Linear BSP Trees for Sets of Hyperrectangles with Low Directional Density

by

Petr Tobola
Karel Nechvíle

FI MU Report Series FIMU-RS-2000-04

Linear BSP Trees for Sets of Hyperrectangles with Low Directional Density

Petr Tobola, Karel Nechvíle[1]

Faculty of Informatics, Masaryk University
Botanická 68a, 602 00 Brno
Czech Republic
{ptx,kodl}@fi.muni.cz

Abstract

We consider the problem of constructing of binary space partitions (BSP) for a set S of n hyperrectangles in arbitrary dimensional space. If the set S fulfills the low directional density condition defined in this paper then the resultant BSP has $O(n)$ size and it can be constructed in $O(n \log^2 n)$ time in $\mathcal{R}^3$. The low directional density condition defines a new class of objects which we are able to construct a linear BSP for. The method is quite simple and it should be appropriate for practical implementation.

keywords: BSP, rectangle, tree, partitioning

1 Introduction

Many of computer graphics and computational geometry problems concern processing of object sets in two and three-dimensional space. Such tasks can be usually solved successfully and effectively, if a scene is simplified by a suitable partitioning of the space into subspaces.

A scene can be divided in many ways. We have to decide which information will be important for us and that's why we will require its maintenance or highlighting. A natural way to perform the partitioning is to make a linear cut of the space with a hyperplane splitting the space (and possibly some of the objects) into two parts.

[1]Support was provided by the grant 201/98/K041

Informally: *Binary Space Partition*, or *BSP* (initially introduced by Schumacker [18]) is a recursive partitioning of the space with objects by suitable hyperplane. The partitioning process is repeated for new arising subspaces until only one fragment of any object occurs in detached subspace. We suppose objects do not intersect each other, otherwise we would not be able to ensure finishing of the splitting.

The BSP for a set of objects can be naturally expressed as a tree structure. The splitting hyperplanes and objects lying within them are stored in nodes of BSP tree. Each node of BSP tree is associated with a convex region which is a part of the original space. This convex region is created by splitting the space by hyperplanes associated with ancestors of given node. We can observe that convex regions associated with nodes of the same level generate a resolution of the original space.

The BSP trees have a wide usage in many areas of computer science. They are used, for example, in hidden surface removal using painters algorithm [11], visibility solution [19], shadow generation [7], objects modeling [14, 20], surface approximation [3], or robot motion planning [5].

When we split the space by a hyperplane then some objects can be unwillingly divided into two or more parts. In such way, the original scene will be divided into a lot of fragments. However, the efficiency of algorithms benefiting from BSP depends on the size of consequential BSP. This is the reason for necessity to select the split hyperplanes carefully.

In the past, a lot of attention was dedicated to the development of algorithms which construct BSP trees of a small size. Initially, several heuristic methods were developed (for example [4, 11, 10, 19, 20]), which however can create an excessive size tree for unpropitious cases ($\Omega(n^2)$ in the plane and $\Omega(n^3)$ in the space). The first provable bounds were obtained by Paterson and Yao [15, 16]. They showed [15], that the optimal size of BSP in the space in the worst case is $\Theta(n^2)$ and in the plane is $O(n \log n)$. The next result of these authors [16] was the optimal size BSP algorithm for the set of orthogonal objects in the space in the worst case $\Theta(n^{3/2})$ and in the plane in the worst case $\Theta(n)$.

However, most of randomly created BSP trees have reasonable behavior for practical scenes. Their sizes are considerably smaller than the worst case determined boundary. Modern algorithms try to use these properties to construct nearly linear BSP trees. Pankaj K. Agarwal et al. [1] solved the problem of a construction of BSP tree for a set of fat orthogonal rectangles (the fat objects are intuitive objects without extremely skinny and long parts). Their algorithm creates BSP trees of $n2^{O(\sqrt{\log n})}$ size for scene of n fat rectangles and of $n\sqrt{m}2^{O(\sqrt{\log n})}$ size for scene of $(n-m)$ fat rectangles. The running time is linear to output BSP tree size. In the next paper [2] they compared implementation of this algorithm with other BSP algorithms. It was shown that their algorithm is really applicable in practice.

Mark de Berg et al. have extensively studied the problem of BSP in the plane [8]. They showed existence of a linear size BSP for sets of line segments where the ratio between the lengths of the longest and the shortest segment is bounded by a constant, for sets of fat objects and for homothetic objects. They also proposed effective algorithms to construct it (in the time $O(n \log\log n)$, $O(n \log n)$ and $O(n \log^2 n)$).

In [9], de Berg was engaged in moving of the problem of BSP for sets of fat objects into higher dimensional spaces. His algorithm offers linear BSP trees also with only a little worse running time $(O(n \log^2 n))$. Nevertheless, it is simple and more convenient for practical implementation.

Nguyen Viet Hai [12] published an algorithm creating linear BSP trees for set of *r-bounded* hyperrectangles in $\mathcal{R}^d$. The advantage of this algorithm is that it ensures balance of resultant BSP tree. Moreover, the algorithm works in optimal $O(n \log n)$ time. We will compare the algorithms of Nguyen Viet Hai [12], Mark de Berg [9] and our proposed method in the last part of this paper.

The algorithm proposed in this paper extends our previous work [21] dedicated to BSP trees for sets of segments in the plane. We proposed an algorithm creating linear BSP for set of segments with so-called *low directional density* in $O(n \log^3 n)$ time. Here we extend this method into higher dimensional spaces. Over against this generalisation we have lost the advantage of arbitrary orientation of objects and we work with axes oriented hyperrectangles only. This quite simple method can provide BSP trees of linear size under condition of so-called *low directional density of hyperrectangles.*

This paper is organized as follows: Section 2 presents same basic notions and definitions. In section 3, we prove existence of a linear BSP for a set of axes oriented rectangles with *low directional density* in $\mathcal{R}^3$ and describe main ideas of our algorithm. Section 4 extends our considerations for sets of hyperrectangles in $\mathcal{R}^3$ and in section 5 we describe an efficient algorithm to construct the linear BSP tree. The comparison with other algorithms and conclusion follows in section 6.

2 Preliminary

We start with formal definition of the binary space partition:

Definition 2.1: *A binary space partition tree $\mathcal{B}$ for a set S of pairwise disjoint, $(d-1)$-dimensional, polyhedral objects in $\mathbb{R}^d$ is a tree recursively defined as follows*[2]*:*

Each node v in $\mathcal{B}$ represents a convex region $\mathcal{R}_v$ and a set of objects $S_v = \{s \cap \mathcal{R}_v | s \in S\}$, that intersect $\mathcal{R}_v$. The region associated with the root is $\mathbb{R}^d$ itself. If S_v is empty, then node v is a leaf of $\mathcal{B}$. Otherwise, we partition v's region $\mathcal{R}_v$ into two convex regions by a cutting hyperplane H_v. At v, we store $\{s \cap H_v | s \in S_v\}$, the set of objects in S_v, that lie in H_v. If we let H_v^+ be the positive halfspace and H_v^- the negative halfspace bounded by H_v, the regions associated with the left and right children of v are $\mathcal{R}_v \cap H_v^-$ and $\mathcal{R}_v \cap H_v^+$, respectively. The left subtree of v is a BSP for set of objects $S_v^- = \{s \cap H_v^- | s \in S\}$ and the right subtree of v is a BSP for set of objects $S^- + v = \{s \cap H^- + v | s \in S\}$. The size of $\mathcal{B}$ is the number of nodes in $\mathcal{B}$.

Both the Mark de Berg's *unclutteredness* [9] and the Nguyen Viet Hai's *r-boundednes* [12] are based on properties of scene objects and the dimension of that one corresponds to the dimension of the original space. The first idea of our algorithm comes out from the observation that the dimension of the splitting hyperplane is less by one than the split space. Since we can aim our attention to the space and objects contained in the splitting hyperplane only.

The second idea follows from the *free cuts.* We show an example of the free cut in two-dimensional space with set of segments but it can be generalized for arbitrary dimension. If a segment is split into three or more parts, then we can bring a splitting hyperplane containing the median segment without additional splitting of another segment. In such way, this segment can be excluded from further consideration (see figure 1).

We generalize this idea and define so-called *ε-free cuts*, which can cut only constant number (ε) of other segments in our algorithm. For algorithm's intentions, we suppose that any object (hyperrectangle) has *extended low directional density* defined in the next part of this paper. We believe, that many realistic scenes fit to this condition.

