

# Compressing spatio-temporal trajectories

Joachim Gudmundsson<sup>1</sup>, Jyrki Katajainen<sup>2,\*</sup>, Damian Merrick<sup>1,3</sup>, Cahya Ong<sup>4</sup>, and Thomas Wolle<sup>1</sup>

<sup>1</sup> National ICT Australia\*\*, Sydney, Australia

{joachim.gudmundsson, thomas.wolle, damian.merrick}@nicta.com.au

<sup>2</sup> Department of Computing, University of Copenhagen, Denmark

jyrki@diku.dk

<sup>3</sup> School of Information Technologies, University of Sydney, Australia

<sup>4</sup> Department of Engineering, University of New South Wales, Australia

**Abstract.** Trajectory data is becoming increasingly available and the size of the trajectories is getting larger. In this paper we study the problem of compressing spatio-temporal trajectories such that the most common queries can still be answered approximately after the compression step has taken place. In the process we develop an  $O(n \log^k n)$ -time implementation of the Douglas-Peucker algorithm, where  $k = 2$  or  $k = 3$  depending on the type of approximation, in the case when the polygonal path of  $n$  vertices given as input is allowed to self-intersect.

## 1 Introduction

Technological advances of location-aware devices, surveillance systems, and electronic transaction networks produce more and more opportunities to trace moving individuals. Consequently, an eclectic set of disciplines including geography [8], database research [10], animal-behaviour research [13], surveillance and security analysis [18], and transport analysis [15] shows an increasing interest in movement patterns of various entities moving in various spaces over various times scales (see also the survey by Gudmundsson et al. [9]).

Large sets of data on the movement of entities create the problem of storing, transmitting, and processing this data. Hence, simplifying this data becomes an important problem. Trajectories are similar to polygonal paths as both represent a sequence of vertices (i.e. locations) in space. Simplifying polygonal paths is a well-researched area in cartography, geographic information systems, digital image analysis, and computational geometry. However, trajectories differ from polygonal paths, because trajectories do not only contain information about a sequence of locations, but also *when* an entity has been at these locations. Therefore, simplifying trajectories differs from simplifying polygonal paths, as we might wish to preserve some temporal information. The movement of a point object  $p$  is described by a sequence of coordinates given at  $n$  time steps  $\langle (x_1, y_1, t_1), \dots, (x_n, y_n, t_n) \rangle$ , and we assume that for all  $i \neq j : (x_i, y_i) \neq (x_j, y_j)$ . The aim is to simplify the trajectory such that both spatial and temporal information is maintained.

We consider path-simplification algorithms as they might be extended or generalised to trajectory-simplification algorithms. In this paper we focus on a path-simplification algorithm that computes a simplified path containing only vertices of the original path, such that all

---

\* Part of this research was conducted when the author visited Sydney Research Laboratory at National ICT Australia. The research of this author was partially supported by the Danish Natural Science Research Council under contract 272-05-0272 (project “Generic programming—algorithms and tools”).

\*\* National ICT Australia is funded through the Australian Government’s Backing Australia’s Ability initiative, in part through the Australian Research Council.

vertices of the original path are at most  $\varepsilon$  away from the simplification. We call such a simplified path an  $\varepsilon$ -*simplification*.

Imai and Iri [14] formulated the path-simplification problem in terms of graph theory: construct a directed acyclic graph that models all possible edges in a simplification and then use breadth-first search to compute a shortest path in the graph. Their algorithm runs in  $O(n^2 \log n)$  time. Chan and Chin [5], and Melkman and O'Rourke [16] improve the running time of their algorithm to quadratic. Most of the known algorithms use  $O(n^2)$  time and space. A notable exception is the algorithm by Agarwal and Varadarajan [1] that achieves  $O(n^{4/3+\delta})$  time and space, where  $\delta > 0$  is an arbitrarily small constant. However, their algorithm only works for the  $L_1$  metric. Since the problem of developing a near-linear time algorithm for computing the optimal  $\varepsilon$ -simplification remains unsolved, several heuristics have been proposed.

The most widely used heuristic is the Douglas-Peucker method [6] (together with its variants), originally proposed for simplifying curves under the Hausdorff error measure. For a real number  $\varepsilon > 0$ , the polygonal path  $\langle v_1, \dots, v_n \rangle$  is approximated as follows. If every vertex  $v_i$ , for  $1 < i < n$ , has a distance smaller than  $\varepsilon$  to the line  $\ell$  determined by  $v_1$  and  $v_n$ , accept the line segment  $(v_1, v_n)$  as an approximation for the whole path. Otherwise, split the path at the vertex furthest from line  $\ell$  and recursively approximate the two pieces. A straightforward implementation requires  $O(n)$  time to find the point furthest from line  $\ell$ . Since the recursion depth can be linear in the worst case, it follows that the running time is bounded by  $O(n^2)$ .

A crucial aspect of simplification algorithms is the choice how the distance between a point and a line segment is measured. Originally in the Douglas-Peucker algorithm, the Euclidean distance between a point and a line is used (*line model*), where the line is defined by the corresponding line segment. This can lead to counter-intuitive simplifications as illustrated in Fig. 1. The solid black line segments form the path  $P$ , and the distance from  $p$  to the grey dotted line is much smaller than the distance from  $p$  to the solid grey line segment. In the line model,  $P$  could be simplified to be the small grey solid line segment, which might not be what we wish. That is why we also use the Euclidean distance from a point to a line segment (*line-segment model*). Apart from the difference between the line model and the line-segment model, many other distance functions are possible.

Even though the Douglas-Peucker algorithm does not output the minimum number of vertices and its worst-case running time is  $O(n^2)$ , it is often used due to its simplicity and efficiency in practice. However, in the case when the path is assumed to be non-self-intersecting, or even monotone, faster methods have been developed. Hershberger and Snoeyink [11] showed that in the line model the running time can be improved in the case when the path does not self-intersect by making use of the fact that the furthest point has to be a vertex of the convex hull of the point set. Allowing  $O(n \log n)$  preprocessing they showed how the furthest point can be found in  $O(\log n)$  time. This was later improved further to  $O(n \log^* n)$  in [12] by the same authors. We will use a similar approach with two crucial differences: the input path may self-intersect, and we consider both the line and the line-segment models.

The contribution of this paper is threefold:

1. We consider the problem of simplifying trajectories and propose a way of modelling such trajectories in 3-dimensional space which generalises the results by Cao et al. [4]. In the same paper Cao et al. argued that most spatio-temporal queries are composed of five types of queries. We show that all the queries can be approximated in our proposed model.

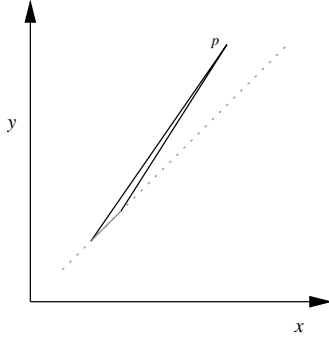
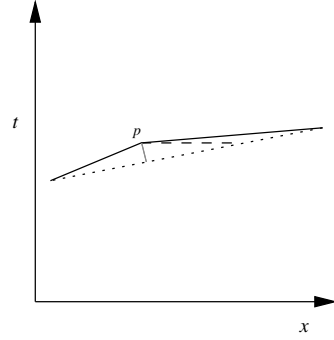


Fig. 1: An example that is bad for the line model.

Fig. 2: An example that is bad for  $E_u$ .

2. In the process we present an  $O(n \log^2 n)$ -time (line model) and an  $O(n \log^3 n)$ -time (line-segment model) implementation of the Douglas-Peucker algorithm in the plane in the case when the polygonal path can self-intersect.
3. We propose a simple algorithm that produces an approximate Douglas-Peucker path simplification in 3-dimensional space. That is, given two real values  $\varepsilon > 0$  and  $\delta > 0$ , the output of our variant of the Douglas-Peucker algorithm is a  $(1 + \delta)\varepsilon$ -simplification. The running time of our algorithm is  $O(\frac{1}{\delta^2} n \log^2 n)$  in the line model and  $O(\frac{1}{\delta^2} n \log^3 n)$  in the line-segment model.

The paper is organised as follows. In Section 2 we propose and motivate a more general model for trajectory simplification. In Section 3 we present our implementations of the Douglas-Peucker algorithm in the case when the polygonal path may self-intersect. Finally, in Section 4 we present a fast approximation algorithm for trajectory simplification using the model proposed in Section 2. Due to space constraints, many proofs have been omitted and can be found in the appendix.

## 2 Modelling trajectories

According to Cao et al. [4] most spatio-temporal queries are composed of the following five types of queries: *where-at*, *when-at*, *intersect*, *nearest-neighbour* and *spatial-join*. We state the semantics of the two most basic queries *where-at* and *when-at* on a trajectory  $T = \langle (x_1, y_1, t_1), \dots, (x_n, y_n, t_n) \rangle$  as follows. See [4] for details on the other queries.

- *where-at*( $T, t$ ) returns the location of the entity corresponding to  $T$  at time  $t$  according to  $T$ . If  $t < t_1$  or  $t > t_n$ , then the answer is undefined.
- *when-at*( $T, x, y$ ) returns the time  $t$  at which a moving object on trajectory  $T$  is expected to be at location  $(x, y)$ . If the location is not on the trajectory, then the answer is undefined.

Also the notion of soundness of distance functions is discussed in [4]. For a trajectory  $T$ , let  $q(T)$  denote the answer of some spatio-temporal query  $q$  with input  $T$ . To make the dependence on both  $\varepsilon$  and the underlying distance function *dist* explicit, we let a  $(dist, \varepsilon)$ -simplification denote a simplification that is computed using *dist*.

