Capturing the Shape of a Point Set With a Line Segment

Nathan van Beusekom
Department of Mathematics and Computer Science, TU Eindhoven, the Netherlands

Marc van Kreveld
Department of Information and Computing Sciences, Utrecht University, the Netherlands

Max van Mulken
Department of Mathematics and Computer Science, TU Eindhoven, the Netherlands

Marcel Roeloffzen
Department of Mathematics and Computer Science, TU Eindhoven, the Netherlands

Bettina Speckmann
Department of Mathematics and Computer Science, TU Eindhoven, the Netherlands

Jules Wulms
Department of Mathematics and Computer Science, TU Eindhoven, the Netherlands

Abstract

Detecting location-correlated groups in point sets is an important task in a wide variety of application areas. In addition to merely detecting such groups, the group's shape carries meaning as well. In this paper, we represent a group's shape using a simple geometric object, a line segment. Specifically, given a radius r, we say a line segment is *representative* of a point set P of n points if it is within distance r of each point $p \in P$. We aim to find the shortest such line segment. This problem is equivalent to stabbing a set of circles of radius r using the shortest line segment. We describe an algorithm to find the shortest representative segment in $O(n \log h + h \log^3 h)$ time, where h is the size of the convex hull of P. Additionally, we show how to maintain a stable approximation of the shortest representative segment when the points in P move.

Keywords and phrases Shape descriptor, Stabbing, Rotating calipers

Digital Object Identifier 10.57717/cgt.v4i1.74

1 Introduction

Studying location-correlated groups or clusters in point sets is of interest in a wide range of research areas. There are many algorithms and approaches to find such groups; examples include the well-known *k-means clustering* [25] or *DBSCAN* [19]. In addition to the mere existence of such groups, the group's characteristics can carry important information as well. In wildlife ecology, for example, the perceived shape of herds of prey animals contains information about the behavioral state of animals within the herd [32]. Since shape is an abstract concept that can get arbitrarily complex, it is often useful to have a simplified representation of group shape that can efficiently be computed. A natural starting point is to consider the simplest possible representation that can describe a group's shape: a line segment. Representing a group with a suitable line segment conveys information about both extent and direction, and can serve as a foundation for more complex shape descriptors.

When the points represent animals moving in the plane, the shape of the group evolves over time. The representative line segment may change in orientation and length as the group moves, collapse to a single point when the shape loses elongation, or disappear altogether if the animals disperse (see Figure 1). To reason about such dynamics, we must first establish a well-defined model in the static setting that specifies the conditions under which a line segment provides an adequate representation of a point set.

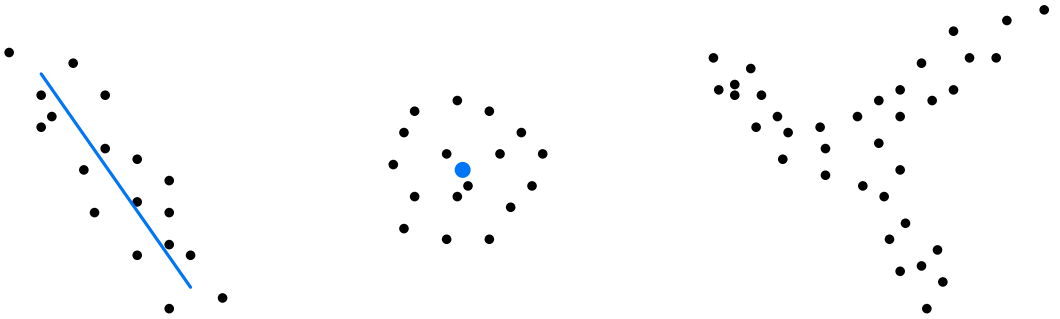

Figure 1 Examples of groups that can be represented by a line segment (left), by a single point (middle), and by neither a line segment nor a single point (right).

There are a few simple ways to define a line segment for a set of points. We can use the width of the point set to define a narrowest strip, put tight semi-circular caps on it, and use the centers of these semi-circles as the endpoints of the line segment. We can also use the focal points of the smallest enclosing ellipse, and use them as the endpoints. Alternatively, we can use a maximum allowed distance from the points to the line segment, and use the shortest line segment possible. The first and third option are based on a hippodrome shape (the Minkowski sum of a line segment and a disk). Note that it is useful to define a threshold distance to rule out that points are arbitrarily far from the defining line segment. In particular, the case that a line segment is *not* a suitable representation should exist in the model, and also the case where the line segment can become a single point. There are multiple other options besides the three given, for example, by using the first eigenvector of the points, or the diameter.

In this paper we study the model given by the third option: given a set P of n points in general position, we want to find the shortest line segment q_1q_2 such that all points are within distance r. This model has several advantages: (i) It is a simple model. (ii) It naturally includes the case that no line segment represents the points, or a single point already represents the points. (iii) It guarantees that all points are close to the approximating line segment. (iv) It has desirable properties when the points move: in the first two options, there are cases where the points intuitively remain equally stretched in the direction of the line segment, but points moving orthogonally away from it yields a shorter(!) line segment. This issue does not occur in the chosen model. Moreover, it was studied before in computational geometry, and we can build on existing algorithmic methods and properties.

The first algorithm published that solves the optimization version of the static problem (in fact, the first option) uses $O(n^4 \log n)$ time, for a set of n points [26]. This was improved by Agarwal et al. [1] to $O(n^2\alpha(n)\log^3 n)$, where $\alpha(n)$ is the extremely slowly growing functional inverse of Ackermann's function. The first subquadratic bound was given by Efrat and Sharir [18], who presented an $O(n^{1+\varepsilon})$ time algorithm, for any constant $\varepsilon > 0$. They use the fixed-radius version as a subroutine and then apply parametric search. Their fixed-radius algorithm already has the bound of $O(n^{1+\varepsilon})$, as it uses vertical decompositions of a parameter space in combination with epsilon-nets. They remark that their methods can solve the shortest stabber problem for unit disks within the same time, which is our problem.

In this paper we present an improved static result and new kinetic results. We solve the static version in $O(n \log^3 n)$ time by exploiting the geometry of the situation better, which allows us to avoid the use of parametric search and epsilon-nets. Our new algorithm uses a rotating calipers approach where we predict and handle events using relatively simple data

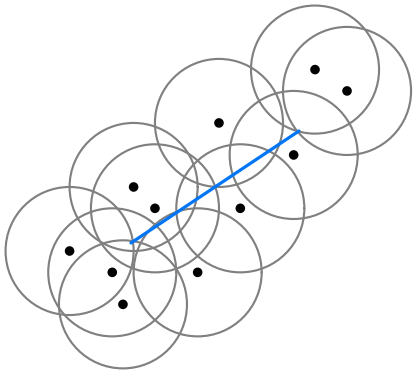

Figure 2 The line segment (blue) must hit every circle of radius r, centered at the points in P.

structures. We still use a key combinatorial result from [18] in our efficiency analysis. For the kinetic problem, we are interested in developing a strategy to maintain a line segment that can not flicker rapidly between "on" and "off", and whose endpoints move with bounded speed. To accomplish this, we must relax (approximate) the radius around the line segment in which points can be. We show that with constant speeds and a constant factor approximation in radius, the endpoints of the line segment move at a speed bounded by a linear function in r, while also avoiding frequent (dis)appearances of the line segment. These results complement recent results on stability [28, 29].

Related work. A number of shape descriptors have been proposed over the years. A few popular ones are the *alpha shape* of a point set [16] and the *characteristic shape* [13], both of which generate representative polygons. Another way to generate the shape of a point set is to fit a function to the point set [7, 24, 34]. Bounding boxes and strips are much closer related to the line segment we propose. In the orientation of the first eigenvector, bounding boxes (and strips) have been shown to not capture the dimensions of a point set well [12]. Optimal bounding boxes and strips, of minimum area and width, respectively, align with a convex hull edge and can be computed in linear time given the convex hull [20, 33]. Problems of finding one or more geometric objects that intersect a different set of geometric objects are known as *stabbing* problems [17], and several variants have been studied [6, 11, 31]. As mentioned, stabbing a set of unit circles with the shortest line segment was studied in [18]. The inverse variant, line segments stabbed by one or more circles, has also been studied [8, 27].

Recently, considerable attention has been given to stability of structures under the movement of a set of points or motion of other objects. Stability is a natural concern in, for example, (geo)visualization and automated cartography: In air traffic control, planes may be visualized as labeled points on a map, and the labels are expected to smoothly follow the locations of the moving points [9]. Similarly, for interactive maps that allow, for example, zooming and panning, labels should not flicker in and out of view [3, 21, 22, 30]. In computational geometry, only the stability of k-center problems was studied [4, 10, 14, 15], until Meulemans et al. introduced a framework for stability analysis [28]. Applying the framework to shape descriptors, they proved that an $O(1)$-approximation of an optimal oriented bounding box or strip moves only a constant-factor faster than the input points [29].

2 Computing the Shortest Representative Segment

Given a set P of n points and a distance bound r, we show how to construct the shortest segment q_1q_2 with maximum distance r to P. Our algorithm uses the rotating calipers approach [33]. We start by finding the shortest representative segment for fixed orientation α, after which we rotate by π radians while maintaining the line segment, and return the shortest one we encounter. Note that, even though a representative segment does not exist for every orientation, we can easily find an initial orientation α for which it does exist using rotating calipers; these are the orientations at which the rotating calipers have width $\leq 2r$. Although our input point set P can be of any shape, the following lemma shows that it suffices to consider only the vertices of its convex hull $\mathsf{CH}(P)$.

▶ **Lemma 1.** *If a line segment q_1q_2 intersects all circles defined by the vertices of convex hull $\mathsf{CH}(P)$, then q_1q_2 also intersects all circles defined by the points in P.*

Proof. If we assume q_1q_2 crosses each circle defined by vertices of $\mathsf{CH}(P)$, then each vertex of $\mathsf{CH}(P)$ has a distance of at most r to q_1q_2. Any point $p \in P$ that is on a convex hull edge vu for $v, u \in \mathsf{CH}(P)$ must also be at distance at most r to q_1q_2. To see why, let q_v and q_u be the closest points on q_1q_2 to v and u, respectively. Then, let $V(t)$ and $Q(t)$ be parameterizations of vu and q_vq_u obtained by linear interpolation over these segments. That is, for $t \in [0, 1]$, we have $V(t) = v + t(u - v)$ and $Q(t) = q_v + t(q_u - q_v)$. Now let $C(t) = V(t) - Q(t) = (1-t)(v-q_v)+t(u-q_u)$. As such, $C(t) \leq \max(C(0), C(1))$ for $t \in [0, 1]$. Since $C(0) \leq r$ and $C(1) \leq r$, we must have $C(t) \leq r$ for any $t \in [0, 1]$. Furthermore, for any $t \in [0, 1]$, $C(t)$ is at least the distance from $V(t)$ to q_1q_2. Therefore, any point $p \in P$ that is on convex hull edge vu must be at distance at most r to q_1q_2.

Each point $p \in P$ that is not on the convex hull also has distance at most r to q_1q_2. To see this, let q_p be the point on q_1q_2 closest to such a point p, and consider the ray emanating from q_p towards p. If we extend this ray past p, it will eventually hit the convex hull in a point c_p. To finish the proof, observe that q_p is also the closest point on q_1q_2 for c_p: either p and c_p both orthogonally project onto q_1q_2, or they both have the same endpoint of q_1q_2 as their closest point. Thus, p is closer to q_1q_2 than c_p, and hence each point in P must have a distance of at most r to q_1q_2. ◀

For the remainder of this section we work exclusively with points on the convex hull. That is, given n input points, we compute $\mathsf{CH}(P)$ in $O(n \log h)$ time, where h is the size of the convex hull [5], and discard all points that are not convex hull vertices.

2.1 Fixed orientation

We describe how to find the shortest representative segment with fixed orientation α. Using rotating calipers [33], we can find all orientations in which a representative segment exists, and pick α such that a solution exists. For ease of exposition and without loss of generality, we assume α to be horizontal. Let the *left* and *right half-circle* of a circle C be the half-circle between angles $\pi/2$ and $3\pi/2$ and between angles $3\pi/2$ and $5\pi/2$, respectively. Lemma 1 permits us to consider only points in $\mathsf{CH}(P)$, thus when considering the set $\mathcal{C}_P$ of circles of radius r centered at the points of P, we consider only those centered at points in $\mathsf{CH}(P)$.

Observe that every horizontal line that crosses all circles must lie below the bottom-most top horizontal tangent τ_1 and above the top-most bottom horizontal tangent τ_2 of all circles (see Figure 3). If τ_1 lies below τ_2, then there exists no horizontal line that crosses all circles. To place q_1q_2 in the strip between τ_1 and τ_2, we can define regions R_1, R_2 in which endpoints q_1 and q_2 must be placed such that q_1q_2 intersects all circles (see Figure 3).

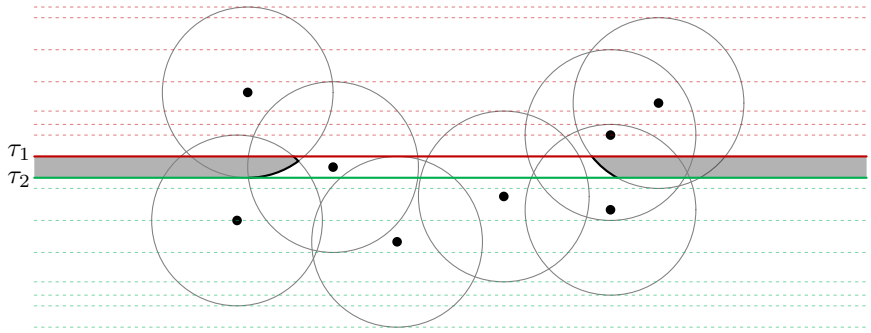

Figure 3 Two extremal tangents τ_1 and τ_2 for horizontal orientation α. The shortest line segment of orientation α that intersects all circles, ends at the boundary of the gray regions.

Given a collection of left/right half-circles, let the left/right *envelope* be a subset of the union of these half-circles such that an infinite horizontal ray shot to the left/right originating from any point in the left/right envelope is unobstructed. The region R_1 is then defined as the set of points below or on τ_1 and above or on τ_2 and to the left of the left envelope of all right half-circles. The region R_2 is defined analogously using the right envelope of left half-circles. We use S_1 and S_2 to denote the envelope boundary of R_1 and R_2 respectively. Note that S_1 and S_2 are convex and consist of a sequence of circular arcs from the left and right half-circles respectively. If R_1 and R_2 intersect, then we can place a single point in their intersection at distance at most r from all points in P. Otherwise, note that q_1 and q_2 must be on S_1 and S_2, respectively; otherwise, we can move q_1 onto S_1 or q_2 onto S_2, shortening q_1q_2 and still intersecting all circles.

We will show that we can compute S_1 and S_2 in $O(h)$ time. First, we show that the half-circles on S_1 and S_2 appear in order of the convex hull.

▶ **Lemma 2.** *The order of the circular arcs in S_1 or S_2 matches the order of their corresponding centers in* $\mathsf{CH}(P)$.

Proof. We show the proof for S_1, the argument for S_2 is analogous. If S_1 consists of only two arcs, the argument is trivial. As such, assume S_1 consists of at least three arcs.

Let s_i and s_j be any two adjacent circular arcs on S_1, centered at p_i and p_j, such that a clockwise walk starting at the intersection between τ_1 and S_1 encounters s_j before s_i. Let s_k be another circular arc on S_1, centered at p_k, that is not s_i or s_j. See Figure 4 for an example. We will show that p_i must lie clockwise from p_j on $\mathsf{CH}(P)$.

Recall that S_1 consists of arcs of the right half-circles centered at the vertices of the convex hull of P. Additionally, note that any two right half-circles intersect only once. The pairwise order of appearance of two arcs on S_1 is dependent on the vertical order of their center points. To see why, consider the left half-circle of radius r centered at the intersection between s_i and s_j. Both p_i and p_j must be on this left half-circle, otherwise their right half-circles do not intersect. Now if s_i appears clockwise from s_j on S_1, then p_i must lie clockwise from p_j on the left half-circle centered at the intersection between s_i and s_j. As such, p_i lies above p_j.