[2]We take up the definition of Agarwal [1]

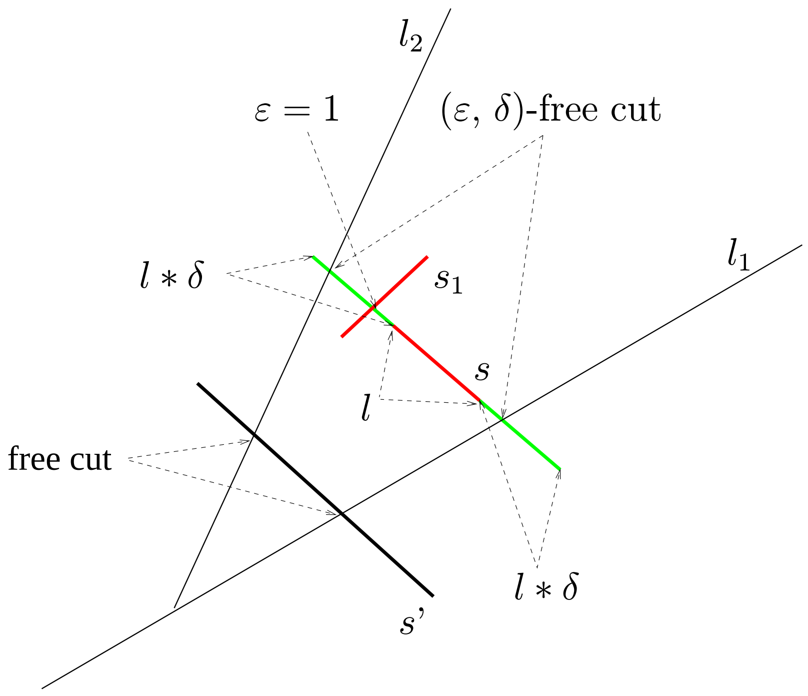


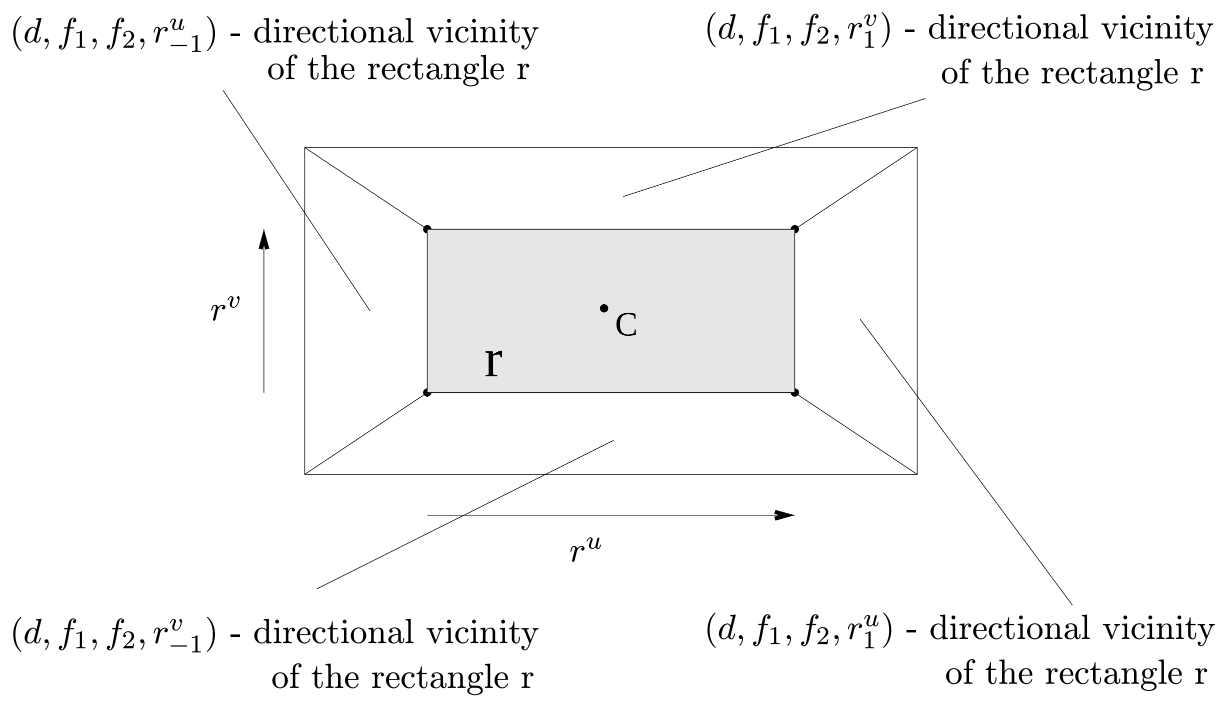
Figure 1: Free cut

Definition 2.2: *Let r be a rectangle in the space with vertices $r[X_j]; j \in \{1, ..., 4\}$ and side vectors $\vec{r^u} = X_2 - X_1$ and $\vec{r^v} = X_3 - X_2$, as you can see on the figure 2. Point C be center of the rectangle r. W.l.o.g. we can suppose that $|\vec{r^u}| > |\vec{r^v}|$. Then $(\delta, f_u, f_v, r^{\{u,v\}}_{\{1,-1\}})$-directional neighbourhood of rectangle r (we will mark it $\Omega(\delta, f_u, f_v, r^{\{u,v\}}_{\{1,-1\}})$) is a union of set of points defined as follows:*

- $\Omega(\delta, f_u, f_v, r^u_1) = \{C \pm c_1\vec{r^v_i} + c_2\vec{r^u_i}\}$
- $\Omega(\delta, f_u, f_v, r^u_{-1}) = \{C \pm c_1\vec{r^v_i} - c_2\vec{r^u_i}\}$
- $\Omega(\delta, f_u, f_v, r^v_1) = \{C \pm c_3\vec{r^u_i} + c_4\vec{r^v_i}\}$
- $\Omega(\delta, f_u, f_v, r^v_{-1}) = \{C \pm c_3\vec{r^u_i} - c_4\vec{r^v_i}\}$

where $c_1 \in \langle 0, ..., f_v\rangle$, $c_2 \in \langle \frac{1}{2}, ..., \frac{1}{2} + f_u\rangle$, $c_3 \in \langle 0, ..., f_u\rangle$, $c_4 \in \langle \frac{1}{2}, ..., \frac{1}{2} + f_v\rangle$. $f_{\{u,v\}} = f_{\{u,v\}}(\delta, r)$ is non-negative above unlimited function increasing with δ. The (δ, f_u, f_v, r)-directional neighbourhood of rectangle r (we will mark it $\Omega(\delta, f_u, f_v, r)$) is a union $\cup\Omega(\delta, f_u, f_v, r^{\{u,v\}}_{\{1,-1\}})$.

Let $f_u = \delta, f_v = \delta$. In this case, the definition is intuitively extension of the definition for set of segments in the plane [21]. *We call the $\Omega(\delta, f_u, f_v, r)$ neighbourhood* **simple directional neighbourhood** *and sign $\Omega_s(\delta, r)$.*



(d, f_1, f_2, r^u_{-1}) - directional vicinity of the rectangle r
(d, f_1, f_2, r^v_1) - directional vicinity of the rectangle r
r^v
C
r
r^u
(d, f_1, f_2, r^v_{-1}) - directional vicinity of the rectangle r
(d, f_1, f_2, r^u_1) - directional vicinity of the rectangle r

Figure 2: Directional neighbourhood of rectangle r

Definition 2.3: *Let R be a set of rectangles in the space, $r_i \in R$ and ε be an integer constant. We say, that rectangle r_i is:*

- **free** *if* $\Omega(\infty, f_u, f_v, r_i) \cap R = \emptyset$
- ε**-free,** *if* $|\Omega(\infty, f_u, f_v, r_i) \cap R| < \varepsilon$

Definition 2.4: *We say, that a rectangle $r_i \in R$ has $(\varepsilon, \delta, f_u, f_v)$* **-low directional density,** *iff $|\Omega(\delta, f_u, f_v, r_i) \cap R| \leq \varepsilon$, whereas ε is a integer constant and $\delta > 0$ is a real constant.*

We say, that a set of rectangles R has $(\varepsilon, \delta, f_u, f_v)$-**low directional density,** *iff any rectangle $r \in R$ has $(\varepsilon, \delta, f_u, f_v)$-low directional density.*

Let us define the **simple low directional density** *of r and R for the $\Omega_s(\delta, r)$ neighbourhood.*

3 BSP of rectangles in the space

Lemma 3.1: *There is a set R of axes aligned rectangles with simple low directional density that no linear BSP exists.*

Proof: Obviously, any hyperrectangle in $\mathcal{R}^3$ is bounded by six rectangles from $\mathcal{R}^2$. A BSP for the set of bounding rectangles is implicitly BSP for the set of hyperrectangles.

