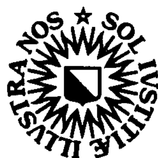
New Results on Binary Space Partitions in the Plane

M. de Berg and M. de Groot and M. Overmars

UU-CS-1994-51
December 1994



Utrecht University
Department of Computer Science
Padualaan 14, P.O. Box 80.089,
3508 TB Utrecht, The Netherlands,
Tel. : ... + 31 - 30 - 531454

New Results on Binary Space Partitions in the Plane

M. de Berg and M. de Groot and M. Overmars

Technical Report UU-CS-1994-51
December 1994

Department of Computer Science
Utrecht University
P.O.Box 80.089
3508 TB Utrecht
The Netherlands

ISSN: 0924-3275

New Results on Binary Space Partitions in the Plane

Mark de Berg* Marko de Groot* Mark Overmars*

Abstract

We prove the existence of linear size binary space partitions for sets of objects in the plane under certain conditions that are often satisfied in practical situations. In particular, we construct linear size binary space partitions for sets of fat objects, for sets of line segments where the ratio between the lengths of the longest and shortest segment is bounded by a constant, and for homothetic objects. For all cases we also show how to turn the existence proofs into efficient algorithms.

Keywords: binary space partition, fat objects, homothets, line segments.

1 Introduction

Problems where the input is a set of objects in the plane or in space are often solved by partitioning the space into subspaces, and then solving the problem on the subspaces recursively. A natural way to perform the partitioning is to make a linear cut of the space, that is, to split the space (and possibly some of the objects) with a hyperplane. The splitting process is repeated for each of the half-spaces with the corresponding sets of (fragments of) objects. This continues until there is at most one fragment of an object left in each of the subspaces. Such a partitioning scheme is called a *binary space partition*, or *bsp* for short, see figure 1. A binary space partition is naturally modeled as a tree structure: a *binary space partition tree*, or *bsp tree*. The nodes of the bsp tree store the splitting hyperplanes; the leaves correspond to the cells (subspaces) in the final partitioning and store (the fragment of) the object that is left in that cell. Binary space partition trees are popular in many application areas. In computer graphics, for example, bsp trees are used for efficient implementations of the painter's algorithm [10]. In this algorithm one tries to "paint" the objects in a back-to-front order onto the screen. Thus objects in the

*Dept. of Computer Science, Utrecht University, P.O.Box 80.089, 3508 TB Utrecht, the Netherlands.
Supported by the Dutch Organisation for Scientific Research (N.W.O.) and by ESPRIT Basic Research Action No. 7141 (project ALCOM II: *Algorithms and Complexity*).

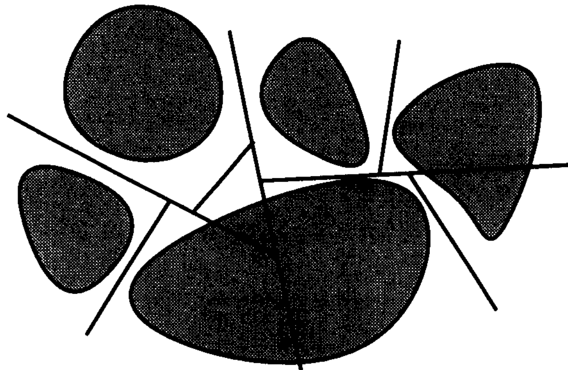


Figure 1: a binary space partition

front are painted on top of objects in the back, resulting in a correct view of the scene. Note that a depth order does not always exist, since there can be cyclic overlap among the objects. When the objects are stored in a bsp tree, however, then a back-to-front order for the object fragments in the tree can easily be obtained for any given viewing direction by traversal of the tree. Other uses of bsp trees in computer graphics include shadow generation [6]. In geometric modelling bsp trees have been used for the implementation of set operations on polyhedra [13, 19] and in robotics for (approximate) cell decomposition methods [1].

The efficiency of algorithms that are based on binary space partitions depends strongly on the *size* of the bsp that is used, that is, on the total number of fragments created by the final partitioning. Hence, it is important to choose the splitting hyperplanes in such a way that the fragmentation is kept as low as possible.

Several results in this direction have been obtained by Paterson and Yao [15, 16]. They proved that for any set of n line segments[1] in the plane there exists a bsp of size $O(n \log n)$, when the segments are orthogonal then there exists a bsp of $O(n)$ size. In both cases the bsp can be computed in $O(n \log n)$ time. The result on arbitrary segments had, in fact, already been proved by Preparata in his paper on point location [17]. For orthogonal objects the same result (with a slightly better constant for the combinatorial bound) was achieved by d'Amore and Franciosa [7]. Paterson and Yao have conjectured that any set of segments in the plane admits a bsp of linear size, but until now this conjecture is still open. Paterson and Yao also proved bounds on bsp trees in higher dimensions: they have shown that any set of $(d-1)$-simplices in d-space admits a bsp of size $O(n^{d-1})$, and any d-dimensional configuration of n axis-parallel line segments, or orthogonal rectangles, admits a bsp of size $O(n^{d/(d-1)})$. In three-dimensional space they have given lower bound constructions which match their upper bounds, namely $\Omega(n^2)$ for the general case and $\Omega(n\sqrt{n})$ for the axis-parallel case. Note that these lower bounds do not apply in higher

[1]Here and in the sequel we assume that the input objects are disjoint, since otherwise no bsp exists where each cell contains at most one object.

dimensions, unless we require that a binary space partition completely decomposes lower-dimensional subconfigurations.

In this paper we prove the existence of linear size binary space partitions for three classes of objects in the plane, namely for fat objects, for line segments where the ratio between the lengths of the longest and the shortest one is bounded by a constant, and for homothetic objects. We also give efficient algorithms for these three cases. For fat objects and for line segments with bounded length ratios our algorithms run in $O(n \log n \log\log n)$ time and for homothets we obtain an $O(n \log n)$ algorithm.

The methods for fat objects and for homothets, described in sections 2 and 3, both transform the problem to a problem on orthogonal line segments. The latter problem is then solved using the algorithm of Paterson and Yao [16]. In the case of homothets the transformation is relatively simple. For fat objects the transformation is more involved. Here we prove the following result which is of independent interest: for any set of fat objects in the plane there exists a linear size orthogonal subdivision such that any region of the subdivision contains a constant number of objects.