**Definition 1.** Let  $T$  be a trajectory and  $T'$  its  $(dist, \varepsilon)$ -simplification. The distance function *dist* is sound for query  $q$ , if for each  $\varepsilon$  there exists a bound  $\delta$ , such that  $|q(T, \cdot) - q(T', \cdot)| \leq \delta$ .

For the *where-at* query  $|q(T, t) - q(T', t)|$  refers to the Euclidean distance between the two points given as answers, and for the *when-at* query  $|q(T, x, y) - q(T', x, y)|$  means the difference between the two returned times.

Cao et al. [4] define the following distance functions between a point  $p_m$  and a line segment  $\overline{p_i p_j}$  in 3-dimensional space:  $E_2$  (2-dimensional Euclidean distance),  $E_3$  (3-dimensional Euclidean distance),  $E_u$  ( $E_u(p_m, \overline{p_i p_j}) = \sqrt{(x_m - x_c)^2 + (y_m - y_c)^2}$  where  $p_c$  is the point on  $\overline{p_i p_j}$  with  $t_m = t_c$ ) and  $E_t$  ( $E_t(p_m, \overline{p_i p_j}) = |t_m - t_c|$  where  $p_c$  is the point on the 2-dimensional projection of  $\overline{p_i p_j}$  onto the  $xy$ -plane that is closest to the 2-dimensional projection of  $p_m$  onto the  $xy$ -plane). They also show that only the distance function  $E_u$  is sound for the *where-at* query, and only the distance function  $E_t$  is sound for the *when-at* query. Hence, they propose to use a combined distance function based on  $E_u$  and  $E_t$  which is sound for both queries. This approach combines the strength of both single distances, but also their weaknesses. This combined distance function results in the worst compression ratio among the researched distance functions.

We argue that using  $E_u$  gives rise to another problem. Consider the trajectory as shown in Fig. 2, where an entity moves with high speed along the  $x$ -axis (i.e.  $y = 0$ ) and changes slightly its speed. (The effect can be amplified by repeating this pattern.) From a practical point of view we might wish to simplify this trajectory to a line segment, as we are not interested in preserving the marginal speed changes of an entity (e.g. a car) on a long line segment (e.g. a motorway). However, with the  $E_u$  distance we are unable to do so. To see this note that e.g. in Fig. 2 the shortest distance (grey line) from vertex  $p$  of the trajectory (solid line) to a point on the simplification (dotted line) is very small compared to the distance (dashed line) between  $p$  and a point on the simplification that has the same time as  $p$ , which is the distance defined by  $E_u$ .

## 2.1 Our model

As in [4], we think of a trajectory as a polygonal path in 3-dimensional space. The  $x$ - and  $y$ -dimensions correspond to the two spatial dimensions in which the entities move. The third dimension is the time  $t$ , which enables us to preserve temporal information. If we want to apply a path-simplification algorithm on such a 3-dimensional path, we need a distance measure between points (or lines or line segments) in 3-dimensional space. The two spatial dimensions have the same physical units, but the time dimension has a different unit. We choose to use the Euclidean distance in 3-dimensional space and therefore propose to use a conversion parameter  $\alpha$  that transforms time units into space units. Given a point  $p$  in 3-dimensional space, the 3-dimensional ball  $B_p$  with centre at  $p$  and radius  $\varepsilon$  contains exactly those points within distance at most  $\varepsilon$  from  $p$ . Hence, if we would like to know whether point  $p'$  is within distance  $\varepsilon$  of  $p$ , then this is the same as asking whether  $p'$  is inside  $B_p$ .

In our distance function  $dist_\alpha$ , the impact of  $\alpha$  can be seen in two different ways: either as ‘stretching’ the  $t$ -axis or as ‘flattening’ the ball  $B_p$ . In the former, we can say that the bigger  $\alpha$ , the longer the time axis (i.e. the more spatial length units that correspond to one time unit), and always consider a perfect ball  $B$  as basis for the distance between two points. In the latter, we keep the coordinate system fixed, but the bigger  $\alpha$  the flatter the ball  $B$  in the  $t$ -dimension. Formally, the distance function is defined as follows.

**Definition 2.** *The distance  $dist_\alpha$  between a point  $p_m = (x_m, y_m, t_m)$  and a line segment  $\overline{p_i p_j}$  is the shortest Euclidean distance in 3-dimensional space from  $p_m$  to a point  $p_c$  on  $\overline{p_i p_j}$  where*

1 time unit is equivalent to  $\alpha$  space units, i.e.:

$$\text{dist}_\alpha(p_m, \overline{p_i p_j}) = \sqrt{(x_m - x_c)^2 + (y_m - y_c)^2 + \alpha \cdot (t_m - t_c)^2}$$

The three distance functions  $E_2$ ,  $E_3$ , and  $E_u$  defined in [4] are special cases of our distance function, namely  $\text{dist}_0 \equiv E_2$  (where ‘ $\equiv$ ’ denotes equivalence),  $\text{dist}_1 \equiv E_3$ , and  $\text{dist}_\infty \equiv E_u$ . Choosing  $\alpha = 0$  renders the time information irrelevant, and hence it is equivalent to projecting the line segment onto the  $xy$ -plane and using the Euclidean distance on it. This distance function has the advantage that it does simplify trajectories as shown in Fig. 2, but it is not sound for the *where-at* query. The other extreme,  $\alpha \rightarrow \infty$ , denoted as  $\text{dist}_\infty$ , means the ball  $B_p$  is flattened into a 2-dimensional disk, which is parallel to the  $xy$ -plane. This means that the distance between a point  $p$  and a line segment  $\overline{p_i p_j}$  is the Euclidean distance between  $p$  and  $p'$ , where  $p'$  is the point on  $\overline{p_i p_j}$  that has the same time value. This distance function has the advantage that it is sound for the *where-at* query, but it does not simplify trajectories like the one in Fig. 2.

Apart from being more general, our approach to be able to choose  $\alpha$  has the advantage of allowing any distance function between  $\text{dist}_0 \equiv E_2$  and  $\text{dist}_\infty \equiv E_u$ . Intuitively, we can fine-tune the trade-off between ‘soundness’ and ‘sensible simplification’, and we can prove  $\text{dist}_\alpha$  to be sound for all  $\alpha$  under certain conditions. To make this more precise, we incorporate the speed of entities in our considerations, where the speed  $s_\ell$  along the line segment  $\ell$  is defined as the distance in the  $xy$ -plane divided by the time difference corresponding to  $\ell$ .

**Theorem 1.** *Let  $\ell = \overline{p_i p_j}$  be a line segment with speed  $s_\ell$  that is part of a  $(\text{dist}_\alpha, \varepsilon)$ -simplification of the trajectory  $T = \langle p_1, \dots, p_i, \dots, p_j, \dots, p_n \rangle$ , and let  $t$  be any moment of time with  $i \leq t \leq j$ . Then we have:*

$$|\text{where-at}(T, t) - \text{where-at}(\ell, t)| \leq \delta_s := \frac{\varepsilon}{\sin(\arctan \frac{\alpha}{s_\ell})}$$

*Proof.* Let  $p_m = (x_m, y_m, t_m)$  be a point on the trajectory  $T$  and  $p_k = (x_k, y_k, t_k)$  be a point on the line segment  $\ell = \overline{p_i p_j}$  with  $t = t_m = t_k$ , see Fig. 3. Let  $p_c$  be the point on the line through  $\ell$  that is closest to  $p_m$ . We know that the angle  $\angle p_h p_c p_m$  is a right angle for all  $p_h$  on  $\ell$  such that  $p_h \neq p_c$ . The spatial error is the distance between  $p_k$  and  $p_m$ :

$$|\text{where-at}(T, t) - \text{where-at}(\ell, t)| = |p_k p_m| = \frac{|p_c p_m|}{\sin(\angle p_c p_k p_m)} \leq \frac{\varepsilon}{\sin(\angle p_c p_k p_m)}$$

Let  $\gamma$  denote the angle between  $\ell = \overline{p_i p_j}$  and the  $xy$ -plane, and note that  $\gamma \leq \angle p_c p_k p_m \leq \frac{\pi}{2} - \gamma$ . Therefore,  $\sin \gamma \leq \sin(\angle p_c p_k p_m)$ . Now, note that  $\tan \gamma = \frac{\alpha \cdot (t_j - t_i)}{\sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}}$ . Since  $s_\ell = \frac{\sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}}{t_j - t_i}$ , we obtain  $\tan \gamma = \frac{\alpha}{s_\ell}$ . Altogether, we obtain:

$$\frac{\varepsilon}{\sin(\angle p_c p_k p_m)} \leq \frac{\varepsilon}{\sin \gamma} = \frac{\varepsilon}{\sin(\arctan \frac{\alpha}{s_\ell})}$$

□

The previous theorem tells us that the bigger  $\frac{\alpha}{s_\ell}$  becomes the smaller gets  $\delta_s$ . Hence, the distance function  $\text{dist}_\alpha$  is sound according to Definition 1 for the *where-at* query for any  $\alpha > 0$  as long as  $0 < s_\ell < \infty$ . However, in practice only a restricted range of values for  $\alpha$  might be

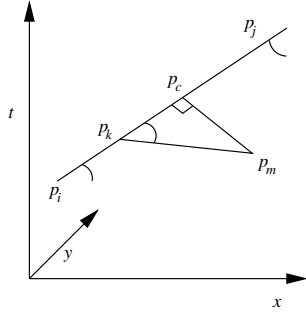
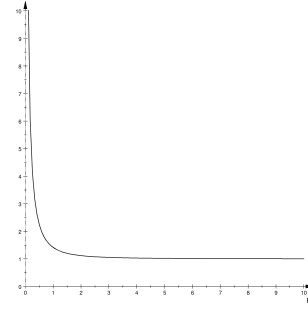


Fig. 3: Illustrating the proof of Theorem 1.

