アルゴリズム 36−10
（1993. 11. 25）

三角形族の中での直交探索

徳山 豪
日本ＩＢＭ東京基礎研究所

空間内の三角形または線分の集合に対して、軸方向の直交領域と交わる図形を全て列挙する問題（直交探索）に対して、効率の良いアルゴリズムを考える。もっとも興味深いのは、探索の効率が、データの個数以外の様々な幾何学的パラメターに依存している事である。

Orthogonal queries in triangles

Takeshi Tokuyama
IBM Research, Tokyo Research Laboratory
1623-14, Shimotsuruma, Yamato-shi, Kanagawa, 242 Japan. Email:ttoku@vnet.ibm.com

We present an efficient orthogonal query data structure in a set of segments or triangles in space. The most important feature of our results is that the efficiency of the data structure is highly dependent on the geometric properties of the input set, as well as its cardinality. (1) Given a set of n segments in d-dimensional space, we give a data structure of $O(m)$ space ($m \geq n \log^{d-1} n$) that allows us to count the segments intersecting an orthogonal query window W in $\tilde{O}(\sqrt{K/m})$ time. Here, K is the complexity of the arrangement of the images of segments projected onto axial subspaces. (2) Given a set of n triangles in d-dimensional space, we give a data structure of $O(m)$ space that allows us to report the triangles intersecting an orthogonal query window in $\tilde{O}(\sqrt{K/m} + \sqrt{M}/m^{1/3})$ time, not including the time for the output. Here, M is a parameter that coincides with the number of intersecting pairs of triangles if $d = 3$, and K is as the same as in (1) for the set of edges of triangles.

1 Introduction

In textbooks [19, 23], the orthogonal range-searching problem is stated as follows: "Given a set S of n objects in a d-dimensional space, construct a data structure for counting (or reporting) the objects intersecting a given orthogonal query window". For the case in which the objects are a set of points, the problem has been well investigated, and an $O(n \log^{d-1} n)$ space $O(\log^{d-1} n)$ time query structure can be obtained by using the range search tree method [15]. Chazelle reduced the space requirement by a facor of $O(\log \log n)$ for $d = 2$. The space-time tradeoff of orthogonal range searching is also an important topic [22, 21], and it is known that $\Omega((\log n / \log \log n)^{d-1})$ space is necessary (on a pointer machine) for a polylogarithmic query time structure [8].

For non-point objects, several results (theoretically [6], practically [12]) are known for orthogonal quereis in a set of orthogonal segments in a plane. The most general case, in which the objects and the windows are non-orthogonal, is studied as an application of space subdivisions and simplex range searching by Dobkin-Edelsbrunner [10].

In this paper, we study orthogonal queries in non-orthogonal objects, particularly (possibly intersecting) segments and triangles in d-dimensional space. In order to distinguish the orthogonal queries in a set of non-point objects from orthogonal queries in a set of points, we sometimes call it *orthogonal clipping*. Figures 1 and 2 give examples of two- and three-dimensional cases.

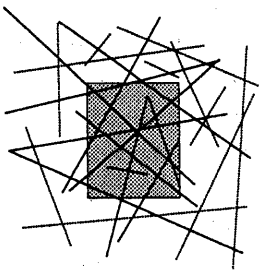

Figure 1: Orthogonal clipping of segments in a plane

An immediate application of orthogonal clipping is the design of clipping functions in CAD

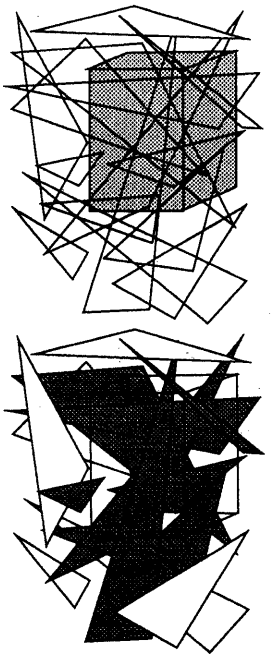

Figure 2: Orthogonal clipping of triangles in space

(when $d = 2$ or 3) and window systems [14]. It also gives efficient clipping of maps, which can be used in geographic data base systems and central navigation systems (when $d = 2, 3, 4$).