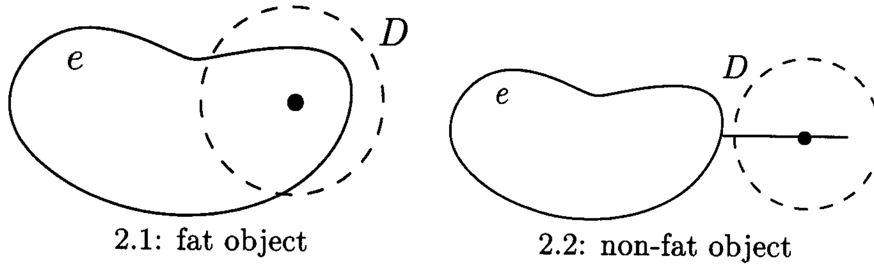
The strategy for segments with bounded length ratio, described in section 4, is to first construct a partitioning where each segment is cut at least once. This implies that all segments inside any cell of this initial partitioning are attached to the boundary of that cell. We then show how to solve the subproblem inside a cell efficiently.[2]

The merit of this work is twofold. First of all, we prove that the conjecture of Paterson and Yao is true for several interesting special cases. The techniques that we use in our proofs—especially for the case of segments with bounded length ratios—might be useful to prove the general conjecture, although we have not been able to do so yet. Secondly, although tight worst-case bounds are known on binary space partitions in 3-space, we feel that this case has not been solved satisfactorily from a practical point of view. Indeed, the quadratic lower bound example given in [15] is rather artificial. It would be very useful if one could establish better bounds for three-dimensional bsp trees under certain conditions on the input objects that are satisfied in practical situations. We believe that two of the conditions that we study in the planar case—fatness and bounded length ratios—are realistic for many three-dimensional applications and we hope that our planar results can be generalized to three dimensions.

2 Fat Objects

In this section we address the problem of finding a binary space partition for a set of n non-intersecting fat objects of constant complexity in 2-space. Intuitively, an object is called *fat* if it contains no extremely "skinny" parts. (See below for a more formal definition of fatness.) In practice, this is often the case. Our method for fat objects consists of three stages. In the first stage we transform the problem on non-intersecting fat objects to a problem on non-intersecting orthogonal line segments. The next stage solves the latter problem using the orthogonal partitioning algorithm of Paterson and Yao [16]. The final

[2] For the last step Chazelle [3] has independently obtained similar results.



2.1: fat object 2.2: non-fat object

stage completes this orthogonal subdivision to an $O(n)$ size binary space partition for the set of fat objects. Fat objects can formally be defined as follows, see van der Stappen et al.[20].

Definition 2.1 *Let $e \subseteq R^2$ be an object and let k be a positive constant. The object e is k-fat if for each disc D that has its center inside e and whose boundary intersects the boundary of e the following holds: $k \cdot area(e \cap D) \geqslant area(D)$. We call a set E of objects a set of fat objects if there is a constant k such that all objects in E are k-fat.*

Let E be a set of n fat objects of constant complexity. The following property [20] of fat objects will be crucial to our solution.

Lemma 2.1 *Let E be a set of non-intersecting fat objects in the plane and let $c > 0$ be a constant. Let $e \in E$ be an object and let δ be the diameter of its minimal enclosing circle. Then the number of objects $e' \in E$ with a minimal enclosing circle with diameter at least δ intersecting any rectangular box with side length $c \cdot \delta$ is bounded by a constant.*

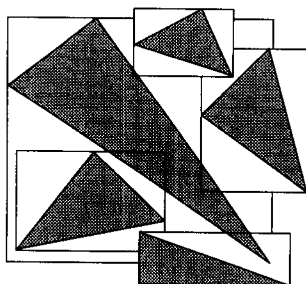
We now transform the set E of fat objects to a set $L = L(E)$ of axis-parallel line segments. First we sort the objects in order of increasing size of their minimal enclosing circle. Let $\{e_1, e_2, \ldots, e_n\}$ be the sorted set of objects and let b_i $(1 \leqslant i \leqslant n)$ be the bounding box of e_i, that is, b_i is the smallest axis-parallel rectangle that contains e_i. Lemma 2.1 shows that only $O(1)$ objects from $\{e_{i+1}, \ldots, e_n\}$ will intersect the interior of b_i. Second, for each edge s of b_i we add $s - \cup_{j=1}^{i-1} b_j$ to L. Intuitively, it is as if the boxes are added one by one, where b_i is added *behind* the bounding boxes that already have been added, as in figure 2.3. So, the set L consists of parts of the edges of the bounding boxes b_i.

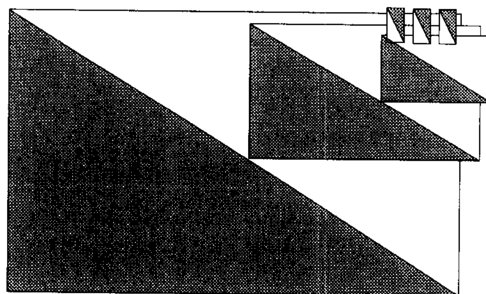
The arrangement $\mathcal{A}(L)$ induced by the line segments in L has the following property.

Lemma 2.2 *The interior of any cell C in $\mathcal{A}(L)$ is intersected by $O(1)$ objects $e \in E$.*

Proof: This is easy to see if we take the intuitive view of the construction, where we add the bounding boxes one behind the other. In this view we create new cells of $\mathcal{A}(L)$ when we add a bounding box b_i. These new cells are the portions where b_i is "visible". Notice that such a cell will not change by the addition of subsequent bounding boxes, since we only add parts of the edges of these boxes that lie outside the current union of bounding boxes. Since the cells where b_i is visible lie outside $\cup_{j=1}^{i-1} b_j$ they cannot be intersected by

any object e_j with $j < i$. Furthermore, by Lemma 2.1 each cell is intersected by only a constant number of objects e_j with $j > i$. □


2.3: the arrangement


2.4: quadratic complexity

If we now apply the orthogonal partition algorithm of Paterson and Yao to the set L of orthogonal segments, then we create a linear size binary space partition such that any cell in this partition contains only a constant number of objects from E. Unfortunately the number of segments in L can be $O(n^2)$, as can be seen in Figure 2.4. Many cells in the arrangement $\mathcal{A}(L)$, however, appear to be empty, that is, their interiors do not intersect any object. Indeed, we prove that if we only consider the cells that are intersected by the boundary of at least one object then the complexity of the arrangement reduces to $O(n)$. To analyze the complexity of all such cells in $\mathcal{A}(L)$, we distinguish between *rectangular cells*, which have exactly four vertices, and *non-rectangular cells*, which have more than four vertices.