Fig. 4: The dependency of  $\frac{1}{\sin(\arctan R)}$  on  $R$ .

sensible, as illustrated in Fig. 4. For instance setting  $\alpha = s_{\max}$ , where  $s_{\max}$  is the maximum speed along the trajectory, results in  $\delta_s \leq \sqrt{2} \cdot \varepsilon$  for the entire trajectory. Also values smaller than  $s_{\max}$  might make sense for  $\alpha$  in practice. In this case, the slower the speed on a line segment of the simplification is, the smaller  $\delta_s$  is.

In the same way as for the *where-at* query we also obtain that the *when-at* query is sound for  $dist_\alpha$ , if  $\alpha \neq 0$  and  $s_\ell > 0$ .

**Theorem 2.** *Let  $\ell = \overline{p_i p_j}$  be a line segment with speed  $s_\ell$  that is part of a  $(dist_\alpha, \varepsilon)$ -simplification of the trajectory  $T = \langle p_1, \dots, p_i, \dots, p_j, \dots, p_n \rangle$ , and let  $(x, y)$  be any point that lies exactly once on both the projections of  $\ell$  and  $\langle p_i, \dots, p_j \rangle$  onto the  $xy$ -plane. Then we have:*

$$|when-at(\langle p_i, \dots, p_j \rangle, x, y) - when-at(\ell, x, y)| \leq \delta_t := \frac{\varepsilon}{\sin(\arctan \frac{s_\ell}{\alpha})}$$

Hence, the smaller  $\frac{s_\ell}{\alpha}$  is, the bigger  $\delta_t$  is (cf. Fig. 4). While in practice it is sensible to assume that the speed of entities is bounded from above, it is unreasonable to assume that all entities have a minimum speed; this would forbid an entity to be stationary. Being able to choose  $\alpha$  allows a user to fine-tune the trade-off between spatial and temporal soundness of  $dist_\alpha$ , as reflected by Theorems 1 and 2.

Cao et al. show in [4] that, if a distance function is sound for the *where-at* query, then it is also sound for the *nearest-neighbour* and *intersect* queries, and hence, Theorem 1 carries over to those queries, too. The *spatial-join* is special in the sense that the query itself uses a distance function between trajectories. For  $\alpha_1 \leq \alpha_2$  we have that  $dist_{\alpha_1}(p_m, \overline{p_i p_j}) \leq dist_{\alpha_2}(p_m, \overline{p_i p_j})$ . From results in [4] it then follows that  $dist_{\alpha_2}$  is sound for the *spatial-join* that uses the Hausdorff distance function based on  $dist_{\alpha_1}$  as distance function between trajectories.

We believe that the definition of the *when-at* query as given in [4] is too strict. When considering the soundness of a distance function, we compare the original trajectory  $T$  and its simplification  $T'$ . Although all points of  $T'$  have a distance to  $T$  of at most  $\varepsilon$ , we could expect that  $when-at(T, x, y)$  or  $when-at(T', x, y)$  is undefined for almost all points  $(x, y)$ , which renders any reasoning about soundness to be difficult. Hence, we propose different semantics for the *when-at* query (similar considerations can also be made for other queries). As simplified trajectories are approximations anyway, we allow a query region instead of a query point.

- *apx-when-at*( $T, x, y, \lambda$ ) returns a time  $t$  at which a moving object on trajectory  $T$  is expected to be within distance  $\lambda$  from location  $(x, y)$ . If there is no such location on the trajectory, then the answer is undefined.

It seems impossible to prove the soundness of this query in the same sense as above. However, we can prove that *apx-when-at* will report a point at time  $t$  for which it holds that the entity must have been close to  $(x, y)$  at some point in time that is close to  $t$ . That is, we can prove a ‘soundness’ bound that has both a spatial and temporal error. To simplify the statement of the theorem we define *set-apx-when-at* $(T, x, y, \lambda)$  as reporting the set of time points when the trajectory  $T$  is within distance  $\lambda$  from  $(x, y)$ . For a trajectory  $T$  we use  $T(t)$  to denote the position in the  $xy$ -plane of the entity along  $T$  at time  $t$ .

**Theorem 3.** *Let  $P$  be a  $(\text{dist}_\alpha, \varepsilon)$ -simplification of the trajectory  $T = \langle p_1, \dots, p_n \rangle$ . Given a query point  $q = (x, y)$  in the  $xy$ -plane, let  $t_1$  be the time reported by *apx-when-at* $(P, x, y, \lambda + \varepsilon)$ . There exists a time point  $t_2$  in *set-apx-when-at* $(T, x, y, \lambda + 2\varepsilon)$  such that  $|t_1 - t_2| \leq \varepsilon/\alpha$  and  $|T(t_1) - P(t_2)| \leq \varepsilon$ .*

*Proof.* From the definition of *apx-when-at* we have that  $|P(t_1) - q| \leq \lambda + \varepsilon$ . Since  $P$  is a  $(\text{dist}_\alpha, \varepsilon)$ -simplification of the trajectory  $T$  there exists a point  $T(t_2)$  on  $T$  such that  $|P(t_1) - T(t_2)| \leq \varepsilon$  which implies that  $|t_1 - t_2| \leq \varepsilon/\alpha$ . It remains to prove that  $t_2 \in \text{set-apx-when-at}(T, x, y, \lambda + 2\varepsilon)$ . This is equivalent to proving that the distance between  $T(t_2)$  and  $q$  is at most  $\lambda + 2\varepsilon$ , which is obviously true since  $|T(t_2) - q| \leq |T(t_2) - P(t_1)| + |P(t_1) - q| \leq \lambda + 2\varepsilon$ .  $\square$

Note that if we set  $\alpha$  to be greater than the largest speed of the entity then both *where-at* and *apx-when-at* can be sound for small errors at the same time for any input path. This is the first time any such bound has been shown using a single distance function, even though it is approximate in both time and space.

We also implemented our algorithms and performed initial experiments for artificially generated data. We conclude that adding the scaling parameter  $\alpha$  into the Douglas-Peucker algorithm in the 3-dimensional case increases the running time only marginally. Furthermore, we can see huge reductions in the number of vertices in the output, which in the end is the goal of trajectory simplification. Depending on the chosen values for  $\varepsilon$  and  $\alpha$ , we could observe compression rates between 70% and 95%. Cao et al. [4] report compression rates of up to 98% for real data. Real data is likely to contain more structure and hence, allows for higher compression rates. More interesting might be the influence of  $\alpha$ . We could observe that, for a fixed  $\varepsilon$ , the bigger  $\alpha$  became the more temporal information was preserved.

### 3 A fast implementation of the Douglas-Peucker algorithm for self-intersecting polygonal paths

In this section we present a fast implementation of the Douglas-Peucker algorithm in the case when the polygonal path may self-intersect. We consider two variants of the algorithm, one which works in the line-segment model and another which works in the line model. As mentioned in the introduction, Hershberger and Snoeyink gave an  $O(n \log^* n)$ -time algorithm working in the line model when the path does not self-intersect. In the self-intersecting case, only the trivial  $O(n^2)$  time bound is known, to the best of the authors’ knowledge.

As a first step, we will prove that just as in the line model the furthest point has to be a vertex on the convex hull of the point set. For simplicity we will throughout this section assume that no three points lie on a line.

**Lemma 1.** *Given a set of  $n \geq 3$  points  $S$  and a line segment  $\ell$ , the maximum distance between  $S$  and  $\ell$  is defined between a vertex  $p$  on the convex hull of  $S$ .*

It is now easy to see that an important subproblem that we need to consider is the following.

*Problem 1.* (Line-segment furthest-point queries (LSFP-queries)) Preprocess an ordered set of  $n$  points  $p_1, \dots, p_n$  in convex position in the plane into a data structure supporting the following query: given a line segment  $(p_i, p_j)$ ,  $1 \leq i < j \leq n$ , report the point  $p_k$  that is furthest from  $(p_i, p_j)$  such that  $i < k < j$ .

Below we will prove that the LSFP-query problem can be transformed into the following problem with only a small loss in time and space complexity.

*Problem 2.* (Half-plane furthest-point queries (HPFP-queries)) Preprocess  $n$  points  $p_1, \dots, p_n$  in convex position in the plane into a data structure supporting the following query: given a point  $q$  and a directed line  $\ell$ , report the point  $p_i$  that is furthest from  $q$  subject to being to the left of  $\ell$ .

**Lemma 2.** *A set  $S$  of  $n$  points in convex position in the plane can be preprocessed in  $2F(n) + O(n \log n)$  time using  $O(n) + S(n)$  space such that LSFP-queries can be answered in  $2Q(n) + O(\log n)$  time, where  $F(n)$  is the preprocessing time needed to store  $S$  in a data structure of size  $S(n)$  that answers HPFP-queries in  $Q(n)$  time.*

The  $O(n \log n)$  time bound in the above lemma comes from the fact that we need to compute the convex hull of  $S$ . However, if the points in  $S$  are sorted with respect to their  $x$ -coordinates in increasing order, then this step can be done in  $O(n)$  time. Unfortunately this improvement will not affect the overall time complexity of the Douglas-Peucker algorithm.