The orthogonal clipping problem can be reduced to the *dual* of the simplex range searching problem, and has a lower bound induced from that of simplex range searching [7]. In simplex range searching, the geometric properties of the data set have little effect on the efficiency of a query structure; indeed, the lower bound holds on an uniformly distributed point set. In contrast, we show that the geometric properties of the configuration of the input objects strongly influence the efficiency of orthogonal clipping. We define parameters indicating the geometric property of a configuration, and construct a data structure that has a much better performance if these parameters are small.

Let us start with a set of n segments in a plane. Let K be the complexity (i.e. the number of intersections and endpoints) of the arrangement of those segments.

If we can use $O(K+n \log n)$ space, an $O(\log n)$-time query structure can be constructed by using

a planar partition [16] and a persistent tree structure [20]. However, $O(K)$ space is often too huge. We study the space-query trade off so that we can store the data in a given amount of storage.

Because of a lower bound given by Chazelle [7], an $\Omega(n/\sqrt{m}\log n)$ query time is needed for an $O(m)$-space data structure (in a slightly more general model) if we consider the set of lines (where $K = n(n-1)/2$), since the problem involves the dual problem of range searching with respect to (non-isothetic parallel-sided) slabs. However, K is practically smaller than $O(n^2)$ in many cases. Moreover, it may also happen theoretically in many cases; for example, $K = O(n^{1.5})$ if we consider the set of $O(n)$ segments obtained by projecting the 1-skeleton of a Voronoi diagram of $\sqrt{n}$ points in space [13]. Chazelle's lower bound implies an $\Omega(\sqrt{K/m}/\log n)$ lower bound for orthogonal clipping, and it is shown to be tight within polylogarithmic factors even in higher dimensional cases.

In general, given a set of n segments in d-dimensional space, we define K as the sum of the complexity of arrangements obtained by projecting the segments to axis-parallel two-dimensional subspaces (i.e, x-y, y-z, and z-x planes). Obviously, $K = O(n^2)$, and smaller in practice.

Theorem 1.1 *For any $m \geq n\log^{d-1} n$, a data structure of $O(m)$ space and $\tilde{O}((K/m)^{\frac{1}{2}})$ query time can be constructed for orthogonal query on n segments in R^d.*

The problem is a kind of dynamic computational geometry [2], if we consider the d-th coordinate as the time parameter. The set of segments can be considered as a set of points that are linearly moving (inserted and deleted at certain time epochs). If we consider the range tree for $(d-1)$-dimensional orthogonal range searching in those points, the number of combinatorial changes is $\tilde{O}(K+n)$. Hence, if $m = K$, the theorem implies a logarithmic query *persistent search structure* [11] for orthogonal range searching. The theorem gives the space-query trade-off for the persistent structure.

Our data structure is a hierarchical structure based on a segment tree, each of whose nodes has a simplex range-searching structure. Moreover, we use a clustering of the set of segments into sparse sets and dense sets, and also balancing of the space complexity for the simplex range-searching structure in each node in order to obtain the space complexity of the total data structure.

Next, we consider the orthogonal query in a set of triangles (called a *castle in the air* [4]) in d-dimensional space. Let K be the complexity that defined above associated with the set of edges of triangles, and let M be the number of intersecting pairs of triangles.

Theorem 1.2 *For any $m \geq n\log^{d-1} n$, there exists a data structure of $O(m)$ space such that an orthogonal reporting query is can be done in $\tilde{O}((K/n)^{\frac{1}{2}} + \sqrt{M}/n^{1/3})$ time, excluding the time for output.*

As a corollary, we can query the L_∞-nearest triangle to the query point in $\tilde{O}((K/n)^{\frac{1}{2}}+\sqrt{M}/n^{1/3})$ time.

The rest of the paper is organized as follows: in Section 2, we discuss the *tubular hammock query* which is the main tool for devising algorithms sensitive to K and M. In Section 3 and 4, we deal with orthogonal queries in segments and triangles respectively.

2 Hammock query

In this section, we give a construction of an efficient data structure for the *tubular hammock query* defined below. Although we only use it for the case where $d = 2, 3$ in orthogonal clipping, we formulate it in general dimensional case. We consider a d-dimensional space with an orthonormal coordinate system $(x_1, x_2, .., x_d)$. Let $\mathcal{H}$ be an arrangement of n hyperplanes in d-dimensional space. Given a $(d-1)$-dimensional convex region Q, we define a tube $Z(Q) = \{\vec{x} \in R^d | (x_1, .., x_{d-1}) \in Q\}$. We call $\mathcal{Z}(Q) = Z(Q) \cap \mathcal{A}(\mathcal{H})$ a *tubular hammock*, or a *hammock* for short. If $d = 2$, it coincides with the hammock defined by Chazelle [5]. Let M be the sum of n and the number of pairs of hyperplanes intersecting inside the hammock. Obviously, $M = O(n^2)$.