Paterson and Yao [16] showed, that the BSP lower bound of hyperrectangles in the space is $\Omega(n^{3/2})$. The worst case (three-dimensional grid) is shown in the figure 3.

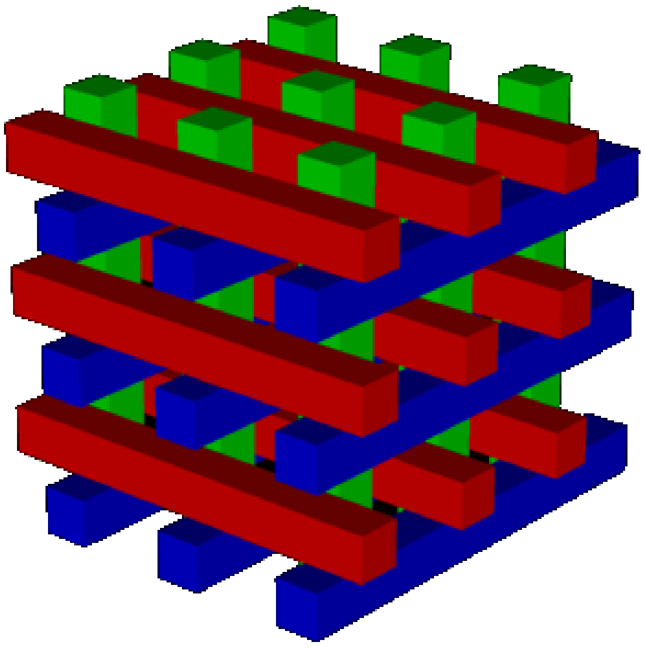


Figure 3: *The worst case of hyperrectangles in the space.*

Let us shrink the two shorter sides of any hyperrectangle into third of the original size. In this way, a free space around the hyperrectangles arises. The lower bound of BSP is preserved, but the $\Omega_s(1, r_b)$ neighbourhood of any bounding rectangle r_b is crossed by no other rectangle because the size of the neighbourhood conforms the size of the original hyperrectangle. Therefore, any bounding rectangle has $(0, 1, \delta, \delta)$-*low directional density* and the whole set of bounding rectangles has *simple low directional density.* □

Definition 3.2: *Let $f_u = (1 + 2\delta)$, $f_v = \frac{|\vec{r^u}|}{|\vec{r^v}|}(1 + 2\delta)$. We call the $\Omega(\delta, f_u, f_v, r)$ neighbourhood of r* **extended directional neighbourhood** *and sign $\Omega_e(\delta, r)$. We call the set of rectangles R with $(\varepsilon, \delta, f_u, f_v)$-low directional density the set with* **extended low directional density***.*

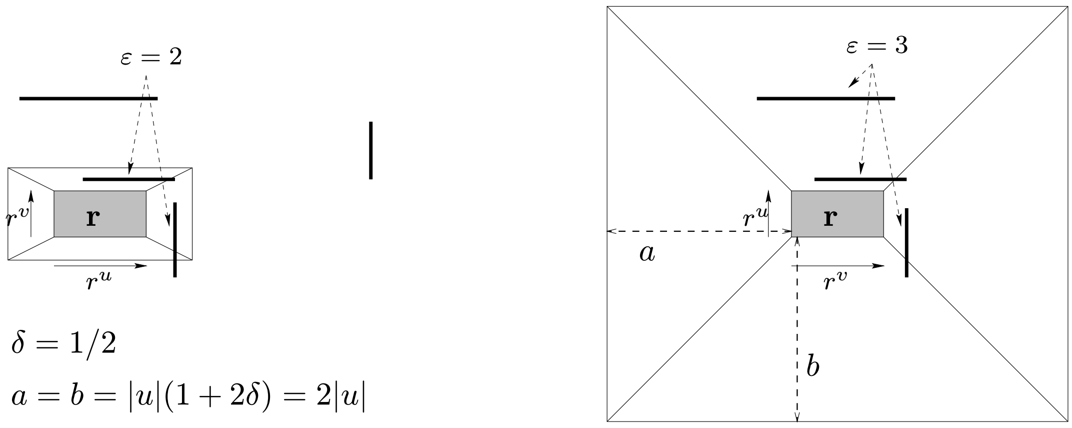


Figure 4: *Simple low directional density* (on the left) and *Extended low directional density* (on the right) of rectangle r parallel with xy plane.

In the consecutive text, we use the following Lemma proved in [21].

Lemma 3.3: *Let S, B be non-empty sets of segments in the plane which fulfil the following conditions:*

1. $n = |S| \leq |B| = n + k$
2. *There is such injective mapping $\sigma : I \to J; I = \{1, ..., n\}, J = \{1, ..., n+k\}$ and real constant α, that the following claim holds for all $i \in I$: $(|s_i| \leq \alpha |b_{\sigma(i)}|) \wedge (s_i \parallel b_{\sigma(i)})$, where $|s_i|$ means the length of segment $s_i \in S$ and $|b_{\sigma(i)}|$ means the length of segment $b_{\sigma(i)} \in B$.*

Furthermore, let v be an arbitrary non-zero vector such that $\exists(s_i) : s_i \nparallel u$ and p be a line parallel with v. Then the following statement holds: $\exists(p) : |p \cap S| \leq \alpha |p \cap B|$.

Lemma 3.4: *Let R be a set of axes parallel rectangles with extended low directional density. Then a linear BSP for the set R exists.*

Proof: We use both the notions of simple and extended low directional density during this proof. Most of cuts are going through the simple directional neighbourhood. Nevertheless, it is not possible to use only such cuts as follows from Lemma 3.1. Hence, we are forced to use the extended directional neighbourhood in the adverse cases.

At first, we build up three pairs of auxiliary sets of segments B_1, S_1, B_2, S_2 and B_3, S_3 in the following way:

Let us project the set $\{r|r \in R\}$ of original rectangles onto the x axis. The set of segments S_1 contains the projected rectangles $S_1 = \{s|s = Proj_x(r), r \in R\}$. For the sake of simplicity, we will suppose, that the endpoints are in general position (i.e. no two endpoints have the same x-coordinate).

The degenerate cases could be simply solved by lexicographical ordering on the points of original rectangles. Each endpoint of $s \in S_1$ can be considered as projection of an unique point p_{max} (p_{min}) $\in r$ maximal (minimal) in the standard lexicographical ordering.

The simple directional neighbourhood $\Omega_s(\delta, r)$ belonging to r is a part of rectangle enclosing r. Let us project the set $\{\Omega_s(\delta, r)|r \in R\}$ onto the axis x. We get a set of segments. Let us split each segment $Proj_x(\Omega_s(\delta, r))$ into two parts by subtraction of the $Proj_x(r)$ from the one. We get two resultant segments: b_1 with lower x coordinates and b_2 with higher x coordinates associated with the segment s, as you can see in figure 5. It follows from the definition of $\Omega_s(\delta, r)$ that $|b_1| = |b_2| = \delta|s|$. The set B_1 is an unification of all segments b_1 and b_2 generated by the set of $\{\Omega_s(\delta, r)|r \in R\}$. Note, that the degenerate cases are treated by lexicographical ordering as well and we get two zero length segments b_1 and b_2 associated with the zero length segment s.

The sets S_2, B_2, S_3 and B_3 are created in the same way by projection onto the y and z axes. Considerations about sets S_2,B_2 and S_3, B_3 are symmetric as about S_1 and B_1 and will be omitted in the next text.

