_4 - a_4)x &= x_2(x_3 - a_3) - (x_4 - a_4)x_3 + (x_3 - a_3)x_4 \end{aligned}$$

From this equation we can extract the coordinates of the intersection point (x, y) between wq and ℓ_e:

$$\begin{aligned} x &= \frac{x_2(x_3 - a_3) - (x_4 - a_4)x_3 + (x_3 - a_3)x_4}{(x_3 - a_3)x_1 - (x_4 - a_4)} \\ y &= x_1\frac{x_2(x_3 - a_3) - (x_4 - a_4)x_3 + (x_3 - a_3)x_4}{(x_3 - a_3)x_1 - (x_4 - a_4)} - x_2 \end{aligned}$$

Lastly we substitute the algebraic expression for (x, y) into the formula for ρ and we linearize the predicate:

$$\begin{aligned} 0 \geq\ & a_1(x_2(x_3 - a_3) - (x_4 - a_4)x_3 + (x_3 - a_3)x_4) - \\ & x_2 - x_1(x_2(x_3 - a_3) - (x_4 - a_4)x_3 + (x_3 - a_3)x_4) + x_2 \\ 0 \geq\ & [-a_1a_3](x_2) + [a_3](x_1x_2) + [a_1](x_2x_3) + [a_1a_4](x_3) + \\ & [-a_4](x_1x_3) + [-a_1a_3](x_4) + [a_3](x_1x_4) + [-1](x_1x_2x_3) \end{aligned}$$

Thus we found a predicate $F(\vec{x}, \vec{a})$ with:

$$\begin{aligned} (f_1, f_2, f_3, f_4, f_5, f_6, f_6, f_8) &= (x_2, x_1x_2, x_2x_3, x_3, x_1x_3, x_4, x_1x_4, x_1x_2x_3) \\ (g_0, g_1, g_2, g_3, g_4, g_5, g_6, g_7, g_8) &= (0, -a_1a_3, a_3, a_1, a_1a_4, -a_4, -a_1a_3, a_3, -1) \end{aligned}$$

It follows that we can map every variate vertex to a point in $\mathbb{R}^8$ using the f-maps provided by the predicate. Any query consisting of the halfplane $\rho^{\neg q}$ defined by ρ and q gets mapped to a halfspace in $\mathbb{R}^8$. The halfplane $\rho^{\neg q}$ contains the variate vertex defined by q, w, and e if and only if its representative point lies in this halfspace. ◀

This theorem now immediately follows:

▶ **Theorem 12.** *Let P be a simple polygon with n vertices. We can store P in a data structure of size $\mathcal{O}(n^2 \log^3 n)$ that allows us to answer trajectory visibility queries between a static and a linearly moving entity that may cross P in $\mathcal{O}(n^{\frac{3}{4}+\varepsilon})$ time. Building the data structure takes $\mathcal{O}(n^2 \log^3 n)$ time.*

Polygonal domains. In case P is a polygonal domain we use a similar approach; we build a subdivision $\mathcal{S}$ in which all points in a cell have a visibility polygon V_q with the same combinatorial structure, and then traverse $\mathcal{S}$ while maintaining V_q in a partially persistent data structure. To obtain $\mathcal{S}$ we simply we simply take all $\mathcal{O}(n^2)$ lines defined by pairs of polygon vertices. The subdivision $\mathcal{S}$ is the arrangement of these lines and has $\mathcal{O}(n^4)$ complexity. We obtain a traversal of $\mathcal{S}$ by computing an Euler tour of a spanning tree of the dual of $\mathcal{S}$. We conclude

▶ **Theorem 13.** *Let P be a polygonal domain with n vertices. We can store P in a data structure of size $\mathcal{O}(n^4 \log^3 n)$ that allows us to answer trajectory visibility queries between a static and a linearly moving entity that may cross P in $\mathcal{O}(n^{\frac{3}{4}+\varepsilon})$ time. Building the data structure takes $\mathcal{O}(n^4 \log^3 n)$ time.*

Next, we investigate the variants where the entities can walk through edges of P, and/or where P is a polygonal domain. We switch to different techniques, focused on representing the set of all potential visibility polygons in P compactly. Our representation supports efficient intersection queries of the visibility polygon of a specific point q with the trajectory of r. We obtain the following result.