### 3.1 Half-plane furthest-point queries

The HPFP-query problem was first studied by Aronov et al. [3] and they showed the following two results:

**Fact 1** (Corollary 5 in [3]) *There is a data structure that requires  $O(n^{1+\beta})$  space and preprocessing time, and supports HPFP-queries in  $O(2^{1/\beta} \log n)$  time on  $n$  points in convex position, for any real number  $\beta > 0$ .*

**Fact 2** (Corollary 11 in [3]) *There is a data structure that requires  $O(n \log^3 n)$  space and polynomial preprocessing time, and supports HPFP-queries in  $O(\log n)$  time on  $n$  points in convex position.*

As an alternative we present a data structure that has slightly higher query time, but because of smaller preprocessing time and smaller space consumption our approach leads to a more efficient implementation of the Douglas-Peucker algorithm.

**Lemma 3.** *One can preprocess a planar set  $S$  of  $n$  points in convex position in  $O(n \log n)$  time using  $O(n \log n)$  space such that HPFP-queries on  $S$  can be answered in  $O(\log^2 n)$  time.*

### 3.2 Path simplification in the line-segment model

In this section we merge the results into one single data structure. In particular, we study the problem of preprocessing a polygonal path  $P$  with  $n$  vertices such that, given a line segment  $\ell$  and a subpath  $P'$  of  $P$ , the point in  $P'$  furthest from  $\ell$  is reported.

We will prove the following lemma.

**Lemma 4.** *A polygonal path  $P = \langle v_1, v_2, \dots, v_n \rangle$  with  $n$  vertices in the plane can be preprocessed in*

$$O(n \log^2 n) + \sum_{i=0}^{\log n} 2^{i+1} F\left(\frac{n}{2^i}\right) \text{ time, using } O(n \log n) + \sum_{i=0}^{\log n} 2^i S\left(\frac{n}{2^i}\right) \text{ space}$$

such that, given a line segment  $\ell$  and a subpath  $P' = \langle v_i, \dots, v_j \rangle$  of  $P$ , the point in  $P'$  furthest from  $\ell$  can be reported in time

$$O(\log^2 n) + 2 \sum_{i=0}^{\log n} Q\left(\frac{n}{2^i}\right),$$

where  $F(n)$  is the preprocessing time needed to construct a data structure of size  $S(n)$  that can answer HPFP-queries in  $Q(n)$  time.

The standard Douglas-Peucker algorithm iterates over at most  $n$  line segments. Thus, by combining Lemmas 2, 3 and 4 we obtain the following theorem.

**Theorem 4.** *(line-segment model) For a polygonal path  $P$  with  $n$  vertices in the plane, the Douglas-Peucker algorithm can be implemented in time  $O(n \log^3 n)$  using  $O(n \log n)$  space.*

Note that in Lemma 4 presorting could be used to improve the preprocessing time by a logarithmic factor, but this does not have any effect on the asymptotic efficiency of the Douglas-Peucker algorithm.

### 3.3 Path simplification in the line model

Even if Theorem 4 also holds in the line model, the inclusive structure of the distance queries is not fully utilised. It turns out that path simplification is easier in the line model. Next we show how both the time and the space bounds can be improved by a logarithmic factor. The tools used in this improved construction are basically the same as those used before. The main reason for obtaining this improvement is that a vertex of a convex hull furthest from a line can be reported fast by binary search [17] by determining the two tangents parallel to the given line and returning the furthest of the vertices on these tangents.

The algorithm operates in four steps. First, the vertices on the given polygonal path  $P$  of size  $n$  are partitioned into canonical sets whose size is a power of 2. Let the collection of these sets be  $\mathcal{P} = \{P_1, P_2, \dots, P_h\}$ . The size of  $P_1$  should be the largest power of 2 no greater than  $n$ , the size of  $P_2$  the largest power of 2 no greater than  $n - |P_1|$ , and so on. That is,  $h \leq \lceil \log n \rceil$ . Second, the canonical sets of  $\mathcal{P}$  are presorted according to their  $x$ -coordinate. Let  $\mathcal{S}$  be the corresponding collection of sorted sets of vertices. Also, associate each vertex in a sorted set with its index in the polygonal path. Third, the convex hulls of the canonical sets are computed. Let the resulting collection be  $\mathcal{C}$ . Due to presorting, the computation of

each convex hull only takes linear time if we use Graham’s convex-hull algorithm. Fourth, the recursive subroutine, to be described next, is called with  $\varepsilon$ ,  $P$ ,  $\mathcal{P}$ ,  $\mathcal{S}$ , and  $\mathcal{C}$ .

Assume that the input of the recursive subroutine is real number  $\varepsilon$  and polygonal path  $\langle v_i, v_{i+1}, \dots, v_k \rangle$  together with the corresponding collections of canonical sets, sorted sets, and convex hulls. The functioning of the recursive subroutine is as follows:

1. Compute the furthest point between the polygonal path and the line  $\ell$  determined by  $v_i$  and  $v_k$ . This is done by computing the furthest point between  $\ell$  and the convex hulls, one by one, and by determining the overall furthest point. Let  $v_j$  be this vertex.
2. If the distance between line  $\ell$  and vertex  $v_j$  is less than or equal to  $\varepsilon$ , return the line segment  $(v_i, v_k)$  as a simplification for  $P$  and stop this branch of recursion.
3. Split the polygonal path into two subpaths  $\langle v_i, v_{i+1}, \dots, v_j \rangle$  and  $\langle v_j, v_{j+1}, \dots, v_k \rangle$ . Correspondingly, split the canonical set containing  $v_j$  into smaller canonical sets whose size is a power of 2. This is done by repeatedly halving the canonical set containing  $v_j$  until  $v_j$  forms a singleton set. For each canonical set created during this process, compute the sorted set of vertices by scanning the sorted set corresponding to the parent canonical set. Finally, compute the convex hulls of the new canonical sets created. After halving a canonical set, it and the corresponding sorted set and convex hull are disposed.
4. Call the recursive routine for both subpaths together with the corresponding collections of canonical sets, sorted sets, and convex hulls.

Let us now analyse the performance of this algorithm for a polygonal path of  $n$  vertices. The amount of work done in the three first steps of the main routine is dominated by that required by sorting, i.e. the running time is  $O(n \log n)$ . In the recursive subroutine in connection with each halving, sorted sets are scanned and convex hulls may be computed, both requiring time linear on the size of the subpaths considered. Since each vertex is involved in  $O(\log n)$  halvings, the overall running time of all splits is  $O(n \log n)$ . At each recursive step, in the furthest-point calculation the number of convex hulls to be considered is bounded by  $O(\log n)$  and each distance computation between a line and a convex hull takes  $O(\log n)$  time. Naturally, the number of recursive calls is linear in the worst case. Therefore, the total running time of the algorithm is  $O(n \log^2 n)$ . At any given point in time, each vertex can be in at most one canonical set. Hence, the space bound is  $O(n)$ .

The above discussion can be summarised as follows:

**Theorem 5.** *(line model) For a polygonal path  $P$  with  $n$  vertices in the plane, the Douglas-Peucker algorithm can be implemented in time  $O(n \log^2 n)$  using  $O(n)$  space.*

#### 4 A fast implementation of the Douglas-Peucker algorithm in 3-dimensional space

In this section, we present a fast, approximate version of the Douglas-Peucker algorithm in  $\mathbb{R}^3$ . The algorithm can be used for 3-dimensional paths or for trajectories with two spatial dimensions and one temporal dimension. In addition to taking as input a distance error threshold  $\varepsilon$ , it takes a real number  $\delta > 0$ , and produces a simplified path that is within a distance of  $(1 + \delta)\varepsilon$  from every vertex of the original path. It is possible to set  $\varepsilon = \frac{\varepsilon^*}{1+\delta}$  to obtain a distance error bound of exactly some desired value  $\varepsilon^*$ . In this case,  $\delta$  does not affect the distance threshold, but a larger  $\delta$  may result in a larger number of vertices in the

simplified path. As for the original Douglas-Peucker algorithm, this approach is a heuristic, and we present no bound on the number of vertices.

The general idea of the algorithm is as follows. First, we project the vertices of the original path onto  $O(1/\delta^2)$  rotations of the  $xy$ -plane, equally spaced in angle about the  $y$ - and  $z$ -axes, yielding a 2-dimensional projection of the original path that may contain self-intersections. One of the 2-dimensional algorithms of Section 3 is then executed on each of the planes, up to the point where the simplified path is to be split at a vertex. At this point, a split vertex has been chosen for each projection plane, based on the distance in the projection between that vertex and the proposed simplified line segment. From these potential split vertices, take the one with the maximum distance to the line segment over all of the projection planes. Split at this vertex in all planes, and continue executing. We will show that the original distance between the vertex and the line segment in  $\mathbb{R}^3$  is at most  $(1 + \delta)$  times the maximum projected distance over all of the planes. This property allows us to construct an approximate simplification efficiently in 3-dimensional space.

We start by defining a set  $\Psi$  of projection planes. Given two angles  $0 \leq \alpha \leq \pi$  and  $0 \leq \beta \leq 2\pi$ , let  $\psi_{\alpha,\beta}$  be the plane obtained by rotating the  $xy$ -plane around the  $y$ -axis by  $\alpha$  radians and around the  $z$ -axis by  $\beta$  radians, i.e. the plane with normal vector  $\langle \sin \alpha \cos \beta, \sin \alpha \sin \beta, \cos \alpha \rangle$ . Suppose we wish to perform  $k = \lceil 2\pi/\arccos(1/(1 + \delta)) \rceil$  discrete rotations around the  $y$ -axis, and  $2k$  around the  $z$ -axis. The angle between successive rotations around either of the axes will be  $\theta = \frac{\pi}{k}$ . Note that for any real  $\delta > 0$ , it holds that  $0 < \theta < \frac{\pi}{4}$ . Now we can define a set of projection planes  $\Psi = \{\psi_{i\theta,j\theta} \mid i, j \in \mathbb{Z}, 0 \leq i < (k/2), 0 \leq j < k\}$ .