Now we have two cases: s_k is visited before or after s_j and s_i on the clockwise walk from τ_1. We consider the case where s_k is visited before s_j and s_i; the other case is analogous.

If s_k is visited before s_j and s_i, then p_k must lie below both p_j and p_i. Additionally, the right half-circle centered at p_k must intersect the right half-circle centered at p_j. However, since s_i and s_j are adjacent, this intersection must appear counterclockwise from the

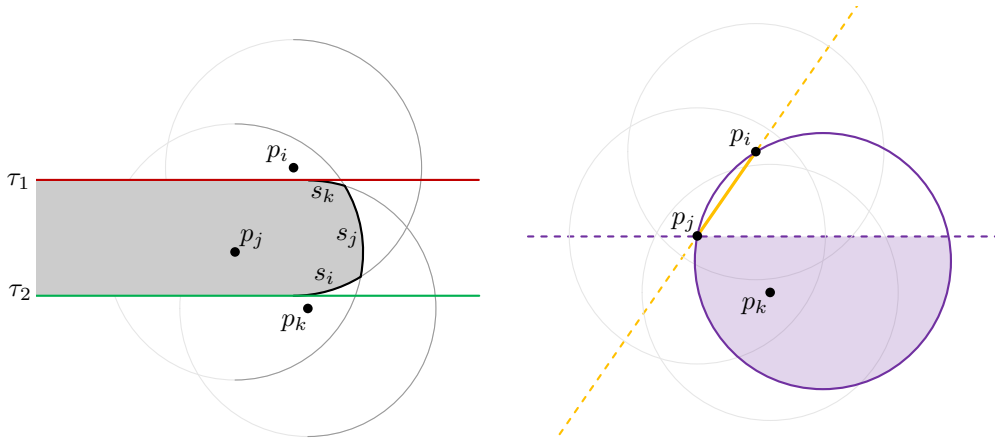

Figure 4 Arcs s_i, s_j and s_k, centered at p_i, p_j and p_k, appear in clockwise order on S_1 (left). The purple shaded area denotes the area in which p_k must lie (right).

intersection between the right half-circle centered at p_i and the right half-circle centered at p_j. This means p_k must be interior to the radius r circle through p_i and p_j. Otherwise, the intersection between s_i and s_j lies outside the radius r circle centered at p_k, which would mean s_k intersects s_i instead of s_j (see Figure 4). As such, since p_k must be both in this circle as well as below p_j, it must lie to the right of the line from p_j to p_i, and therefore cannot lie between p_j and p_i on the convex hull.

The argument for the case where s_k is visited after s_i and s_j is analogous. As such, if s_i and s_j appear in clockwise order on S_1, then p_i and p_j must lie in clockwise order on $\mathsf{CH}(P)$. Since this holds for any adjacent pair of arcs s_i and s_j on S_1, the order of arcs in S_1 must be the same as the order of corresponding center points on $\mathsf{CH}(P)$. ◀

In the following lemma we show how to compute S_1 and S_2 in linear time, given the tangents τ_1 and τ_2, which can easily be found in linear time.

▶ **Lemma 3.** *Sequences S_1 and S_2 can be constructed in $O(h)$ time, where $h = |\mathsf{CH}(P)|$.*

Proof. Assume a solution exists (this can be checked in $O(1)$ time by simply comparing τ_1 and τ_2). Given tangents τ_1, τ_2, and convex hull $\mathsf{CH}(P)$ of size h, we describe how to construct S_1, in clockwise order, starting at τ_1. Construction of S_2 is analogous. We can find the first arc on S_1 by checking all intersections between τ_1 and the right half-circles, and identifying the extremal intersection in $O(h)$ time (see Figure 3). We add the part of the half-circle that lies between τ_1 and τ_2 to S_1. We then process each point p_i along the convex hull in clockwise order from the point defining our initial arc.

Let $s_1, \ldots, s_k$ denote the circular arc pieces for S_1 constructed so far. Let p_i be the next point clockwise on the convex hull. Let c_i denote the right half-circle centered at p_i and let c_k be the supporting right half-circle of s_k. By Lemma 2, if a part of c_i appears on S_1, it must appear adjacent to s_k. We check for the intersection between c_i and c_k, and perform updates to S_1 depending on where this intersection is. There are three cases: (1) c_i and c_k do not intersect, (2) c_i and c_k intersect above τ_1 or below τ_2, (3) c_i and c_k intersect between τ_1 and τ_2. We discuss each of these cases separately.

(1) If c_i and c_k do not intersect, then c_i must lie entirely to the right of c_k in the strip between τ_1 and τ_2 (since otherwise s_k cannot be part of S_1). As such, c_i does not contribute to S_1, and we do not perform any updates.

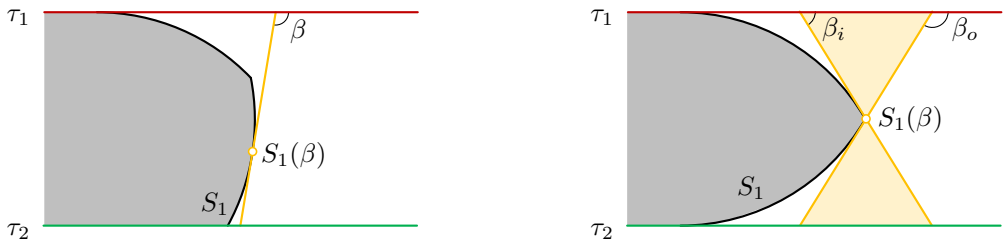

Figure 5 $S_1(\beta)$ indicates the location where the tangent of S_1 (yellow) makes an angle β with τ_1. In the right case, all values $\beta_i < \beta < \beta_o$ map to the vertex.

(2) If c_i and c_k intersect above τ_1 or below τ_2, then c_i lies to the right of c_k between τ_1 and τ_2 as well (since otherwise s_k cannot be part of S_1). Therefore, c_i again does not contribute to S_1, and we do not perform any updates.

(3) If c_i and c_k intersect between τ_1 and τ_2, we need to update S_1 depending on where this intersection lies on c_k.

(3a) If c_i intersects s_k, then we shorten s_k to end at its intersection with c_i, and the entire arc of c_i clockwise from its intersection with s_k to S_1.

(3b) If c_i does not intersect s_k, this means that the intersection between c_i and c_k must lie counterclockwise from s_k. This means, s_k cannot contribute to S_1, since it lies entirely to the right of c_i. We remove s_k from S_1, and do the full comparison again using c_i and c_{k-1}.

We analyze the time it takes to construct S_1 using a charging argument on these cases. Each occurrence of cases (1), (2) and (3a) can be charged to the prospective new arc that is considered; since we handle vertices in order of the convex hull, and vertices are never revisited, at most one of these cases is applicable to each prospective arc. Each occurrence of case (3b) can be charged to the arc that is permanently removed from S_1. Since we never revisit vertices, each arc is removed at most once. Each of these cases can be handled in $O(1)$ time by simply checking for intersections between circles. This leads to $O(1)$ occurrences charged to each considered vertex, each of which can be resolved in $O(1)$ time. As such, since we consider $O(h)$ vertices, construction of S_1 takes $O(h)$ time. ◀

Next, we must place q_1 and q_2 on S_1 and S_2, respectively, such that q_1q_2 is shortest. To this end, for some $\beta \in [0, 180]$, we create parameterizations $S_1(\beta)$ and $S_2(\beta)$ that determine the location on S_1 and S_2 where the angle between its tangent line and τ_1 is β (see Figure 5 (left)). For any value of β where no such tangent line exists (e.g. due to vertices on S_1 or S_2), we must have some vertex v for which $\beta_i < \beta < \beta_o$, where β_i and β_o are the associated tangent angles of the incoming/outgoing arcs of v (see Figure 5 (right)). In that case, let $S_1(\beta)$ or $S_2(\beta)$ map to the vertex v. Note that v can also be the intersection point of S_1 or S_2 with τ_1 or τ_2, in which case $\beta_i = 0$ or $\beta_o = 180$.

▶ **Lemma 4.** *Let $S_1(\beta_1)S_2(\beta_2)$ be a line segment with orientation α. Setting $\beta_1 = \beta_2$ minimizes $|S_1(\beta_1)S_2(\beta_2)|$ over all line segments of orientation α between S_1 and S_2.*

Proof. Without loss of generality, assume that α is the horizontal orientation. Consider the function $f(y)$ that describes the length $|S_1(\beta_1)S_2(\beta_2)|$ of the horizontal line segment between S_1 and S_2, parametrized by its vertical placement y between τ_1 and τ_2. The rate of change of this function over y is characterized by the tangents of S_1 and S_2, and as such is proportional to $\beta_2 - \beta_1$ (unless $S_1(\beta_1)$ or $S_2(\beta_2)$ is at a vertex, in which case the rate of change is dependent on the tangent of the outgoing arc). See Figure 6. Additionally, since S_1 and S_2 are convex, β_1 and β_2 are monotone over S_1 and S_2. As such, the function

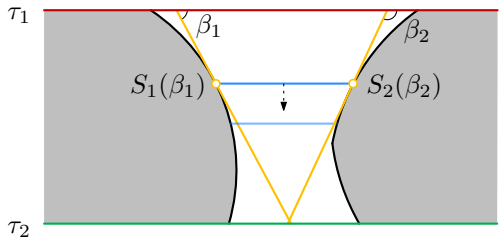

Figure 6 The change in length of the blue line segment as we move it down is dependent on β_1 and β_2.

$f(y)$ is unimodal, i.e. it has a single minimum. This means we minimize $f(y)$ if we place $S_1(\beta_1)S_2(\beta_2)$ such that the rate of change of $f(y)$ is zero, which happens when $\beta_1 = \beta_2$, and the claim follows. ◀

Given S_1 and S_2, we can simply perform a linear scan over S_1 and S_2 to find the optimal value of β. Thus, by Lemmata 3 and 4, we can compute the shortest representative segment of fixed orientation α in $O(h)$ time.

2.2 Rotation

In this section, we describe a rotational sweep performed by our algorithm to find the shortest representative segment over all orientations. After finding the shortest representative segment for a fixed orientation, we sweep through all orientations α by performing a continuous clockwise rotation. We will maintain the key attributes necessary to compute the shortest segment: the tangents τ_1, τ_2, envelope boundaries S_1, S_2, and the placement of endpoints q_1 and q_2 on S_1 and S_2. During the continuous rotation, these attributes will also be changing continuously. This can be handled by maintaining each attribute as a function of α. However, at certain discrete moments, the combinatorial structure of these attributes may change. The central idea of our approach is to forecast and handle these combinatorial changes as they occur. Our approach is very similar to how a Kinetic Data Structure [2] maintains attributes in a continuously changing system.

To efficiently detect and handle combinatorial changes, we maintain a set of *certificates*, which are conditions that certify the combinatorial structure of the attributes we are maintaining. The idea is that, whenever a certificate is violated, we must perform combinatorial updates to one or more of our attributes. We call such a certificate violation an *event*. The certificates are stored in an event queue, sorted by the orientation at which they are violated. This way, at each event, we update the maintained attributes and recompute relevant certificates prior to proceeding the rotation.

We identify four distinct types of certificates:

1. Endpoints q_1 and q_2 remain on the same arcs or vertices of S_1 and S_2.
2. The combinatorial structures of S_1 and S_2 remain unchanged.
3. The order of τ_1 and τ_2 remains unchanged.
4. Tangents τ_1 and τ_2 remain tangent to circles belonging to the same convex hull vertices.

Recall that the order of τ_1 and τ_2 determines whether a solution exists or not. That is, assuming α is horizontal, a representative segment exists if and only if τ_1 lies above (or on) τ_2. As such, the third certificate in the above list certifies whether there exists a valid solution

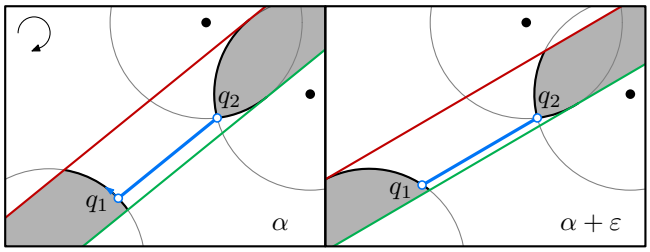

Figure 7 Rotating α clockwise causes q_1 and q_2 to either rotate counterclockwise over S_1 or S_2 (q_1), or remain stationary at a vertex (q_2).

or not. The other three certificates certify the combinatorial structure of the attributes we must maintain to find q_1q_2.

When one of the above certificates fails, it triggers an event. Each event causes one or more attributes to change, and one or more certificates to be recomputed. We distinguish the following six types of events, corresponding to the certificates above:

1a. q_1q_2 moves onto/off τ_1 or τ_2.
1b. q_1 or q_2 moves onto/off an intersection between two arcs on S_1 or S_2.
2a. τ_1 or τ_2 is tangent to S_1 or S_2 and rotates over a (prospective) vertex of S_1 or S_2.
2b. τ_1 or τ_2 is not tangent to S_1 or S_2 and rotates over a (prospective) vertex of S_1 or S_2.
3. τ_1 and τ_2 are the same line.
4. τ_1 and τ_2 are parallel to a convex hull edge.

We make the distinction between event types 1a/1b and 2a/2b, since these events should be handled slightly differently from one another, and their certificates are constructed in a slightly different manner. Therefore, in the remainder of this section, we will treat these subtypes of events separately, even though they certify the same condition.

Note that, since α is rotated in clockwise direction, the segment q_1q_2 also rotates in clockwise direction. This means q_1 and q_2 move over S_1 and S_2 in *counterclockwise* direction (or remain stationary if they are on a vertex, see Figure 7).

Since the shortest line segment q_1q_2 in orientation α is completely determined by τ_1, τ_2, S_1, and S_2, the above list forms a complete description of all possible events: (1) q_1 and q_2 move continuously/remain stationary until they move onto a different arc/vertex of S_1 or S_2, (2) S_1 and S_2 undergo a combinatorial change only when the tangents τ_1 and τ_2 hit intersections between circles in $\mathcal{C}_P$; otherwise S_1 and S_2 change continuously along the arcs on S_1 and S_2, (3) τ_1 and τ_2 are parallel, and for their order to change by continuous rotation and translation they must first become the same line, and (4) τ_1 and τ_2 are uniquely defined by a circle and a direction, so their continuous trajectories change only when they are defined by a new circle.

We maintain at most two certificates for events of type 1a (one for each of τ_1 and τ_2), type 1b (one for each of q_1 and q_2), and type 4 (one for each of τ_1 and τ_2), and a single type 3 certificate. Additionally, there must be exactly one type 2 (a or b) certificate for each endpoint of S_1 and S_2, so four in total. As such, we have $O(1)$ certificates at any time, which are stored in an event queue Q, ordered by orientation of appearance. Since Q is constant size, insert, remove, and search operations on Q can be performed in $O(1)$ time.

We will describe below how all events over a rotational sweep from orientation α to $\alpha + \pi$ can be handled in $O(h \log^3 h)$ time in total. Combined with the computation of the convex hull of P this yields the following theorem. Note that in the worst case P is in convex position, and $n = h$.

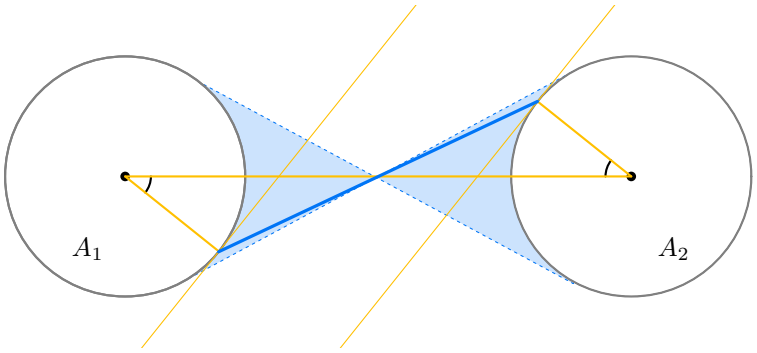

Figure 8 The shortest segment (of this orientation) between A_1 and A_2 must lie within $\mathcal{W}$ (blue), or it does not exist.