▶ **Theorem 14.** *Let P be a simple polygon with n vertices. We can store P in a data structure of size $\mathcal{O}(n^2 \log^3 n)$ that allows us to answer trajectory visibility queries between a static and a linearly moving entity that may cross P in $\mathcal{O}(n^{\frac{3}{4}+\varepsilon})$ time. Building the data structure takes $\mathcal{O}(n^2 \log^3 n)$ time. If P is a polygonal domain, we can still obtain this result using $\mathcal{O}(n^4 \log^3 n)$ space and $\mathcal{O}(n^{\frac{3}{4}+\varepsilon})$ query time.*

6 A data structure for queries in polygonal domains

Finally, in Appendix C we consider the most general version of the problem, where both entities are moving in a polygonal domain (and/or can move through walls). Now the set of visibility lines from one trajectory to the other does not form a single hourglass but consist of linearly many hourglasses, thus the visibility glass does not dualize to a convex polygon. As a result, the solutions based on partition trees no longer lead to sublinear query times. Instead, we discuss a different approach based on cutting trees, leading to polylogarithmic query time at the cost of much higher space usage. We obtain the following result.

▶ **Theorem 15.** *Let P be a polygonal domain with n vertices. We can store P in a data structure of size $O(n^{3k})$, for some sufficiently large constant k, that allows us to answer trajectory visibility queries in $\mathcal{O}(\log^k n)$ time. Building the data structure takes $\mathcal{O}(n^{3k})$ time.*

References

1 P. K. Agarwal and J. Matoušek. Dynamic half-space range reporting and its applications. *Algorithmica*, 13(4):325–345, April 1995. doi:10.1007/BF01293483.

2 Pankaj K. Agarwal, Jivr'i Matouvsek, and Micha Sharir. On range searching with semialgebraic sets. II. *SICOMP*, 42(6):2039–2062, 2013. doi:10.1137/120890855.

3 Boris Aronov, Leonidas J Guibas, Marek Teichmann, and Li Zhang. Visibility queries and maintenance in simple polygons. *DCG*, 27(4):461–483, 2002.

4 Michael Ben-Or. Lower bounds for algebraic computation trees. In *STOC*, pages 80–86, 1983.

5 Marc Benkert, Joachim Gudmundsson, Florian Hübner, and Thomas Wolle. Reporting flock patterns. *Comput. Geom.*, 41(3):111–125, 2008. doi:10.1016/j.comgeo.2007.10.003.

6 Marshall Bern, David Dobkin, David Eppstein, and Robert Grossman. Visibility with a moving point of view. *Algorithmica*, 11(4):360–378, 1994.

7 P. Bovet and S. Benhamou. Spatial analysis of animals' movements using a correlated random walk model. *J. Theoretical Biology*, 131(4):419–433, 1988. doi:10.1016/S0022-5193(88)80038-9.

8 Gerth Stølting Brodal and Riko Jacob. Dynamic planar convex hull. In *FOCS*, pages 617–626, 2002.

9 Kevin Buchin, Maike Buchin, Joachim Gudmundsson, Maarten Löffler, and Jun Luo. Detecting commuting patterns by clustering subtrajectories. *IJCGA*, 21(03):253–282, 2011. doi:10.1142/S0218195911003652.

10 Jean Cardinal and Udo Hoffmann. Recognition and complexity of point visibility graphs. *DCG*, 57(1):164–178, 2017.

11 Timothy M Chan. Optimal partition trees. *DCG*, 47(4):661–690, 2012.

12 Bernard Chazelle. Cutting hyperplanes for divide-and-conquer. *DCG*, 9(2):145–158, 1993.

13 Bernard Chazelle, Herbert Edelsbrunner, Michelangelo Grigni, Leonidas Guibas, John Hershberger, Micha Sharir, and Jack Snoeyink. Ray shooting in polygons using geodesic triangulations. *Algorithmica*, 12(1):54–68, 1994.