Hammock query (Figure 3) Given a query segment lying in $Z(Q)$, count (or report) the hyperplanes intersecting it.

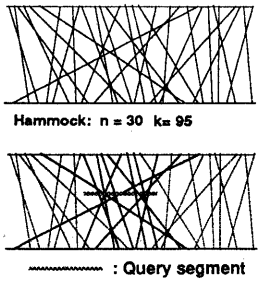

Figure 3: Hammock query

In the dual setting, n hyperplanes are transformed into n points in the dual space. The segment intersection counting query is therefore a special case of simplex range searching. We use the following result of Matoušek [18]:

Theorem 2.1 *There exists an $O(n)$ space and an $O(n^{1-1/d})$ query-time range-searching data structure. The preprocessing time is $O(n^{1+\delta})$ for any positive constant δ. For $n < m < n^2$, there exists an $O(m)$ space and an $O(\frac{n}{m^{1/d}}\log^3 n)$ query-time range-searching structure, constructed in $O(n^{1+\delta} + m(\log n)^{\delta})$ preprocessing time. The query time is reduced to $O(\frac{n}{m^{1/d}}\log^{1.5} n)$ if the ranges are wedges or slabs.*

We devote the rest of this section to proving the following theorem:

Theorem 2.2 *There exists an $O(m)$-space data structure $m \geq n$ such that the number of hyperplanes intersected by a query segment in the tube can be given in $O((1+\frac{\sqrt{M}}{m^{1/d}})\log^{2+1/d} n)$ time. The hyperplanes are reported in an additional time proportional to the output size.*

Proof The outline of the construction of the data structure is decomposed in stages as follows: In the first stage, we classify the hyperplanes into two classes, *tame* and *wild*, by using a parameter r initially set as $3M/n$, and give a data structure for the tame hyperplanes. In the next stage, we reclassify the wild hyperplanes into *tame* and *wild*, using a new parameter r, and give a data structure for the (new) tame hyperplanes. We continue this process until there are no wild segments.

Let us give the initial stage construction. We choose an arbitrary point q of Q as the reference point. Let $l(q)$ be the line penetrating q and orthogonal to $x_d = 0$. We sort the intersection points of the hyperplanes with the line $l(q)$, and give a total ordering to the set of hyperplanes according to it. One by one, we remove the hyperplanes (called *wild*) that intersect more than $2M/n$ hyperplanes, until no hyperplane intersects $2M/n$ other hyperplanes(Figure 4). So far, there are at most $s = n/2$ wild hyperplanes. The rest of the hyperplanes are temporally classified as *tame*. We choose every $r = 3\lfloor M/n \rfloor$-th tame hyperplane as a partition hyperplane (Figure 5). It is easy to see that a pair of partition hyperplanes never intersect; thus the partition planes decompose the tube into at most n/r trapezoids, which we call *buckets* (Figure 6). We reclassify hyperplanes intersecting partition hyperplanes as wild.

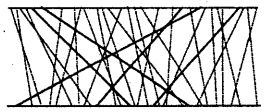

Figure 4: Wild segments (before reclassification)

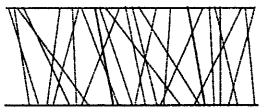

Figure 5: Partition lines

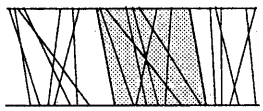

Figure 6: Bucket partition of segments

At most $(n-s)/r \times 2M/n = 2(n-s)/3$ hyperplanes are reclassified; hence, the total number of wild hyperplanes is at most $2n/3 + s/3 < 5n/6$. By definition, each hyperplane is contained in a bucket; thus, we have a clustering of hyperplanes. Each bucket contains at most r hyperplanes. We provide Matoušek's $O(rm/n)$-space $O(r(rm/n)^{-1/d})$-query data structure (called the range-query structure) for each bucket. We also construct an interval tree over $\{1, 2, .., n/r\}$,

so that the tame hyperplanes in the union of buckets $B_i, B_{i+1}, .., B_j$ are retrieved as a union of $\log n$ sets (named *principal sets*) for any $i < j$, where B_i is the i-th bucket from the left. This approach requires $O(n \log n)$ space, but the requirement can be easily reduced to $O(n)$ space for counting queries by storing only the number of hyperplanes to buckets. For reporting queries, the space can also be reduced to $O(n)$, by representing the point set as a disjoint union of $O(n/\log n)$ subsets of size $O(\log n)$ and storing the pointers to those point sets in each bucket instead of each point set.