Lemma 2.3 *The complexity of all non-rectangular cells in $\mathcal{A}(L)$ is $O(n)$ in total.*

Proof: There are two types of vertices in $\mathcal{A}(L)$: corners of bounding boxes and T-junctions, which arise when an edge of a bounding box is placed "behind" another bounding box. The second type of vertex is convex with respect to all three cells that it is incident to, the first type of vertex is convex with respect to one of the incident cells and reflex with respect to the other. Hence, the total number of reflex vertices over all cells in $\mathcal{A}(L)$ is $4n$. It remains to observe that the complexity of a non-rectangular cell is linear in the number of reflex vertices of that cell. □

To bound the number of rectangular cells, we bound the number of intersection points that appear between the boundaries of the bounding boxes and the boundaries of the input objects. Using Lemma 2.1 we can show the following.

Lemma 2.4 *The total number of rectangular cells in $\mathcal{A}(L)$ that are intersected by the boundary of an object in E is $O(n)$.*

Proof: If a cell is intersected by the boundary of an object then either it completely contains the object, or the boundary of the object intersects the boundary of the cell. Obviously, there are at most n cells of the first type. As for the second type, we observe that each object is of constant complexity, so its boundary intersects the boundary of a bounding box only $O(1)$ times. We add only those parts of the edges of a bounding box to L that lie outside the bounding boxes of all previously added boxes. These facts together with Lemma 2.1 imply that there are $O(n)$ intersections in total between edges in L and boundaries of objects in E. Hence, the number of rectangular cells whose boundary is intersected by the boundary of an object in E is $O(n)$ in total. □

We discard from L all those segments that contribute neither to the boundary of any non-rectangular cell nor to the boundary of any rectangular cell that is intersected by the boundary of an object. Now we are left with an orthogonal subdivision $\mathcal{A}(L)$ of linear complexity in total. Each cell of the arrangement is intersected by only a constant number of object boundaries. We obtain the following lemma.

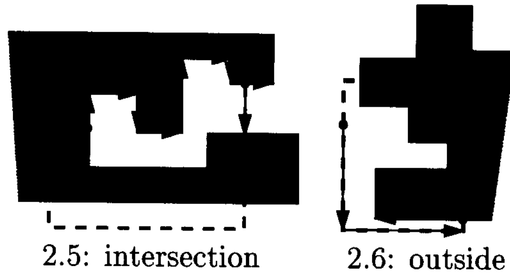
Lemma 2.5 *For any set E of n non-intersecting fat objects of constant complexity in the plane there exists an $O(n)$ size orthogonal subdivision $\mathcal{A}(L)$ such that each cell $\mathcal{C}$ in $\mathcal{A}(L)$ is intersected by a constant number of objects in E.*

Now that we have analysed the complexity of the orthogonal subdivision $\mathcal{A}(L)$ while leaving out the empty cells, we show how to compute this subdivision. Notice that we cannot simply compute all cells of $\mathcal{A}(L)$ and discard the empty cells afterwards because the complexity of the initial arrangement can be quadratic, resulting in a complexity of the algorithm that is at least quadratic.

We first compute the intersection points of the boundary of each bounding box b_i with the boundaries of the input objects $e_{i+1}, \cdots, e_n$. Lemma 2.2 shows there are $O(n)$ of these intersection points in total. Because the boundaries of the objects have constant complexity they can be split into a constant number of axes-monotone curves.

Lemma 2.6 *The intersection points of the boundary of each bounding box with the boundaries of the $O(n)$ input objects can be found in time $O(n \log n \log\log n)$.*

Proof: Starting with the biggest object e_n, no other object in E with minimum enclosing circle bigger than the minimum enclosing circle of e_n will intersect the bounding box b_n of e_n. Now assume that the boundaries of the objects $e_{i+1}, \cdots, e_n$ are stored in a datastructure such that the intersection of these boundaries with a rectangular box can be found efficiently. For e_i we find the objects in $e_{i+1}, \cdots, e_n$ intersecting b_i by searching with b_i in this datastructure. Subsequently we add the boundery of e_i to this datastructure and repeat this procedure for e_{i-1}. Because the boundaries of the objects have constant complexity and therefore can be split into a constant number of axes-monotone curves, according to Overmars [14] the datastructure can be implemented such that finding those k object boundery segments intersecting a (bounding) box takes time $O(k + \log^2 n)$. The



2.5: intersection 2.6: outside

application of dynamic fractional cascading improves the query time to $O(k+\log n \log\log n)$ [4, 5, 12]. Insertion in the dynamic datastructure can be done in time $O(\log n \log\log n)$. The boundaries of n objects of constant complexity are inserted, so $O(n \log n \log\log n)$ time in total is needed. The datastructure requires $O(n \log n)$ storage. □

We store with each object e_i the intersection points of the boundary of this object with the boundaries of the bounding boxes $b_1, \cdots, b_{i-1}$ ($i > 1$), so these intersection points can easily be retrieved.

Let L initially only consist of the four edges of the bounding box b_1. With each edge of b_1 we store the boundary of b_1 itself, so we can easily determine for any point if it lies in the interior of the bounding box. We add fragments of the boundaries of the bounding boxes b_i to L by processing the boxes in their order as follows. For each box b_i we only add those fragments $b_i - \cup_{j=1}^{i-1} b_j$ of its boundary to L that give rise only to a cell $\mathcal{C}$ for which $\mathcal{C} \cap e_i$ is non-empty. With each fragment of a segment of b_i that is added to L we store the boundary of b_i itself. Notice that in this way we compute an arrangement that is somewhat different from the one of which we analysed the complexity. There we added all fragments of $b_i - \cup_{j=1}^{i-1} b_j$ of the boundary of b_i to L that gave rise to a cell $\mathcal{C}$ for which $\mathcal{C} \cap E$ was non-empty. But the number of cells we compute now is still linear, since we add only fewer fragments of segments of b_i to L. Furthermore the number of object fragments within each cell $\mathcal{C}$ remains constant: because all objects $e_1, \cdots, e_{i-1}$ remain fully contained within $\cup_{j=1}^{i-1} b_j$, any cell created by fragments of the boundary of b_i will still only be intersected by the objects $e_{i+1}, \cdots, e_n$ and by e_i itself, so the number of objects intersecting the cell is $O(1)$ by lemma 2.1.