**Lemma 5.** *Given a plane with normal vector  $\hat{n}$ , there exists a plane  $\psi^* \in \Psi$  with normal vector  $\hat{n}^*$  such that the angle between  $\hat{n}$  and  $\hat{n}^*$  is no more than  $\theta$ .*

*Proof.* Let  $\phi$  be the angle between  $\hat{n}$  and the  $y$ -axis, and let  $\gamma$  be the angle between  $\hat{n}$  and the  $z$ -axis. By construction, there is a plane  $\psi_{\alpha,\beta} \in \Psi$  such that  $|\alpha - \phi| \leq \theta/2$  and  $|\beta - \gamma| \leq \theta/2$ . Let  $\psi^* = \psi_{\alpha,\beta}$ . Since  $\hat{n}$  needs to be rotated twice by at most  $\theta/2$  to meet  $\hat{n}^*$ , the angle between  $\hat{n}$  and  $\hat{n}^*$  is at most  $2 \cdot (\theta/2) = \theta$ .  $\square$

Given a point  $p \in \mathbb{R}^3$  and a plane  $\psi \in \Psi$ , let  $proj(p, \psi)$  be the orthogonal projection of  $p$  onto the plane  $\psi$ , defined as the point of intersection between  $\psi$  and the line orthogonal to  $\psi$  passing through  $p$ . To prove an approximation bound, we first need a bound on the distance between two projected points from their original distance in  $\mathbb{R}^3$ .

**Lemma 6.** *Given two points  $p, q \in \mathbb{R}^3$ , it holds that*

$$|\overline{pq}| \cos \theta \leq \max_{\psi \in \Psi} |\overline{proj(p, \psi)proj(q, \psi)}| \leq |\overline{pq}|$$

In the Douglas-Peucker algorithm, we are not only interested in the distance between two points, but also in the distance between a point and a line. We therefore need to look at the projection of the triangle given by the point and two points on the line.

**Lemma 7.** *Given three points  $p, q, r \in \mathbb{R}^3$  such that  $\angle pqr > 2\theta$ , it holds that*

$$dist(q, \overline{pr}) \geq \max_{\psi \in \Psi} dist(proj(q, \psi), \overline{proj(p, \psi)proj(r, \psi)}) \geq \frac{dist(q, \overline{pr})}{\sqrt{2 - \cos^2 \theta}}$$

We are now ready to state the final result of this section.

**Theorem 6.** *Given a real number  $\delta > 0$ , a  $(1 + \delta)$ -approximate Douglas-Peucker simplification can be computed in the line model in  $O(\frac{1}{\delta^2}n \log^2 n)$  time using  $O(\frac{1}{\delta^2}n)$  space, and in the line-segment model in  $O(\frac{1}{\delta^2}n \log^3 n)$  time using  $O(\frac{1}{\delta^2}n \log n)$  space.*

## Acknowledgements

We would like to thank Adel Ahmed for useful discussions.

## References

1. P. K. Agarwal and K. R. Varadarajan. Efficient algorithms for approximating polygonal chains. *Discrete & Computational Geometry*, 23(2):273–291, 2000.
2. A. Aggarwal, L. J. Guibas, J. B. Saxe, and P. W. Shor. A linear-time algorithm for computing the Voronoi diagram of a convex polygon. *Discrete and Computational Geometry*, 4(1):591–604, 1989.
3. B. Aronov, P. Bose, E. D. Demaine, J. Gudmundsson, J. Iacono, S. Langerman, and M. Smid. Data structures for halfplane proximity queries and incremental Voronoi diagrams. In *Proceedings of the 7th Latin American Symposium on Theoretical Informatics*, volume 3887 of *Lecture Notes in Computer Science*, pages 80–92. Springer-Verlag, 2006.
4. H. Cao, O. Wolfson, and G. Trajcevski. Spatio-temporal data reduction with deterministic error bounds. *The VLDB Journal*, 15(3):211–228, 2006.
5. W. S. Chan and F. Chin. Approximation of polygonal curves with minimum number of line segments. In *Proceedings of the 3rd International Symposium on Algorithms and Computation*, volume 650 of *Lecture Notes in Computer Science*, pages 378–387. Springer-Verlag, 1992.
6. D. Douglas and T. Peucker. Algorithms for the reduction of the number of points required to represent a digitised line or its caricature. *The Canadian Cartographer*, 10(2):112–122, 1973.
7. H. Edelsbrunner, L. J. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM Journal on Computing*, 15(2):317–340, 1986.
8. A. U. Frank. Life and motion of socio-economic units. In *Socio-Economic Units: Their Life and Motion*, volume 8 of *GISDATA*, pages 21–34. Taylor & Francis, 2001.
9. J. Gudmundsson, P. Laube, and T. Wolle. Movement patterns in spatio-temporal data. In *Encyclopedia of GIS*. Springer-Verlag, to appear.
10. R. H. Güting and M. Schneider. *Moving Objects Databases*. Morgan Kaufmann Publishers, 2005.
11. J. Hershberger and J. Snoeyink. Speeding up the Douglas-Peucker line-simplification algorithm. In *Proceedings of the 5th International Symposium on Spatial Data Handling*, pages 134–143. IGU Commission on GIS, 1992.
12. J. Hershberger and J. Snoeyink. Cartographic line simplification and polygon CSG formulæ in  $O(n \log^* n)$  time. *Computational Geometry—Theory and Applications*, 11(3–4):175–185, 1998.
13. I. A. R. Hulbert. GPS and its use in animal telemetry: The next five years. In *Proceedings of the Conference on Tracking Animals with GPS*, pages 51–60. Macaulay Insitute, 2001.
14. H. Imai and M. Iri. Computational-geometric methods for polygonal approximations of a curve. *Computer Vision, Graphics, and Image Processing*, 36(1):31–41, 1986.
15. M. P. Kwan. Interactive geovisualization of activity-travel patterns using three dimensional geographical information systems: A methodological exploration with a large data set. *Transportation Research, Part C*, 8(1–6):185–203, 2000.
16. A. Melkman and J. O’Rourke. On polygonal chain approximation. In *Computational Morphology*, pages 87–95. North-Holland, 1988.
17. M. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *Journal of Computer and System Sciences*, 23(2):166–204, 1981.
18. F. Porikli. Trajectory distance metric using hidden Markov model based representation. In *Proceedings of the 6th IEEE International Workshop on Performance Evaluation of Tracking and Surveillance*. IEEE, 2004.

## A Omitted Proofs

**Lemma 1.** *Given a set of  $n \geq 3$  points  $S$  and a line segment  $\ell$ , the maximum distance between  $S$  and  $\ell$  is defined between a vertex  $p$  on the convex hull of  $S$ .*

*Proof.* Consider a point  $p \in S$  that realises the maximum distance between  $S$  and  $\ell$ . We need to prove that  $p$  lies on the convex hull of  $S$ . For simplicity we assume that  $\ell$  is horizontal. We will have two cases:

(a) If  $p$  has a perpendicular projection onto  $\ell$ , then consider the line  $\ell'$  through  $p$  and parallel to  $\ell$ , see Fig. 5a. If  $p$  lies on an edge  $e$  of the convex hull, then it is easily seen that one can move  $p$  to one of the endpoints of the edge such that the distance does not decrease. If  $p$  does not lie on the convex hull, then there must be a point  $p'$  of  $S$  that lies above  $\ell'$ , otherwise  $p$  would be on the convex hull. However, since  $p'$  lies above  $\ell'$  it follows that  $d(p, \ell) < d(p', \ell)$ , which is a contradiction and thus  $p$  must lie on the convex hull.

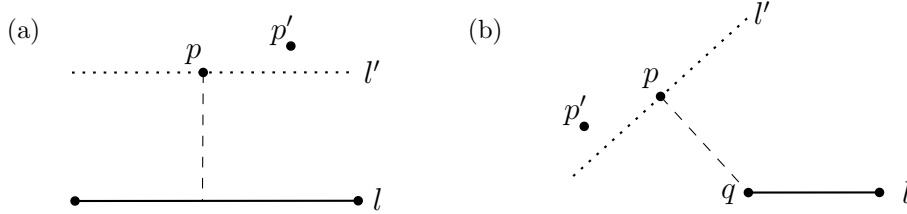


Fig. 5: Illustrating the proof of Lemma 1.

(b) If  $p$  does not have a perpendicular projection onto  $\ell$ , then let  $q$  be the endpoint of  $\ell$  closest to  $p$ , as illustrated in Fig. 5b. Consider the line  $\ell'$  through  $p$  and orthogonal to the line through  $q$  and  $p$ . Assume that  $p$  does not lie on the convex hull. Then there must be a point  $p'$  of  $S$  that lies on the opposite side of  $\ell'$  compared to  $\ell$ , otherwise  $p$  would be on the convex hull. However, this implies that  $d(p, \ell) < d(p', \ell)$ , which is a contradiction and thus  $p$  must lie on the convex hull.  $\square$

**Lemma 2.** *A set  $S$  of  $n$  points in convex position in the plane can be preprocessed in  $2F(n) + O(n \log n)$  time using  $O(n) + S(n)$  space such that LSPF-queries can be answered in  $2Q(n) + O(\log n)$  time, where  $F(n)$  is the preprocessing time needed to store  $S$  in a data structure of size  $S(n)$  that answers HPFP-queries in  $Q(n)$  time.*

*Proof.* As a first step, compute the convex hull of  $S$  in  $O(n \log n)$  time by sorting the points in angular order. Assume that  $\ell = (p, q)$  is horizontal and that  $p$  lies to the left of  $q$ . Partition the points of the convex hull into four sets; the set  $S_T$  of points on the convex hull above  $\ell$  and with a perpendicular projection onto  $\ell$ , the set  $S_B$  of points on the convex hull below  $\ell$  and with a perpendicular projection onto  $\ell$ , the set  $S_L$  of points on the convex hull whose nearest point on  $\ell$  is the left endpoint of  $\ell$ , and finally, the set  $S_R$  of points on the convex hull whose nearest point on  $\ell$  is the right endpoint of  $\ell$ . Partitioning into sets  $S_T$ ,  $S_B$ ,  $S_L$ , and

$S_R$  with respect to  $\ell$  can be carried out by computing the intersection points, if any, between the convex hull and a directed line. Each of these sets consists of up to three pieces of the precomputed convex hull.