14 Bernard Chazelle and Leonidas J. Guibas. Visibility and intersection problems in plane geometry. *DCG*, 4:551–581, 1989. doi:10.1007/BF02187747.

15 Vasek Chvátal. A combinatorial theorem in plane geometry. *JCTB*, 18(1):39–41, 1975.

16 Mark De Berg. *Ray shooting, depth orders and hidden surface removal*, volume 703. Springer Science & Business Media, 1993.

17 Mark De Berg, Marc Van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational geometry*. Springer, 1997.

18 Yago Diez, Matias Korman, André van Renssen, Marcel Roeloffzen, and Frank Staals. Kinetic all-pairs shortest path in a simple polygon. *EuroCG*, pages 21–24, 2017. abstract.

19 David P. Dobkin and David G. Kirkpatrick. Fast detection of polyhedral intersection. *TCS*, 27(3):241–253, 1983. ICALP. doi:10.1016/0304-3975(82)90120-7.

20 Somayeh Dodge, Robert Weibel, and Ehsan Forootan. Revealing the physics of movement: Comparing the similarity of movement characteristics of different types of moving objects. *Computers, Environment and Urban Systems*, 33(6):419–434, 2009.

21 James R. Driscoll, Neil Sarnak, Daniel D. Sleator, and Robert E. Tarjan. Making data structures persistent. *JCSS*, 38(1):86–124, 1989. doi:10.1016/0022-0000(89)90034-2.

22 Frédo Durand. A multidisciplinary survey of visibility. *ACM Siggraph course notes Visibility, Problems, Techniques, and Applications*, 2000.

23 Steve Fisk. A short proof of chvátal's watchman theorem. *JCTB*, 24(3):374, 1978.

24 Leila De Floriani and Paola Magillo. Algorithms for visibility computation on terrains: a survey. *Environ. Plann. B*, 30(5):709–728, 2003.

25 S. Gaffney and P. Smyth. Trajectory clustering with mixtures of regression models. In *KDD*, pages 63–72, 1999.

26 Subir K. Ghosh and Partha P. Goswami. Unsolved problems in visibility graphs of points, segments, and polygons. *CSUR*, 46(2):22:1–22:29, December 2013. doi:10.1145/2543581.2543589.

27 Joachim Gudmundsson, Patrick Laube, and Thomas Wolle. Movement patterns in spatio-temporal data. In *Encyclopedia of GIS.*, pages 1362–1370. Springer, 2017. doi:10.1007/978-3-319-17885-1_823.

28 Leonidas J. Guibas and John Hershberger. Optimal shortest path queries in a simple polygon. *J. Comput. Syst. Sci.*, 39(2):126–152, 1989. doi:10.1016/0022-0000(89)90041-X.

29 Leonidas J. Guibas, John Hershberger, Daniel Leven, Micha Sharir, and Robert Endre Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2:209–233, 1987. doi:10.1007/BF01840360.

30 Eliezer Gurarie, Russel D. Andrews, and Kristin L. Laidre. A novel method for identifying behavioural changes in animal movement data. *Ecology Letters*, 12(5):395–408, 2009. doi:10.1111/j.1461-0248.2009.01293.x.

31 Patrick Laube, Marc J. van Kreveld, and Stephan Imfeld. Finding REMO - detecting relative motion patterns in geospatial lifelines. In *SDH*, pages 201–215, 2004. doi:10.1007/3-540-26772-7_16.

32 J.G. Lee, J. Han, and K.Y. Whang. Trajectory clustering: a partition-and-group framework. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 593–604, 2007.

33 Xiaojie Li, Xiang Li, Daimin Tang, and Xianrui Xu. Deriving features of traffic flow around an intersection from trajectories of vehicles. In *Proc. 18th International Conference on Geoinformatics*, pages 1–5. IEEE, 2010.

34 J. Matoušek. Efficient partition trees. *DCG*, 1992.

35 Jiří Matoušek. Range searching with efficient hierarchical cuttings. *DCG*, 10(2):157–182, 1993. doi:10.1007/BF02573972.

36 Ether Moet. *Computation and complexity of visibility in geometric environments.* PhD thesis, Utrecht University, 2008.