▶ **Theorem 5.** *Given a point set P consisting of n points and a radius r, we can find the shortest representative segment in $O(n \log h + h \log^3 h)$ time, where $|\mathsf{CH}(P)| = h$.*

Event handling. Prior to the rotational sweep, we initialize our algorithm by computing and storing the shortest representative segment for a fixed initial orientation α. We also store this α, as well as the value $\beta = \beta_1 = \beta_2$ that realizes this segment according to Lemma 4 as α_i and β_i. These values of α_i and β_i will be used in the handling of events of type 1a and 1b to maintain the shortest representative segment encountered during the rotational sweep.

In the following, we assume that each event occurs at a specific orientation α, and that a sufficiently small $\varepsilon > 0$ exists such that no other events occur in the interval $(\alpha - \varepsilon, \alpha + \varepsilon)$, except possibly at α itself. That is, multiple events may occur simultaneously at orientation α, but no events are arbitrarily close to each other in orientation. Moreover, we make a general position assumption such that no three points are co-linear or co-circular for a circle with radius r. This ensures that vertices of S_1 and S_2 are defined by at most two disks, and τ_1 and τ_2 are tangent to at most two disks at a time.

When multiple events of different types occur simultaneously, we handle them in reverse order of their type numbers: events of type 4 are handled first, followed by type 3, type 2b, and so on. This ordering reflects the dependency hierarchy among the event types. Specifically, observe that each event depends only on attributes maintained by events of higher types in the list. For instance, events of type 2a and 2b (which update the combinatorial structure of S_1 and S_2) depend only on the input points and the current extremal tangents τ_1 and τ_2, which are updated in events of type 4. Conversely, the placement of q_1 and q_2 (type 1a and 1b events) depend on an up-to-date combinatorial structure of S_1 and S_2. By processing the events of higher type first, we ensure that all attributes are correctly updated before use.

Before we describe how to handle the events, we first provide more insight into how the segment q_1q_2 moves during our rotational sweep. To this end, we distinguish two cases: (1) both q_1 and q_2 are on arcs of S_1 and S_2, or (2) q_1 or q_2 is on a vertex of S_1 or S_2.

In the first case, q_1 and q_2 are located on arcs of S_1 and S_2 respectively. Let A_1 and A_2 denote the full radius r circles that contain these arcs, and consider the double wedge $\mathcal{W}$ between the two inner bi-tangents of A_1 and A_2 (see Figure 8). Note that A_1 and A_2 have equal radii, which means that this wedge is symmetric with respect to A_1 and A_2. As such, any line segment ℓ between two points on A_1 and A_2 that have tangents with the same slope must lie within $\mathcal{W}$. By Lemma 4, the shortest representative segment of orientation α must lie between these two arcs of S_1 and S_2, and must therefore lie within $\mathcal{W}$, unless one of τ_1 or

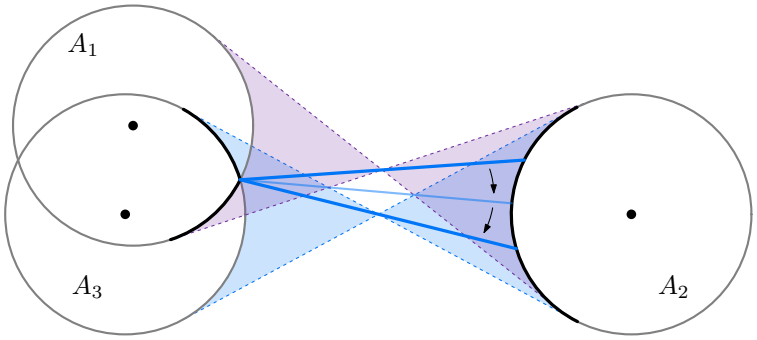

Figure 9 If q_1 is at a vertex, q_1q_2 is temporarily not contained in any double-wedge.

τ_2 prevents this placement.

In the second case, q_1 or q_2 is located on a vertex v of S_1 or S_2. Recall that, by Lemma 4, $|q_1q_2|$ is minimized for a given value of α if they are placed at $S_1(\beta_1)$ and $S_2(\beta_2)$ such that $\beta_1 = \beta_2$. If both q_1 and q_2 are at a vertex, to ensure $\beta_1 = \beta_2$ as we increase α, one of q_1 or q_2 must also immediately move off its corresponding vertex. As such, we only consider cases where one of q_1 and q_2 lies at a vertex.

Without loss of generality, let q_1 be at a vertex of S_1 and q_2 on an arc of S_2; the other case is analogous. Let A_1 and A_3 denote the full radius r circles that contain the incoming and outgoing arcs of v, respectively, and let A_2 denote the full radius r circle that contains the arc on which q_2 is located. While q_1 is at v, q_1q_2 is not contained in any double-wedge defined by the two inner bi-tangents of two circles; since q_1 is at a vertex, q_1q_2 does not actually describe a shortest segment of a given orientation between two circles. Instead, q_1q_2 is temporarily not contained fully in any double wedge $\mathcal{W}$. During this interval of orientations q_1 remains at v and q_2 continues moving counterclockwise over A_2 until q_1q_2 is contained in the double-wedge defined by the inner bi-tangents of A_3 and A_2, at which point q_1 will move onto the outgoing arc of v. See Figure 9.

We will use these observations later to efficiently compute certificates of type 1a.

(1a) q_1q_2 moves onto/off τ_1 or τ_2. We describe, without loss of generality, how to handle the event involving τ_1; the case for τ_2 is analogous. See Figure 10 for an example of this event.

An event of this type is triggered whenever the optimal segment q_1q_2 would move onto or off τ_1. In the following, let A_1 and A_2 be the full radius r circles that correspond to the arcs of S_1 and S_2 on which q_1 and q_2 are currently located. If q_1 (or q_2) is currently on an intersection of two arcs, let A_1 (or A_2, respectively) be the full radius r circle that

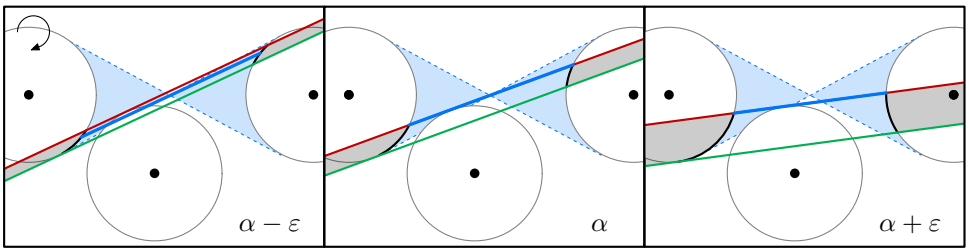

Figure 10 When q_1 or q_2 is at a vertex of S_1 or S_2, it stops moving.

corresponds to the arc counterclockwise along S_1 (or S_2) from this intersection. Additionally, let $\mathcal{W}$ be the double wedge between the inner bi-tangents of A_1 and A_2.

Constructing type 1a certificates. If both q_1 and q_2 are on arcs of S_1 and S_2, we can use the double wedge $\mathcal{W}$ to compute the next orientation at which q_1q_2 moves onto or off τ_1. Let the *focal point* $\chi_{\mathcal{W}}$ of $\mathcal{W}$ be the intersection between the two inner bi-tangents of A_1 and A_2. It follows from our observations above that the shortest segment of orientation α between A_1 and A_2 must pass through $\chi_{\mathcal{W}}$. This means that, if $\chi_{\mathcal{W}}$ does not lie between τ_1 and τ_2, the shortest representative segment q_1q_2 of orientation α cannot be the shortest segment of orientation α between A_1 and A_2. Instead, to minimize its length, q_1q_2 must be on τ_1 or τ_2 (whichever is closer to $\chi_{\mathcal{W}}$). This means that a certificate of type 1a (for τ_1) can simply be computed by finding at the next orientation at which the tangent τ_1 hits $\chi_{\mathcal{W}}$, which can be done in $O(1)$ time.

If q_1 or q_2 is at a vertex of S_1 or S_2, finding the certificate is easier. Since q_1 or q_2 is at a vertex, and q_1q_2 can only move onto τ_1 when τ_1 intersects both q_1 and q_2, we can simply find the next orientation at which τ_1 hits the vertex at which q_1 or q_2 is currently located, which can be done in $O(1)$ time.

▶ **Observation 6.** *We can construct a new certificate of type 1a in $O(1)$ time.*

Handling type 1a events. We distinguish two different cases for this event: q_1q_2 moves *onto* or *off* τ_1 at orientation α. In either case, we must update the shortest representative segment encountered so far. At the start of the event handling, we introduced parameters α_i and β_i to do this. The values of α_i and β_i are updated whenever a type 1a or 1b event happens, and represent the orientation α and tangent angle β since we have last updated the shortest representative segment.

If q_1q_2 moves onto τ_1 at orientation α, then q_1q_2 will remain on τ_1 until the next type 1a event. We place new a type 1a certificate into the event queue that is violated when q_1q_2 should move off τ_1, that is, at the the next orientation at which τ_1 hits $\chi_{\mathcal{W}}$. We also update the type 1b certificates in the event queue to reflect the new orientation at which q_1 and q_2 hit the next vertex of S_1 or S_2, respectively. Additionally, let β_o be the angle between τ_1 and the tangent of S_1 at its intersection with τ_1. Observe that, since q_1q_2 moves onto τ_1 at this orientation, this value of β_o induces the shortest segment $S_1(\beta_o)S_2(\beta_o)$ of orientation α between S_1 and S_2. At this point, since the value of β_i is updated whenever a type 1a or 1b event happens, the values $\beta \in [\beta_i, \beta_o]$ induce either a single arc or a single vertex of both S_1 and S_2, which describe the movement trajectories of q_1 and q_2 since the previous event of type 1a or 1b. We can find the shortest line segment between these arcs/vertices in $O(1)$ time using Lemma 4, which is the shortest representative segment encountered since the last type 1a or 1b event. We store this segment if it is shorter than the shortest representative segment encountered so far, and update the value of α_i to the current α and β_i to β_o.

If q_1q_2 moves off τ_1 at orientation α, then q_1q_2 will remain off τ_1 until the next type 1a event. We place two (one for each of τ_1 and τ_2) new type 1a certificates into the event queue that are violated when q_1q_2 should move onto τ_1 or τ_2, that is, at the next orientation at which τ_1 or τ_2 hits $\chi_{\mathcal{W}}$. We also update the type 1b certificates in the event queue to reflect the new orientation at which q_1 and q_2 hit the next vertex of S_1 or S_2, respectively. Additionally observe that, since the value of α_i is updated whenever a type 1a or 1b event happens, the values $\alpha_a \in [\alpha_i, \alpha]$ induce a single arc of both S_1 and S_2, which describe the movement trajectories of q_1 and q_2 since the previous event of type 1a or 1b. Finding the shortest line segment of orientation $\alpha_a \in [\alpha_i, \alpha]$ between two circular arcs is a $O(1)$ time calculation. This segment is the shortest representative segment encountered since the

last type 1a or 1b event, so again we store this segment if it is shorter than the shortest representative segment encountered so far, and update the value of α_i to the current α and β_i to β_o.

▶ **Lemma 7.** *During rotation of orientation α by π radians, type 1a events happen at most $O(h)$ times, and we can resolve each occurrence of such an event in $O(1)$ time.*

Proof. Let A_1 and A_2 denote the full radius r circles that contain the arcs of S_1 and S_2 on which q_1 and q_2 are currently located (or the outgoing arc of the vertex on which q_1 or q_2 is located). Each tangent of the circle to which τ_1 is incident that crosses through the focal point of $\mathcal{W}$ induces one type 1a event. Since at most two tangents of any circle intersect a single point, as long as A_1, A_2, and the circle to which τ_1 is incident do not change, there can be at most two events of type 1a. The circles A_1 and A_2 can change whenever a type 1b event happens, and the circle to which τ_1 is incident can change whenever a type 4 event happens. As such, we can charge these two type 1a events to occurrences of type 1b and type 4 events. We show later, in Lemmata 9 and 17, that events of type 1b and 4 happen at most $O(h)$ times. As such, there can be at most $O(h)$ occurrences of type 1a events.

Constructing a new type 1a certificate takes $O(1)$ time by Observation 6. We show later, in Observation 8, that we can also construct type 1b certificates in $O(1)$ time. Inserting a new certificate into the constant-size event queue takes $O(1)$ time. Computing the shortest representative segment between β_i and β_o or α_i and α can be done in $O(1)$ time as well. As such, events of type 1a can be resolved in $O(1)$ time. ◀

(1b) q_1 or q_2 moves onto/off an intersection between two arcs on S_1 or S_2. We describe, without loss of generality, how to handle the event involving q_1, S_1, and τ_1; the cases for q_2, S_2, and τ_2 are analogous. See Figure 11 for an example of this event.

An event of this type is triggered whenever q_1 would move onto or off an intersection of two arcs on S_1. Let this intersection, which is described by a vertex of S_1, be denoted $v \in S_1$, and let the outgoing (e.g. counterclockwise) arc of v be denoted a.

Constructing type 1b certificates. We distinguish three cases for type 1b certificates. For each of these cases, we describe how to compute the orientation α_c at which the new certificate will be violated.

(1) If q_1 is not on τ_1 and moves onto v, the new certificate will be violated at orientation α_c such that shortest segment $S_1(\beta_v)S(\beta_v)$ has orientation α_c, where β_v is the angle between τ_1 and the tangent of a at v.

(2) If q_1 is not on τ_1 and moves off v, the new certificate will be violated at orientation α_c such that the shortest segment $S_1(\beta_a)S_2(\beta_a)$ has orientation α_c, where β_a is the angle between τ_1 and the tangent at the counterclockwise endpoint of a.

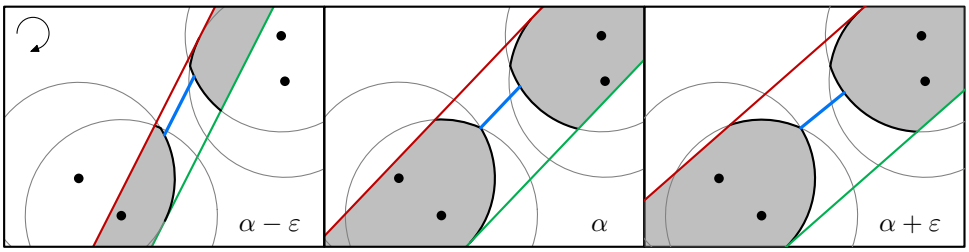

Figure 11 When q_1 or q_2 is at a vertex of S_1 or S_2, it stops moving.

(3) If q_1q_2 is on τ_1 and q_1 moves over v, observe that τ_1 hits v at orientation α as well. This means that a type 2b or 2b event will trigger before this type 1b event (crucially, however, this type 1b event will still occur), and S_1 will be updated prior to this event taking place. As such, there are two options: either v is part of S_1 at orientation $\alpha + \varepsilon$, or it is not.

(3a) If it is not, q_1 cannot not stop at v, but will continue moving over the outgoing arc a of v. This means that the new type 1b certificate will be violated at orientation α_c such that τ_1 hits the counterclockwise endpoint of a at orientation α_c.

(3b) If v is part of S_1 at orientation $\alpha + \varepsilon$, q_1 will stop at v if and only if the value of β on S_2 at q_2 is included in the range of values of β covered by v. This means that, if q_1 remains at v, the new type 1b certificate fails at orientation α_c such that shortest segment $S_1(\beta_v)S(\beta_v)$ has orientation α_c, where β_v is the angle between τ_1 and the tangent of a at v. If q_1 does not remain at v, q_1q_2 must remain on τ_1 which means that the new type 1b certificate fails at orientation α_c such that τ_1 hits the counterclockwise endpoint of a at orientation α_c.