The point in  $S_T$  with the largest distance to  $\ell$  is the point with the largest  $y$ -coordinate. Obviously, one can find this point by performing a binary search along the pieces containing  $S_T$ . Symmetrically, the same can be done with the set  $S_B$ . It remains to find the points in  $S_L$  and  $S_R$  that are furthest from  $p$  and  $q$ , respectively.

Consider the set  $S_L$ . The points in  $S_L$  lie to the left of the vertical line through  $p$  directed from bottom to top, while the remaining points of  $S$  lie to the right of that line. Thus the point in  $S_L$  furthest from  $\ell$  can be obtained by performing an HPFP-query with the point  $p$  and the directed line determined by  $\ell$  and  $p$  on  $S_L$ . Symmetrically, the same can be done with  $S_R$ . This concludes the proof of the lemma.  $\square$

**Lemma 3.** *One can preprocess a planar set  $S$  of  $n$  points in convex position in  $O(n \log n)$  time using  $O(n \log n)$  space such that HPFP-queries on  $S$  can be answered in  $O(\log^2 n)$  time.*

*Proof.* To simplify the description we consider  $S$  to be a simple path with  $n$  vertices ordered along the convex hull. We build a balanced binary tree  $\mathcal{T}$  above the vertices listed in counter-clockwise order along the convex hull. The subset of points stored in the leaves of a subtree rooted at a node  $\nu$  is called the *canonical subset* of  $\nu$ , and is denoted  $S(\nu)$ . For any internal or leaf node  $\nu$  we store a furthest-point Voronoi diagram (FPVD) of the canonical subset, together with a point-location structure (PLS) that allows for fast point-location queries in the FPVD. We claim (\*) that both the FPVD and the PLS can be built in  $O(|S(\nu)|)$  time using  $O(|S(\nu)|)$  space such that, given a query point  $p$ , the point in  $S(\nu)$  furthest from  $p$  can be found in  $O(\log |S(\nu)|)$  time.

To prove the claim (\*), consider a set  $S$  of  $n$  points in convex position in the plane. The FPVD of  $S$ , denoted  $\mathcal{F}(S)$ , can be computed in  $O(n)$  time using  $O(n)$  space when the input is a convex polygon [2]. Note that  $\mathcal{F}(S)$  forms a tree that partitions the plane into convex unbounded faces. Next, construct a bounding rectangle  $B$  that contains  $S$  and all the vertices in  $\mathcal{F}(S)$ . Let  $\mathcal{D}(S)$  denote the subdivision obtained from the union of  $B$  and  $\mathcal{F}(S)$  where all the edges of  $\mathcal{F}(S)$  outside  $B$  has been removed. It is easily seen that each face in  $\mathcal{D}(S)$  is a convex face. Edelsbrunner et al. [7] showed that for any subdivision where the faces are  $y$ -monotone a PLS can be built in  $O(n)$  time using  $O(n)$  space such that queries can be answered in  $O(\log n)$  time. Obviously, a convex face is also  $y$ -monotone, thus the claim holds. Since the FPVD and the PLS can be constructed in linear time for each node in  $\mathcal{T}$ , it follows that the total time to build  $\mathcal{T}$  is  $O(n \log n)$  and it requires  $O(n \log n)$  space.

Given a query point  $p$  and a directed line  $\ell$ , an HPFP-query is performed as follows. If  $\ell$  does not intersect the convex hull of  $S$ , then we simply report the point furthest from  $p$  in  $S$ . Otherwise, let  $p_l$  and  $p_r$  be the two intersection points between  $\ell$  and the convex hull of  $S$  and assume that  $p_l$  lies above  $p_r$  along  $\ell$ . For simplicity we assume that the first point of  $S$  in the order along the convex hull does not lie to the left of  $\ell$ , if it does we simply have to perform two queries instead of one, and choose the point among the two reported points that is furthest from  $\ell$ . The two points  $p_l$  and  $p_r$  can be found in  $O(\log n)$  time by binary search along the convex hull of  $S$ . Search with  $p_l$  and  $p_r$  in  $\mathcal{T}$  until we get a node  $\nu_{split}$  where the search path splits. Without loss of generality, assume that  $p_l$  (resp.  $p_r$ ) lies in the left (right)

subtree of  $\nu_{split}$ . Since the points stored in the leaves of  $\mathcal{T}$  are ordered along the convex hull, the search paths from the root to  $p_l$  and  $p_r$  are well defined.

From the left child of  $\nu_{split}$  we continue the search with  $p_l$ , and at every node  $\nu$  where the search path of  $p_l$  goes left, we perform a furthest-point query with  $p$  in  $\mathcal{F}(S_R(\nu))$ , where  $S_R(\nu)$  is the point set stored in the right child of  $\nu$ . As described above, this can be performed in  $O(\log |S_R(\nu)|)$  time. Similarly, we continue the search with  $p_r$  at the right child of  $\nu_{split}$ , and at every node  $\nu$  where the search path of  $p_r$  goes to the right we perform a furthest-point query with  $p$  in  $\mathcal{F}(S_L(\nu))$ , where  $S_L(\nu)$  is the point set stored in the left child of  $\nu$ . In effect, we performed  $O(\log n)$  queries on a collection of  $O(\log n)$  subtrees that together contain exactly the points along the convex hull of  $S$  between  $p_l$  and  $p_r$ . The point furthest from  $\ell$  reported from the  $O(\log n)$  queries is reported to have the largest distance to  $\ell$  among all the points on the convex hull between  $(p_l, p_r)$  (i.e. to the left of  $\ell$ ).

Since we perform  $O(\log n)$  queries, and each query requires  $O(\log n)$  time, the total query time is  $O(\log^2 n)$  time. As a result, the claim of Lemma 3 follows.  $\square$

**Lemma 4.** *A polygonal path  $P = \langle v_1, v_2, \dots, v_n \rangle$  with  $n$  vertices in the plane can be preprocessed in time*

$$O(n \log^2 n) + \sum_{i=0}^{\log n} 2^{i+1} F\left(\frac{n}{2^i}\right),$$

using

$$O(n \log n) + \sum_{i=0}^{\log n} 2^i S\left(\frac{n}{2^i}\right)$$

space such that, given a line segment  $\ell$  and a subpath  $P' = \langle v_i, \dots, v_j \rangle$  of  $P$ , the point in  $P'$  furthest from  $\ell$  can be reported in time

$$O(\log^2 n) + 2 \sum_{i=0}^{\log n} Q\left(\frac{n}{2^i}\right),$$

where  $F(n)$  is the preprocessing time needed to construct a data structure of size  $S(n)$  that can answer HPFP-queries in  $Q(n)$  time.

*Proof.* Consider a tree structure containing the points of  $P$ . The subset of points stored in the leaves of a subtree rooted at a node  $\nu$  is called the canonical subset of  $\nu$ , and is denoted  $P(\nu)$ . We build a 2-level data structure  $\mathcal{C}$  where the main tree of  $\mathcal{C}$  is a balanced binary search tree  $\mathcal{T}$  built on the order of the points in  $P$ . For any internal or leaf node  $\nu$  in  $\mathcal{T}$ , the canonical subset  $P(\nu)$  is stored in an LSFP-structure including the convex hull  $\mathcal{H}(\nu)$  of  $P(\nu)$  (as well as other data structures needed to support HPFP-queries).

A distance query is performed as follows. Search with  $v_i$  and  $v_j$  in  $\mathcal{C}$  until we get a node  $\nu_{split}$  where the search path splits. From the left child of  $\nu_{split}$  we continue the search with  $v_i$ , and at every node  $\nu$  where the search path of  $v_i$  goes left, we perform an LSFP-query with  $\mathcal{H}(\nu_R)$  and the segment  $(v_i, v_j)$ , where  $\nu_R$  is the right child of  $\nu$ . Similarly, we continue the search with  $v_j$  at the right child of  $\nu_{split}$ , and at every node  $\nu$  where the search path of  $v_j$  goes to the right we perform an LSFP-query with  $\mathcal{H}(\nu_L)$  and the segment  $(v_i, v_j)$ , where  $\nu_L$  is the left child of  $\nu$ . In effect, we performed  $O(\log n)$  queries on a collection of  $O(\log n)$

subtrees that together contain exactly the points along  $P$  between  $v_i$  and  $v_j$ . The largest value reported from the  $O(\log n)$  LSFP-queries is reported as the largest distance between  $(v_i, v_j)$  and a point along the path from  $v_i$  to  $v_j$ .