37 Ketan Mulmuley. Hidden surface removal with respect to a moving view point. In *STOC*, pages 512–522. ACM, 1991.

38 J. Nievergelt and E. M. Reingold. Binary Search Trees of Bounded Balance. *SICOMP*, 2(1):33–43, 1973.

39 Joseph O'Rourke. *Art Gallery Theorems and Algorithms.* Oxford University Press, Inc., New York, NY, USA, 1987.

40 Michel Pocchiola and Gert Vegter. The visibility complex. *IJCGA*, 6(3):279–308, 1996.

41 Andreas Stohl. Computation, accuracy and applications of trajectories – a review and bibliography. *Atmospheric Environment*, 32(6):947–966, 1998. doi:10.1016/S1352-2310(97)00457-3.

42 Emo Welzl. Constructing the visibility graph for n-line segments in $o(n^2)$ time. *IPL*, 20(4):167–171, 1985.

A Transforming two entites into an algebraic curve.

Throughout this paper, we study the line-of-sight between two entities that each move along a linear trajectory (possibly at different but constant speeds) during the time $t \in [0, 1]$. Consider the line $g(t)$ through the two entities at time t. We can dualize $g(t)$ to a point using classical point-line dualization. In Section 3 we claim that the continuous dualization of the line through the two entities traces a degree-2 curve segment denoted by γ in the dual. Here we show why this is the case:

▶ **Lemma 1.** *The segment γ is a segment of a quadratic curve with 5 degrees of freedom.*

Proof. For algebraic convenience we say that entity q walks from (a_1, a_2) to $(a_1 + a_3, a_2 + a_4)$ and that entity r walks from (a_5, a_6) to $(a_5 + a_7, a_6 + a_8)$. Note that the speed of entity q is $\|(a_3, a_4)\|$ and that the speed of entity r is $\|(a_7, a_8)\|$. We can parametrize the position of entity q and r on the time $t \in [0, 1]$ as follows:

$$q(t) = \begin{pmatrix} x_{q(t)} \\ y_{q(t)} \end{pmatrix} = \begin{pmatrix} a_1 + a_3 t \\ a_2 + a_4 t \end{pmatrix} \quad r(t) = \begin{pmatrix} x_{r(t)} \\ y_{r(t)} \end{pmatrix} = \begin{pmatrix} a_5 + a_7 t \\ a_6 + a_8 t \end{pmatrix} \tag{1}$$

At all times, the line $g(t)$ is the line through the points $q(t)$ and $r(t)$. We say that at all times, $g(t)$ has slope and offset $(\alpha(t), \beta(t))$. The parametrisation of $g(t)$ then becomes:

$$g(t) = \begin{pmatrix} \alpha(t) \\ \beta(t) \end{pmatrix} = \begin{pmatrix} \frac{y_{r(t)} - y_{q(t)}}{x_{r(t)} - x_{q(t)}} \\ \alpha(t) \cdot x_{q(t)} - y_{q(t)} \end{pmatrix} = \begin{pmatrix} \frac{a_6 - a_2 + (a_8 - a_4)t}{a_5 - a_1 + (a_7 - a_3)t} \\ \alpha(t)(a_1 - a_3 t) - a_2 - a_4 t \end{pmatrix} \tag{2}$$

If the time t lies between 0 and 1, this parametric equation traces our curve segment γ and if we take t over all of $\mathbb{R}$, the parametric equation traces a full curve which we denote by Γ. To show the degree of the curve Γ we rewrite the parametrized curve to a canonical form that drops the dependence on t. First we take the formula for the β-coordinate and isolate t:

$$t = \frac{\alpha(t)a_1 - a_2 - \beta(t)}{\alpha(t)a_3 + a_4}$$

We then take the formula for the α-coordinate and remove the fraction by multiplying both sides with $((a_5 - a_1) + (a_7 - a_3)t)$. Note that this expression is only zero if the line $g(t)$ is vertical. Refer to below on how to avoid such degeneracies.

