Linear BSP Trees for Sets of Hyperrectangles with Low Directional Density

Petr Tobola

Karel Nechvíle

Faculty of Informatics
Masaryk University, Botanická 68a
625 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 space with constant dimension. 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 $\mathbb{R}^3$. 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, partitioning, hyperrectangle

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. 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.

Informally: *Binary Space Partition*, or *BSP* (initially introduced by Schumacker [16]) is a recursive partitioning of the space with objects by a suitable hyperplane. The partitioning process is repeated for new subspaces until only one fragment of any object occurs in each 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 readily see that convex regions associated with nodes of the same level form 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 [10], visibility solution [17], shadow generation [7], objects modeling, 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. If this is the case, the original scene will be divided into lot of fragments. However, the efficiency of algorithms benefiting from BSP depends on the size of the resulting BSP. This is why the split hyperplanes must be selected 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, 10, 17]), which however can create tree of excessive size under unfavorable circumstances($\Omega(n^2)$

in the plane and $\Omega(n^3)$ in the space). The first provable bounds were obtained by Paterson and Yao [13, 14]. They showed [13], that the optimal size of BSP in the worst case is $\Theta(n^2)$ in the space and $O(n \log n)$ in the plane. The next result of these authors [14] was the optimal size BSP algorithm for the set of orthogonal objects with $\Theta(n^{3/2})$ in the space in the worst case and $\Theta(n)$ in the plane in the worst case .

However, most of randomly created BSP trees have reasonable behavior for practical scenes. Their sizes are considerably smaller than the worst case boundary. Modern algorithms try to use these properties to construct nearly linear BSP trees. Pankaj K. Agarwal et al. [1] solved the problem of 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. Their algorithm is linear with respect to 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 [11] published an algorithm creating linear BSP trees for set of *r-bounded* hyperrectangles in $\mathbb{R}^d$. 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 [11], 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 [18] 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 some basic notions and definitions. Section 3 is devoted to BSP of axes aligned rectangles in $\mathbb{R}^3$ and contanis definiton of *low directional density.* Section 4 extends our considerations for sets of hyperrectangles in $\mathbb{R}^3$ and contains a description of pseudo-code of BSP construction algorithm. In section 5, we describe how to write the pseudo-code efficiently. The comparison with other algorithms and conclusion follows in section 6.

Because of the lack of space we omitted proofs of Lemmas and some parts from this paper. You can find it in [19].

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*[1]*:*

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 *uncluttered*ness [9] and the Nguyen Viet Hai's *r-boundedness* [11] 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-

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

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).

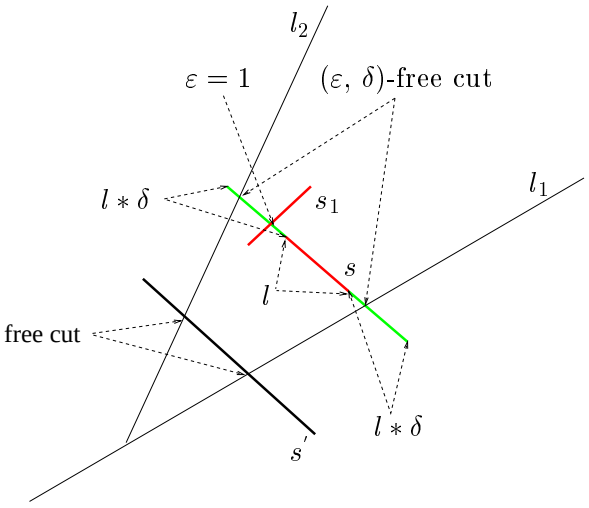

Figure 1: Free cut

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.

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:*

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

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 [18]. *We call the $\Omega(\delta, f_u, f_v, r)$ neighbourhood* **simple directional neighbourhood** *and sign $\Omega_s(\delta, r)$.*

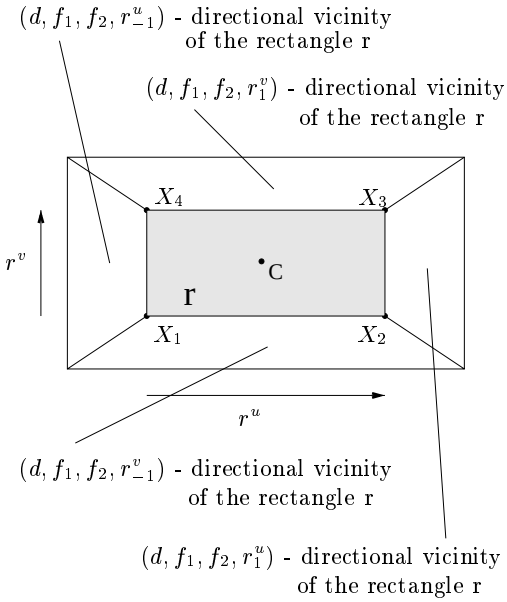

Figure 2: Directional neighbourhood of rectangle r

Definition 2.3: *Let R be a set of rectangles in the space, $r_i \in R$ and ε be an integer constant. The rectangle r_i is called* **free** *if $\Omega(\infty, f_u, f_v, r_i) \cap R = \emptyset$. The rectangle r_i is called* ε**-free** *if $|\Omega(\infty, f_u, f_v, r_i) \cap R| < \varepsilon$.*

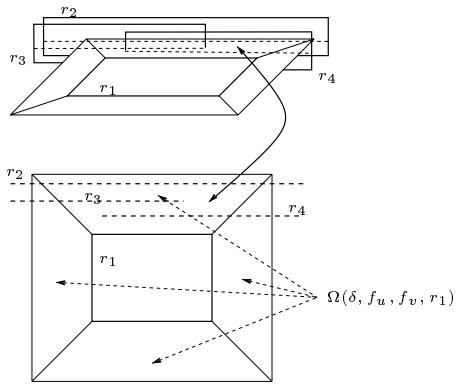

Figure 3: There are three rectangles crossing the $\Omega(\delta, f_u, f_v, r_1)$ on the figure. Specifically, the rectangles r_2, r_3 and r_4.