Let  $F'(n)$  be the preprocessing needed to construct a data structure of size  $S'(n)$  that can answer LSFP-queries in  $Q'(n)$  time. Using Lemma 2 the preprocessing time is

$$O(n \log n) + \sum_{i=0}^{\log n} 2^i \cdot F'(\frac{n}{2^i}) = O(n \log^2 n) + \sum_{i=0}^{\log n} 2^{i+1} F'(\frac{n}{2^i}),$$

the query time is

$$O(\log n) + \sum_{i=1}^{\log n} Q'(\frac{n}{2^i}) = O(\log^2 n) + 2 \sum_{i=0}^{\log n} Q(\frac{n}{2^i})$$

and the amount of space is

$$O(n) + \sum_{i=0}^{\log n} 2^i \cdot S'(\frac{n}{2^i}) = O(n \log n) + \sum_{i=0}^{\log n} 2^i S(\frac{n}{2^i}).$$

This concludes the proof of the lemma.  $\square$

**Lemma 6.** *Given two points  $p, q \in \mathbb{R}^3$ , it holds that*

$$|\overline{pq}| \cos \theta \leq \max_{\psi \in \Psi} |\overline{proj(p, \psi)proj(q, \psi)}| \leq |\overline{pq}|$$

*Proof.* It follows from Lemma 5 that for any plane containing  $\overline{pq}$  with normal  $\hat{n}$ , there exists a plane  $\psi^* \in \Psi$  with normal  $\hat{n}^*$  such that the angle  $\omega$  between  $\hat{n}$  and  $\hat{n}^*$  is between 0 and  $\theta$ . The length of  $\overline{pq}$  projected onto  $\psi^*$  is  $|\overline{proj(p, \psi^*)proj(q, \psi^*)}| = |\overline{pq}| \sin((\frac{\pi}{2}) - \omega) = |\overline{pq}| \cos \omega$ . Clearly, the maximum value of  $|\overline{proj(p, \psi)proj(q, \psi)}|$  over all  $\psi \in \Psi$  occurs when  $\omega$  is smallest. Hence,  $|\overline{pq}| \cos \theta \leq \max_{\psi \in \Psi} |\overline{proj(p, \psi)proj(q, \psi)}| \leq |\overline{pq}|$ .  $\square$

**Lemma 7.** *Given three points  $p, q, r \in \mathbb{R}^3$  such that  $\angle pqr > 2\theta$ , it holds that*

$$dist(q, \overline{pr}) \geq \max_{\psi \in \Psi} dist(proj(q, \psi), \overline{proj(p, \psi)proj(r, \psi)}) \geq \frac{dist(q, \overline{pr})}{\sqrt{2 - \cos^2 \theta}}$$

*Proof.* Let  $\psi$  be the plane containing  $p, q$  and  $r$ . By Lemma 5, there is a plane  $\psi^* \in \Psi$  within an angle of  $\theta$  from  $\psi$ . Consider the line  $\ell$  of intersection between  $\psi$  and  $\psi^*$ . Note that we can translate  $\psi^*$  and hence  $\ell$  arbitrarily and obtain the same orthogonal projection; only the orientation of  $\psi^*$  and  $\ell$  are ultimately important. Without loss of generality, assume that  $\psi$  is the  $xy$ -plane and that  $\ell$  is the vertical line passing through the point  $r$ , as in Fig. 6(a).

To simplify the analysis, we rotate the projected points  $proj(p, \psi^*)$ ,  $proj(q, \psi^*)$  and  $proj(r, \psi^*)$  back into the  $xy$ -plane and call the rotated points  $p'$ ,  $q'$  and  $r'$ , respectively. Let  $t_p$  be the point on  $\ell$  that is closest to  $p$  (Fig. 6(b)). Then  $|t_p p'| = |t_p \overline{proj(p, \psi^*)}| \geq |\overline{t_p p}| \cos \theta$ . Similarly,  $|\overline{t_q q'}| \geq |\overline{t_q q}| \cos \theta$  and  $|\overline{t_r r'}| \geq |\overline{t_r r}| \cos \theta$ .

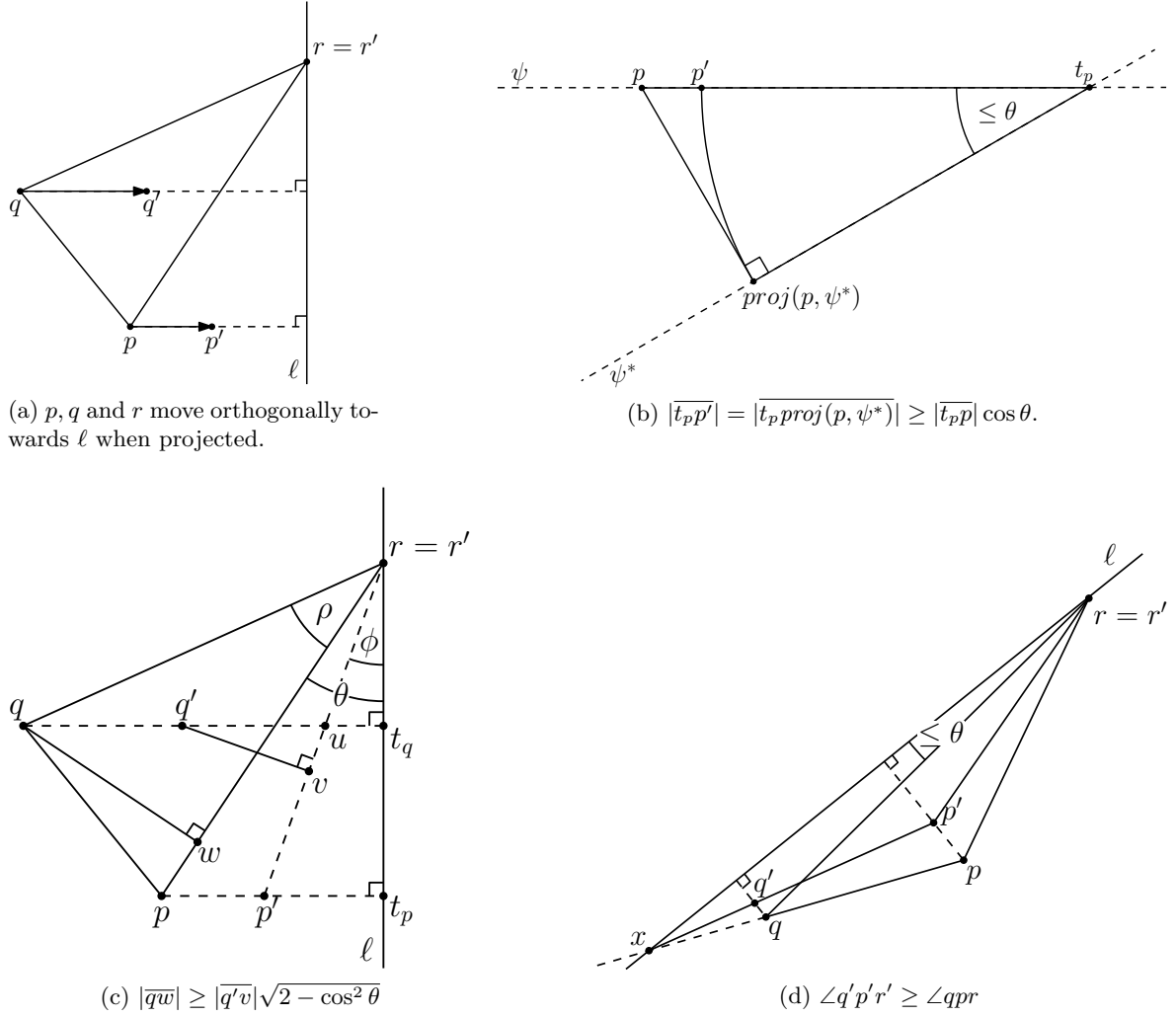


Fig. 6: Illustrating the proofs of Lemma 7 and Theorem 6.

Let  $\rho$  be the angle in the triangle at  $r$ , let  $\phi$  be the angle between  $\ell$  and  $\overline{p'r}$  (see Fig. 6(c)). Define the point  $u$  as the intersection of  $\overline{qt_q}$  with  $\overline{p'r}$ ,  $v$  as the intersection of  $\overline{p'r}$  with its perpendicular through  $q'$ , and  $w$  as the intersection of  $\overline{p'r}$  with its perpendicular through  $q$ .

Let  $h_{old} = |\overline{qw}| = \text{dist}(q, \overline{p'r})$  and  $h_{new} = |\overline{q'v}| = \text{dist}(q', \overline{p'r'})$ . Our aim is to find the ratio  $h_{old}/h_{new}$ . Note that we can always choose  $\psi^* \in \Psi$  such that the angle between  $\ell$  and  $\overline{p'r}$  is at most  $\theta$ , as a direct consequence of Lemma 5. Furthermore, since  $\angle pqr > 2\theta$ ,  $\psi^*$  can be chosen such that the movement of  $q$  to  $q'$  is bounded by the extension of the edges  $\overline{pq}$  and  $\overline{qr}$  to the line  $\ell$ . The best case clearly occurs when the angle between  $\ell$  and  $\overline{p'r}$  is zero, and we obtain  $h_{old}/h_{new} = 1$ . We now analyse the case when this angle is  $\theta$ . First, we calculate  $\phi$  from  $\triangle p't_p r$ :

$$\begin{aligned}
|\overline{p't_p}| &= |\overline{pt_p}| \cos \theta \\
|\overline{pt_p}| &= |\overline{rt_p}| \tan \theta \\
|\overline{p't_p}| &= |\overline{rt_p}| \tan \theta \cos \theta = |\overline{rt_p}| \sin \theta \\
\tan \phi &= \frac{|\overline{p't_p}|}{|\overline{rt_p}|} = \frac{|\overline{rt_p}| \sin \theta}{|\overline{rt_p}|} = \sin \theta \\
\phi &= \arctan(\sin \theta)
\end{aligned} \tag{1}$$