Observation: Let p be a plane parallel to plane yz and r be a rectangle parallel to plane xy. Then the next cases can occur (see figure 6):

1. The plane p miss both of the rectangle r and its directional neighbourhood $\Omega_s(\delta, r)$.

2. The plane p intersect the rectangle r and the neighbourhoods $\Omega_s(\delta, r^i_{\{1,-1\}})|(i = u) \vee (i = v)$.

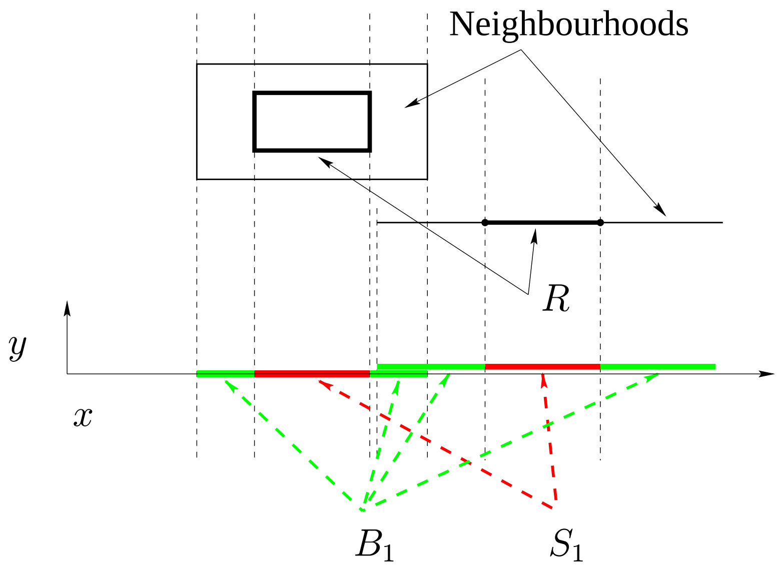


Figure 5: The sets S_1 and B_1

3. The plane p intersect the neighbourhoods $\Omega_s(\delta, r^i_{\{1,-1\}})|(i = u) \vee (i = v)$ and $\Omega_s(\delta, r^j_1)|j \neq i$
4. The plane p intersect the neighbourhoods $\Omega_s(\delta, r^i_{\{1,-1\}})|(i = u) \vee (i = v)$ and $\Omega_s(\delta, r^j_2)|j \neq i$

When the cut (3) or the cut (4) is used, then the boundaries between adjacent directional neighbourhoods are intersected (figure 6). Hence, the plane r_p containing the rectangle r intersects only constant number of original rectangles contained in $\Omega_s(\delta, r^j_1)$ (case 3) or $\Omega_s(\delta, r^j_2)$ (case 4) and unknown number of original rectangles contained in other neighbourhoods after using such cut. We call such cut (3) or (4) *effective cut* and the rectangle r *1-side free* after using both such cuts. If we use four effective cuts through all neighbourhoods $\Omega(\delta, r^{\{i,j\}}_{\{1,-1\}})$, then we can use *auto-partition* onto the rectangle r cutting only constant number of original rectangles.

In addition, let $Proj(p)$ be the projection of the plane p onto the plane xy (i.e. a line). Then the case (1) occurs iff $Proj(p)$ miss segments $s \in S_1$ and $b_1, b_2 \in B_1$ associated with r and its neighbourhoods. The case (2) occurs iff $Proj(p)$ crosses a segment $s \in S_1$ associated with r. The cases (3) (or (4)) occurs iff $Proj(p)$ crosses a segment b_1 (or b_2)$|b_1, b_2 \in B_1$.

The *simple low directional density* of a set R results from the *extended low directional density*. Hence, $\forall b_{\{1,2\}} \in B_1 : |s| \leq 1/\delta |b_{\{1,2\}}|$, where $1/\delta$ is a

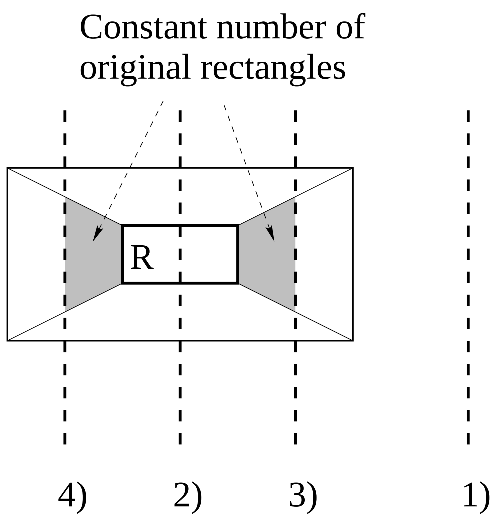


Figure 6: Intersections of r by a plane parallel with the yz plane.

constant. It follows from definition, that $n = |S_1| \leq |B_1| = 2n$ and $s \parallel b_{\{1,2\}}$. The assumptions of Lemma 3.3 are satisfied for the sets S_1 and B_1 and thus we can find such line $l = Proj(p)$, that $|l \cap S_1| \leq 1/\delta |l \cap B_1|$.

The BSP construction algorithm proceeds with loops consisting of two sections. In the first section, we process all ε-free rectangles and dispose them from S_i. In the second section, we split the original set R by a plane p and associated sets S_i and B_i by a line $l = Proj(p)$ into two portions according to Lemma 3.3, provided that S_i is not empty. The algorithm starts with $i = 1$.

Section 1:

```
while There are any $\varepsilon$-free rectangles in $R$ do
   begin
(1)  Pick an arbitrary $\varepsilon$-free rectangle $r \in R$ up;
(2)  Determine the plane $p$ containing $r$;
(3)  Eliminate all segments $b \in B_j | b \cap p \neq \emptyset$ from the sets $B_j | j \in \{1, ..., 3\}$;
(4)  Use $p$ as the splitting plane for sets $R$ and $S_j \cup B_j | j \in \{1, ..., 3\}$;
(5)  if $|S_i| > 1$ then recurse on the resultant sets;
   end;
```

Section 2:

```
if The sets $S_i$ are not empty then
```

```
  begin
(6) if There is not possibility to select a line $l$ according to Lemma 3.3 then
      // $|l \cap B|/|l \cap S| \geq \delta$
        Choose a new $B_i$, $S_i|i \in \{1, ..., 3\}$ not disturbing the Lemma 3.3 conditons;
        // If any sets $B_i$, $S_i$ satysfying the Lemma 3.3 conditions doesn't exist
        // then the rectangle with its longest side is $\varepsilon$-free (using extended low
        // direcitonal density).
(7) Select a line $l$ according to Lemma 3.3 and associated plane $p$;
(8) Eliminate all segments $b \in B_j|b \cap p \neq \emptyset$ from the sets $B_j|j \in \{1, ..., 3\}$;
(9) Use $p$ as the splitting plane for the sets $R$ and $S_j \cup B_j|j \in \{1, ..., 3\}$;
(10) if $|S_i| > 1$ then recurse on the resultant sets;
  end;
```

An example of splitting of a rectangle from lines (4) or (9) is shown in figure 8.

Now, we have to certify, that the assumptions of Lemma 3.3 are satisfied in section 2 of the algorithm and that the algorithm finishes with BSP tree of linear size.

We can observe, that there are exactly two segments $b_{\{1,2\}} \in B$ belonging to segment $s \in S$ at the start of this algorithm. A segment $b \in B$ is discarded in a section of the algorithm only in case, when it is crossed by (or contained in) a BSP splitting line.

Let r be a rectangle derived by several steps of the algorithm. We denote r^{or} the original rectangle, which r has been derived from.

We can not certify the assumptions of Lemma 3.3 in the case that there is a segment $s \in S$ such that no $b_{\{1,2\}}$ belonging to s exists. Such segment corresponds to a 1-side free or ε-free original rectangle r. We analyze these two possibilities now.

Firstly: the rectangle r is ε-free. Nevertheless, all ε-free rectangles have been discarded before entering the section 2. Hence, this event can not occur.

Secondly: the rectangle r is 1-side free[3] but r is not ε-free. In this case, we have to change the parameter i and continue the algorithm with new sets S_i, B_i. We show that a convenient parameter i always exists.

[3]We have to remind, that the cuts are drawn through the *simple directional neighbourhood* using the Lemma 3.3.

Let us suppose that we have sets $S_j^{'}$, $B_j^{'}|j \in \{1, ..., 3\}$ derived by the algorithm from the initial sets S_j, B_j and associated with a convex region R of a node of the BSP tree. In addition, we can not select any parameter i to continue the algorithm. In this case, any set $S_i^{'}|i \in \{1, ..., 3\}$ contains at least one segment s such that there is no $b \in B_i^{'}$ associated with s. The segment s is derived from an original rectangle r^{or}. Let $\mathcal{M}$ be a set of all such rectangles (i.e. the set of all original rectangles, the 1-side free rectangles in R was derived from) for the sets S_i. Because the rectangles are axes aligned, every rectangle projects either as copy of the original rectangle or as a line segment. Let us select the rectangle $r^{or} \in \mathcal{M}$ with maximal length of the longer side and non zero area and mark this original rectangle as A. W.l.o.g. we can suppose, that the selected rectangle is parallel to the plane xy. The length of longer side of A is L, as you can see on figure 7. Now we transfer our considerations onto the xy-coordinate.