Let L_{i-1} ($1 < i \leqslant n+1$) be the set of axis-parallel line segments after the objects $b_1, \cdots, b_{i-1}$ have been handled, $\mathcal{A}(L_{i-1})$ be the arrangement induced by L_{i-1}, and $\mathcal{U}(L_{i-1})$ denote the set of (fragments of) line segments of L_{i-1} that lie on the boundary of the union of cells of $\mathcal{A}(L_{i-1})$. Let L^* be the fragments of the boundary of b_i such that $L_i = L^* \cup L_{i-1}$. To compute L^* and $\mathcal{U}(L_i)$ we start shooting rays along the edges of $\mathcal{U}(L_{i-1})$ and along the edges of b_i. Each ray will hit an edge of $\mathcal{U}(L_{i-1})$ or b_i along which the next ray can be shot. The rays initially start at the intersection points of $\mathcal{U}(L_{i-1})$ with the boundary of e_i, and at the intersection points between the boundary of e_i and the boundary of b_i (if both boundaries intersect along a line segment only the endpoints of the line segment are considered). We call these points origins for ray shooting. Because the object boudaries

are of constant complexity there are only a constant number of origins for each object. Shooting the rays is guided by the following rules:

- Shooting along the edges of $\mathcal{U}(L_{i-1})$ located inside b_i we continue until the boundary of b_i is hit, see figure 2.5. We use the intersection point of this ray and the boundary of b_i as an origin for the next ray along the boundary of b_i outside $\cup_{j=1}^{i-1} b_j$. We mark all the (fragments of) edges of $\mathcal{U}(L_{i-1})$ walked through.
- Shooting along the boundary of b_i outside $\cup_{j=1}^{i-1} b_j$ we continue until an edge of $\mathcal{U}(L_{i-1})$ is hit. All edges (or fragments) traversed along the boundary of b_i are added to L^*. We continue ray shooting starting from this intersection point along the edges of $\mathcal{U}(L_{i-1})$ located inside b_i, see figure 2.6. If we do not hit any edge of $\mathcal{U}(L_{i-1})$, we can add all edges of b_i to L^*.

Observe that we can start shooting at an origin in two directions along an edge. The ray shooting is continued until no more origins are left to start a ray from.

Lemma 2.7 *L^* contains exactly all those fragments of the boundary of b_i that create cells C for which $C \cap e_i$ is non-empty.*

Proof: According to the rules above, all cells that contain an intersection point of e_i and $\mathcal{U}(L_{i-1})$, or an intersection of the boundary of e_i and b_i are found. In case a cell does not contain an origin on its boundary, there are two possibilities. Either the object e_i is completely contained within the interior of the cell or the cell is completely contained within the interior of e_i. Both cases are impossible because such cells cannot contain an edge of b_i in their boundary.

No empty cells are created because the origins are either intersections of the boundary of e_i with $\mathcal{U}(L_{i-1})$ or points where e_i touches b_i. □

Thus, the new cells induced by the addition of b_i are found by computing their boundary using a rayshooting scheme. The boundary of the new cells are formed by those parts of the boundary of b_i, added to L^*, and fragments of $\mathcal{U}(L_{i-1})$ that are marked. We refer to de Berg[8]; a structure exists for ray shooting into a fixed direction in a set of axis-parallel polygons with $O(n)$ vertices in total, such that queries can be answered in $O(\log n \log\log n)$ time with a structure that uses $O(n \log n)$ storage. An axis-parallel polygon of constant complexity can be inserted into or deleted from the structure in $O(\log n \log\log n)$ amortized time.

Next to finding the new cells, $\mathcal{U}(L_i)$ is computed by updating $\mathcal{U}(L_{i-1})$. However it will not suffice to delete all marked (fragments of) edges of $\mathcal{U}(L_{i-1})$ from and insert all edges of L^* into $\mathcal{U}(L_{i-1})$. There can be fragments of $\mathcal{U}(L_{i-1})$ within the interior of the new cells that do not intersect the boundary of these cells and thus are not marked by the rayshooting scheme. To find these isolated parts of $\mathcal{U}(L_{i-1})$ all the new cells, which are all simple, are partitioned into rectangular windows by Chazelle's [2] linear time triangulation algorithm. With each of these windows we query into $\mathcal{U}(L_{i-1})$ for any isolated parts

of $\mathcal{U}(L_{i-1})$. Note that according to lemma 2.5 the overal complexity of the orthogonal subdivision $\mathcal{A}(L)$ is $O(n)$, so at most $O(n)$ queries will be performed. As mentioned above according to Overmars [14] windowing in a set of line segments segments takes time $O(k + \log^2 n)$, using a dynamic datastructure. The application of dynamic fractional cascading improves the query time to $O(k+\log n \log\log n)$ [4, 5, 12]. At most $O(n)$ isolated parts are reported in total, lemma 2.5. Insertion in the dynamic datastructure can be done in time $O(\log n \log\log n)$. The dynamic datastructure requires $O(n \log n)$ storage.

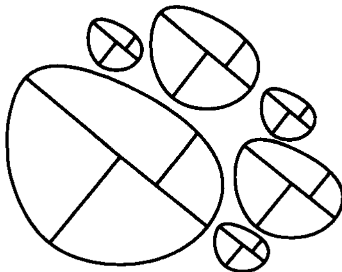
Thus, $\mathcal{U}(L_i)$ can be computed from $\mathcal{U}(L_{i-1})$ by deletion from $\mathcal{U}(L_{i-1})$ of the marked (fragments of) edges of $\mathcal{U}(L_{i-1})$, deletion from $\mathcal{U}(L_{i-1})$ of the isolated parts of $\mathcal{U}(L_{i-1})$ within the interiors of the new cells by windowing in $\mathcal{U}(L_{i-1})$, and insertion in $\mathcal{U}(L_{i-1})$ of all segments in L^*.

Theorem 2.1 *For any set E of n non-intersecting fat objects of constant complexity in the plane there exists an $O(n)$ size orthogonal subdivision $\mathcal{A}(L)$ such that each cell C in $\mathcal{A}(L)$ is intersected by a constant number of objects in E. The subdivision can be computed in time $O(n \log n \log\log n)$ using $O(n \log n)$ space.*

Proof: According to lemma 2.6 finding the intersection points between the boundaries of the input objects and the bounding boxes takes time $O(n \log n \log\log n)$. At most $O(n)$ queries are done, twice for each vertex of a bounding box and twice for each intersection between the boundary of an object and the boundary of a bounding box. According to lemma 2.5 at most $O(n)$ (fragments of) bounding box boundaries are inserted into and deleted from the datastructure. So, this step takes time $O(n \log n \log\log n)$ using $O(n \log n)$ storage [8]. Partitioning the new cells into rectangular windows takes time $O(n)$ in total, lemma 2.5. In total at most $O(n)$ window queries are performed taking time $O(n \log n \log\log n)$ and storage $O(n \log n)$ [14, 4, 5, 12]. At most $O(n)$ isolated boundary fragments will be reported. □

Now we have a scheme to construct an $O(n)$ size orthogonal subdivision $\mathcal{A}(L)$ for a set of non-intersecting fat objects of constant complexity, where each cell of the subdivision contains a constant number of objects. If we apply the orthogonal partition algorithm of Paterson and Yao to the set L of orthogonal segments, then we create a linear size binary space partition such that any cell in this partition contains only a constant number of objects from E. If we can be sure that we need only $O(1)$ partition lines to further subdivide the space within each cell such that subspaces result containing one object fragment, we are able to construct an $O(n)$ size binary space partition for the set of input objects. Thus if the set of input objects is a set of non-intersecting convex fat objects of constant complexity or a set of non-intersecting (non-convex) polygonal fat objects of constant complexity we are able to construct an $O(n)$ size binary space partition. We obtain the following theorems.