Lemma 2.3 *If the above structure is used, the tame hyperplanes intersecting a query segment can be queried in $O(\log n + r(rm/n)^{-1/d})$ time.*

Proof When the query segment is given, we first count the intersecting tame hyperplanes. We first locate the buckets in which the endpoints of the segment lie (this takes $O(\log n)$ time). In each such bucket, we count the intersections in $O(r(rm/n)^{-1/d})$ time by using the range-query structure. The hyperplanes in the buckets penetrated by the segment are reported (or counted) by using the segment tree structure. □

Next, we must deal with wild hyperplanes. Let there be n_1 wild hyperplanes and M_1 intersecting pairs of wild hyperplanes. We set $r_1 = 3M_1/n_1$. If $r_1 \leq 3M \log^d n/n$, we create the same data structure as given in the initial stage, and go to the next stage. Note that we use n_1 space for the data structure in this case, and that the query time in this stage is $O((M/n)^{1-1/d} \log n + \log n)$. It is easy to see that $(M/n)^{1-1/d} \leq M^{1/2}/n^{1/d}$, because $M \leq n^2$.

Otherwise, $r_1 = 3M_1/n_1 > 3M \log n/n$, and we make the same bucket subdivision, but give an $O(\frac{mr_1}{n_1 \log n})$ space range searching structure for each bucket. The space complexity is $O(m/\log n)$ and the query time is $O(r_1(M_1 m/n_1^2 \log n)^{-1/d} \log^{1.5} n)$.

Because $M_1 \leq n_1^2$, $r_1(M_1 m/n_1^2)^{-1/d} = m^{-1/d} n_1^{2/d-1} M_1^{1-1/d} \leq m^{-1/d} M_1^{1/2}$. Hence, the query time is $O(M_1^{1/2} m^{-1/d} \log^{1.5+1/d} n)$.

Because the number of wild hyperplanes is reduced to at most half in each stage, the number of stages is at most $\log n$. Thus, the total space complexity of the structure is $O(n)$, and the query time is $O(\frac{\sqrt{M}}{m^{1/d}} \log^{2+1/d} n)$. □

We make the following claim about the preprocessing time.

Proposition 2.4 *The hammock query structure can be constructed in $O(n^{1+\epsilon}) + \tilde{O}(m)$ time for $d \leq 3$ and in $O(n^2) + \tilde{O}(m)$ time for $d \geq 4$, provided that Q has constant number of faces.*

Remark. It is not crucial that $Z(Q)$ should be tubular. In fact, it suffices that $Z(Q)$ is convex, and that there exists a line segment located in $Z(Q)$ intersecting all hyperplanes of $A(H)$ that intersect $Z(Q)$.

Remark 2. For intersection query with respect to a general line segment in a set of segments in a plane, Dobkin-Edelsbrunner [10] gives an $O(n)$-space $O(n^{0.695})$-query time algorithm for the counting query in a set of segments. However, the solution is independent of K, and the best known data structure for the counting segment intersection query among non-intersecting line segments using $O(npoly \log n)$ space takes $\tilde{O}(\sqrt{n})$ query time [1].

3 Orthogonal clipping

3.1 Orthogonal clipping of segment

Given a set S of n segments in d-dimensional space R^d, we preprocess it, so that the set of segments intersecting an window W parallel to the axis is counted (or reported) efficiently.

Let $K_{i,j}$ be the complexity of the arrangement obtained by projecting segments to the principal two-dimensional subspaces spanned by x_i and x_j. We define $K = \sum_{i<j} K_{i,j}$. Obviously, $K = O(n^2)$.