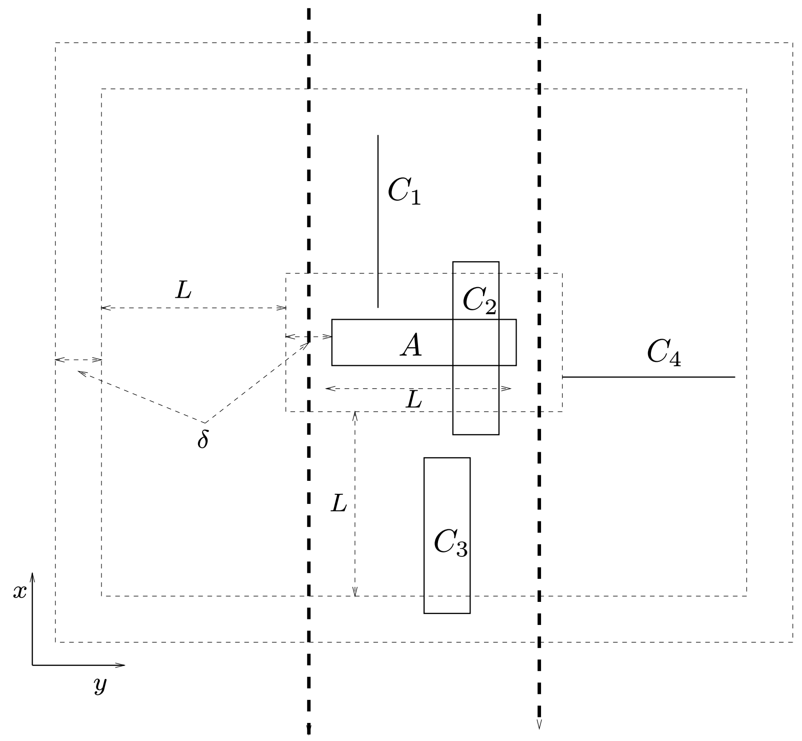


Figure 7: Proof

It results from the definition of *extended directional neighbourhood* that the width of A's neighbourhood is at least $2\delta L + L$ in both x and y directions. In additional, A is bounded by two cuts parallel with a coordinate axis (w.l.o.g. we can suppose that it is the x axis) and going through its *simple directional neighbourhood*. Further, we carry out two cuts through A's neighbourhood to bound the rectangle and to be able to use the ε-cut.

Again, let C be a rectangle $r_1^{or} \in \mathcal{M}$ and C is bounded by two cuts parallel with the y coordinate axis. Let L_1 be the length of the longer side of C. We can distinguish the next cases:

1. $A = C$. But then the rectangle r associated with A is bounded by four axes parallel cuts. Hence r is ε-free - a contradiction.

2. $A \neq C$. Then:

   (a) There are points $u \in A$ and $v \in C$ such that $|u, v|_{xy} < \delta L$, where $|u, v|_{xy}$ denotes the distance of points u,v in xy plane (see fig. 7, rectangles C_1 (or C_2), A). Because C is bounded by two cuts parallel with y coordinate axis and going through its *simple directional neighbourhood*, we can show that C with its *simple directional neighbourhood* lies whole inside of A with its *extended directional neighbourhood*. Precisely: Let w be a point inside of C with its *simple directional neighbourhood*. Then $|u, w|_{xy} < |u, v|_{xy} + L_1 + \delta L_1 < 2\delta L + L$. C is bounded by two cuts parallel with the y coordinate axis and going through its *simple directional neighbourhood*. From the previous considerations, both the cuts are going through the *extended directional neighbourhood* of A. Hence r is ε-free - a contradiction.

   (b) There are points $u \in A$ and $v \in C$ such that $|u, v|_{xy} < \delta$ does not exist (see fig. 7, rectangles C_3, A). Then $\forall(u \in A, v \in C)|u, v|_{xy} > \delta$. Nevertheless, there is a cut parallel with y coordinate axis going through the *simple directional neighbourhood* of B. Such cut must separate the rectangle r and the rectangle r_1. Hence, r and r_1 can not belong to the set $\mathcal{M}$ concurrently - a contradiction.

We have proved in the previous part, that a segment b belonging to s always exists when we are entering part (7) of the algorithm. Therefore $|S_i| \leq |B_i|$. In the whole algorithm, no changes of length of any segment b_{i_j} occur. Only a segment $s \in S_i$ can be split and hence shortened. It shows, that $\forall i \in \{1, ..., n\} : |s_i| \leq \alpha |b_{\sigma(i)}|$. The rest of assumptions of Lemma 3.3 is apparently true.

It remains to prove that this algorithm finishes with linear BSP tree for an input set of segments S. In order to bound the resultant BSP tree size, we summarize the total number of cuts in the course of the algorithm.

It is clear, that the nodes of resultant BSP tree contain only ε-free rectangles (or fragments of rectangles). Hereafter, at least one segment $b \in B_j$ or a ε-free rectangle (or its part) is treated in each pass of a section of the algorithm and no quite new segment $b \in B$ can arise in any new subregion. Hence, the algorithm finishes after a finite number of steps.

There are two kinds of partition used in the algorithm:

The partitioning in the first section of the algorithm uses the ε-free cuts. It means that the number of split rectangles by the directional neighbourhood of a rectangle is at most ε.

The partitioning in the second section of the algorithm uses a splitting plane derived from a splitting line of the Lemma 3.3. Given by the condition from Lemma 3.3, the number of crossed segments from the set S is less or equal $1/\delta$ times the number of crossed segments from the set B. Furthermore, to each rectangle (or part of rectangle) belongs exactly one segment $s \in S_i$ and at most two segments $b_{\{1,2\}} \in B_i$ for each $i \in \{1, 2, 3\}$.

Let us select an arbitrary fixed $i \in \{1, 2, 3\}$. If the objects are split only by the cuts perpendicular to i-axis and no object is split by a cut perpendicular to another axis, then at moast $2n\frac{1}{\delta} + n\varepsilon$ objects (rectangles) are split by this cuts overall (we have to cross $2n$ segments $b \in B_i$ and we use n free cuts). Hence, we have at most $n + 2n\frac{1}{\delta} + n\varepsilon = (1 + \frac{2}{\delta} + \varepsilon)n$ resultant objects.

Now, we have to take into consideration, that we are working with planar rectangles in $\mathcal{R}^3$. Any planar rectangle is 2-dimensional object. When a rectangle is split, a new segment $s \in S_{\{j,k\}}$ and at most two new segments b_1, $b_2 \in B_{\{j,k\}} | j \neq i, k \neq i$ can arise in a new subregion as you see at figure 8. Thereof, the number of generated objects (subrectangles) is bounded by $(1 + \frac{2}{\delta} + \varepsilon)^2 n$.

Together, the maximal number of all generated subrectangles is bounded by

$$(1 + \frac{2}{\delta} + \varepsilon)^2 n = O((1 + \frac{1}{\delta} + \varepsilon)^2 n)$$

Because δ and ε are constants, the number of objects in the BSP tree is at most $O(n)$ and the resultant BSP tree has linear size. $\square$

Remark: An appropriate directional neighbourhood could be defined for any polygon (not necessarily axes aligned) but it can be too large and hence

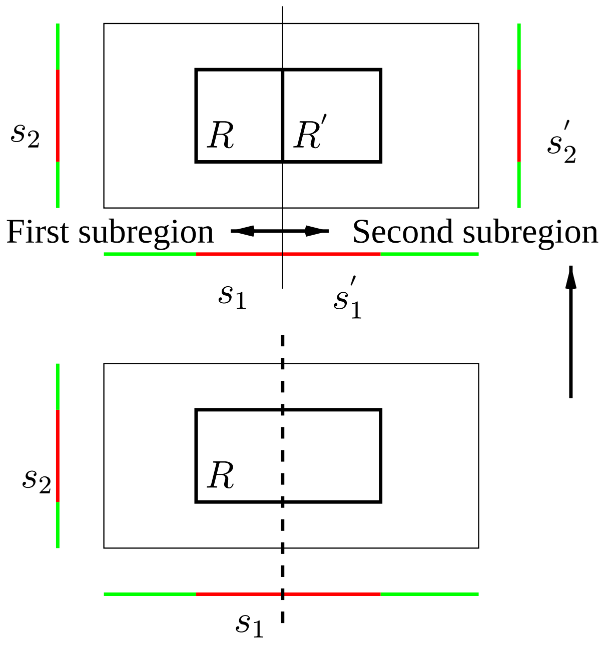


Figure 8: The rectangle R is divided into two parts.

useless. Moreover, there are technical problems with description and realization of ε-free cuts.