Theorem 2.2 *For any set of n non-intersecting constant-complexity fat convex objects in the plane and for any set of n non-intersecting constant-complexity fat arbitrary polygons in the plane there exists an $O(n)$ size bsp, which can be computed in time $O(n \log n \log\log n)$ using $O(n \log n)$ space.*



3.7: line segment representation

3 Convex Homothets

In the previous section we constructed a linear size binary space partition for a set of fat objects in the plane by reducing the problem to finding a bsp for a set of orthogonal segments. It turns out that the same strategy can be applied to obtain a linear size bsp for a set of convex *homothets* as well, as we explain in this section. (A set of objects is a set of homothets if all objects are identical except for their scale and position in space. For example, a set of discs of varying radius in the plane is a set of homothets.) Homothets are fat, so we could use the results of the previous section. But the fact that they are identical except for their scale and position in space makes a much more simple strategy possible. Let H be the input set of n convex non-intersecting homothets. We assume that the homothets have constant complexity. The transformation consists thereof that we represent each homothet $h \in H$ by three non-intersecting orthogonal line segments, as follows. As the first line segment we choose the longest line segment contained in h. (If this segment is not uniquely defined then we take of all longest segments the one with smallest slope.) This line segment s cuts the homothet h into two convex halves h^- and h^+. For each of these halves we again choose a longest line segment $s^- \subseteq h^-$ and $s^+ \subseteq h^+$ contained in this half and such that s^- and s^+ are orthogonal to s. See figure 3.7 for an illustration. We denote the resulting set of $3n$ line segments by L_H. The following lemma is straightforward.

Lemma 3.1 *A set H of n non-intersecting convex homothets of constant complexity in the plane can be transformed in linear time into a set L_H of $3n$ non-intersecting line segments with the following properties:*

- *inside each homothet in H there are three segments in L_H*
- *there are two orthogonal axes such that each segment in L_H is parallel to one of them*
- *each line segment that is parallel to one of these two orthogonal axes and that intersects the boundary of a homothet in H twice intersects at least one segment in L_H.*

By applying the orthogonal partition algorithm of Paterson and Yao to the set L_H of line segments, we create an $O(n)$ size binary space partition $\mathcal{B}(L_H)$ for L_H with the property that no cell of $\mathcal{B}(L_H)$ contains a fragment of a line segment of L_H in its interior. Let $\mathcal{C}$ denote a single cell of the subdivision $\mathcal{B}(L_H)$.

Lemma 3.2 *At most* 4 *homothets intersect the interior of each cell* $\mathcal{C}$ *of the binary space partition* $\mathcal{B}(L_H)$.

Proof: Note that each cell $\mathcal{C}$ in $\mathcal{B}(L_H)$ is a rectangle. Each corner point of $\mathcal{C}$ can be contained in the interior of only one homothet, because the homothets do not intersect. We claim that no other homothets than those containing a corner point of $\mathcal{C}$ in their interior can intersect the interior of $\mathcal{C}$. Assume for a contradiction that such a homothet exists. There are two cases to consider. In the first case the boundary of the homothet intersects an edge of the cell $\mathcal{C}$. Because the homothet is convex and does not contain a corner $\mathcal{C}$ in its interior, its boundary intersects this edge exactly twice. But according to Lemma 3.1 this edge would intersect a line segment in L_H which would then intersect the interior of $\mathcal{C}$, contradicting the definition of the binary space partition $\mathcal{B}(L_H)$. The second case to consider is when the homothet lies entirely inside $\mathcal{C}$. Since there are three segments of L_H inside each homothet, this again contradicts the definition of the binary space partition $\mathcal{B}(L_H)$. □

So we construct the set L_H of orthogonal segments according to Lemma 3.1. Next we compute a binary space partition for L_H using the orthogonal partition algorithm of Paterson and Yao [16]. During the partitioning we keep track of the homothet fragments that appear in the cells of the partition. Only two situations occur, see [16]. When a line segment is cut by a partition line or the line segment is contained in a partition line, fragments of the corresponding homothet appear in both new cells neighboring the partition line. Second, when a line segment is hit upon in a T-decomposition fragments of the related homothet appear in all three new cells created by the T-decomposition. Note that with each (fragment of) a line segment a unique homothet is related, and the homothets are of constant complexity, so computing the fragments will take only constant time. Putting it all together we obtain the following theorem.

Theorem 3.1 *Given a set* H *of* n *non-intersecting convex homothets of constant complexity, a binary space partition of size* $O(n)$ *for* H *can be constructed in time* $O(n \log n)$.

Proof: First we construct in $O(n)$ time the set L_H of orthogonal segments according to Lemma 3.1. Next we construct in time $O(n \log n)$ an $O(n)$ size binary space partition for L_H using the orthogonal partition algorithm of Paterson and Yao [16]. According to Lemma 3.2 there are at most four homothet fragments inside each of the $O(n)$ cells of this partition. Each fragment is convex, because it is the intersection of a convex homothet and a rectangle. Hence, in constant time per cell we can add a constant number of additional partition lines to separate the fragments within each cell, resulting in an $O(n)$ size bsp for H. □

Remark 3.1 In fact, we have proved the existence of a linear size bsp for arbitrary sets of convex non-intersecting homothets; the assumption that the homothets have constant complexity is only used to guarantee that we can compute the line segments representing the homothets in constant time per homothet. If the homothets do not have constant complexity, but are, e.g., convex polygons with N vertices each then computing the segments representing one of the homothets takes $O(N)$ time [18]. From this we can find the segments for the other homothets by translation and scaling. Furthermore, a line separating two homothets (which we need in the final stage of the algorithm) can be found in $O(\log N)$ time [9]. So in this case we can find a bsp of $O(n)$ size in $O(N + n \log N)$ time.

4 Line Segments with Bounded Length Ratios

Let S be a set of n disjoint line segments in the plane such that the ratio between the length of the longest segment and the length of the shortest one is bounded by a constant c. In this section we show that S admits a linear size binary space partition (with the constant of proportionality depending on c.)

Our method consists of two stages. In the first stage we construct a partitioning such that each segment is intersected at least once (but not too many times, of course). This implies that inside a cell of the partitioning all segments are connected to the boundary of that cell. The next stage constructs for each cell a linear size bsp tree on the segments inside the cell.