We can obtain  $h_{old}$  from  $\triangle qwr$ :

$$h_{old} = |\overline{qw}| = |\overline{qr}| \sin \rho \tag{2}$$

Observe that  $\angle q'uv = \angle rut_q$ , and therefore  $\angle vq'u = \phi$ . Now we can consider  $h_{new}$ :

$$\begin{aligned}
h_{new} &= |\overline{q'v}| = |\overline{q'u}| \cos \phi \\
|\overline{q'u}| &= |\overline{q't_q}| - |\overline{ut_q}| \\
|\overline{q't_q}| &= |\overline{qt_q}| \cos \theta \\
|\overline{qt_q}| &= |\overline{qr}| \sin(\rho + \theta)
\end{aligned}$$

So we have:

$$|\overline{q't_q}| = |\overline{qr}| \sin(\rho + \theta) \cos \theta \tag{3}$$

Now calculate  $|\overline{ut_q}|$ :

$$\begin{aligned}
|\overline{ut_q}| &= |\overline{rt_q}| \tan \phi \\
|\overline{rt_q}| &= |\overline{qr}| \cos(\rho + \theta) \\
|\overline{ut_q}| &= |\overline{qr}| \cos(\rho + \theta) \tan \phi
\end{aligned} \tag{4}$$

Combining (3) and (4) gives us  $|\overline{q'u}|$ :

$$\begin{aligned}
|\overline{q'u}| &= |\overline{q't_q}| - |\overline{ut_q}| \\
&= |\overline{qr}| \sin(\rho + \theta) \cos \theta - |\overline{qr}| \cos(\rho + \theta) \tan \phi \\
&= |\overline{qr}| (\sin(\rho + \theta) \cos \theta - \cos(\rho + \theta) \tan \phi)
\end{aligned}$$

We can now calculate  $h_{new}$ :

$$\begin{aligned}
h_{new} &= |\overline{q'v}| \\
&= |\overline{q'u}| \cos \phi \\
&= |\overline{qr}| \cos \phi (\sin(\rho + \theta) \cos \theta - \cos(\rho + \theta) \tan \phi)
\end{aligned}$$

Substituting (1) into this, we obtain:

$$\begin{aligned}
h_{new} &= |\overline{qr}| \cos(\arctan(\sin \theta)) (\sin(\rho + \theta) \cos \theta - \cos(\rho + \theta) \sin \theta) \\
&= |\overline{qr}| \cos \left( \arcsin \left( \frac{\sin \theta}{\sqrt{\sin^2 \theta + 1}} \right) \right) \sin((\rho + \theta) - \theta) \\
&= |\overline{qr}| \cos \left( \arccos \left( \sqrt{1 - \frac{\sin^2 \theta}{\sin^2 \theta + 1}} \right) \right) \sin \rho \\
&= |\overline{qr}| \sqrt{\frac{\sin^2 \theta + 1 - \sin^2 \theta}{\sin^2 \theta + 1}} \sin \rho \\
\therefore h_{new} &= \frac{|\overline{qr}| \sin \rho}{\sqrt{2 - \cos^2 \theta}} \tag{5}
\end{aligned}$$

We can now combine (2) and (5) to calculate the desired ratio:

$$\begin{aligned}
\frac{h_{old}}{h_{new}} &= \frac{|\overline{qr}| \sin \rho \sqrt{2 - \cos^2 \theta}}{|\overline{qr}| \sin \rho} \\
\therefore \frac{h_{old}}{h_{new}} &= \sqrt{2 - \cos^2 \theta} \tag{6}
\end{aligned}$$

Hence, when the angle between  $\psi$  and  $\psi^*$  is  $\theta$ , it holds that  $\overline{\text{dist}(\text{proj}(q, \psi^*), \text{proj}(p, \psi^*)\text{proj}(r, \psi^*))} \geq \text{dist}(q, \overline{pr})/\sqrt{2 - \cos^2 \theta}$ . Clearly, the angle between  $\ell$  and  $\overline{pr}$  must be lower than the angle between  $\psi$  and  $\psi^*$ , and thus the same inequality holds for angles less than  $\theta$ . Therefore  $\text{dist}(q, \overline{pr}) \geq \max_{\psi \in \Psi} \overline{\text{dist}(\text{proj}(q, \psi), \text{proj}(p, \psi)\text{proj}(r, \psi))} \geq \text{dist}(q, \overline{pr})/\sqrt{2 - \cos^2 \theta}$ .  $\square$

**Theorem 6.** *Given a real number  $\delta > 0$ , a  $(1 + \delta)$ -approximate Douglas-Peucker simplification can be computed in the line model in  $O(\frac{1}{\delta^2} n \log^2 n)$  time using  $O(\frac{1}{\delta^2} n)$  space, and in the line-segment model in  $O(\frac{1}{\delta^2} n \log^3 n)$  time using  $O(\frac{1}{\delta^2} n \log n)$  space.*

*Proof.* First, consider the line model. Construct a set of planes  $\Psi$  as described earlier in this section. Project the input points onto each of these planes. Now execute the algorithm of Section 3 on each plane, and whenever a split is to be performed, choose the split vertex with the maximum projected distance over all of the projection planes.

Let  $q$  be the chosen split vertex, and  $p$  and  $r$  be the endpoints of the current simplified line segment. By Lemma 7, we know that as long as  $\angle pqr > 2\theta$ , there is some plane in which the actual distance from  $q$  to the line through  $\overline{pr}$  in  $\mathbb{R}^3$  is at most  $\sqrt{2 - \cos^2 \theta}$  times the projected distance. The same result also gives us that in all planes the projected distance is at most equal to the actual distance. Since we are interested in the distance to a line and not a line segment, we can always choose  $p$  and  $r$  on the same line that are sufficiently far apart to guarantee that  $\angle pqr > 2\theta$ , and therefore Lemma 7 guarantees an approximation bound for every point-to-line distance.

Recall our definition of  $\theta$ :

$$\theta = \frac{\pi}{\left\lceil \frac{2\pi}{\arccos(\frac{1}{1+\delta})} \right\rceil}$$

By this definition,  $0 < \theta < \frac{\pi}{2}$ . Then we have:

$$\theta \leq \frac{\pi}{\left(\frac{2\pi}{\arccos\left(\frac{1}{1+\delta}\right)}\right)}$$

$$2\theta \leq \arccos\left(\frac{1}{1+\delta}\right)$$

$$\cos(2\theta) \geq \frac{1}{1+\delta}$$

$$1+\delta \geq \frac{1}{\cos(2\theta)}$$

$$\geq \cos(2\theta)$$

$$\therefore 1+\delta \geq \cos \theta \tag{7}$$

$$\text{and } 1+\delta \geq \sqrt{2 - \cos^2 \theta} \tag{8}$$

Combining equation (8) with equation (6) from Lemma 7 gives:

$$\frac{h_{old}}{h_{new}} \leq 1 + \delta$$

Therefore, in the line model, we have a  $(1 + \delta)$ -approximation. To obtain this, we must execute the algorithm of Section 3 on  $2k^2$  planes, where  $k = \lceil \pi / \arccos(1/(1 + \delta)) \rceil$ . But  $2k^2 \leq (1/\delta)^2 + \kappa$ , for a constant  $\kappa \approx 57$ . Hence the total time required is  $O(\frac{1}{\delta^2} n \log^2 n)$  and the amount of space used  $O(\frac{1}{\delta^2} n)$ .

It remains to consider the line-segment model. Clearly, whenever the closest point to  $q$  on the line through  $\overline{pr}$  lies between the segment endpoints  $p$  and  $r$ , the situation is identical to the line model, and the theorem holds. If this is not the case, then the actual distance in  $\mathbb{R}^3$  is  $\min(|\overline{pq}|, |\overline{qr}|)$ . Assume without loss of generality that the distance is  $|\overline{pq}|$ . Now consider the projection of  $p, q$  and  $r$  onto a given plane  $\psi^* \in \Psi$ , and define  $p', q'$  and  $r'$  as in Lemma 7.

If  $\angle p'q'r' \geq \frac{\pi}{2}$ , the distance between  $q$  and  $\overline{pr}$  in the plane  $\psi^*$  will be  $|\overline{p'q'}|$ . By Lemma 6 we know that  $|\overline{p'q'}| \geq |\overline{pq}| \cos \theta$ , and thus by equation (7)  $|\overline{pq}|/|\overline{p'q'}| \leq 1 + \delta$ . Furthermore, we can always choose  $\psi^* \in \Psi$  such that the angle between  $\ell$  and  $\overline{qr}$  is at most  $\theta$ , as in Fig. 6(d). Let  $x$  be the point of intersection between  $\ell$  and the line through  $\overline{pq}$ . Since both  $x$  and  $r$  are on  $\ell$ , they do not move when projected (i.e.  $x' = x, r' = r$ ). The point  $q$  is on the line segment  $\overline{px}$ , and therefore  $q'$  must be on the line segment  $\overline{p'x}$ . The point  $p'$  is closer to  $\overline{rx}$  than  $p$ , and hence  $\angle q'p'r' = \angle xp'r \geq \angle qpr \geq \frac{\pi}{2}$ .

Thus there is always a plane  $\psi^* \in \Psi$  such that the line-segment distance between  $q$  and  $\overline{pr}$  in  $\mathbb{R}^3$  is at most  $(1 + \delta)$  times the line segment distance of  $\text{proj}(q, \psi^*)$  from  $\text{proj}(p, \psi^*)\text{proj}(r, \psi^*)$ . Apart from differences in the 2-dimensional algorithms of Section 3, the line-segment model algorithm is equivalent to the line model algorithm, and hence runs in  $O(\frac{1}{\delta^2} n \log^3 n)$  time using  $O(\frac{1}{\delta^2} n \log n)$  space.  $\square$