$$\alpha(t)(a_5 - a_1) + \alpha(t)(a_7 - a_3)t = (a_6 - a_2) + (a_8 - a_4)t$$

We substitute the value for t into this equation, and remove the fraction by multiplying both sides with $(\alpha(t)a_3 + a_4)$:

$$\begin{aligned} \alpha(t)(a_5 - a_1) \cdot (\alpha(t)a_3 + a_4) + \alpha(t)(a_7 - a_3) \cdot (\alpha(t)a_1 - a_2 - \beta(t)) = \\ (a_6 - a_2) \cdot (\alpha(t)a_3 + a_4) + (a_8 - a_4) \cdot (\alpha(t)a_1 - a_2 - \beta(t)) \Rightarrow \\ \alpha(t)^2 a_3(a_5 - a_1) + \alpha(t)a_4(a_5 - a_1) + \\ \alpha(t)^2 a_1(a_7 - a_3) - \alpha(t)a_2(a_7 - a_3) - \alpha(t)\beta(t)(a_7 - a_3) = \\ \alpha(t)a_3(a_6 - a_2) + a_4(a_6 - a_2) + \alpha(t)a_1(a_8 - a_4) - a_2(a_8 - a_4) - \beta(t)(a_8 - a_4) \end{aligned}$$

Lastly we show that this equation provides a linearization as defined in Appendix B by separating polynomials based on $\alpha(t)$ and $\beta(t)$ from polynomials based on $a_1 \dots a_8$.

$$[\alpha(2)^2](a_3(a_5 - a_1) + a_1(a_7 - a_3)) + \tag{3}$$

$$[\alpha(t)](a_4(a_5 - a_1) - a_2(a_7 - a_3) - a_3(a_6 - a_2) - a_1(a_8 - a_4)) - \tag{4}$$

$$[\alpha(t)\beta(t)](a_7 - a_3) + [\beta(t)](a_8 - a_4) + \alpha(t)(a_1(a_8 - a_4))[1](a_2(a_8 - a_4) - a_4(a_6 - a_2)) \tag{5}$$

◀

Dealing with degeneracies in this paper. Observe that in Equation 2 it is possible to divide by zero. Note that this occurs only if there is a moment in time where the line-of-sight between the two entities is a vertical segment. This situation occurs throughout this paper, and in this case the dual of their line-of-sight is also not well defined. Generally we cannot construct the dual of a visibility glass if in the primal it contains any vertical lines of sight.

This is a common degeneracy with visibility queries and dualization algorithms in computational geometry and it can be solved as follows: For any two (input or query) segments, one can split the time interval into two disjoint intervals, such that in one interval the segment is never vertical and in the second interval the segment is never horizontal. Note that only one split is needed, which can be calculated in constant time, because the entities move linearly. Therefore we solve the algorithmic question or the data structure question by solving two separate inputs or queries, where for the time interval that can contain vertical but not horizontal lines-of-sight, we consider a rotated version of the plane.

Similarly, whenever we consider any visibility glass, we split it into lines that are steeper than $y = x$ and lines that are not. One such set will never contain horizontal lines and the other will never contain vertical lines. Therefore, for each set of lines we can construct the dual of the visibility glass in an appropriate rotated version of the plane.

B Semi-algebraic range searching

Throughout this paper we make extensive use of the semi-algebraic (or linearization) techniques from Agarwal et al. [2]. We describe the technique in detail for completeness.

Let X be a set of n geometric objects in $\mathbb{R}^d$, where each object is parametrized by a vector $\vec{x}$. If X is a set of points, then the most natural parametrization is a vector of its coordinates. But X could be a more complicated algebraic object such as a line, in that case the most natural parametrization is a two-dimensional vector containing its slope and offset.

We denote by Γ the family of geometric regions (called semi-algebraic ranges) in $\mathbb{R}^d$ where each region $G \in \Gamma$ is bounded by an algebraic curve γ which is parametrized by a vector $\vec{a}$. Two examples of such a family is the set of all disks and the set of all disks of radius 1. An arbitrary disk can be represented as a vector in many ways: three non-colinear points define a unique circle so one could represent a circle as a six-dimensional vector which specifies the coordinates of these points. However the larger the representation vector, the more complicated the linearization process becomes. The most efficient representation of a circle is by a three-dimensional vector specifying its center and radius. The family of disks of radius 1 has fewer degrees of freedom than the family of all disks, and thus their representation can be more efficiently represented (e.g. as a 2-dimensional vector specifying only its center).