Let N be the number of segments intersecting with W. Let N_1 be the number of endpoints in W, and N_2 the number of intersections between segments and δW (the boundary of W). If a segment intersects δW twice, we count both intersections. The following lemma is easy to see:

Lemma 3.1 $N = 1/2(N_1 + N_2)$

It is easy to count N_1, since it is the orthogonal query on a point set. Hence, orthogonal clipping of segments can be reduced to the problem of counting (or reporting) the segments with each facet. We give d-dimensional space a Cartesian

coordinate system $\{x_1, .., x_{d-1}, z\}$. A segment is called (α, β)-bound if the segment has its lower (resp. upper) endpoint on $z = \beta$ (resp. $z = \alpha$). The orthogonal clipping problem is reduced to the following:

Flat orthogonal clipping: Let $\mathcal{L}$ be a set of n (α, β)-bound segments in d-dimensional space $(d \geq 2)$. Given (ζ, W), where W is an orthogonal region of R^{d-1}, report the segments of $\mathcal{L}$ whose points of intersection with $z = \zeta$ are located in W.

Proposition 3.2 *There exists a data structure for flat orthogonal clipping in $O(m)$ space $(m \geq n \log^{d-2} n)$ and $O((1 + \frac{K}{m})^{1/2} \log^{(5d-5)/2} n)$ query time.*

Although above proposition is the key proposition, we omit the proof of it because of space limitation.

Theorem 3.3 *Suppose that we are given n segments in d-dimensional space. For any $m \geq n \log^{d-1} n$, a data structure with $O(m)$ space is constructed, and orthogonal clipping query is done in $O((1 + \frac{K}{m})^{1/2} \log^{(5d-3)/2} n)$ time.*

Proof Clearly, it suffices to prove the theorem for the case in which $m = n \log^{d-1} n$. We use the segment tree trick [19]. A hyperplane (or its subset) orthogonal to the z-axis is said to be horizontal. Because of Lemma 3.1, it suffices to count (or report) the segments intersecting the upper horizontal face F of the query window W. We project segments of S onto the z-axis to obtain a set $\mathcal{I}$ of intervals. Let $z_1, .., z_{2n}$ be the set of points, which are projected images of endpoints, on the z-axis. We make an interval tree according to this set of points.

We store the intervals of $\mathcal{I}$ in an *interval tree* to create a *segment tree*. An interval in $\mathcal{I}$ is subdivided into at most $\log n$ principal intervals of the interval tree. Accordingly, each segment of S is also cut into at most $\log n$ subsegments, and stored in the associated nodes. The subsegments stored in a node naturally form a hammock. Let the hammock associated with a node q contain $n(q)$ segments and $K(q)$ intersections. We make a data structure of $O(K/n)^{1/2} \log^{2d-2} n)$ query time, using $O(\max\{n(q) \log^{d-2} n, (K(q) n \log^{d-1} n)/K\})$ space. It is clear that the total space complexity of the data structure is $O(n \log^{d-1} n)$.

The query is performed as follows: Suppose a face F of the query window lies on the horizontal hyperplane $y = a$. Then, we find all the principal intervals containing a. We perform hammock query for the hammock at the nodes associated with these principal intervals. The total query time is obviously $O((K/n)^{1/2} \log^{2d-1} n)$ time, which is $O((K/n \log^{d-1} n)^{1/2} \log^{(5d-3)/2} n)$. □

3.2 Orthogonal clipping of triangles

Orthogonal clipping of triangles: Given a set S of n triangles in d-dimensional space R^d, we preprocess it, so that the set of triangles intersecting an axis parallel orthogonal window W is reported efficiently.

Note that we consider only the reporting problem, in contrast to the orthogonal clipping of segments. Although we deal mainly with the case in which $d = 3$, we start with the general case. Because of limitation of space, we omit proofs in this section.

Given a query window W, a triangle T intersects W if (1) one of the vertices of T is located in W, (2) one of the edges of T intersects a facet of W or (3) T intersects one of the $(d-2)$-dimensional faces (called *ridges*) of W.

Cases (1) and (2) can be handled by using clipping in segments. Therefore, we concentrate on case (3).

Let F be a ridge of W. We can assume that F is orthogonal to the two-dimensional space H spanned by x_1 and x_2. Let p be the point, that is the image of F via the orthogonal projection to H.

If F intersects T, then the projection of T onto H contains p. However, the converse is not true. Let $aff(T)$ be the two-dimensional affine space containing T.

Lemma 3.4 *T intersects F if and only if the image of T projected onto H contains p and $aff(T)$ intersects F.*

We first construct the set $Pr(S)$ of images of a set S of triangles projected onto H, that makes a set of triangles in the plane H. Next, we query

the triangles in $Pr(S)$ containing p on H, so that the output is given as a union of *canonical* subsets of $Pr(S)$. We can ensure that a point is contained in at most $O(\log n)$ canonical subsets.