Stabbing the segments. To construct a bsp such that each segment is stabbed at least once we proceed as follows. Assume without loss of generality that the length of the shortest segment is one. First we add a number of vertical partition lines in the following way. Define $\ell(x^*)$ as the vertical line with x-coordinate x^*. The vertical lines that we use are the lines of the form $\ell(\frac{i}{2}\sqrt{2})$,for some integer i, that intersect at least one segment in S. In between any pair of lines (and to the left of the leftmost line, and to the right of the rightmost line) we next add horizontal partition lines in the same manner. That is, between two adjacent vertical lines ℓ and ℓ' we add all the horizontal partition lines—which are segments connecting ℓ and ℓ'—with y-coordinates $\frac{i}{2}\sqrt{2}$ for some integer i, provided that they intersect at least one segment in S.

As remarked earlier, the fact that any segment is intersected at least once implies that any segment inside a cell of the partitioning is *anchored*, that is, it intersects the boundary of that cell. The above partitioning scheme yields the following result.

Lemma 4.1 *Let S be a set of n line segments in the plane such that the ratio between the length of the longest segment in S and the length of the shortest one is bounded by a constant c. Then there is a binary space partition of size $(2c + 2)n - 2c + 2$ into cells $C_1, \dots, C_k$ such that the fragments of the segments in S that lie in any cell C_i are anchored. Furthermore, $\sum_{i=1}^{k} |S(C_i)| \leqslant (2c+2)n-2c+2$, where $S(C_i)$ is the set of segment fragments inside C_i.*

Proof: Consider a segment s of length c and let the projection of s along the vertical lines have length a. The maximum number of splitting lines that can intersect the interior of s is $\lceil a/(\frac{1}{2}\sqrt{2}) \rceil = \lceil a\sqrt{2} \rceil$. Similarly, the number of horizontal splitting lines that cut s is $\lceil \sqrt{c^2-a^2}\sqrt{2} \rceil$. Hence the total number of fragments into which s is cut is $\lceil a\sqrt{2} \rceil + \lceil \sqrt{c^2-a^2}\sqrt{2} \rceil \leqslant \sqrt{2}(a+\sqrt{c^2-a^2})+2$. This expression is maximized for $a = \frac{1}{2}\sqrt{2}c$. The number of fragments is $2c+2$. The maximum number of fragments for a segment of length 1 is 4. So the total number of fragments is at most $(2c+2)(n-1)+4 = (2c+2)n-2c+2$. The number of splitting lines that we can have is maximized when no splitting line intersects more than one segment. The maximum number of lines intersecting (the interior or endpoints of) a segment of length c is $\sqrt{2}(a+\sqrt{c^2-a^2})+2$. Hence the total number of splitting lines is bounded by $(2c+2)(n-1)+4 = (2c+2)n-2c+2$. □

Notice that the partitioning of Lemma 4.1 is, strictly speaking, not a bsp for S, since there will be more than one segment left in the cells of the partitioning.

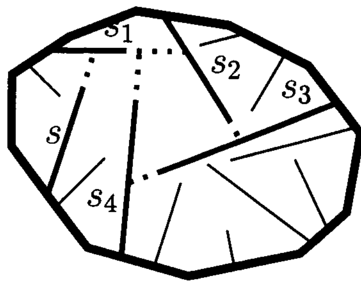
The case of anchored segments. Let $\mathcal{C}$ be a convex cell and let $S(\mathcal{C})$ be a set of anchored segments. We start with a simple special case, where all segments are anchored at the same edge of $\mathcal{C}$. Here we can obtain a linear size bsp by taking splitting lines containing the segments, starting with the segment that extends farthest from the edge.

Lemma 4.2 *Let $\mathcal{C}$ be a convex polygon, and let $S(\mathcal{C})$ be a set of segments inside $\mathcal{C}$ that are all anchored at the same edge of $\mathcal{C}$. Then there exists a bsp for $S(\mathcal{C})$ inside $\mathcal{C}$ of size $|S(\mathcal{C})|$.*

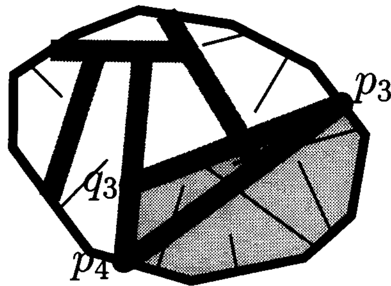
Proof: Let e be the edge of $\mathcal{C}$ at which the segments are anchored. For each segment in $S(\mathcal{C})$ consider its endpoint that is not on e. Sort the segments according to decreasing distance of these endpoints to e. If we have two segments s and s' such that the endpoint of s is further from e than the endpoint of s', then the extension of s does not intersect s'. Hence, if we construct a bsp by taking the partition lines containing the segments in $S(\mathcal{C})$ according to this order, then no segment will be fragmented. □

We now consider the case where the segments are incident to different edges of the boundary of the cell. Our strategy is to partition $\mathcal{C}$ into a number of subcells in such a way that the new fragments that we generate by cutting segments lie in only one of the subcells. Moreover, all these fragments are incident to a single edge of that subcell, so we can employ Lemma 4.2 to partition it. The remaining cells are partitioned recursively.

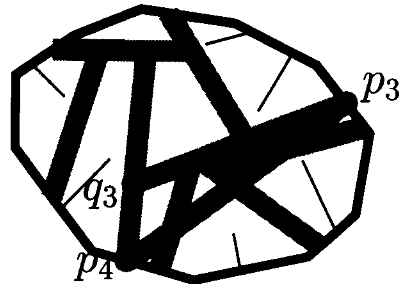
Let s be an arbitrary segment in $S(\mathcal{C})$. The segment s defines a so-called *successor sequence*, denoted by $\mathcal{S}(s)$, which we use to obtain the partitioning into subcells. The successor sequence of s is a sequence $s_0, s_1, \ldots, s_m$ of segments in $S(\mathcal{C})$, defined as follows. The first segment s_0 of the sequence is s itself. Extend s_0 until it either hits another segment or it hits the boundary of $\mathcal{C}$. The extension of s_0 (including s_0 itself) obtained in this way is denoted by $\text{ext}(s_0)$. If $\text{ext}(s_0)$ hits the boundary of $\mathcal{C}$ then $\mathcal{S}(s) = s_0$ and we are ready. Otherwise $\text{ext}(s_0)$ hits another segment. This segment—the successor of s_0, denoted by $\text{successor}(s_0)$—is the next segment s_1 of $\mathcal{S}(s)$. The rest of $\mathcal{S}(s)$ is found in a



4.8: successors



4.9: breaking the cycle



4.10: recurse

similar manner: Suppose that the part of $\mathcal{S}(s)$ that we have found so far is $s_0, s_1, \ldots, s_i$. To find the next segment in the sequence we extend s_i until it either hits the boundary of $\mathcal{C}$, or it hits the extension $\text{ext}(s_j)$ of one of the already added segments, or it hits a new segment s'. In the first two cases the successor sequence has been completed. In the last case we add $\text{successor}(s_i) := s'$ as the next segment s_{i+1} to $\mathcal{S}(s)$ and repeat the process. See Figure 4.8 for an illustration. The process must end after at most $|S(\mathcal{C})|$ steps, because a segment is added at most once to the successor sequence.