Given X and Γ, we are interested in preprocessing X such that for any range $G \in \Gamma$, we can report which objects of X intersect G. To accomplish this, we first want to derive what we have dubbed a *predicate function* $F(\vec{x}, \vec{a}) \leq 0$. The predicate function F takes any instance of X (parametrized by $\vec{x}$) and any instance of Γ (parametrized by $\vec{a}$) and outputs a real number. The object intersects the range if and only if the output value is lesser then or equal to zero. At this point we present an example: let X be a set of two-dimensional points parametrized by $\vec{x} = (x_1, x_2)$ and Γ be the set of arbitrary disks, each parametrized by $\vec{a} = (a_1, a_2, r)$. The disk parametrized by $\vec{a}$ has center (a_1, a_2) and radius r. Any point (x_1, x_2) is contained in this disk if and only if $F(\vec{x}, \vec{a}) = (a_1 - x_1)^2 + (a_2 - x_2)^2 - r^2 \leq 0$ and thus we have found our predicate function.

The predicate function $F(\vec{x}, \vec{a})$ could be seen as a map from the parameter space of our intersection problem to the boolean space $\{0, 1\}$ and thus F partitions our parameter space into areas where the answer is *yes* or areas where the answer is *no*. The idea behind

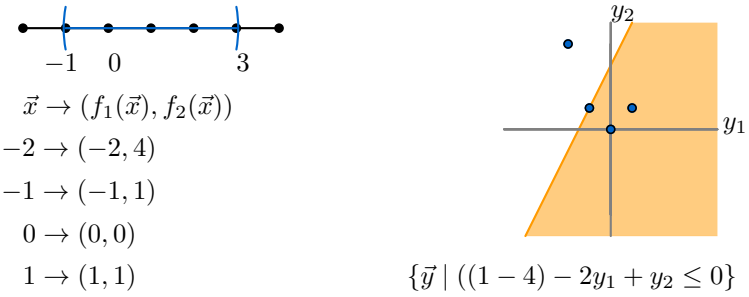

Figure 11 Consider the family Γ of 1-dimensional unit disks. We can parametrize their border by supplying a 1-dimensional center point and a radius $\vec{a} = (a_1, a_2)$. Any 1-dimensional point $\vec{x} = (x_1)$ is contained in a disk $G \in \Gamma$ if and only if $(a_1 - x_1)^2 - a_2^2 \leq 0 \Rightarrow [a_1^2 - a_2^2] + [-2a_1](x_1) + (x_1^2) \leq 0$. If we linearize this predicate function, we get that $(g_0, g_1, g_2) = (a_1^2 - a_2^2, -2a_1, 1)$ and $(f_1, f_2) = (x_1, x_1^2)$. It follows that any intersection query between a disk G and a set of points X can be answered using a halfspace emptyness query. In the figure we show an example for the points $(-2, -1, 0, 1)$ and the disk G with center 1 and radius 2.

semi-algebraic range searching is that we search for an intersection in this parameter space, as opposed to searching in the space where our problem lives. However, the border of these areas do not have to be particularly nice. In our example. each of the *yes* areas is bounded by a quadratic surface. This is where *linearization* comes in. We transform the parameter space through a polynomial map into a k-dimensional space where the boundary of the *yes* spaces becomes a linear-complexity surface. At this point, we wish to mention that in full generality, this map does not have to be a polynomial map. But for the purpose of this paper, we can restrict the results from [2] so that at all times, all the functions that we regard are bounded degree polynomials.