Let us describe the query structure for the set of triangles containing q in a plane. We can cut each triangle in S with a horizontal line, and make a pair of triangles each of which has a horizontal edge. Hence, we may assume that each triangle of S has a horizontal edge. Note that this transformation is not permitted if we consider the counting query.

Let $y_1, .., y_{2n}$ be the set of the y coordinate values of the vertices of the triangles. We construct an interval tree on $y_1, .., y_{2n}$. Then, we can reduce the problem to the following *trapezoidal query* with adding an $O(\log n)$ factor for the space complexity. **Trapizoidal query** Given a set of n trapezoids in a x-y plane, whose upper edges are on the horizontal lines $y = \alpha$ and whose lower edges are on $y = \beta$, for a query point p, report all trapezoids containing p.

Proposition 3.5 *We can construct a data structure for trapizoidal query with $O(m)$ space and $O((K/m)^{1/2} \log^3 n)$ query time for $K \geq m \geq n \log n$, where the output is formed as a union of sets of trapezoids.*

We now have the set of canonical subsets, and we describe how to use it in the clipping problem. For a canonical subset A, we define the corresponding subset of triangles as $Pr^{-1}(A)$. Let $J(A)$ be the set of all two-dimensional affine subspaces corresponding to the triangles of $Pr^{-1}(A)$.

Because of Lemma 3.4, it suffices to report the set of affine spaces in $J(A)$ intersecting F for each A. Combining with a result of range searching, we have the following:

Theorem 3.6 *There exists an $O(m)$-space data structure ($m \geq n \log^{d-1} n$) such that orthogonal clipping in n triangles is done in $\tilde{O}((K/m)^{1/2} + M^{1/2}/m^{1/3})$ time, where (1) M is the number of pairs of triangles intersecting each other if they are projected to axial three-dimensional spaces, and (2) K is the number of pairs of edges of triangles intersecting each other if they are projected to axial two-dimensional spaces.*

We briefly investigate the orthogonal clipping of simplices in three-dimensional space. Given n simplices in three-dimensional spaces, we use the above method to report the simplices intersecting a query window W except those tha contain W. Hence, it suffices to report the simplices containing a particular vertex v of the window, besides clipping the faces of the simplices. This can be done similarly to the clipping in triangles as follows: We project all the faces of the triangles to the x-y plane, and report the triangles containing the projected image of v as a union of canonical sets. For each canonical set, we project back the triangles, and make a tubular hammock. We locate the point v in the hammock, and report the simplices (represented by pairs of planes) containing v.

Corollary 3.7 *There exists an $O(m)$-space data structure ($m \geq n \log^2 n$) such that the orthogonal clipping in n simplices in three-dimensional space is done in $\tilde{O}((K/m)^{1/2} + M^{1/2}/m^{1/3})$ time, where (1) M is the number of pairs of simplices intersecting each other, and (2) K is the number of pairs of edges of triangles intersecting each other if they are projected to an axial two-dimensional subspace.*

4 Nearest-neighbour search

Let us consider a set of triangles in d-dimensional space, and consider the triangle nearest to a query point with respect to L_∞ metric. This problem can be solved by *parametric searching*, using the orthogonal clipping algorithm.

Proposition 4.1 *There exists an $O(m)$-space data structure ($m \geq n \log^{d-1} n$) such that the L_∞-nearest triangle to a query point is found in $\tilde{O}((K/m)^{1/2} + M^{1/2}/m^{1/3})$ time, using the notation introduced in the previous section.*

The following is a problem in a central navigation system: "Among n moving objects in (three-dimensional) space, query the object approaching nearest to a query point during a given query time-interval." If the objects move piecewise linearly, the problem is to report the line segment (in 4-dimensional space) nearest to the query point (precisely speaking, to the vertical line segment which is the trajectory of the query point). Thus, if we consider the L_∞ distance, this problem can also be solved in $\tilde{O}((K/m)^{1/2})$ time.