In any of the above cases, the certificates can be found in $O(1)$ time by finding the next vertex/arc along S_1 and/or S_2 in counterclockwise direction.

▶ **Observation 8.** *We can construct a new certificate of type 1b in $O(1)$ time.*

Handling type 1b events. We describe how to handle an event triggered by a certificate violation for each of the three cases above.

If q_1 is not on τ_1 and and moves onto v at α, then q_1 was moving over an arc of S_1 at $\alpha - \varepsilon$. In that case, q_1 will remain at v at orientation $\alpha + \varepsilon$. We must recompute the type 1a certificate that is currently in the event queue to reflect the new placement of q_1. Additionally, we place a new type 1b certificate into the event queue that is violated when q_1 should move off the vertex, that is, when the final value of β at v is reached. Recomputing the shortest representative segment encountered since the last event of type 1a or 1b works the same as in the type 1a event handling; we repeat it here for completeness. To this end, let β_o be the angle between τ_1 and the tangent at v of the incoming arc of v. Observe that, since q_1 moves onto v at this orientation, this value of β_o induces the shortest segment $S_1(\beta_o)S_2(\beta_o)$ of orientation α between S_1 and S_2. At this point, since the value of β_i is updated whenever a type 1a or 1b event happens, the values $\beta \in [\beta_i, \beta_o]$ induce a single arc or a single vertex of S_1 and S_2, which indicate the movement trajectory of q_1 and q_2 since the previous event of type 1a or 1b. We can find the shortest segment between these arcs/vertices in $O(1)$ time using Lemma 4, which is the shortest representative segment encountered since the last type 1a or 1b event. We store this segment if it is shorter than the shortest representative segment encountered so far, and update the values of α_i to the current α and β_i to β_o.

If q_1 is not on τ_1 and and moves off v at α, then q_1 was already placed at v at $\alpha - \varepsilon$. In that case, q_1 should start moving in counterclockwise direction over a. We must recompute the type 1a certificate that is currently in the event queue to reflect the new placement of q_1. Additionally, we place a new type 1b certificate into the event queue that is violated when q_1 encounters the next vertex. To recompute the shortest representative segment, let β_o be the angle between τ_1 and the tangent at v of a. The remainder of the computation to find the shortest representative segment encountered since the last type 1a or 1b event is the same as described above.

If q_1q_2 is on τ_1 and q_1 moves over v, then q_1 was moving over an arc of S_1 at $\alpha - \varepsilon$. In that case, q_1 will either continue moving in counterclockwise direction over a, or remain at v. We must recompute the type 1a certificate that is currently in the event queue to reflect the new placement of q_1. Additionally, we place a new type 1b certificate into the event queue

that is violated when q_1 either encounters the next vertex, or when it should move off v. To recompute the shortest representative segment observe that, since the value of α_i is updated whenever a type 1a or 1b event happens, the values $\alpha_a \in [\alpha_i, \alpha]$ induce a single arc of both S_1 and S_2, which describe the movement trajectories of q_1 and q_2 since the previous event of type 1a or 1b. Finding the shortest line segment of orientation $\alpha_a \in [\alpha_i, \alpha]$ between two circular arcs is a $O(1)$ time calculation. This segment is the shortest representative segment encountered since the last type 1a or 1b event, so again we store this segment if it is shorter than the shortest representative segment encountered so far, and update the value of α_i to the current α and β_i to β_o.

▶ **Lemma 9.** *During rotation of orientation α by π radians, type 1b events happen at most $O(h)$ times, and we can resolve each occurrence of such an event in $O(1)$ time.*

Proof. Since S_1 is convex, q_1 moves monotonically counterclockwise over S_1. As such, because we perform a rotational sweep of π radians, by Lemma 2 each convex hull point p induces at most a constant number of events (two for each vertex on S_1 adjacent to the arc centered at p). As such, this event happens at most $O(h)$ times.

Constructing a new type 1b certificate takes $O(1)$ time by Observation 8. Inserting a new certificate into the constant-size event queue takes $O(1)$ time. ◀

(2a) τ_1 or τ_2 is tangent to S_1 or S_2 and rotates over a (prospective) vertex of S_1 or S_2. We describe, without loss of generality, how to handle the event involving τ_1 and S_1; the case for τ_2 and S_2 is analogous. See Figure 12 for an example of this event.

Constructing type 2a certificates. First, observe that we can compute certificates of this type in $O(1)$ time: let C_i be the circle to which τ_1 is tangent. Then, to construct a certificate, we find the orientation at which τ_1 hits the vertex of S_1 that is on C_i or, if that vertex does not yet exist, the intersection point between S_1 and C_i.

▶ **Observation 10.** *We can construct a new certificate of type 2a in $O(1)$ time.*

Handling type 2a events. Let vertex v be the vertex of the convex chain S_1 that is intersected by τ_1 at orientation α. Then either vertex v is a vertex of S_1 at orientation $\alpha - \varepsilon$ but not at $\alpha + \varepsilon$, or vice versa.

In the prior case, at orientation α the arc to which τ_1 is tangent is completely removed from S_1. Vertex v becomes the endpoint of S_1 and starts moving along the next arc of S_1. If the affected arc or vertex appeared in a type 1b certificate in the event queue, it is updated to reflect the removal of the arc and the new movement of the vertex. Additionally, we place a new type 2b certificate into the event queue.

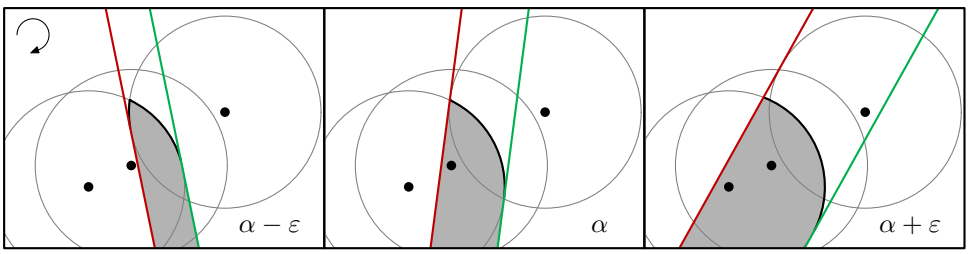

Figure 12 When τ_1 or τ_2 hits an intersection of its defining circle that is also on S_1 or S_2, an arc is removed from S_1 or S_2.

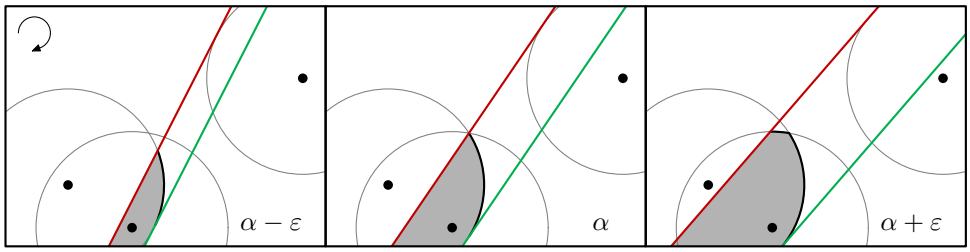

Figure 13 When τ_1 or τ_2 hits an intersection of two circles, an arc needs to be added to S_1 or S_2.

In the latter case, at orientation α an arc of the incident circle to τ_1 needs to be added to S_1. If the arc that was previously the outer arc of S_1 appeared in a type 1b certificate in the event queue, it may need to be updated to reflect the addition of the new arc. Additionally, we place a new type 2a certificate into the event queue.

▶ **Lemma 11.** *During rotation of orientation α by π radians, type 2a events happen at most $O(h)$ times, and we can resolve each occurrence of such an event in $O(\log^2 h)$ time.*

Proof. Since the circle to which τ_1 is incident changes in order of the convex hull, during a full rotational sweep τ_1 can be incident to the circle centered at each convex hull vertex only over one interval. Additionally, the rotational movement direction of the intersection between S_1 and τ_1 reverses only with events of type 2a, or when the circle to which τ_1 is incident changes (which happens once per convex hull edge). As such, events of type 2a can only occur $O(h)$ times.

To handle a type 2a event, we must add/remove the circular arc to which τ_1 is incident to S_1, which we can do in $O(1)$ time. Additionally, we possibly compute a new type 1b certificate, as well as a new type 2a or type 2b certificate, which can be done in $O(1)$ and $O(\log^2 h)$ time by Observations 8 and 10 and Lemma 12, respectively. Then, if necessary, we must remove the old type 1b certificate from the event queue, and add the new certificates, all in $O(1)$ time. ◀

(2b) τ_1 or τ_2 is not tangent to S_1 or S_2 and rotates over a (prospective) vertex of S_1 or S_2. We describe, without loss of generality, how to handle the event involving τ_1 and S_1; the case for τ_2 and S_2 is analogous. See Figure 13 for an example of this event. The following lemmata, describing construction and handling of type 2b certificates and events, are proven in Section 2.3, using an additional data structure which is described in detail in Section 2.4. Note that the event handling for this event includes both updating the combinatorial structure of S_1, as well as the recomputation of type 1b, 2a and 2b certificates.

▶ **Lemma 12.** *We can construct a new certificate of type 2b in $O(\log^2 h)$ time.*

▶ **Lemma 13.** *During rotation of orientation α by π radians, τ_1 or τ_2 hits a vertex of S_1 or S_2 at most $O(h \log h)$ times, and we handle each occurrence of this event in $O(\log^2 h)$ time.*

(3) τ_1 and τ_2 are the same line. When this event takes place, τ_1 and τ_2 are the inner bi-tangents of their two respective defining circles. See Figure 14 for an example.

Constructing type 3 certificates. Observe that this event takes place at orientation α when α is an orientation at which the incident circles of τ_1 and τ_2 have an inner bi-tangent. As such, we can compute certificates of this type in $O(1)$ time by simply finding the inner bi-tangents of the circles τ_1 and τ_2 are incident to.

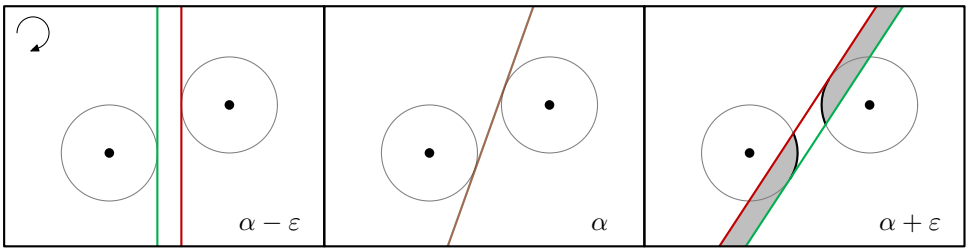

Figure 14 When τ_1 and τ_2 are the same line, they are an inner bi-tangent of their two defining circles.

▶ **Observation 14.** *We can construct a new certificate of type 3 in $O(1)$ time.*

Handling type 3 events. We distinguish two different cases for this event: either there is a solution at $\alpha - \varepsilon$ and no solution at $\alpha + \varepsilon$, or vice versa.

If there was a solution at $\alpha - \varepsilon$ and there is none at $\alpha + \varepsilon$, we simply stop maintaining q_1q_2, S_1 and S_2 until there exists a solution again. As such, we remove all type 1a, 1b, 2a, and 2b certificates from the event queue and place a new type 3 certificate into the event queue that is violated at the next orientation where τ_1 and τ_2 are the same line.

If there was no solution at $\alpha - \varepsilon$ and there is a solution at $\alpha + \varepsilon$, we must recompute S_1, S_2, and q_1q_2 at orientation α. At orientation α, S_1 and S_2 are single vertices where τ_1 and τ_2 intersect the extremal half-circles of the arrangement. Then, q_1q_2 is the line segment between these single vertices of S_1 and S_2. We place new type 1a, 1b, 2a, and 2b certificates into the event queue reflecting the newly found S_1, S_2, q_1 and q_2. Additionally, we insert a new type 3 certificate that is violated at the next orientation where $\tau_1 = \tau_2$.

▶ **Lemma 15.** *During rotation of orientation α by π radians, type 3 events happen at most $O(h)$ times, and we can resolve each occurrence of such an event in $O(\log^2 h)$ time.*

Proof. Observe that τ_1 and τ_2 form an inner bi-tangent of their respective circles when they are the same line (see Figure 14), and each pair of circles has two such inner bi-tangents. As described previously, τ_1 and τ_2 change incident circles only when they are perpendicular to a convex hull edge. This means that, once τ_1 is perpendicular to convex hull edge p_ip_j, for example, τ_1 will not become incident to the radius r circle centered at p_i again during a rotation of π radians. The same goes for τ_2. As such, we can charge each occurrence of a type 3 event to the circle incident to either τ_1 or τ_2 depending on which one will be first to be incident to a convex hull edge. Since each pair of circles has only two inner bi-tangents, this means we will charge at most four type 3 events to each circle (two for each inner bi-tangent for each of τ_1 and τ_2). Hence, type 3 events happen at most $O(h)$ times.

If there was a solution at $\alpha - \varepsilon$, we compute a new type 3 certificate, which can be done in $O(1)$ time by Observation 14. Then, we remove and insert certificates to/from a constant size event queue, which can be done in $O(1)$ time.

If there was no solution in $\alpha - \varepsilon$, then, symmetrically, there must be a solution in $\alpha + \varepsilon$. We can find the two arcs that contain q_1 and q_2 by finding the outermost intersections between τ_1 (or τ_2) and circles in $\mathcal{C}_P$. To do this efficiently, we require an additional data structure, which is described later in Section 2.4. By Lemma 21, we can query this data structure in $O(\log^2 h)$ time to obtain starting vertices for S_1 and S_2.

Next, we recompute the type 1a, 1b, and 3 certificates, which by Observations 6, 8, and 14 can be done in $O(1)$ time. Lastly, we compute a constant number of new type 2a and type

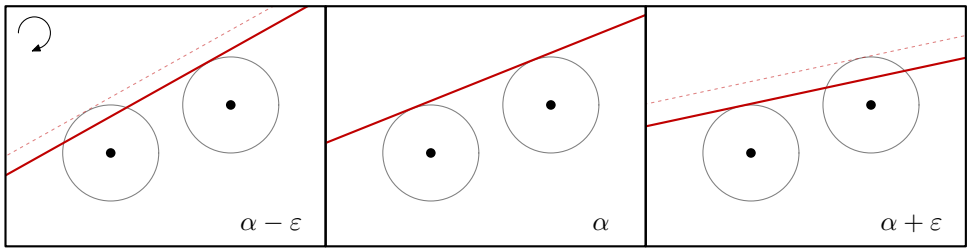

Figure 15 When the defining circle of τ_1 or τ_2 changes, τ_1 or τ_2 is parallel to a convex hull edge.

2b certificates. By Observation 10 and Lemma 12, this can be done in $O(1)$ and $O(\log^2 h)$ time. Adding these new certificates into the constant size event queue takes $O(1)$ time. ◀

(4) τ_1 and τ_2 are parallel to a convex hull edge. Let (u, v) be the convex hull edge to which τ_1 and τ_2 are parallel. Note that, when this event happens, either τ_1 or τ_2 is an outer bi-tangent to the radius r circles centered at u and v. We describe, without loss of generality, how to handle the event involving τ_1; handling τ_2 is analogous. See Figure 15 for an example.

Constructing type 4 certificates. Observe that we can compute certificates of this type in $O(1)$ time, since these certificates depend only on the orientation of the next convex hull edge.