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 (see figure 3).*

*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 axis aligned rectangles with simple low directional density that no linear BSP exists.*

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**.

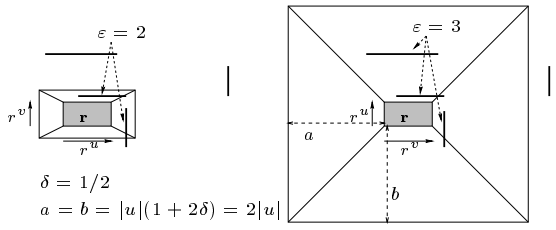

Figure 4: *Simple low directional density* (on the left) and *Extended low directional density* (on the right) of rectangle r parallel with xy plane.

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 $\exists(s_i) : s_i \nparallel v$ 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 rectangles with extended low directional density. Then a linear BSP for the set R exists.*

4 BSP of hyperrectangles in the space

Obviously, every hyperrectangle $E \in \mathbb{R}^3$ 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 axis 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 $\vec{e}_x, \vec{e}_z$.*
   - $f_x = 1 + 2\delta$
   - $f_y = \delta$
   - $f_z = \frac{|\vec{r}^x|}{|\vec{r}^z|}(1 + 2\delta)$

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

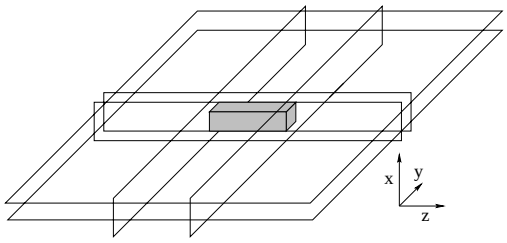

Figure 5: The $\Omega_1(\delta, E)$ neighbourhood.

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.*

Now, we can describe the pseudo-algorithm of construction of BSP:

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 6. 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.

Any segment $b \in B$ expresses the neighbourhood of a segment $s \in S$.

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.

The BSP construction algorithm proceeds with loops consisting of two sections until only one fragment remains in the resultant subspace. 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 ε-free rectangles in R **do**
begin
(1) Pick an arbitrary ε-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) Recurse on the resultant sets;
end;

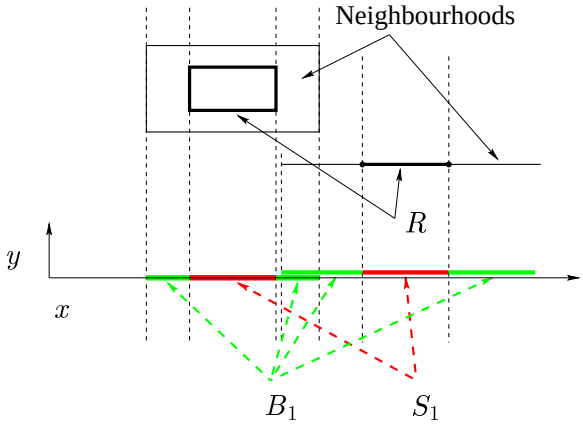

Figure 6: The sets S_1 and B_1

Section 2:

begin
(6) **if** There is not possibility to select a line l according to Lemma 3.3 // $|l \cap B_i|/|l \cap S_i| \geq \delta$
then 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 ε-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) Recurse on the resultant sets;
end;

An example of splitting of a rectangle from lines (4) or (9) is shown in figure 7.

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.

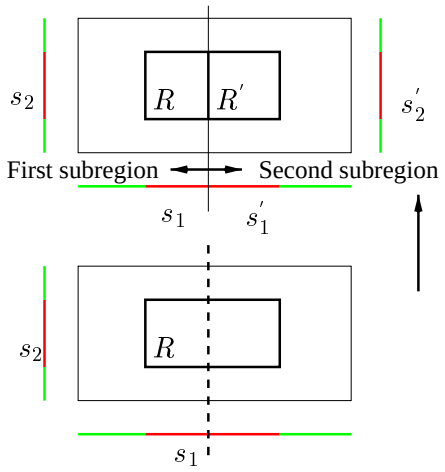

Figure 7: The rectangle R is divided into two parts.

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 [11].

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 [15]:

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** *$\{[B[v], E[v]] : v$ 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 [12].

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 di-

rectional neighbourhoods to the input set of hyperrectangles.

5.2 The trade-off algorithm

Our implementation corresponds to the presented pseudo-code. 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 $T_i(1, K)$ 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. $|C^<|$ ($|C^>|$) – the number of segments s lying entirely in the left (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.*

Theorem 5.4: *Let E be a set of hyperrectangles with (ε, δ)-low directional density in the $\mathbb{R}^3$ space. 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 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. It can be shown that the *low directional density* is independent of *r-boundedness* [11] and *unclutteredness* [9]. Since it enlarges class of objects, which we can create linear BSP for.

The algorithm can be simply extended for any constant dimension. However, the size of constant of resultant BSP increases exponentially with respect to the space dimension. The time and space complexity of the algorithm is $O(n \log^{d-1} n)$ in d-dimensional space.

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, 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.

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

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

[13] 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.

[14] M. S. Paterson and F. F. Yao. Optimal binary space partitions for orthogonal objects. Research Report 158, Univ. Warwick, 1990.

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

[16] 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.

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

[18] 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.

[19] P. Tobola and K. Nechvíle. Linear bsp trees for sets of hyperrectangles with low directional density. Technical report, Masaryk Univ., Brno, 2000.