Given such a linearization, we can solve our problem using halfspace range searching. Specifically, we rewrite the function $F(\vec{x}, \vec{a})$ to the form: $F(\vec{x}, \vec{a}) = g_0(\vec{a}) + \sum_{i=1}^{k} g_i(\vec{a}) f_i(\vec{x})$ where f_i and g_i are polynomials dependent only on $\vec{x}$ and $\vec{a}$ respectively. In our example we had: $F(\vec{x}, \vec{a}) = (a_1 - x_1)^2 + (a_2 - x_2)^2 - r^2 \leq 0$. To linearize this function we need to first expand the squares: $F(\vec{a}, \vec{x}) = a_1^2 - 2a_1x_1 + x_1^2 + a_2^2 - 2a_2x_2 + x_2^2 - r^2$. This immediately gives a straight-forward linearization of seven terms. However, we can reduce the number of terms by grouping variables and writing: $F(\vec{x}, \vec{a}) = [a_1^2 + a_2^2 - r^2] + [-2a_1](x_1) + [-2a_2](x_2) + [1](x_1^2 + x_2^2)$ and obtain a linearization where: $(g_0, g_1, g_2, g_3) = (a_1^2 + a_2^2 - r^2, -2a_1, -2a_2, 1)$ and $(f_1, f_2, f_3) = (x_1, x_2, x_1^2 + x_2^2)$. We get the term g_0 (which is not attached to any polynomial dependent on x) "for free" and this thus becomes a three-dimensional linearization.

Agarwal et al. prove that you can map any d-dimensional point $\vec{x}$ to the k-dimensional point $f(\vec{x}) = (f_1(\vec{x}), f_2(\vec{x}), \ldots f_k(\vec{x}))$, and any query range to the k-dimensional halfspace $G(\vec{a}) = \left\{\vec{y} \in \mathbb{R}^k \mid g_0(\vec{a}) + \sum_i^k g_i(\vec{a}) y_i \leq 0\right\}$ and that $\vec{x}$ intersects G if and only if $f(\vec{x})$ is contained in $G(\vec{a})$. Consider our example. The map f that we found, is the well-known paraboloid projection: We map all the two-dimensional points onto a three-dimensional paraboloid and they are contained in a circle $G \in G$ if and only if the points lie within a halfplane cutting the paraboloid. Refer to Figure 11 for an even shorter example.

C Two moving entities in a polygonal domain

We investigate the two cases where (1) the entities can walk through edges of P and (2) P is a polygonal domain simultaneously. Let k be an unspecified constant. We prove that it is possible to preprocess a polygonal domain P with n vertices in $\mathcal{O}(n^k)$ time, such that for any

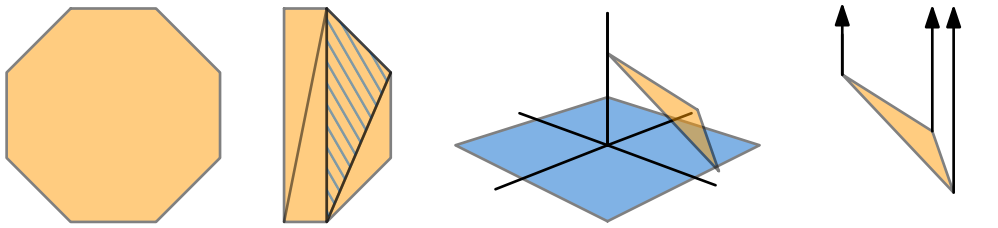

Figure 12 Unfortunately we cannot draw figures in four dimensions. So we illustrate the mapping from a visibility glass to a three-dimensional volume instead using the map $F_1(a, b)$ based on only the edge e_1 and not the edge e_2.

two entities q and r that each traverse a line segment possibly through edges of P, we can determine if there is a moment when q and r are mutually visible in sub-linear time. Our approach almost certainly does not generate an optimal solution with respect to its space requirement and query time. However, it is a non-trivial proof that sub-linear query times are achievable. We obtain these results, by transforming the visibility query in the plane into an intersection query in $\mathbb{R}^4$ and then immediately applying semi-algebraic range searching.

Let e_1 and e_2 be two edges of P and denote their visibility glass by $L(e_1, e_2)$. We say that e_1 lies on the line $y = x_1x - x_2$ and e_2 lies on the line $y = x_3x - x_4$. Let $\ell :: y = ax - b$ be a line through $L(e_1, e_2)$ with positive slope and let e_1 lie below e_2 along ℓ. The x-coordinate of the intersection between ℓ and the edges is given by $F_1(a, b) = \frac{b-x_2}{a-x_1}$ and $F_2(a, b) = \frac{b-x_4}{a-x_3}$.