▶ **Observation 16.** *We can construct a new certificate of type 4 in $O(1)$ time.*

Handling type 4 events. Suppose that, without loss of generality, u was the previous extremal vertex in direction $\theta - \varepsilon$. Then v is extremal in direction $\theta + \varepsilon$. As such, τ_1 is incident to the radius r circle centered at u at $\alpha - \varepsilon$, and to the radius r circle centered at v at $\alpha + \varepsilon$. When this happens, we insert a new type 4 certificate into the event queue that is violated at the orientation of the next convex hull edge. Additionally, we must recompute the certificates of type 1a, 1b, 2a, 2b and 3 that are currently in the event queue, since these are all dependent on τ_1.

▶ **Lemma 17.** *During rotation of orientation α by π radians, type 4 events happen at most $O(h)$ times, and we can resolve each occurrence of such an event in $O(\log^2 h)$ time.*

Proof. Since there are $O(h)$ convex hull edges, and this event happens at most once per convex hull edge, this event occurs at most $O(h)$ times. By Observations 6, 8, 10, 12 and 14 and Lemma 16, we can construct new type 1a, 1b, 2a, 2b, 3 and 4 certificates in $O(1)$ and $O(\log^2 h)$ time. Deletions from and insertions into the event queue take $O(1)$ time. ◀

2.3 Finding and maintaining the half-circle envelopes

In this section, we describe an additional data structure necessary to maintain half-circle envelopes S_1 and S_2 efficiently. We can use this data structure to construct and handle violations of type 2b certificates efficiently, as well as to find new starting positions of S_1 and S_2 after a type 3 event.

Let $p_1, \ldots, p_h$ be the vertices of the convex hull in clockwise order. At a given orientation α, let circle C_i be the circle to which τ_1 is incident, centered at point p_i. We use v_τ to denote the intersection point between τ_1 and S_1, if it exists, which is simultaneously an endpoint of S_1. Let C_j be the circle on which v_τ is located, centered at point p_j. This implies that the arc on S_1 intersected by τ_1 belongs to circle C_j. Then, during our rotational sweep, v_τ

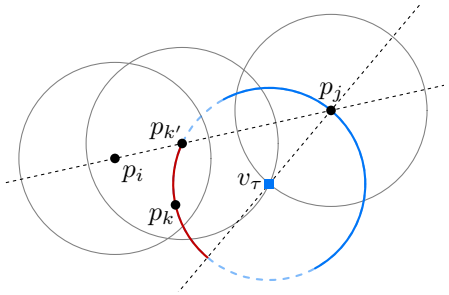

Figure 16 Point p_k is placed on the red arc, which is a subset of the (blue dashed) convex semi-circle centered at v_τ, and disjoint from the (blue solid) concave semi-circle centered at v_τ.

is moving over C_j. A type 2b event takes place when v_τ hits the intersection of C_j with another circle C_k centered at point p_k.

If, before a type 2b event, the arc of C_j on S_1 was shrinking due to the movement of v_τ, then C_j is fully removed from S_1 at the event, and v_τ continues moving over S_1. Constructing the certificate in this case is very easy, since all we need to do is walk over S_1 from v_τ to find the next vertex. As such, for the remainder of this section, we consider only the more complicated type 2b event, where the arc of C_j on S_1 is growing due to the movement of v_τ.

In that case, when the type 2b event happens, an arc of C_k is added to S_1, and v_τ starts moving over C_k instead of C_j. As such, to construct a type 2b certificate, we must find the intersection between C_j and another circle C_k centered at a point $p_k \in P$, such that the intersection between C_j and C_k is the first intersection hit by v_τ. To do this, we will first state some characteristics of C_k and p_k.

First, observe that simply finding the first intersecting circle C_k of C_j is not necessarily enough. We are only interested in the semi-circles of all circles in $\mathcal{C}_P$ that span the same angular interval as the convex chain S_1 for a given orientation α. As such, let the *convex semi-circle* of a given circle C be the semi-circle of C that is convex with respect to S_1. Conversely, let the *concave semi-circle* of C be the opposite semi-circle of C. Then, circle C_k appears on S_1 after a hit by v_τ at orientation α if v_τ is on the convex semi-circle of C_k at orientation α. If this is not the case, C_k should be skipped. We show that, for this reason, we never have to consider points p_l, with $l < i$ or $j < l$, when constructing a type 2b certificate.

▶ **Lemma 18.** *Let C_i be the circle defining τ_1, and let C_j be the circle on which v_τ is located, for $i \neq j$. Let C_k be the first circle hit by v_τ during rotation at orientation α. If $k < i$ or $j < k$, then at orientation α, v_τ lies on the concave semi-circle of C_k with orientation α.*

Proof. Without loss of generality assume p_i is positioned left of p_j, then assuming that $k < i$ or $j < k$, p_k must lie below the line through p_i and p_j (since $i < j$ and the points are ordered in clockwise order). See Figure 16 for the following construction.

Consider point $p_{k'}$ placed on the line through p_i and p_j, such that v_τ could be the bottom intersection of C_j and a radius-r circle centered at $p_{k'}$. Let p_k be a point placed on the circle of radius r centered at v_τ. If p_k is placed on the other side of the line through v_τ and p_j, compared to $p_{k'}$, then v_τ would enter C_k at orientation α, which does not induce an event. As such, p_k must be on the same side of the line through v_τ and p_j as $p_{k'}$. Additionally, since $k < i$ or $j < k$, p_k must be below the line through p_i and p_j.

It is easy to see that all points on the convex semi-circle of v_τ will have v_τ on their concave semi-circle. Since the arc on which p_k is placed is a strict subset of the concave semi-circle of v_τ, any placement of p_k must have v_τ on its concave semi-circle. ◀

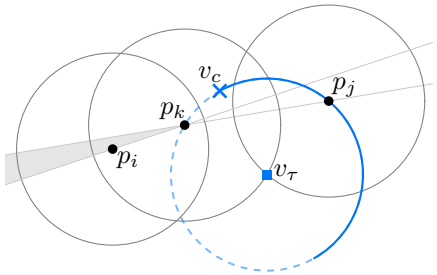

Figure 17 Point p_l must be located in the gray cone.

Lemma 18 implies that, while constructing a type 2b certificate, we need to consider only candidate points p_k such that $i < k < j$. Note that, if $i = j$, we get a type 2a event. Then, all that is left is to find the first circle C_k with $i < k < j$ that is intersected by v_τ as it moves over C_j. To this end, we describe a data structure that allows us to perform a circular ray shooting query along C_j from the orientation at which the certificate must be constructed.

Data structure. Our data structure is essentially a balanced binary tree $\mathcal{T}$ on the vertices of the convex hull in clockwise order, where each node stores an associated structure. For any i, let D_i be the disk bounded by circle C_i. Suppose a node in $\mathcal{T}$ is the root of a subtree with $p_i, \ldots, p_j$ in the leaves. Then its associated structure stores the boundary of $\bigcap_{i \le l \le j} D_l$ as a sorted sequence of circular arcs. Given a range (i, j) we can query this data structure with a (circular) ray in $O(\log^2 h)$ time to find the first intersection of the ray with the boundary of $\bigcap_{i \le l \le j} D_l$. Section 2.4 describes the data structure and query algorithms in more detail.

Event handling. Whenever a certificate of type 2b is violated at orientation α, it means some circle C_k is hit by v_τ at orientation α. It is still possible, however, that this hit happens on the concave semi-circle of C_k. In that case, C_k should not be added to S_1, and v_τ should simply continue moving over its original trajectory. We call these events, where a type 2b certificate is violated but S_1 is not updated, *internal events*. To handle an internal event, we merely need to construct another new type 2b certificate and continue our rotational sweep. In this case, however, we do not have to search the entire range $p_i, \ldots, p_j$ when constructing a new certificate, as shown in the following lemma.

▶ **Lemma 19.** *Let C_i be the circle incident to τ_1, and let v_τ be at the intersection of circles C_j and C_k, where C_j is the circle that defines the current trajectory of v_τ and where v_τ is on the concave semi-circle of C_k. Then if the next circle hit by v_τ is C_l for $i < l < k$, this circle is hit on its concave semi-circle.*

Proof. Let C_l be the next circle hit by v_τ for $i < l < k$, and see Figure 17 for the following construction. Since v_τ is on the concave semi-circle of p_k, p_k must be on the convex semi-circle of v_τ. Furthermore, as C_k was hit by v_τ, p_k must lie on the same side of the line through p_j and v_τ as p_i. Let the endpoint of the concave semi-circle of v_τ that lies clockwise from p_k be denoted v_c, and observe that $d(p_k, p_j) > d(v_c, p_j)$, where $d(a, b)$ denotes the Euclidean distance between a and b. Additionally, since we consider only points on the convex hull, point p_l must lie above line $p_i p_k$ but below line $p_k p_j$, resulting in a cone with its apex at p_k.

Every point in this cone is further away from p_j than p_k: Since v_τ is part of S_1, placing q_1 at v_τ must yield a valid solution for $q_1 q_2$. This means that all points must be in the Minkowski sum of a radius r disc and the ray with orientation α originating from v_τ, and

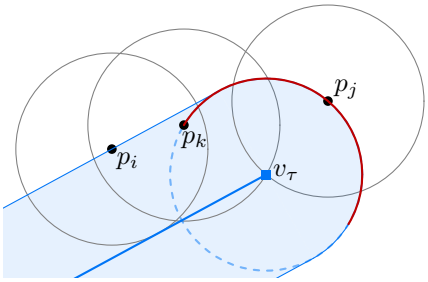

Figure 18 All points in $\mathsf{CH}(P)$ must be in the blue shaded area. When an internal event happens, the red semi-circle is a witness that (p_k, p_j) is conjugate.

hence p_k lies below the line $v_c p_i$ (see Figure 18). Finally, p_j lies on the concave semi-circle around v_τ, and p_k lies on the convex semi-circle around v_τ, which shows that the cone lies on the far side of p_k with respect to p_j. This implies that $d(p_l, p_j) > d(p_k, p_j)$.

During our monotone rotational sweep, v_c must continuously move closer to p_j as we continue our sweep. Therefore, when v_τ intersects C_l, we must have $d(p_l, p_j) > d(v_c, p_j)$: This directly implies that v_c must lie clockwise from p_l on the radius r circle centered at v_τ. Thus, v_τ lies on the concave semi-circle of C_l. ◀

Lemma 19 implies that, after an internal event with circle C_l, it is sufficient to consider only points p_k with $l < k < j$ when constructing a new type 2b certificate.

When a type 2b certificate is violated and v_τ is on the convex semi-circle of C_k, however, we do need to update S_1 to reflect v_τ moving over the intersection between C_k and C_j. Additionally, we must construct new certificates: We possibly need to compute a new type 1b certificate, as well as either a type 2a certificate or a new type 2b certificate using C_k as the new trajectory of v_τ and searching for the next hit with C_l for $i < l < k$. We are now ready to prove Lemmata 12 and 13.

▶ **Lemma 12.** *We can construct a new certificate of type 2b in $O(\log^2 h)$ time.*

Proof. Let c_i be the circle incident to τ_1, C_j be the circle over which v_τ is currently moving, and α be the current orientation. To construct a type 2b certificate, we find the first circle C_k with $i \leq l < k < j$ that is intersected by v_τ. Here, if this certificate is constructed during an internal event, l is the index of the circle C_l intersected by v_τ during that event. Otherwise, $l = i$. Since v_τ moves over C_j, we can use the data structure described in Section 2.4 to perform a circular ray shooting query on the sequence $C_l, \ldots, C_{j-1}$ with starting point v_τ to find C_k in $O(\log^2 h)$ (Lemma 20). This gives us the point p_k to include in the certificate, and we can find the orientation at which p_k is hit by drawing the tangent of C_i through the intersection point between C_j and C_k. ◀

▶ **Lemma 13.** *During rotation of orientation α by π radians, τ_1 or τ_2 hits a vertex of S_1 or S_2 at most $O(h \log h)$ times, and we handle each occurrence of this event in $O(\log^2 h)$ time.*

Proof. To show that this event happens $O(h \log h)$ times during a π rotation, we need the following definition. Let an ordered pair (p_g, p_h) of vertices of $\mathsf{CH}(P)$ be *conjugate* if we can place a semi-circle of radius r so that it hits p_g and p_h, p_h lies clockwise from p_g on this semi-circle, and the semi-circle does not intersect the interior of $\mathsf{CH}(P)$. Efrat and Sharir prove that any convex polygon with n vertices has at most $O(n \log n)$ conjugate pairs if the vertices are placed in general position [18]. We charge each occurrence of a type 2b event to

a conjugate pair, and prove that each pair is charged only a constant number of times. This immediately yields the bound of $O(h \log h)$ on the number of type 2b events.

Consider an internal type 2b event. We charge these events to the pair (p_k, p_j). To this end, we show that this pair of points must be conjugate. Consider orientation α at which this event takes place. At that point, we can draw a circle of radius r, centered at v_τ, through p_k and p_j. Since, by definition of an internal event, v_τ is on the concave semi-circle of p_k, p_k must be on the convex semi-circle centered around v_τ.

Now again observe that, since v_τ is part of S_1, placing q_1 at v_τ must yield a valid solution for q_1q_2. This means that all points must be in the Minkowski sum of a radius r disc and the ray with orientation α originating from v_τ. See Figure 18. Therefore, the concave semi-circle of radius r centered at v_τ does not intersect $\mathsf{CH}(P)$. If we rotate this semi-circle counterclockwise until one of its endpoints coincides with p_k, we obtain a semi-circle through p_k and p_j that does not intersect $\mathsf{CH}(P)$. This means it is a witness that (p_k, p_j) is conjugate.

Next, consider a type 2b event that is not internal. The same argument as above holds, except p_k is already on the concave semi-circle of radius r centered at v_τ. Since this semi-circle does not intersect $\mathsf{CH}(P)$, that semi-circle is a direct witness that (p_k, p_j) is conjugate.

Every conjugate pair only induces at most one type 2b event. In order for conjugate pair (p_k, p_j) to induce a second type 2b event, v_τ must again hit the intersection between p_k and p_j while it is moving in the same angular movement direction. This can only happen if we perform a full 2π rotational sweep, or if v_τ first moves over this intersection in the opposite direction. The prior is not possible in a π rotational sweep. The latter is only possible if p_k is first involved in a type 2a event. But then, $k = i$, and by construction p_k can never be involved in another type 2b certificate.

Handling a type 2b event consists of updating S_1 and the movement trajectory of v_τ, which can be done in $O(1)$ time. Additionally, we construct new type 1b certificate and either a type 2a or type 2b certificate, which can be done in $O(1)$ and $O(\log^2 h)$ time by Observations 8 and 10 and Lemma 12. ◀

2.4 Data structure and queries

We describe a data structure that stores the points of $\mathsf{CH}(P)$, so that for a given query subsequence $p_i, \ldots, p_j$ and another object q (a radius-r circle or a line), we can efficiently find the two intersections of $\bigcap_{i \le l \le j} D_l$ with q, or decide that they do not intersect. Here, D_l is the disk of radius r centered on p_l. Since the common intersection of disks is convex, a line will intersect it at most twice. Furthermore, since the query circle has the same radius as each of the disks D_l, it will intersect the common intersection of any subset of these disks at most twice as well. In all queries we will perform, any pair of points in the query sequence $p_i, \ldots, p_j$ are at most $2r$ apart. Furthermore, we are always interested in the intersection that is the *exit*, when we direct the line or when we direct a circular ray along the circle from a given starting point. Computing both intersections of the line or circle immediately gives this desired exit intersection as well.