5 Concluding remarks

There are several problems for future research: (1) Investigate orthogonal clipping in simplices in d-dimensional space. (2) For higher-dimensional cases, let N be the complexity of a hammock. Then, can we obtain a bound of $\tilde{O}((N/m)^{1/d})$, which is better than $\tilde{O}(M^{1/2}/m^{1/d})$, for the hammock query? (3) How efficiently can we query if we use $O(n)$ space? This problem is difficult even for a set of points, unless we permit $O(n^{\epsilon})$ query time. (4) Give a practical algorithm; we should avoid using sophisticated range searching methods that have huge constant factors. We can use the practical bucketing technique [3] for the subroutine, although we lose the theoretical bound.

Acknowledgement

The author gratefully acknowledges the advice of K. Kuse, who introduced him the problem of clipping a straight-line graph with moderate intersections in application to window systems. He also thanks P. K. Agarwal for suggesting a way of simlifying the data structure.

References

[1] P. Agarwal, Ray Shooting and Other Applications of Spanning Trees with Low Stabbing Number, *Proc. 5th ACM Comput. Geom.,* (1989), 315-325.

[2] M. Atallah, Some Dynamic Computational Geometry Problems, *Computers and Mathematics with Applications,* **11** (1985), 1171-1181.

[3] T. Asano, M. Edahiro, H. Imai, M. Iri, and K. Murota, Practical Use of Bucketing Techniques in Computational Geometry, in *Computational Geometry* ed. G. Toussaint , Elsevier, North Holland (1985), 153-194.

[4] B. Aronov and M. Sharir, Triangles in Space or Building (and Analyzing) Castles in the Air, *Combinatorica* **10**, (1990) 137-173. (1988), 381-391.

[5] B. Chazelle, Reporting and Counting Segment Intersections, *J. Comput. System Sci.* **32** (1986) 156-182.

[6] B. Chazelle, Filtering Search: A New Approach to Query-Answering *SIAM J. Comput.* **15** (1986) 703-724.

[7] B. Chazelle, Lower Bounds on the Complexity of Polytope Range Searching, *J. Amer. Math. Sci.*,**2** (1989) 637-666.

[8] B. Chazelle, Lower Bounds for Orthogonal Range Searching I. The Reporting Case, *J. ACM*,**37** (1990) 200-212.

[9] B. Chazelle, M. Sharir, and E. Welzl, Quasi-Optimal Upper Bound for Simplex Range Searching and New Zone Theorems, *Proc. 6th ACM Comput. Geom.* (1990) 23-33.

[10] D. Dobkin and H. Edelsbrunner, Space Searching for Intersecting Objects, *Proc. 25th IEEE FOCS* (1984), 387-392.

[11] J. Driscoll, N. Sarnak, D. Slator, and R. Tarjan, Making Data Structure Persistent, *Proc. 18th ACM STOC* (1986), 109-120.

[12] M. Edahiro, K. Tanaka, T. Hoshino, and T. Asano, A Bucketing Algorithm for the Orthogonal Segment Intersection Search Problems and Its Practical Efficiency, *Proc. 3rd ACM Comput. Geom.* (1987) 258-267.

[13] T. Hirata, J. Matoušek, X. Tan, and T. Tokuyama, Complexity of Projected Images of Convex Subdivisions, *Proc. 4th CCCG* (1992) 121-126.

[14] K. Kuse, private communication.

[15] G. Lueker, A Data Structure for Orthogonal Range Queries, *Proc. 19th IEEE FOCS* (1978), 28-34.

[16] K. Mulmuley, A Fast Planar Partition Algorithm, II, *Proc. 5th ACM Comput. Geom.* (1989), 33-43.

[17] J. Matoušek, Efficient Partition Trees, *Proc. 7th ACM Comput. Geom.* (1991), 1-9.

[18] J. Matoušek, Range Searching with Efficient Hierarchical Cuttings, *Proc. 8th ACM Comput. Geom.* (1992), 276-285.

[19] F. Preparata and M. Shamos, *Computational Geometry, an Introduction*, 2nd edition, Springer-Verlag (1988).

[20] N. Sarnak and R. Tarjan, Planar Point Location Using Persistent Search Trees, *Comm. ACM* **29** (1986), 669-679.

[21] P. Vaidya, Space-Time Tradeoffs for Orthogonal Range Queries, Proc. 17th ACM STOC (1985), 169-174.

[22] A.C. Yao, Space-Time Tradeoff for Answering Range Queries, Proc. 14th ACM STOC (1982), 128-136.

[23] F. Yao, Computational Geometry, in Handbook of Theoretical Computer Science A, van Leeuwen ed. Elsevier (1991).