We next describe how to use $\mathcal{S}(s) = s_0, s_1, \ldots, s_m$ to partition $\mathcal{C}$ into subcells. There are two cases to consider.

case (i): $\text{ext}(s_m)$ hits the boundary of $\mathcal{C}$.

In this case we can add the extensions in reverse order as partition lines, that is, we add the partition lines $\text{ext}(s_m), \text{ext}(s_{m-1}), \ldots, \text{ext}(s_0)$. We recurse on the resulting subcells.

case (ii): $\text{successor}(s_m) = s_j$ for some $0 \leqslant j < m$.

This is the elaborate case. The problem is that none of the extensions $\text{ext}(s_i)$ crosses $\mathcal{C}$ completely. Somehow we have to "break" the cycle $s_j, \ldots, s_m$ by extending one of the segments even further. Then we can use that segment to start the partitioning process; the other extensions can then be added in reverse order, as in case (i). However, extending one of the segments further may cause a lot of fragmentation. If we would treat all subcells recursively then we cannot keep the fragmentation under control. Hence, we proceed in a slightly different fashion.

For $0 \leqslant i \leqslant m$, let p_i be the endpoint of $\text{ext}(s_i)$ that is on the boundary of $\mathcal{C}$ and let q_i be the other endpoint. First we add $\overline{p_{m-1}p_m}$, the segment connecting p_{m-1} and p_m, as a partition line. (It may happen that p_m and p_{m-1} are already on the same edge of $\mathcal{C}$, in which case the addition of $\overline{p_{m-1}p_m}$ can be omitted.) Next we extend $\text{ext}(s_{m-2})$ until it hits $\overline{p_{m-1}p_m}$ and add this extension as a partition line. We have now broken the cycle and we can add the extensions $\text{ext}(s_{m-3}), \text{ext}(s_{m-4}), \ldots \text{ext}(s_0), \text{ext}(s_m)$, $\text{ext}(s_{m-1})$ as partition lines. This is illustrated in Figure 4.9 for the example in Figure 4.8.

Because of the way $\mathcal{S}(s)$ is defined, $\overline{p_{m-1}p_m}$ and the extension of $\text{ext}(s_{m-2})$ are the only partition lines that can cut segments of $S(\mathcal{C})$ into fragments. Thus there are at most three subcells in the partitioning that contain fragments of segments, all other subcells contain only segments that have not been cut. These three subcells lie in the convex region enclosed by $\text{ext}(s_{m-1})$, $\overline{q_{m-1}p_m}$ and a part of the boundary of $\mathcal{C}$ connecting p_m to p_{m-1}. This region is depicted shaded in Figure 4.9. First consider the subcell enclosed by $\overline{p_{m-1}p_m}$ and the boundary of $\mathcal{C}$. All fragments in this subcell completely cross the subcell so we can make a *free cut* along each fragment, that is, we can use the fragments themselves as partition lines inside the subcell. All the subsubcells created this way contain only segments that have not been cut so far. Next consider the other two subcells, which are both triangles contained in the triangle Δ defined by the points p_{m-1}, q_{m-1}, and p_m. Observe that $\overline{p_{m-1}p_m}$ is the only edge of Δ that cuts segments. Hence, we can construct a bsp for the segments inside Δ using Lemma 4.2. This means that we do not have to recurse inside Δ anymore. The resulting partitioning for the example of Figure 4.8 is illustrated in Figure 4.10. Note that the extension of $\text{ext}(s_{m-2})$, which cuts Δ into two, also cuts the bsp inside Δ into two, since it has already been added. But this only doubles the size of the bsp.

Lemma 4.3 *The procedure described above yields a valid partitioning of $S(\mathcal{C})$ into subcells $\mathcal{C}_1, \ldots, \mathcal{C}_k$, with $k \geqslant 2$, such that $\sum_{i=1}^{k} |S(\mathcal{C}_i)| \leqslant |S(\mathcal{C})| - k/3$, where $S(\mathcal{C}_i)$ is the set of segments inside $\mathcal{C}_i$.*

Proof: It is easily checked that in both cases the method yields a valid partitioning, that is, any partition line that we add completely crosses the subcell of the current partitioning that it is in. It remains to prove the bound on the size of the partitioning.

If we perform the partitioning according to case (i) then every partition line contains a segment of $S(\mathcal{C})$, (which will not occur in any of the subcells), any segment is contained in at most one partition line, and no segment is fragmented. Since the number of partition lines that is used for a partitioning into k subcells is exactly $k-1$ we have $\sum_{i=1}^{k} |S(\mathcal{C}_i)| = |S(\mathcal{C})| - (k-1) \leqslant |S(\mathcal{C})| - k/3$.

Now consider case (ii). Recall that also in this case the subcells where we recurse contain only segments that have not been cut and no fragments. Hence, it suffices to prove that the number of partition lines used is at most three times the number of segments that are excluded from further consideration.

The first partition line that we add is the line $\overline{p_{m-1}p_m}$. We claim that $\overline{p_{m-1}p_m}$ does not intersect any of the extensions $\text{ext}(s_i)$, $0 \leqslant i \leqslant m$. To see why this is true consider the two segments $\overline{p_{m-1}q_{m-1}}$ and $\overline{q_{m-1}p_m}$. These two segments split $\mathcal{C}$ into two regions. By definition of the successor sequence $\mathcal{S}(s)$ we have that $\text{ext}(s_i)$, $0 \leqslant i \leqslant m-2$, forms a connected sequence of segments that lies completely in one of those regions. Since $\text{ext}(s_m)$ hits $\text{ext}(s_j)$ for some $0 \leqslant j \leqslant m-2$, this cannot be the same region as the region that contains $\overline{p_{m-1}p_m}$.

Now consider the partition lines $\text{ext}(s_i)$, $0 \leqslant i \leqslant m$. Of these lines only $\text{ext}(s_{m-1})$ is cut (namely when we extend $\text{ext}(s_{m-2})$ further) and, hence, actually used twice. If we now

also consider the partition line $\overline{p_{m-1}p_m}$ then we have $m+3$ partition lines that together contain $m+1$ segments that can be excluded from further consideration. Since $m \geqslant 2$, the number of partition lines is less than three times the number of excluded segments.