4 BSP of hyperrectangles in the space

Obviously, every hyperrectangle $E \in \mathcal{R}^3$ is created by set of six bounding rectangles. Hence, if we create a BSP of set of bounding rectangles, we have the BSP of hyperrectangles. The first idea is to use the set of rectangles with *extended low directional density*. Nevertheless, we propose better solution following from new definition of the *low directional density* of set of hyperrectangles. This definition exploits the idea of *extended low directional density* in two directions only. The third direction can have the neighbourhood small. So we can do very flat cuts using this technique.

Definition 4.1: *Let e be a axes aligned hyperrectangle with side vectors $\vec{e}_x$, $\vec{e}_y$ and $\vec{e}_z$. W.l.o.g. we can suppose that $\vec{e}_x \geq \vec{e}_y \geq \vec{e}_z$. We assign two directional neighbourhoods to the hyperrectangle e using the bounding rectangles of e.*

1. *We extend the neighbourhoods in directions $\vec{e}_x, \vec{e}_y$.*
   - $f_x = 1 + 2\delta$
   - $f_y = \frac{|\vec{r^x}|}{|\vec{r^y}|}(1 + 2\delta)$
   - $f_z = \delta$

   $\Omega_1(\delta, E) = \cup_{r_e}(\Omega(\delta, f, r_e))$
2. *We extend the neighbourhoods in directions $\vec{e}_x, \vec{e}_z$.*
   - $f_x = 1 + 2\delta$
   - $f_y = \delta$
   - $f_z = \frac{|\vec{r^x}|}{|\vec{r^z}|}(1 + 2\delta)$

   $\Omega_2(\delta, E) = \cup_{r_e}(\Omega(\delta, f, r_e))$

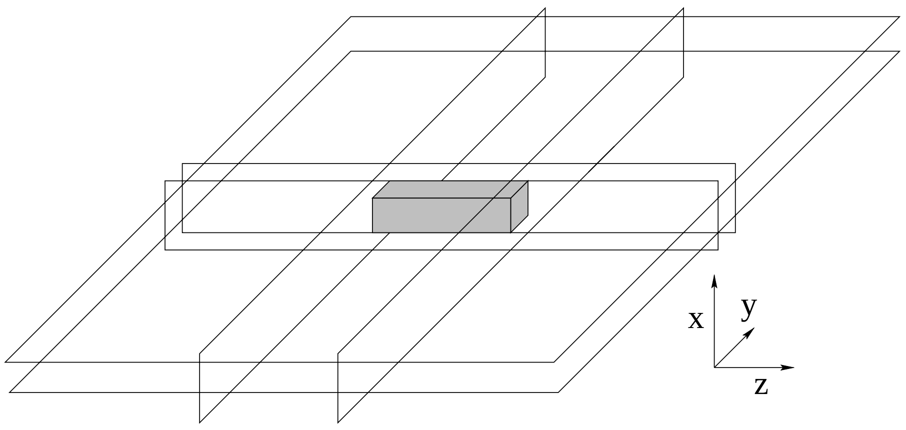


Figure 9: The $\Omega_1(\delta, E)$ neighbourhood.

We say, that the hyperrectangle e has (ε, δ)-low directional density, iff $|\Omega_1(\delta, e) \cap E| < \varepsilon$ or $|\Omega_2(\delta_3, e) \cap E| < \varepsilon$.

Lemma 4.2: *Let E be a set of hyperrectangles with (ε, δ)-low directional density. Then a linear BSP for the set E exists.*

Proof: The proof comes out from the Lemma 3.4. The beginning of this proof corresponds to the proof of Lemma 3.4 exactly. We build up tree auxiliary sets of segments $B_i, S_i | i \in \{1, ..., 3\}$ in the same manner. The Observation and their consequences hold as well. We can use the BSP construction algorithm without changes yet. The only part of the proof we have to top up is the certify, that the assumptions of Lemma 3.3 are satisfied in the section 2 of the algorithm. Specifically the part "Secondly" has to be adapted.

Secondly: the rectangle r is 1-side free but r is not free. In this case, we have to change the parameter i and continue the algorithm with new sets S_i, B_i. We show that a convenient parameter i always exists.

Let $\mathcal{M}$ be a set of rectangles defined in the same way as in Lemma 3.4. Let $A = r^{or} \in \mathcal{M}$ be the rectangle with maximal length of the longer side and bigger length of the shorter side. Let E be the hyperrectangle in the $\mathcal{R}^3$ space bounded by rectangles $r_1, ...r_6; r_1, r_2 \parallel xy$ plane, $r_3, r4 \parallel xz$ plane and $r_5, r_6 \parallel yz$ plane, $A \in \{r_1, ...r_6\}$. W.l.o.g. we can assume, that the functions f_x, f_y, f_z are defined for the neighbourhood extended in directions $\vec{e_x}, \vec{e_y}$ (i.e. $f_x = 1 + 2\delta, f_y = \frac{|\vec{r^x}|}{|\vec{r^y}|}(1 + 2\delta)$ and $f_z = \delta$).

- If the rectangle A fulfils *extended low directional density*, (i.e. the rectangle r is parallel with the xy plane in our case) then a segment b belonging to the segment s of A always exists, as was shown in the Lemma 3.4.

- If the rectangle A is parallel to the xz plane, then the two next cases can occur:

  - The (part of) hyperrectangle E containing the (sub) rectangle A is bounded by two cuts parallel with xy plane. Hence no rectangle (subrectangle of) r_1 or r_2 lies in this subspace. In this case, we can use the A as a rectangle with *extended low directional density*, because the two sides repealing the *extended low directional density* condition has been cut off.
  - A rectangle (subrectangle of) $Q \in \{r_1, r_2\}$ parallel with xy plane exists. It follows from the definition, that this rectangle must be at least as large as the rectangle A.

    If $Q \in \mathcal{M}$, then we can take q instead of r. The rectangle Q has the *extended low directional density* and we can use the technique from the Lemma 3.4.

Lemma 4.3: *Let R be a convex BSP region derived by the proposed algorithm. Let $A \in \mathcal{M}$ be a rectangle with maximal length* *of the longer side. Then the A has maximal length of the longer side among all rectangles $r_i | r_i \cap R \neq 0$.*

Proof: The *simple directional neighbourhood* of A is intersected by two parallel cuts. The distance of these cuts is at least $2\delta L + L$. The width of the neighbourhood of any rectangle $C \notin \mathcal{M}$ is less then $2\delta L + L$ because the neighbourhood of C is intersected by at most one cut. It implies that $C < A$. $\square$

If $Q \notin \mathcal{M}$, then a rectangle $Q_1 \in \mathcal{M} | Q_1$ has a longer side then Q; exists. Nevertheless, (the longest side of A) $\leq$ (the longest side of Q) $<$ (the longest side of Q_1). This is a contradiction with our selection of A as the rectangle with maximal length of its longer side.

- If the rectangle A is parallel to the xy plane, then analogous arguments holds as in the previous point.

The rest of this Lemma corresponds the rest of Lemma 3.4. $\square$

5 Design of the algorithm

The proof presented above provides us a pseudo algorithm how to create a linear BSP tree. However, the construction steps of the algorithm are not elementary and a brute force implementation could be very inefficient.

The second criterion of quality of resultant BSP tree (after size criterion) is balance. Now, we give a formal definition of *best-balanced cut* as used in [12].

Definition 5.1: *Let C be cutting hyperplane in the space, $C^<$ and $C^>$ denote the set of rectangles lying entirely in on of the two halfspaces generated by the cut C. The* **best balanced cut** *is defined to be a cut which minimizes the difference between $C^<$ and $C^>$, i.e.*

$$\delta_C = \| \, |C^<| - |C^>| \, \|,$$

where $\|x\|$ denotes the absolute value of x.