We first break the sequence $p_1, \ldots, p_h$ of $\mathsf{CH}(P)$ into a number of subsequences with useful properties, see Figure 19. We will create maximal subsequences $p_s, \ldots, p_t$ so that the total turning angle is at most $\pi/2$, and the Euclidean distance between p_s and p_t is at most r. These two properties ensure that if we place radius-r disks centered on the points of $p_s, \ldots, p_t$, their common intersection is non-empty. Maximality of the subsequences ensures that for any query, we need to search in only $O(1)$ subsequences. The construction of such subsequences can be done in an incremental greedy manner and is standard. We can assume that there are at least two points in $p_1, \ldots, p_h$ that are more than r apart, otherwise all

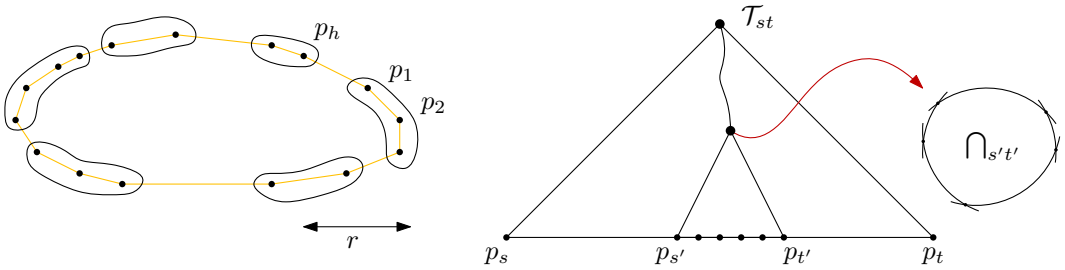

Figure 19 Left, partitioning $p_1, \ldots, p_h$ into subsequences by distance and angle. Right, schematic view of the data structure $\mathcal{T}_{st}$.

points fit in a radius-r circle.

Let $p_s, \ldots, p_t$ be any one of these subsequences, let $C_s, \ldots, C_t$ be the radius-r disks centered on these points, and let $\bigcap_{st}$ be their common intersection. We first describe a data structure for $\bigcap_{st}$, which will later be used as an associated structure for $p_s, \ldots, p_t$.

The queries on $\bigcap_{st}$ that we need to answer are the following: (1) given a circle C of radius r and a point q on it, where q lies inside $\bigcap_{st}$, report the first circular arc hit when q moves counterclockwise along C. (2) Given a query line ℓ, find the intersections with $\bigcap_{st}$. It is clear that the second query yields at most two points. For the first query, since C and $C_s, \ldots, C_t$ all have the same radius, C intersects $\bigcap_{st}$ in at most two points as well. These properties allow us to store the vertices of $\bigcap_{st}$ in sorted order and perform a simple binary search with a radius-r query circle or a line, and find the intersected arcs in $O(\log h)$ time.

The data structure for the subsequence $p_s, \ldots, p_t$ is a balanced binary search tree $\mathcal{T}_{st}$ that stores $p_s, \ldots, p_t$ in the leaves, see Figure 19. For every internal node, with leaves containing $p_{s'}, \ldots, p_{t'}$ $(s \leq s' \leq t' \leq t)$, we store as an associated structure the vertices of $\bigcap_{s't'}$ in sorted order, as just described. The complete data structure $\mathcal{T}$ is the set of all of these trees. Breaking up into subsequences guarantees that every node in any tree in $\mathcal{T}$ has a non-empty associated structure.

It can easily be seen that the data structure has size $O(h \log h)$ and can be constructed in $O(h \log^2 h)$ time (even $O(h \log h)$ construction time is possible).

Circular queries. Suppose we wish to perform a query on the vertices $p_i, \ldots, p_{j-1}$ with a circle C and starting point q. Since $p_i, \ldots, p_{j-1}$ is part of the cyclic sequence $p_1, \ldots, p_h$ of $\mathsf{CH}(P)$, it may be that p_h and p_1 are in the sequence $p_i, \ldots, p_{j-1}$. In that case, instead of querying with $p_i, \ldots, p_{j-1}$, we query with the two sequences $p_i, \ldots, p_h$ and $p_1, \ldots, p_{j-1}$ instead. For ease of description we assume that p_h and p_1 are not part of $p_i, \ldots, p_{j-1}$, so $j - 1 \geq i$.

Let c be the center of C; in our algorithm, c will always be the convex hull vertex p_j. We first test if p_i is farther than $2r$ from p_j: if so, we binary search for the first vertex $p_{i'}$ with $i < i' \leq j$ that is at distance at most $2r$ from p_j and set $i = i'$. Now we are guaranteed that p_i is within distance $2r$ from p_j. Points farther than $2r$ from p_j cannot be answers to the query.

We find the subsequences containing p_i and p_{j-1}; they may be the same, they may be adjacent, or there may be $O(1)$ subsequences in between. It cannot be more because that would violate the greedy choices of the subsequences.

If p_i and p_{j-1} lie in the same subsequence $p_s, \ldots, p_t$, we use the search paths in $\mathcal{T}_{st}$ to p_i and to p_{j-1} and find the highest subtrees between these paths plus the leaves containing

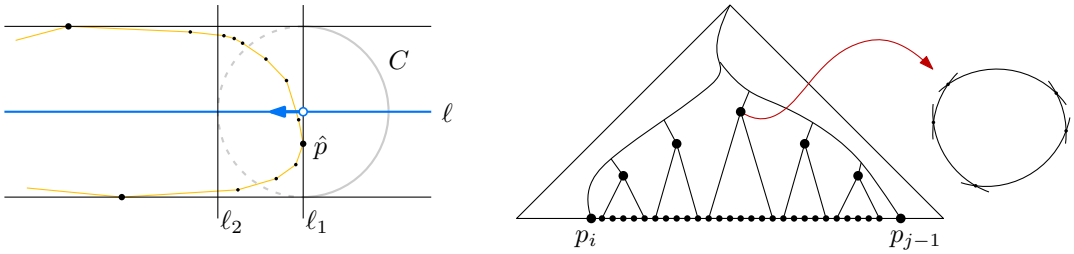

Figure 20 Left, the situation when the leftmost right cap is computed. Right, a query in the data structure shown schematically, with a set of highest subtrees between the search paths to p_i and p_{j-1}.

p_i and p_{j-1}, see Figure 20 (right). At the roots of these subtrees, we query the associated structure as described before. At the leaves, we check the one circle.

If p_i and p_{j-1} lie in two adjacent subsequences, we follow the path to p_i in the tree that contains it, and collect the highest nodes strictly right of the search path, where we query the associated structures of their roots. Then we follow the path to p_{j-1} in the tree for the other subsequence and collect the highest nodes strictly left of the search path, where we query the associated structures of their roots. Again we include the leaves with p_i and with p_{j-1}.

If there were subsequences in between, we additionally query the associated structure of the root of these trees.

In all cases, we choose the best answer (first hit) among the answers from all of the $O(1)$ subsequences and all of the $O(\log h)$ subtrees.

▶ **Lemma 20.** *A circular query in $\mathcal{T}$ can be answered in $O(\log^2 h)$ time.*

Line queries. When at a certain angle α a solution starts to exist again (type 3 event), we must find this first solution so that we can continue the rotating calipers process. Since a solution exists at angle α, the smallest enclosing strip of P must have width at least $2r$. As such, there must be two parallel lines that enclose all points of $\mathsf{CH}(P)$. Additionally, there is one point on each line, and we need to find the caps of the hippodrome: the semi-circles of radius r with their endpoints on the two lines, such that all points of P lie in the hippodrome, and the semi-circles are shifted inwards as much as possible (until they hit a point of P). For ease of description we assume that $\alpha = 0$, so the parallel lines are horizontal. We consider where to place the cap that bounds the hippodrome from the right, moving it as far left as possible.

We first find the rightmost point in $\mathsf{CH}(P)$; let it be $\hat{p}$ and let its x-coordinate be $\hat{p}_x$; see Figure 20 for an illustration. Let $\ell_1 : x = \hat{p}_x$ be the vertical line through the rightmost point, and let $\ell_2 : x = \hat{p}_x - r$. Let ℓ be the horizontal line midway between the two parallel lines that enclose the points. We need to find a point q on ℓ, such that the radius-r circle centered at q has all points of P to the left of (or on) its right semi-circle, and that right semi-circle contains one point of P: the answer to the query. Note that q will lie between ℓ_2 and ℓ_1. Consequently, the answer to the query is a point to the right of ℓ_2. Therefore, we need at most four subsequences from the partition of $\mathsf{CH}(P)$, and in the following, we simply query each of the corresponding trees to find the answer. For simplicity we refer to any such tree as $\mathcal{T}$.

We will find our answer using three separate queries. The first involves all points of P that lie inside C, the radius-r circle that has the parallel lines as tangents and whose

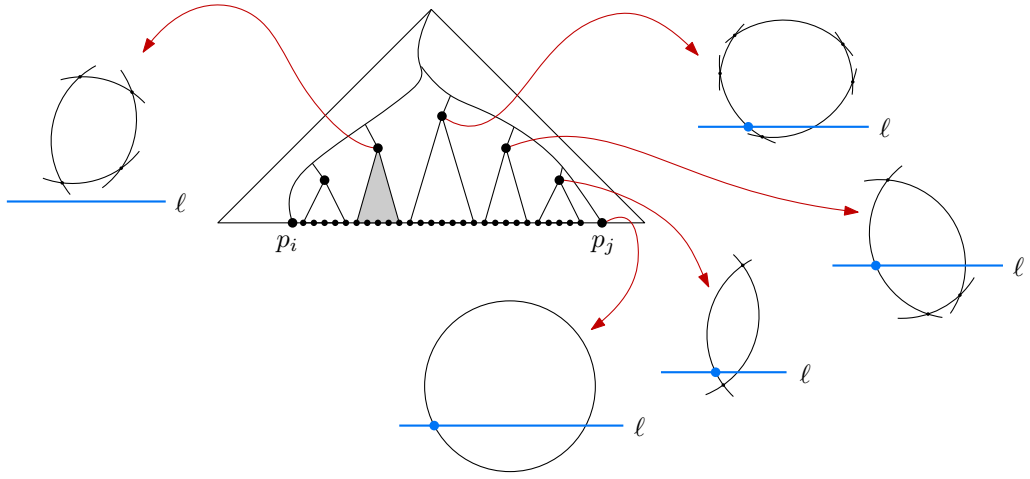

Figure 21 The query in $\mathcal{T}$ among $p_i, \ldots, p_j$ with line ℓ. In the associated structures, from right to left, we find a non-empty intersection four times (blue points are answers in these subtrees), and for the fifth node, ℓ does not intersect the common intersection: we must descend into the grey subtree.

center is $\ell \cap \ell_1$. Point $\hat{p}$ is one of the points inside C; in general the points inside C form a subsequence of $\mathsf{CH}(P)$ that includes $\hat{p}$. If any points beyond the two points on the parallel horizontal lines lie inside C, we ignore them since they are not the desired answer (this may happen when lie to the right of ℓ_2).

We can find the two relevant intersections of $\mathsf{CH}(P)$ and C in $O(\log h)$ time by binary search. For this sequence $p_i, \ldots, \hat{p}, \ldots, p_j$ of points, we know that the circles centered at them all contain the center of C. So, a line intersection query with ℓ in their common intersection gives two intersection points. The left one is the answer we want, and we report the point whose left side of the circle was intersected rightmost. We query the data structure $\mathcal{T}$ with p_i, p_j, and ℓ for this. As before, we in fact query multiple trees and subtrees, and we report the rightmost one of the different answers.

The other two queries are handled in the same way, so we describe only one of them. It concerns the points of $\mathsf{CH}(P)$ that lie outside C, to the right of ℓ_2, and before $\hat{p}$ in the clockwise order. Let the sequence of these points be $p_i, \ldots, p_j$ (overloading notation somewhat). We can find p_i and p_j by binary search on $\mathsf{CH}(P)$ using ℓ_2 and C. The sequence $p_i, \ldots, p_j$ has useful properties, when considering the radius-r disks $D_i, \ldots, D_j$ centered on them. All of these disks intersect the line ℓ; let these intervals be $I_i, \ldots, I_j$. The length of intersection increases monotonically from i to j. The right endpoints of the intervals appear in the order $i, \ldots, j$: the right endpoint of I_l is always left of the right endpoint of I_{l+1}. The left endpoints are not sorted.

The common intersection $\bigcap_{i \le l \le j} D_l$ is not empty, but this common intersection may not intersect ℓ. If it does intersect ℓ, then we can simply find the two intersection points and the left one is our answer (the point whose disk contributes to the common intersection with the circular arc that is hit). We can find it using the data structure $\mathcal{T}$ by querying with p_i, p_j, and ℓ.

When $\bigcap D_l \cap \ell = \emptyset$, then this does not work. Fortunately, we can still use $\mathcal{T}$, but the query algorithm becomes more involved. For $i \le i' \le j' \le j$, let $\bigcap_{i'j'}$ denote the common intersection of the disks $D_{i'}, \ldots, D_{j'}$.

We query $\mathcal{T}$ with i, j, and ℓ. The query with i and j results in $O(\log h)$ subtrees that together represent all points $p_i, \ldots, p_j$, see Figure 20. Each root of such a subtree stores the

non-empty common intersection of the disks corresponding to the points in the leaves of that subtree, as stated in the description of $\mathcal{T}$. We treat these subtrees from higher indices towards the ones with lower indices, so first the subtree for which p_j is the rightmost point. See Figure 21. Suppose we have treated some subtrees and now we need to handle a subtree $T_{i'j'}$ containing the points $p_{i'}, \ldots, p_{j'}$, where $i \leq i' \leq j' \leq j$. The root of this subtree stores $\bigcap_{i'j'}$. We compute $\bigcap_{i'j'} \cap \ell$ in $O(\log h)$ time. Now there are two options: (1) If $\bigcap_{i'j'} \cap \ell = \emptyset$, then the answer to the original query cannot be in $p_i, \ldots, p_{i'-1}$, so we do not need to query any more subtrees that store points with smaller indices than the present one. However, the answer may be in this subtree itself, so we must descend in it. (2) If $\bigcap_{i'j'} \cap \ell \neq \emptyset$, then the answer among $p_{i'}, \ldots, p_{j'}$ is the point whose disk provides the arc of the left intersection point of $\bigcap_{i'j'} \cap \ell = \emptyset$, which implies that we do not need to descend in this subtree. We already have the answer.

So for the selected subtrees from right to left, we first get a sequence with a non-empty intersection of their common intersection $\bigcap$ with ℓ, and then one where the intersection is empty, as illustrated in Figure 21 (we ignore subtrees further to the left). We need to search in this subtree, and do that as follows. We first query the right subtree of the root, and again perform the test whether the common intersection in the right subtree intersected with ℓ is empty or not. If it is empty, then we can ignore the left subtree and by the same argument as before, and we continue the same way in the right subtree until we reach a leaf.

The correctness of this search hinges on the claim that if a subtree gives an empty intersection with ℓ, then no subtree more to the left—with lower indices—can give the answer. The argument is simple, when thinking about the intervals $I_i, \ldots, I_j$. If the intersection for a subtree is empty, we necessarily have a left endpoint among the sequence of right endpoints in this subtree. This left endpoint gives a candidate answer. So, intervals more to the left for which we still need to encounter the right endpoint (reasoning still from right to left) certainly cannot yield a better left endpoint than the one that we know to exist.

The query visits $O(\log h)$ nodes in $\mathcal{T}$, namely the nodes of up to two search paths to leaves and their direct children next to these paths. At every node we spend $O(\log h)$ time for the intersection query of the common intersection with ℓ.

▶ **Lemma 21.** *A line query in $\mathcal{T}$ can be answered in $O(\log^2 h)$ time.*

After analyzing all events that occur during the rotational sweep, we can prove Theorem 5.

▶ **Theorem 5.** *Given a point set P consisting of n points and a radius r, we can find the shortest representative segment in $O(n \log h + h \log^3 h)$ time, where $|\mathsf{CH}(P)| = h$.*

Proof. We initialize the algorithm by computing the convex hull $\mathsf{CH}(P)$. By Lemmata 1, it is sufficient to consider only points in $\mathsf{CH}(P)$ to find a representative segment of P. We use rotating calipers to check that the shortest representative segment is not a point, and find an orientation α in which a solution exists. For this fixed α we find τ_1 and τ_2, compute S_1 and S_2, as well as the shortest line segment q_1q_2 in orientation α. Computing the convex hull can be done in $O(n \log h)$ [5]. By Lemmata 3 and 4, S_1 and S_2 can be initialized in $O(h)$, and we can initialize q_1q_2 in $O(\log h)$ time. As such, initialization of the algorithm can be done in $O(n \log h)$ time in total.