The remaining partition lines are the ones used inside Δ—which may be cut into two by the extension of $\text{ext}(s_{m-2})$—and the free cuts. By Lemma 4.2 the number of partition lines inside Δ is the same as the number of segments inside Δ. It follows that we can charge the remaining partition lines to disappearing segments in such a way that any segment is charged at most three times: once for a free cut and twice inside Δ. □

To obtain the complete bsp for $S(\mathcal{C})$ inside $\mathcal{C}$ we apply the above procedure in $\mathcal{C}$ and recurse on the subcells $\mathcal{C}_i$ with $|\mathcal{C}_i| > 1$. Using Lemma 4.3 we can prove the following.

Lemma 4.4 *Let $\mathcal{C}$ be a convex polygon and let S be a set of segments inside $\mathcal{C}$ that are all anchored at the boundary of $\mathcal{C}$. Then there exists a bsp for $S(\mathcal{C})$ inside $\mathcal{C}$ of size at most $3|S(\mathcal{C})|$.*

Proof: Let $B(m)$ denote the maximum size of the bsp generated if the above procedure is applied to a set of m segments. Note that the partitioning into k subcells described above uses $k-1$ partition lines that each contain at most one fragment of a segment. Hence, by Lemma 4.3 the function $B(m)$ satisfies

$$B(m) \leqslant \sum_{i=1}^{k} B(m_i) + (k-1)$$

with $\sum_{i=1}^{k} m_i \leqslant m - k/3$, with $k \geqslant 2$ and $B(1) = 0$, from which the lemma readily follows. □

Putting it together. Summarizing, we obtain a linear size bsp for a set of segments in the plane with bounded length ratios in the following way. First we construct a partitioning such that each segment is intersected at least once. Then we solve the subproblem inside each cell using the method for anchored segments. We obtain the following result.

Theorem 4.1 *Let S be a set of n line segments in the plane such that the ratio between the length of the longest segment in S and the length of the shortest one is bounded by a constant c. Then there exists a bsp for S of size $O(n)$, which can be computed in $O(n \log^2 n)$ time using $O(n)$ space.*

Proof: According to Lemma's 4.1 and 4.4 the size of the bsp resulting from our method is bounded by

$$\sqrt{2}(c(n-1)+1) + \sum_{i=1}^{k} 3|S(\mathcal{C}_i)| \leqslant$$

$$\sqrt{2}(c(n-1)+1) + 3(c\sqrt{2}+3)n =$$

$$(4c\sqrt{2}+9)n-(c-1)\sqrt{2}.$$

To prove the running time we note that a partition as in Lemma 4.1 can be found in $O(n \log n)$ time. (In fact, to avoid the use of the floor-function we should compute a slightly different partition with the same properties. We leave the details of this as an easy exercise to the reader.) To implement Lemma 4.4 we use a dynamic data structure for ray shooting in connected subdivisions. This data structure stores the subdivision induced by the segments and all partition lines that have already been added. This includes both the partition lines that we add to stab all the segments and all partition lines that we add later. Notice that this subdivision is indeed connected. We maintain only one global data structure, which can be accessed from all the subcells. Whenever we add a partition line we update the subdivision. Using the data structure one can compute the partitioning of Lemma 4.4 with a number of queries and updates that is linear in the number of partition lines that we add. Goodrich and Tamassia [11] have presented a data structure for ray shooting queries in dynamic, connected subdivisions that uses linear space and supports queries and updates in $O(\log^2 n)$ time. Since the total number of partition lines is linear, our algorithm runs in $O(n \log^2 n)$ time and uses $O(n)$ space. □

5 Conclusion

We proved the existence of linear size binary space partitions for three classes of objects in the plane, namely for fat objects, for homothets, and for segments with bounded length ratios. We also presented efficient algorithms for computing linear size binary space partitions for each of these classes. This extends a previous result by Paterson and Yao [15], who constructed linear size bsp's for orthogonal segments in the plane. We have, however, been unable to prove the conjecture of Paterson and Yao that any set of line segments in the plane admits a linear size bsp. This is still the most important problem in this area. We hope that our techniques provide some insight that is useful for proving this conjecture.

References

[1] C. Ballieux. Motion planning using binary space partition. Report inf/src/93-25, Utrecht University, 1993.

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

[3] B. Chazelle. personal communication, 1993.

[4] B. Chazelle and L. J. Guibas. Fractional cascading: I. A data structuring technique. *Algorithmica*, 1:133–162, 1986.

[5] B. Chazelle and L. J. Guibas. Fractional cascading: II. Applications. *Algorithmica*, 1:163–191, 1986.

[6] N. Chin and S. Feiner. Near real time shadow generation using bsp trees. In *SIGGRAPH'90*, pages 99–106, 1990.

[7] F. d'Amore and P. G. Franciosa. On the optimal binary plane partition for sets of isothetic rectangles. *Inform. Process. Lett.*, 44:255–259, 1992.

[8] M. de Berg. *Ray Shooting, Depth Orders and Hidden Surface Removal*, volume 703 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1993.

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

[10] H. Fuchs, Z. M. Kedem, and B. Naylor. On visible surface generation by a priori tree structures. *Comput. Graph.*, 14(3):124–133, 1980.

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

[12] K. Mehlhorn and S. Näher. Dynamic fractional cascading. *Algorithmica*, 5:215–241, 1990.

[13] B. Naylor, J. Amanatides, and W. Thibault. Merging bsp trees yields polyhedral set operations. In *SIGGRAPH'90*, pages 115–124, 1990.

[14] M. H. Overmars. Range searching in a set of line segments. In *Proc. 1st Annu. ACM Sympos. Comput. Geom.*, pages 177–185, 1985.

[15] M. S. Paterson and F. F. Yao. Efficient binary space partitions for hidden-surface removal and solid modeling. *Discrete Comput. Geom.*, 5:485–503, 1990.

[16] M. S. Paterson and F. F. Yao. Optimal binary space partitions for orthogonal objects. *J. Algorithms*, 13:99–113, 1992.

[17] F. P. Preparata. A new approach to planar point location. *SIAM J. Comput.*, 10:473–482, 1981.

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

[19] W. C. Thibault and B. F. Naylor. Set operations on polyhedra using binary space partitioning trees. In *Proc. SIGGRAPH'87*, pages 153–162, 1987.

[20] A.F. van der Stappen, D. Halperin, and M.H. Overmars. The complexity of the free space for a robot moving amidst fat obstacles. *Computational Geometry: Theory and Applications*, 3:353–373, 1993.