It is not possible to construct balanced BSP tree by the proposed algorithm in any time. For example, the sequence of nested cubes with a common center point and exponentially increasing diameters ($\{1, ..., 2^n\}$) has low directional density abundantly. Unfortunately, the presented algorithm can not create balanced BSP tree.

It is clear, that the proposed example is quite artificial. The balanced tree exists for a large class of practical scenes. In the rest of this paper we show an efficient algorithm involving trade-off between balance and size of the resultant tree.

The proposed construction algorithm is based on segment trees discovered by Bentley [6]. Segment tree is a data structure designed to handle intervals on the real line whose extremes belong to a fixed set of $O(n)$ endpoints. The endpoints can be normalized by replacing each of them by its rank in their left-to-right order. W.l.o.g., we may consider these endpoints as the integers in the range $[1, n]$.

We use the definition of F. Preparata and M. Shamos [17]:

Definition 5.2: *The segment tree is a rooted binary tree. Given integers l and r, with $l < r$, the segment tree $T(l, r)$ is recursively built as follows: It consists of a root v, with parameters $B[v] = l$ and $E[v] = r$ (B and E are mnemonic for "beginning" and "end," respectively), and if $r - l > 1$, of a left subtree $T(l, \lfloor(B[v] + E[v])/2\rfloor)$ and a right subtree $T(\lfloor(B[v] + E[v])/2, r\rfloor)$. (The roots of these subtrees are naturally identified as* **LSON**$[v]$ *and* **RSON**$[v]$*, respectively.) The parameters $B[v]$ and $E[v]$ define the interval $[B[v], E[v]] \subseteq [l, r]$ associated with node v. The set of* **intervals** *$\{[B[v], E[v]] : v \text{ a node of } T(l, r)\}$ are the standard intervals of $T(l, r)$. The* **standard intervals** *pertaining to the leaves of $T(l, r)$ are called the* **elementary intervals**. *It is straightforward to establish that $T(l, r)$ is balanced (all leaves belong to two contiguous levels) and has depth $\lceil \log_2(r - l) \rceil$.*

We will maintain the set of segments B_i, S_i in a segment tree extended by extra data. Using this trees, we will be able to select the splitting plane according to Lemma 3.3 effectively.

5.1 Preliminary calculations

Because the definition of low directional density of hyperrectangles depends on neighbourhood, it is necessary to make a preliminary calculation. To this purpose, we use the range trees with fractional cascading technique [13, 22].

We have to choose the better from the possible extended directional neighbourhoods. There are two rectangles parallel with any axes aligned plane: $r_1, r_2 \parallel xy$ and $r_3, r_4 \parallel xz$.

Let $\varepsilon_i = |\Omega_e(\delta, r_i) \cap E|$, where $e \in E$. If $\varepsilon_1 + \varepsilon_2 \leq \varepsilon_3 + \varepsilon_4$ then we select the neighbourhood of rectangles r_1, r_2 else the neighbourhood of rectangles r_3, r_4 in opposite case.

The $\Omega_e(\delta, r)$ can be substituted with a rectangle $v = \Omega_e(\delta, r) \cup r$. This is correct because the number of intersections between the rectangle v and E is the same in the case of nonintersecting hyperrectangles. In the case of intersecting hyperrectangles, we can subtract the number of intersections between E and r. Hence, we have $O(n)$ intersection questions between the bounding rectangles of E and the neighbourhood rectangles (there are exactly $6n$ neighbourhood rectangles). Each intersection query can be computed in $O(\log^2 n)$ time using the range tree with fractional cascading and we obtain the number of bounding rectangles intersecting the neighbourhood rectangle. The range tree can be constructed in $O(n \log^2 n)$ time. Together, we spend $O(n \log^2 n)$ time by computing the extended directional neighbourhoods to the input set of hyperrectangles.

5.2 The trade-off algorithm

Our implementation corresponds to the presented pseudocode. We execute cuts according to Lemma 3.3 (i.e. it holds that $|B_l|/|S_l| \geq \delta$ where $|B_l|$ is number of segments from the set B crossed by the line l and $|S_l|$ is number of segments from the set S crossed by the line l) until 1-side free rectangles make it impossible in any dimension.

Then ε-free ctus are performed. If there is no rectangle bounded by four cuts going through its simple directional neighbourhood, then we use a hyperplane containing the longist rectangle with extended low directional density. It follows from Lemma 4.2, that such rectangle is ε-free.

The main problems appear to select a line l according to Lemma 3.3. If such line exists (line (7) of the algorithm), we eliminate the crossed segments

$b \in B$ (lines (4) and (8) of the algorithm) and split the sets $S_j \cup B_j | j \in \{1, ..., 3\}$ (lines (5) and (10) of the algorithm). If this is done by brute-force manner, then the construction time can be quadratic. The reason of the quadratic time is that we can spend $O(n)$ time by searching for the splitting line and the resultant BSP tree can be very unbalanced. In this case, the recursion $R(n) = R(n-1) + O(n)$ leads to quadratic time.

In order to do it efficiently, we use segment trees. Initially, we have three pairs of sets B_i, S_i, $i \in \{1, ..., 3\}$. We create three segment trees (one for any dimension) $T_{\{1,...,3\}}$ by the following way: The segment tree T_i contains all segments $b \in B_i$ and $s \in S_i$. We will suppose in the rest of the paper, that the endpoints contained in each T are in general position. If this condition doesnt hold, we use the lexicographical ordering as was described in the Lemma 3.4 Moreover, we maintain the next items in each node N of the T:

1. *BS_quotient* – the quotient $|B_N|/|S_N|$ of segments b and s contained in this node N.
2. *best_descendant* – a pointer to a descendant node. Let $N.T_i(l, r)$ be the subtree generated by node N of the $T_i(1, N)$ and let m_i be a line perpendicular to the i axis and intersecting the best quotient $|b|/|s|$ of segments contained in $N.T_i(l, r)$ (if a such line can intersect only b segments, then the line intersecting most b segments is selected). The pointer *best_descendant* selects the descendant node, which contains elementary interval intersected by m_i.
3. *best_quotient* – the numbers $|B_{N.T_i}|$ and $|S_{N.T_i}|$ of segments b and s of the subtree $N.T_i(l, r)$ crossed by the line m_i with best quotient $|b|/|s|$.
4. $|C^<|$ – the number of segments s lying entirely in the left subtree.
5. $|C^>|$ – the number of segments s lying entirely in the right subtree.

We also suppose, that we have cross pointers between segments belonging to identical rectangle and subspace.

It is clear, that the finding of line l according to Lemma 3.3 (line (7)) can be carry out in $O(\log n)$ time by recursively descent using the described segment tree for a set of $O(n)$ segments. Moreover, we can order the descent to select the best balanced cut fulfilling the Lemma 3.3 conditions.

The segment tree $T(l, r)$ is a static structure with respect to the initial set of segments (i.e. the segment trees does not support insertions or deletions of segments with new endpoints). Nevertheless, it can store intervals, whose extremes belong to the set $\{l, ..., r\}$ in a dynamic fashion (that is, supporting insertions and deletions). Since, we can delete a crossed segment b in $O(\log n)$ time (lines (4), (8)).

We have to proceed more properly by splitting a segment s. We can not select the splitting line in any place of the crossed elementary interval because no new endpoint can arise. Hence, we select one of the endpoints of the elementary interval. Now, we can split the segments s in $O(\log n)$ time as well (lines (5), (10)). We should take a note that the new data of the segment tree can be updated in the same time.

The last problem occurs, when we have to split the sets of segments $S_j \cup B_j | j \in \{1, ..., 3\}$. As it has been shown, the splitting plane can be found in $O(\log n)$ time using segment trees and we can determine the bigger of resultant sets (Big_i) and the lesser of resultant sets ($Small_i$). Let us suppose, the set $Small_i$ contains $O(m)$ segments. We take this segments from the segment tree $T_i(l, r)$ away and create a new segment tree for this set from scratch.

In this way, we get two new segment trees (one for each new subset of segments) and the algorithm can continue recursively.

Lemma 5.3: *The proposed algorithm runs in $O(n \log^2 n)$ time and space.*

Proof: The preliminary calculations can be done in $O(n \log^2 n)$ time and space as it has been shown. Next, we have to analyze the trade of algorithm.