Next, we rotate orientation α over π in total, maintaining τ_1, τ_2, S_1, and S_2, as well as q_1q_2. Note that a rotation of π is sufficient, since we consider orientations, which identify opposite directions of a 2π rotation. Throughout the rotation we maintain the shortest representative segment, and return the shortest such line segment found over all orientations. Over the full rotation, we encounter six different types of events. By Lemmata 7–17, these events can be handled in $O(h \log^3 h)$ time in total.

As mentioned in Section 2.1, the shortest representative segment is defined by only τ_1, τ_2, S_1, and S_2. Each tangent is defined by a circle. Hence, τ_1 and τ_2 can only change when they are defined by a new circle. Thus, by Lemma 17, we correctly maintain τ_1 and τ_2 during the full rotation of π radians. Furthermore, S_1 and S_2 can only exist when τ_1 and τ_2 appear in the correct order. By Lemma 15 we correctly maintain when S_1 and S_2 exist. Finally, S_1 and S_2 can only make a discrete change as the tangents τ_1 and τ_2 hit intersections between circles in $\mathcal{C}_P$. If the tangents do not touch a vertex, then S_1 and S_2 must change continuously along the arcs that τ_1 and τ_2 cross. By Lemmata 11 and 13, we correctly maintain S_1 and S_2 when they exist. In conclusion, we correctly maintain τ_1, τ_2, S_1, and S_2 during the full rotation π radians. We also maintain the placement of the shortest line segment between S_1 and S_2 in any orientation using Lemmata 7 and 9. As such, we can return the shortest representative segment over all orientations at the end of our rotational sweep. ◀

3 Stable Representative Segments for Moving Points

In this section, we consider maintaining representative segment q_1q_2 while the points in P move. We first show what happens if we would maintain the optimal solution explicitly under continuous motion of the points.

There are examples where a representative segment exists for a value of r for an arbitrarily short duration. This can happen because one point moves towards a hippodrome shape and another point moves out of it, giving a brief moment with a valid hippodrome. The same effect can be caused by a single linearly moving point that grazes the hippodrome at a join point. These examples are illustrated in Figure 22(a). It can also happen that a minor oscillating movement of a point causes a quick sequence of changes between a valid and no valid segment, see Figure 22(b).

Now, consider the point set P in which the points form a regular k-gon (see Figure 22(c)). When r is equal to half the width of the k-gon, then we can force a discrete change in the placement of q_1 and q_2, with very slow movement of points in P. In this case there is always a valid representative segment, but its endpoints make a jump.

We see that continuously maintaining the optimal solution has two artifacts that are undesirable to a human observer: (1) the segment can appear and disappear frequently within an arbitrarily short time frame, and (2) a segment may jump to a new location, leading to infinitely high speeds of the endpoints even if the points themselves move slowly.

Our goal is therefore to ensure that the segment movement is *stable* over time: We want q_1 and q_2 to move continuously and with bounded speed, preventing discrete or (near) instantaneous changes. We accomplish this as follows. First, we will sample the positions of

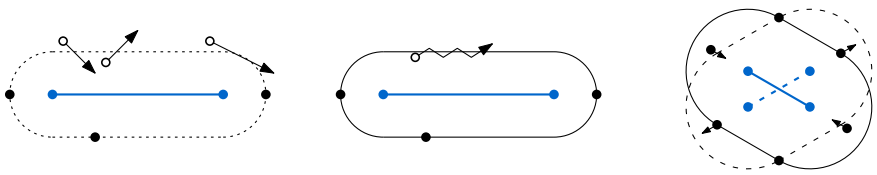

Figure 22 Examples of representative segments (blue) for moving points. **(a)** shows that a representative segment may flicker arbitrarily briefly due to the motion of two points (top left) or one point (top right). **(b)** shows that the representative segment may flicker due to minor oscillations in a movement path. **(c)** Small movements can trigger a discrete change of the representative segment.

the moving points only at integer moments, and then decide immediately to show or not show a solution in the next time unit. This implies that existence and non-existence of a solution lasts at least one time unit, and we avoid the solution (dis)appearing frequently. Second, we make the assumption that the maximum speed of the points is unit. We have to make some bounded speed assumption otherwise we cannot hope to get a bounded speed of the endpoints either. Third, we allow more flexibility in when we have a solution. We do this using a less strict regime on r, and by assuming that $r \geq 1$. Whenever the real solution exists (with the actual r), then we guarantee that we also have a solution. Whenever there is no solution even for a radius of $2\sqrt{2} \cdot r + 4$, then we never give a solution. When the radius is in between these bounds, we may have a solution or not. Our algorithm can then ensure that the speeds of the endpoints are bounded, and the length of our chosen segment always approximates the true optimum (when it exists), at any moment in time, also between the integer sampling moments.

We assume that a point $p \in P$ is described by a trajectory that is sampled at integer timestamps. That is, each point in P is described by $p(i) \to \mathbb{R}^2$ where $i \in \mathbb{Z}$ is the timestamp. In between these integer sampling moments we do not have any knowledge of the movement of the points. We also assign each endpoint of the representative segment a position as a function of $t \in \mathbb{R}$, which means that the representative segment at time t is now defined by $q_1(t)q_2(t)$. Furthermore, we use $D(t)$ and $W(t)$ as the *diameter* and *width* of the point set, which are respectively the maximum pairwise distance of points in P and the width of the thinnest strip containing P. Let $d_1(t), d_2(t) \in P$ be a pair of points defining the diameter. Lastly, we also use the *extent* $E(t)$ of P in the direction orthogonal to the line segment $d_1(t)d_2(t)$. For all of the above definitions, we omit the dependence on t when it is clear from the context.

In the following, we describe how to specify $q_1(t)$ and $q_2(t)$ such that they move with bounded speed, and such that the length of the segment $q_1(t)q_2(t)$ as well as the proximity of the segment to P can be bounded at any time t. In particular, we define such a segment $q_1(t)q_2(t)$ as an *approximating* segment, and prove that the length of an optimal representative segment is approximated by an additive term l, and at the same time the maximum distance from any point in P to the segment $q_1(t)q_2(t)$ is at most $h \cdot r$, for some constant h.

Bounding the approximation factor *and* achieving stability simultaneously is non-trivial. Since the optimal solution often exhibits discrete changes, stable solutions are often achieved by choosing a suitable canonical solution [15, 35].

Algorithm. Our algorithm $A(t)$ is *state-aware*. This means that the output of $A(t)$ is dependent only on the input at or before time t, but it has no knowledge of the input after time t. At every integer timestamp $i \in \mathbb{Z}$, we compute a *canonical solution* $q'_1(i)q'_2(i)$. The endpoints $q'_1(i)$ and $q'_2(i)$ of this canonical solution are placed on the lines orthogonal to the diametric line through $d_1(i)$ and $d_2(i)$, respectively, such that $q'_1(i)q'_2(i)$ lies in the middle of

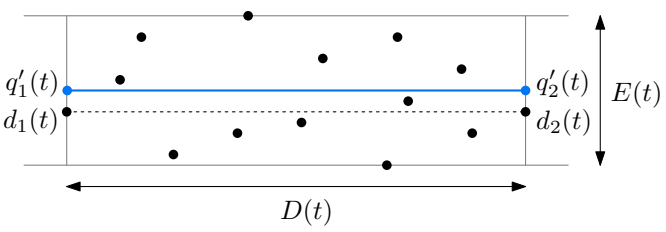

Figure 23 The prospective segment $q'_1q'_2$ in relation to D, E, and diametrical line d_1d_2 at time t.

the narrowest strip containing P in the diametric orientation, see Figure 23.

Our algorithm is now a special kind of state-aware algorithm called a *chasing algorithm* [29]: At every integer time step $i \in \mathbb{Z}$, the algorithm computes the canonical solution $q_1'(i)q_2'(i)$ and then linearly interpolates from $q_1'(i-1)q_2'(i-1)$ to $q_1'(i)q_2'(i)$, arriving there at time $i+1$. At that point, $q_1'(i+1)q_2'(i+1)$ can be computed, and we continue in a similar manner.

However, if the maximum distance from P to the canonical solution becomes too large, we no longer want the algorithm to output any solution. In this case, no (optimal) representative solution exists, and we do not produce an approximating segment either.

Formally, for any timestamp $t \in (i, i+1)$, we linearly interpolate q_1 and q_2 between their previous canonical placements as follows. We define $\alpha = (t-i)$ and set $q_j(t) = (1-\alpha)\cdot q_j'(i-1) + \alpha \cdot q_j'(i)$, for $j \in \{1,2\}$. Then, the output of our algorithm is $A(t) = q_1(t)q_2(t)$ if $E(\lfloor t \rfloor) \leq 2r\sqrt{2}+2$ and $\varnothing$ otherwise.

The above algorithm yields a number of bounds, on the values of $W(t)$ for which algorithm A outputs q_1q_2 or $\varnothing$, on the speed of the endpoints q_1 and q_2, on the length of q_1q_2, and on the distance to P. We prove these bounds in Lemmata 23–27. First, we relate the extent E of P to its width W in the following lemma.

▶ **Lemma 22.** *At any time t, $W(t) \leq E(t) \leq 2 \cdot W(t)$.*

Proof. Consider a diametrical pair $d_1, d_2 \in P$ at some time t. The line segment d_1d_2 has length D, and the extent orthogonal to the line segment has length E. First observe that the width W of P at time t is at most E, as the thinnest strip in the orientation of d_1d_2 has width E. To complete the proof, we show an upper bound of 2 on the ratio E/W. Note that this ratio is maximized when W is as low as possible with respect to E.

To prove the upper bound, we first consider the case where P is rather symmetric, that is, when d_1d_2 is centered in the thinnest strip in the orientation of d_1d_2, and the two points defining the extent in the perpendicular direction are placed symmetrically in the diametral box. Later we show that in other cases the proven upper bound cannot be exceeded. For this symmetric case, see Figure 24(a). Since the strip in the orientation of d_1d_2 cannot be thinner, there must be at least one point located on each line bordering the strip. Since the diameter of P is D, no two points can be further than D apart. This limits the position of the points p_l, p_r on the boundaries of the strip, with respect to d_1, d_2 and each other. To minimize the width of P, we place p_r closer to d_1 and p_l closer to d_2, symmetrically, such that the line segment p_lp_r also has length D. As a result, d_1d_2 and p_lp_r bisect each other and the angle α between d_1d_2 and p_lp_r is equal to $\arcsin \frac{E}{D}$.

To maximize E/W, therefore, we can combine the observations that $E/D = \sin(\alpha)$ and $W = D \cdot \sin(\alpha/2)$ to obtain $E/W = \sin(\alpha)/\sin(\alpha/2)$. The slope of $\sin(\alpha)$ at $\alpha = 0$ is 1, which is the maximum slope of $\sin(\alpha)$. As such, the value $E/W = \sin(\alpha)/\sin(\alpha/2)$ is maximized when $\alpha \to 0$, where $\lim_{\alpha \to 0}(\sin(\alpha)/\sin(\alpha/2)) = 2$.

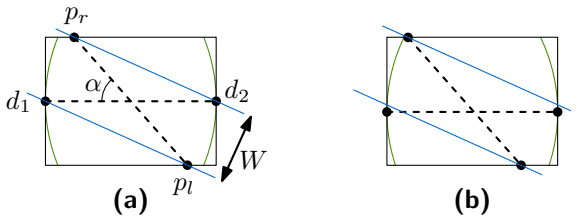

Figure 24 Constructions for Lemma 22. Segments d_1d_2 and p_lp_r are **(a)** symmetric or **(b)** off-center with respect to this bounding box. The (blue) thinnest strip in **(a)** is also shown in **(b)**.

Finally, consider a non-symmetric case: d_1d_2 is not centered in the strip of width E and/or p_l and p_r are not placed symmetrically. Still d_1 and d_2 are horizontally aligned, to be exactly distance D apart, and by definition p_r is closer to d_1, while p_l is closer to d_2 (see Figure 24(b)). To minimize the width, p_l and p_r are also as far apart as possible, and hence exactly at distance D from each other. Since d_1d_2 and p_lp_r no longer bisect each other, the strip of width W in the symmetric case can no longer contain all the points: If the strip is centered on p_lp_r, then either d_1 or d_2 will be outside the strip, as in Figure 24(b), and an analogous reasoning holds for d_1d_2. Finally, the strip of the symmetric case fits both d_1d_2 and p_lp_r individually, but a strip in any other direction will have to accommodate one of both line pieces in a more orthogonal orientation, and thus has to be wider. ◀

We can now prove for which values of $W(t)$ algorithm A is guaranteed to output q_1q_2, and when it is guaranteed to output $\varnothing$.

▶ **Lemma 23.** *At any time t, if $W(t) \leq r\sqrt{2}$ then $A(t) = q_1(t)q_2(t)$.*

Proof. Let $t \in [i, i+1]$ for some $i \in \mathbb{Z}$. For the sake of contradiction, assume $W(t) \leq r\sqrt{2}$ but $A(t) = \varnothing$. By definition, this means that $E(\lfloor t \rfloor) > 2r\sqrt{2} + 2$. Since the points in P have at most unit speed, observe that E can change by at most 2 per time unit. As such, we get

$$2r\sqrt{2} < E(\lfloor t \rfloor) - 2 \leq E(t)$$

But then, by Lemma 22, we have

$$2r\sqrt{2} < E(t) \leq 2 \cdot W(t)$$

which implies $r\sqrt{2} < W(t)$. We have reached a contradiction, and the lemma follows. ◀

▶ **Lemma 24.** *At any time t, if $W(t) > 2r\sqrt{2} + 4$ then $A(t) = \varnothing$.*

Proof. Let $t \in [i, i+1]$ for some $i \in \mathbb{Z}$. For the sake of contradiction, assume $W(t) > 2r\sqrt{2}+4$ but $A(t) = q_1(t)q_2(t)$. By definition, this means that $E(\lfloor t \rfloor) \leq 2r\sqrt{2} + 2$. Since the points in P have at most unit speed, E can change by at most 2 per time unit. As such, we get

$$E(t) \leq E(\lfloor t \rfloor) + 2 \leq 2r\sqrt{2} + 4$$

But then, by Lemma 22, we have

$$W(t) \leq E(t) \leq 2r\sqrt{2} + 4$$

We have reached a contradiction, and the lemma follows. ◀

Next, we prove an upper bound on the speed on q_1 and q_2, as well as on the length of q_1q_2 and its distance to P.

▶ **Lemma 25.** *While $A(t) = q_1(t)q_2(t)$, endpoints q_1 and q_2 move continuously with a speed bounded by $(2r+1)\sqrt{2} + 2$.*

Proof. To bound the speed with which q_1 (and symmetrically q_2) moves, we bound the distance between $q_1'(i)$ and $q_1'(i+1)$, for $i \in \mathbb{Z}$ (see Figure 25(a)). First observe that the diametric pair of points in P determining the orientation of $q_1'(i)q_2'(i)$ may not be the diametric pair at time $i+1$. The points defining this new diametric pair can therefore be located anywhere in the width E bounding box in the orientation of $q_1'(i)q_2'(i)$ at time i. In a single time step, all point can move at most one unit outside this box, and we then

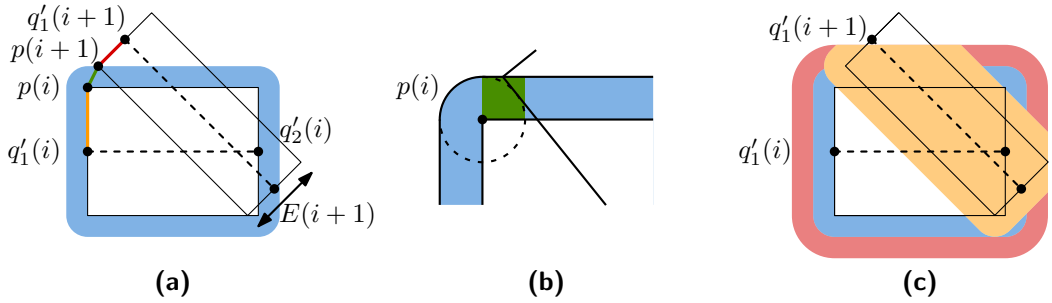

Figure 25 Constructions for Lemmata 25 and 27. **(a)** All points are inside the (horizontal) bounding box at time i and cannot have moved outside the blue border at time $i+1$. We bound the length of the three colored line pieces to bound the speed of q_1. **(b)** When a point p on the minor axis of the horizontal box reaches the major axis of the box at $i+1$, the minor axis of the same box must touch the green square. **(c)** At time $i+2$ all points cannot have moved outside the width-2 red border around the box at time i, and at most one unit outside the box at time $i+1$ (yellow).

consider a new canonical solution $q_1'(i+1)q_2'(i+1)$ in the orientation of the new diametric pair. We bound the distance between $q_1'(i)$ and $q_1'(i+1)$ by half the lengths of the minor axes of the bounding boxes in the orientation of the respective diametric pairs (remember, $q_1'(i)$ and $q_1'(i+1)$ are located in the middle of the respective minor axes) plus the distance between the minor axes of the bounding boxes in the orientation of $q_1'(i)q_2'(i)$ and $q_1'(i+1)q_2'(i+1)$.