▶ **Observation 3.** *Suppose we have a segment on ℓ that starts at the point q and ends at the point r then this segment is contained in $L(e_1, e_2)$ if and only if $F_1(a, b) \leq x_q \leq x_r \leq F_2(a, b)$.*

This observation leads to the following approach for detecting if there is a line-of-sight between q and r: we construct for each of the n^2 pairs of edges e_1, e_2, the two-dimensional area $\Lambda^+(e_1, e_2)$ which is the dualization of all positive slope lines through the visibility glass between e_1 and e_2. For reasons that will become apparent later, we triangulate $\Lambda^+(e_1, e_2)$. Consider any triangle $T^+(e_1, e_2)$ of this triangulation. It represents a collection of positive-slope lines that stab through the visibility glass $L(e_1, e_2)$. We lift $T^+(e_1, e_2)$ to a two-dimensional surface to $\mathbb{R}^4$ with the map that takes a point (a, b) in $T^+(e_1, e_2)$ and that maps it to the point $(a, b, F_1(a, b), F_2(a, b))$. This creates a two-dimensional surface in $\mathbb{R}^4$ which has a constant description size. Now consider the following cylinder-like volume in $\mathbb{R}^4$: $T^*(e_1, e_2) = \{(a, b, c, d) \in \mathbb{R}^4 \mid (a, b, c', d') \in T^+(e_1, e_2) \wedge c' \leq c \wedge c \leq d \wedge d \leq d'\}$. Any point $(a, b, c, d) \in T^*(e_1, e_2)$, represents a line segment that lies on the line $y = ax - b$, whose start point lies below its end point, and whose start and end points lie between e_1 and e_2. Refer to Figure 12 for an example of this transformation in $\mathbb{R}^3$:

Let q and r be given as two line-segment trajectories that do not intersect (if they do intersect, we can always split the visibility query into constantly many visibility queries). Note that we can split q and r into two sub-segments q' and r' where the entity q always has a lower y-coordinate than entity r and where the line through q and r has positive slope. We denote by γ' the continuous dualization of q' and r' according to equation 3. The two-dimensional curve segment γ' can be mapped to a curve segment in $\mathbb{R}^4$ with a mapping that is very similar to our earlier transformation. Each point $(a, b) \in \gamma'$ represents a segment following the line $y = ax - b$ between q and r where q must lie below r. We map the point (a, b) to the point (a, b, c, d) where c and d are the x-coordinates of intersections of the line

$y = ax - b$ with the trajectories of q and r respectively. Coincidentally, this means that we are mapping a point (a, b) to (a, b, x_q, x_r). If γ' intersects $T^*(e_1, e_2)$ then at the time if intersection, the two entities realise a line segment that lies within the visibility glass $L(e_1, e_2)$ and it follows that the entities are mutually visible. Both the volume T^* and the query segment γ' can be parametrized with a constant-length parameter, so the predicate that tests their intersection can be linearized to a constant k number of terms.

It follows that we can create a data structure that stores $\mathcal{O}(n^3)$ of these volumes (one for each triangle in both the positive and negative visibility glasses), each represented by a point in $\mathbb{R}^k$. Specifically, we build a cutting tree on these $\mathcal{O}(n^3)$ points in $\mathcal{O}(n^{3k})$ time. A query supplied as two segments q and r, can be cut into constantly many pairs of segments where for each pair of segments either q is above r or vice versa. For each pair of segments, we derive its corresponding k-dimensional halfspace in $\mathcal{O}(k)$ time and we query the cutting tree in $\mathcal{O}(\log^k n)$ time to see if the halfspace if empty. There is no time when the two entities are mutually visible if and only if each of these queries reports an empty halfspace. Thus we conclude:

▶ **Theorem 15.** *Let P be a polygonal domain with n vertices. We can store P in a data structure of size $O(n^{3k})$, for some sufficiently large constant k, that allows us to answer trajectory visibility queries in $\mathcal{O}(\log^k n)$ time. Building the data structure takes $\mathcal{O}(n^{3k})$ time.*