The steps (1) and (2) of the algorithm can be done in constant time. Moreover, if we search for the best balanced cutting hyperplane containing an ε-free rectangle, then we spend $O(\log n)$ time using the segment tree. The steps (3) and (8) takes $O(\log n)$ time for each deleted segment and the step (7) takes $O(\log n)$ time as was shown in the previous text. The steps (5), (6) and (10) consume constant time apparently.

Now, we aim to bound the steps (4) and (8) and the recursion. In these steps, we have to create the smaller segment tree of size $O(m)$ from scratch (it takes $O(m \log m)$ time) and update the original segment tree (deleting of $O(m)$ segments takes $O(m \log n)$ time). Together: $O(m \log m) + O(m \log n) = O(m \log n)$. Now we use the issue of Lemma 4.2 that the consequential BSP tree has linear size. Thereof, there is such constant c that the number of nodes of the BSP tree is at most cn. The running time of the algorithm can be bounded by sum of $O(n \log^2 n)$ and the recursion

$$R(n) \leq \max_{0 \leq m \leq cn/2} O(m \log n) + T(m) + T(cn - m)$$

which implies that $R = O(n \log^2 n)$. So the running time of the algorithm is bounded by $O(n \log^2 n)$. Because the memory requirements of segment trees are $O(n \log n)$ the recurrence holds for the space as well and the space complexity is identical. □

Theorem 5.4: *Let E be a set of hyperrectangles with (ε, δ)-low directional density. Then the linear size BSP tree can be constructed in $O(n \log^2 n)$ time and $O(n \log^2 n)$ space. Moreover, we can trade-off between balance and size of the resultant tree.*

6 Conclusion

In the proposed paper, we have tried to design an effective algorithm for construction of low size BSP for a set of hyperrectangles. Such BSP can be enormously useful in real-life problems because any set of bounding-boxes of objects forms a set of hyperrectangles.

The presented algorithm creates linear BSP tree for so-called *low directional density* scenes. We compare this class of scenes with *r-boundedness* [12] and *unclutteredness* [9] in the rest of this section.

Definition 6.1: *For $r > 1$, a set S of non-overlapping axis-parallel hyperrectangles in $\mathcal{R}^d$ is said to be* **r-bounded** *if for all $i, 1 \leq i \leq d$, the ratio between the longest and shortest length of x_i-edges of hyperrectangles in S is bounded by r.*

Definition 6.2: *Let S be a d-dimensional scene and let $\kappa \geq 1$ be a parameter. We call S a κ-**cluttered scene**, if any hypercube whose interior does not contain a vertex of one of the bounding boxes of the objects in S is intersected by at most κ objects in S. The clutter factor of a scene is the smallest κ for which it is κ-**cluttered**.*

It can be shown that the low directional density is independent on r-boundedness and unclutteredness. Since it enlarges the class of objects for which we can create linear BSP.

Figure 10 shows an example of hyperrectangles which have low directional density but neither unclutteredness nor r-boundedness is satisfied.

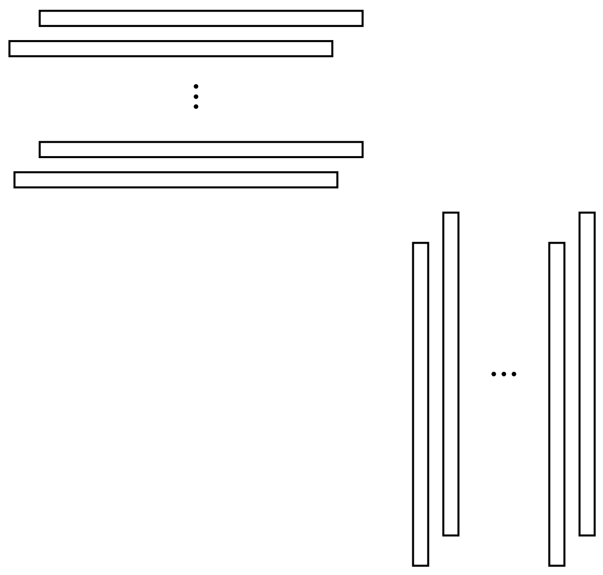


Figure 10: Scene with low directional density

The algorithm can be simply extended for any constant dimension. However, the size of constant of the resultant BSP increases exponentially with respect to the space dimension. The time and the space complexity of the algorithm is $O(\max\{n \log^{d-1} n, n \log^2 n\})$ in d-dimensional space. In three-dimensional space, it is only slightly worse than optimal $O(n \log n)$ time complexity of De Berg [9] and Viet Hai [12] algorithm.

References

[1] P. K. Agarwal, E. F. Grove, T. M. Murali, and J. S. Vitter. Binary space partitions for fat rectangles. In *Proc. 37th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 482–491, October 1996.

[2] Pankaj K. Agarwal, T. Murali, and J. Vitter. Practical techniques for constructing binary space partitions for orthogonal rectangles. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 382–384, 1997.

[3] Pankaj K. Agarwal and Subhash Suri. Surface approximation and geometric partitions. In *Proc. 5th ACM-SIAM Sympos. Discrete Algorithms*, pages 24–33, 1994.

[4] John Milligan Airey. *Increasing Update Rates in the Building Walkthrough System with Automatic Model-space Subdivision and Potentially Visible Set Calculations.* Ph.D. Thesis, Dept. of Computer Science, University of North Carolina, Chapel Hill, 1990.

[5] C. Ballieux. Motion planning using binary space partitions. Technical Report Inf/src/93-25, Utrecht University, 1993.

[6] J. L. Bentley. Solutions to Klee's rectangle problems. Technical report, Carnegie-Mellon Univ., Pittsburgh, PA, 1977.

[7] Norman Chin and Steven Feiner. Near real-time shadow generation using BSP trees. In *Proc. SIGGRAPH '89*, pages 99–106, New York, August 1989. ACM SIGGRAPH.

[8] M. de Berg, M. de Groot, and M. Overmars. New results on binary space partitions in the plane. In *Proc. 4th Scand. Workshop Algorithm Theory*, volume 824 of *Lecture Notes in Computer Science*, pages 61–72. Springer-Verlag, 1994.

[9] Mark de Berg. Linear size binary space partitions for fat objects. In *Proc. 3rd Annu. European Sympos. Algorithms*, volume 979 of *Lecture Notes Comput. Sci.*, pages 252–263. Springer-Verlag, 1995.

[10] H. Fuchs, G. D. Abrams, and E. D. Grant. Near real-time shaded display of rigid objects. *Comput. Graph.*, 17(3):65–72, 1983. Proc. SIGGRAPH '83.

[11] H. Fuchs, Z. M. Kedem, and B. Naylor. On visible surface generation by a priori tree structures. *Comput. Graph.*, 14(3):124–133, 1980. Proc. SIGGRAPH '80.

[12] Nguyen Viet Hai. *Optimal Binary Space Partitions for Orthogonal Objects.* Ph.D. Thesis, Swiss Federal Institute of Technology (ETH), Zurich, 1996.

[13] G. S. Lueker. A data structure for orthogonal range queries. In *Proc. 19th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 28–34, 1978.

[14] B. Naylor, J. A. Amanatides, and W. Thibault. Merging BSP trees yields polyhedral set operations. *Comput. Graph.*, 24(4):115–124, August 1990. Proc. SIGGRAPH '90.

[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. Research Report 158, Univ. Warwick, 1990.

[17] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, 3rd edition, October 1990.

[18] R. A. Schumacker, R. Brand, M. Gilliland, and W. Sharp. Study for applying computer-generated images to visual simulation. Technical Report AFHRL–TR–69–14, U.S. Air Force Human Resources Laboratory, 1969.

[19] S. J. Teller. *Visibility Computations in Densely Occluded Polyhedral Environments.* Ph.D. Thesis, Dept. of Computer Science, University of California, Berkeley, 1992.

[20] W. C. Thibault and B. F. Naylor. Set operations on polyhedra using binary space partitioning trees. *Comput. Graph.*, 21(4):153–162, 1987. Proc. SIGGRAPH '87.

[21] P. Tobola and K. Nechvíle. Linear bsp tree in the plane for set of segments with low directional density. In *Proc. 7th International Conference in Central Europe WSCG '99*, pages 297–304, 1999.

[22] D. E. Willard. *Predicate-oriented database search algorithms.* Ph.D. thesis, Aiken Comput. Lab., Harvard Univ., Cambridge, MA, 1978. Report TR-20-78.