Let p be a point at time i that is located on the minor axis of the time-i bounding box, and consider the bounding box at time $i+1$ (in the direction of $q_1'(i+1)q_2'(i+1)$). A point must be located on each axis of the bounding box at time $t+1$, as otherwise it would not be a bounding box. Thus, each axis must have at least one point at distance at most one from the bounding box at time t. All points lie in this bounding box at time $i+1$, so the point p has to move at least to the boundary of the box at time $t+1$. If it reaches the minor axis, the distance between the minor axes is one, so assume p reaches only the major axis of the box at time $t+1$ (see Figure 25(b)). Since the minor axis at time $t+1$ has to be at distance at most one from the boundary of the box at time t, it can be at distance at most $\sqrt{2}$ from p.

Finally, we add up all the distances. When both $q_1'(i)$ and $q_1'(i+1)$ are used as outputs of our algorithm, neither $A(i+1)$ nor $A(i+2)$ is $\varnothing$. Thus, the minor axes are upper bounded by $E \leq 2r\sqrt{2}+2$, and the distance between $q_1'(i)$ and $q_1'(i+1)$ is upper bounded by $2 * \frac{E}{2} + \sqrt{2} = (2r+1)\sqrt{2}+2$. ◀

▶ **Lemma 26.** *At any time t, if there exists a representative segment of P with radius r and $A(t) = q_1(t)q_2(t)$ then $|q_1(t)q_2(t)| \leq \mathsf{OPT}(t) + 2r + 4$, where $\mathsf{OPT}(t)$ is the length of the shortest representative segment of P with radius r at time t.*

Proof. We first argue that the canonical solution at time $i \in \mathbb{Z}$ has length at most $\mathsf{OPT}(i) + 2r + 2$. Observe that the length of $q_1'(i)q_2'(i)$ is $D(i)$. The shortest possible segment that is at distance at most r from both diametrical points $d_1(i)$ and $d_2(i)$ has length $D(i) - 2r$, and has its each endpoint at distance exactly r from one of the diametrical points. See Figure 26 for an example. Since both diametrical points are in P, the optimal solution therefore has length at least $D(i) - 2r$. Thus, we get

$$|q_1'(i)q_2'(i)| = D(i) \leq \mathsf{OPT}(i) + 2r$$

For $t \in (i, i+1]$, since points in P move with at most unit speed, observe that $|q_1(t)q_2(t)| \leq$

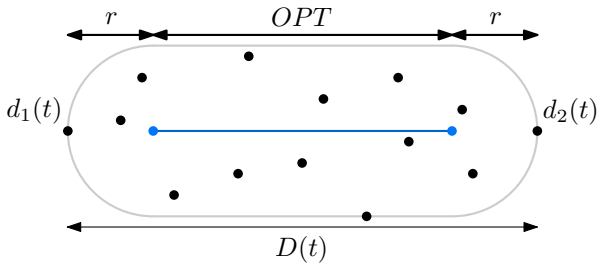

Figure 26 An example of the shortest possible representative segment.

$|q_1'(i)q_2'(i)| + 2$. In addition, OPT can also change at most 2 per time unit. As such, we get

$$|q_1(t)q_2(t)| \leq |q_1'(i)q_2'(i)| + 2 \leq \mathsf{OPT}(i) + 2r + 2 \leq \mathsf{OPT}(t) + 2r + 4.$$ ◀

▶ **Lemma 27.** *While $A(t) = q_1(t)q_2(t)$, the distance from p to q_1q_2 is at most $r\sqrt{2}+3$ for all $p \in P$.*

Proof. To bound the maximum distance between any point p and the line segment q_1q_2 at any point in time, recall that algorithm A interpolates to the canonical solution of the time step before. So consider the canonical solutions at times i and $i+1$ and corresponding bounding boxes in the orientation of $q_1'(i)q_2'(i)$ and $q_1'(i+1)q_2'(i+1)$, respectively (see Figure 25c).

At time $i+1$ the canonical solution of time i is reached, and the points can have moved at most one unit outside the bounding box of time i, and at time $i+2$ the points can have moved one unit further outside. Now consider any $t \in [i+1, i+2]$, in which A interpolated from the canonical solution at time i to the canonical solution at time $i+1$. For any such t a point p is at distance at most $E(i)/2 + 2 \leq r\sqrt{2}+3$ from the canonical solution at time i, which is equal to $A(i+1) = q_1(i+1)q_2(i+1)$.

Additionally, at time $i+1$ all points must be inside the bounding box in the orientation of the canonical solution at time $i+1$. At time $i+2$, the points hence lie at most $E(i+1)/2+1 \leq r\sqrt{2}+2$ away from the canonical solution at time $i+1$, which is $A(i+2) = q_1(i+2)q_2(i+2)$.

Finally, since the points move at unit speed, by linearly interpolating between $q_1(i+1)q_2(i+1)$ and $q_1(i+2)q_2(i+2)$ we ensure that no point is more than $r\sqrt{2}+3$ removed from q_1q_2 at any $t \in [i+1, i+2]$. ◀

We can now combine Lemmata 23–27 to get the following theorem.

▶ **Theorem 28.** *Given a set P of points moving with at most unit speed, algorithm A yields a stable approximating segment with additive length term $l = 2r+4$ and multiplicative distance term $h = \sqrt{2}+3/r$, for which speed of the endpoints is bounded by $(2r+1)\sqrt{2}+2$.*

Since all calculations other than computing the diametral pair can be done in $O(1)$ time, a naive approach to implement our algorithm would simply be to recompute the diametral pair every integer time step, which would result in an $O(n)$ running time per integer time step. However, it is also possible to maintain the diametral pair kinetically using the *Kinetic Data Structure* (KDS) for convex hulls. This results in a KDS that is responsive, compact, and efficient [23].

4 Conclusion

In this paper, we presented an $O(n \log h + h \log^3 h)$ time algorithm to find the shortest representative segment of a point set, improving the previous $O(n \log h + h^{1+\varepsilon})$ time solution. Additionally, we showed how to maintain an approximation of the shortest representative segment in a stable manner, such that its endpoints move with a speed bounded by a linear function in r.

There may be possibilities for improving the running time of our static solution to $O(n \log h + h \log^2 h)$, or even $O(n \log h)$. The $O(h \log^3 h)$ term comes from having to handle $O(h \log h)$ type 2b events in $O(\log^2 h)$ time each. However, it may be possible to show that there are at most $O(h)$ type 2b events, since the conjugate pairs used to bound the number of internal events each have a unique starting point. Additionally, it may be possible to improve the query time of the data structure described in Section 2.4 to $O(\log h)$ time using ideas like fractional cascading, but there is no straightforward way to make this work for the circular query.

References

1 Pankaj K. Agarwal, Alon Efrat, Micha Sharir, and Sivan Toledo. Computing a segment center for a planar point set. *Journal of Algorithms*, 15(2):314–323, 1993. doi:10.1006/jagm.1993.1043.

2 Julien Basch. *Kinetic data structures*. PhD thesis, Stanford University, 1999.

3 Ken Been, Martin Nöllenburg, Sheung-Hung Poon, and Alexander Wolff. Optimizing active ranges for consistent dynamic map labeling. *Computational Geometry: Theory and Applications*, 43(3):312–328, 2010. doi:10.1016/j.comgeo.2009.03.006.

4 Sergei Bespamyatnikh, Binay Bhattacharya, David Kirkpatrick, and Michael Segal. Mobile facility location. In *Proc. 4th Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIALM)*, pages 46–53, 2000. doi:10.1145/345848.345858.

5 Timothy M. Chan. Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discrete & Computational Geometry*, 16(4):361–368, 1996. doi:10.1007/BF02712873.

6 Timothy M. Chan, Thomas C. van Dijk, Krzysztof Fleszar, Joachim Spoerhase, and Alexander Wolff. Stabbing rectangles by line segments–how decomposition reduces the shallow-cell complexity. In *Proc. 29th International Symposium on Algorithms and Computation (ISAAC)*, pages 61:1–61:13, 2018. doi:10.4230/LIPICS.ISAAC.2018.61.

7 Danny Z. Chen and Haitao Wang. Approximating points by a piecewise linear function. *Algorithmica*, 66(3):682–713, 2013. doi:10.1007/s00453-012-9658-y.

8 Merce Claverol, Elena Khramtcova, Evanthia Papadopoulou, Maria Saumell, and Carlos Seara. Stabbing circles for sets of segments in the plane. *Algorithmica*, 80:849–884, 2018. doi:10.1007/s00453-017-0299-z.

9 Mark de Berg and Dirk H. P. Gerrits. Labeling moving points with a trade-off between label speed and label overlap. In *Proc. 21st European Symposium on Algorithms (ESA)*, pages 373–384, 2013. doi:10.1007/978-3-642-40450-4_32.

10 Mark de Berg, Marcel Roeloffzen, and Bettina Speckmann. Kinetic 2-centers in the black-box model. In *Proc. 29th Symposium on Computational Geometry (SoCG)*, pages 145–154, 2013. doi:10.1145/2462356.2462393.

11 José Miguel Díaz-Báñez, Matias Korman, Pablo Pérez-Lantero, Alexander Pilz, Carlos Seara, and Rodrigo I. Silveira. New results on stabbing segments with a polygon. *Computational Geometry*, 48(1):14–29, 2015. doi:10.1016/j.comgeo.2014.06.002.

12 Darko Dimitrov, Christian Knauer, Klaus Kriegel, and Günter Rote. Bounds on the quality of the PCA bounding boxes. *Computational Geometry*, 42(8):772–789, 2009. doi:10.1016/j.comgeo.2008.02.007.

13 Matt Duckham, Lars Kulik, Mike Worboys, and Antony Galton. Efficient generation of simple polygons for characterizing the shape of a set of points in the plane. *Pattern recognition*, 41(10):3224–3236, 2008. doi:10.1016/j.patcog.2008.03.023.

14 Stephane Durocher and David Kirkpatrick. The steiner centre of a set of points: Stability, eccentricity, and applications to mobile facility location. *International Journal of Computational Geometry & Applications*, 16(04):345–371, 2006. doi:10.1142/S0218195906002075.

15 Stephane Durocher and David Kirkpatrick. Bounded-velocity approximation of mobile Euclidean 2-centres. *International Journal of Computational Geometry & Applications*, 18(03):161–183, 2008. doi:10.1142/S021819590800257X.

16 Herbert Edelsbrunner, David Kirkpatrick, and Raimund Seidel. On the shape of a set of points in the plane. *IEEE Transactions on information theory*, 29(4):551–559, 1983. doi:10.1109/TIT.1983.1056714.

17 Herbert Edelsbrunner, Hermann A. Maurer, Franco P. Preparata, Arnold L. Rosenberg, Emo Welzl, and Derick Wood. Stabbing line segments. *BIT Numerical Mathematics*, 22:274–281, 1982. doi:10.1007/BF01934440.

18 Alon Efrat and Micha Sharir. A near-linear algorithm for the planar segment-center problem. *Discrete & Computational Geometry*, 16(3):239–257, 1996. doi:10.1007/BF02711511.

19 Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. 2nd International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 226–231, 1996. doi:10.5555/3001460.3001507.

20 Herbert Freeman and Ruth Shapira. Determining the minimum-area encasing rectangle for an arbitrary closed curve. *Communications of the ACM*, 18(7):409–413, 1975. doi:10.1145/360881.360919.

21 Andreas Gemsa, Martin Nöllenburg, and Ignaz Rutter. Sliding labels for dynamic point labeling. In *Proc. 23rd Canadian Conference on Computational Geometry (CCCG)*, pages 205–210, 2011.

22 Andreas Gemsa, Martin Nöllenburg, and Ignaz Rutter. Consistent labeling of rotating maps. *Journal of Computational Geometry*, 7(1):308–331, 2016. doi:10.20382/jocg.v7i1a15.

23 Leonidas Guibas. Kinetic data structures. In *Handbook of Data Structures and Applications*, pages 377–388. Chapman and Hall/CRC, 2018. doi:10.1201/9781315119335.

24 S. Louis Hakimi and Edward F. Schmeichel. Fitting polygonal functions to a set of points in the plane. *CVGIP: Graphical Models and Image Processing*, 53(2):132–136, 1991. doi:10.1016/1049-9652(91)90056-P.

25 John A. Hartigan and Manchek A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society.*, 28(1):100–108, 1979. doi:10.2307/2346830.

26 Hiroshi Imai, DT Lee, and Chung-Do Yang. 1-segment center problems. *ORSA Journal on Computing*, 4(4):426–434, 1992. doi:10.1287/ijoc.4.4.426.

27 Konstantin Kobylkin. Stabbing line segments with disks: complexity and approximation algorithms. In *Proc. 6th International Conference on Analysis of Images, Social Networks and Texts (AIST)*, pages 356–367, 2017. doi:10.1007/978-3-319-73013-4_33.

28 Wouter Meulemans, Bettina Speckmann, Kevin Verbeek, and Jules Wulms. A framework for algorithm stability and its application to kinetic euclidean MSTs. In *Proc. 13th Latin American Symposium on Theoretical Informatics (LATIN)*, pages 805–819, 2018. doi:10.1007/978-3-319-77404-6_58.

29 Wouter Meulemans, Kevin Verbeek, and Jules Wulms. Stability analysis of kinetic orientation-based shape descriptors. *arXiv preprint arXiv:1903.11445*, 2019.

30 Martin Nöllenburg, Valentin Polishchuk, and Mikko Sysikaski. Dynamic one-sided boundary labeling. In *Proc. 18th International Conference on Advances in Geographic Information Systems (SIGSPATIAL)*, pages 310–319, 2010. doi:10.1145/1869790.1869834.

31 Lena Marie Schlipf. *Stabbing and Covering Geometric Objects in the Plane*. PhD thesis, FU Berlin, 2014.

32 Ariana Strandburg-Peshkin, Damien R. Farine, Margaret C. Crofoot, and Iain D. Couzin. Habitat and social factors shape individual decisions and emergent group structure during baboon collective movement. *eLife*, 6:e19505, 2016. doi:10.7554/eLife.19505.

33 Godfried T. Toussaint. Solving geometric problems with the rotating calipers. In *Proc. 2nd Mediterranean Electrotechnical Conference (MELECON)*, volume 1, page A10, 1983.

34 Der Perng Wang. A new algorithm for fitting a rectilinear x-monotone curve to a set of points in the plane. *Pattern Recognition Letters*, 23(1-3):329–334, 2002. doi:10.1016/S0167-8655(01)00130-1.

35 Jules Joseph Helena Maria Wulms. *Stability of geometric algorithms*. PhD thesis, Department of Mathematics and Computer Science, TU Eindhoven, 2020.