FI MU

Faculty of Informatics
Masaryk University

Linear BSP Tree in the Plane for Set of Segments with Low Directional Density

by

Petr Tobola
Karel Nechvíle

FI MU Report Series FIMU-RS-99-07

 September 1999

Linear BSP Tree in the Plane for Set of Segments with Low Directional Density

Petr Tobola, Karel Nechvíle [1]

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

Abstract: We introduce a new BSP tree construction method for set of segments in the plane. Our algorithm is able to create BSP tree of linear size in the time $O(n \log^3 n)$ for set of segments with low directional density (i.e. it holds for arbitrary segment s_i from such set, that a line created as extension of this segment doesn't intersect too many other segments from the set in a near neighbourhood of s_i) and a directional constant δ belonging to this set.

keywords: BSP, segment, tree, partitioning

1 Introduction

Most of computer graphics and computer 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 suitable recursive partitioning of the space into subspaces.

A global 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 which splits the space (and possibly some of the objects) into two parts.

Informally: *Binary Space Partition*, or *BSP* (initially introduced by Schumacker [Schum69]) 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 because otherwise we would not be able to ensure finishing of the splitting.

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

The BSP for a set of objects can be naturally expressed as a tree structure. The splitting hyperplanes and objects contained 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 [Fuchs80], visibility solution [Telle92], shadow generation [Chin89], objects modelling [Naylo90, Thiba87], surface approximation [Agarw94], or robot motion planing [Balli93].

When we split the space by some hyperplane then some objects can be unwillingly divided into two or more parts. In such way, the original scene can be divided into a lot of fragments. (An example of two alternative BSPs for a set of segments is displayed on figure 1. It provides quite different outcomes.) However, the efficiency of algorithms benefitting from BSP depends on the size of consequential BSP. This is the reason for necessity to select the split hyperplanes carefully.

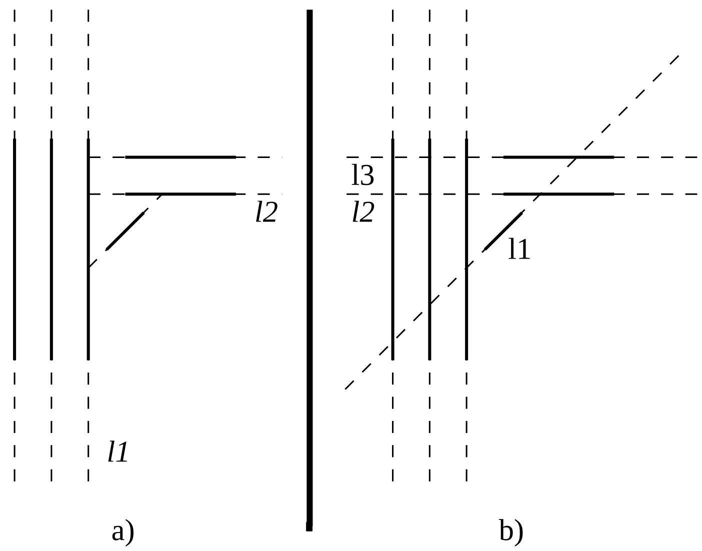

Figure 1: BSP of set of segments in the plane

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 [Airey90, Fuchs80, Fuchs83, Telle92, Thiba87]), 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 [Pater90a, Pater90b]. They showed [Pater90a], 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 [Pater90b] 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 behaviour for practical scenes. Their size 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. [Agarw96] solve 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 [Agarw97] they compared implementation of this algorithm with other BSP algorithms. It was shown that theirs algorithm is really suitable in practice.

Mark de Berg et al. have extensively studied the problem of BSP in the plane [Berg94]. 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, and they proposed effective algorithms to solve it (in the time $O(n \log \log n)$, $O(n \log n)$ and $O(n \log^2 n)$).

In [Berg95], Mark 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.

Here we propose a new BSP trees construction method for sets of segments in the plane. This quite simple method can provide BSP trees of linear size under condition of so-called *low directional density of segments.* Approximately, the principle is as follows: Let us suppose, some given segment satisfies the condition of low directional density. Now we can take an appropriate splitting hyperplane which releases one of directions designated by extension of the segment. After we release both directions of such segment, we can use a splitting hyperplane which contains this segment and creates only constant number of cuts.

Section 2 of this paper contains basic definitions. In section 3 we describe the principles and the functions of the algorithm in detail and prove its complexity. Section 4 contains analysis and concluding comments.

Due to lack of space we defer proofs of Lemma 1 and Lemma 2 to the full version of this paper. You can find it at:
http://www.fi.muni.cz/˜ptx/papers/ldd_bsp.ps.gz

2 Basic definitions

Definition 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 $\mathcal{B}$.

Denotation: Let p be a set of points (segment or line) and S be a set of segments. In the following text we will use this (slightly incorrect) notation for simplification: $p \cap S \equiv \{p \cap s_i\}; s_i \in S$.

Definition 2: Let S be a set of segments in the plane, $s_i \in S$ be a segment with endpoints $s_i[X]$ a $s_i[Y]$. Then $(\delta, s_i[X])$-*directional vicinity of segment* s_i (we will mark it $\Omega(\delta, s_i[X])$ is a set of points $\{s_i[X] + c\vec{u}\}$, where $c \in \langle 0, \delta \rangle$, $\vec{u} = s_i[X] - s_i[Y]$. δ-*directional vicinity of segment* s_i (we will mark it $\Omega(\delta, s_i)$) is a union $\Omega(\delta, s_i[X]) \cup \Omega(\delta, s_i[Y])$.

Definition 3: Let S be a set of segments in the plane, $s_i \in S$ be a segment with endpoints $s_i[X]$ a $s_i[Y]$. In additional let $\Omega_1 = \Omega(\infty, s_i[X])$, $\Omega_2 = \Omega(\infty, s_i[Y])$ and ε be a integer constant. We say that segment s_i is:

[2]We take up the definition of Agarwal [Agarw96]

- *free,* if $\forall j \in \{1,2\} : \Omega_j \cap S = \emptyset$.
- ε-*free,* if $\forall j \in \{1,2\} : |\Omega_j \cap S| < \varepsilon$

Definition 4: We say that a segment s has (ε, δ)-*low directional density,* if the following holds:
$(|\Omega(\delta, s[X]) \cap S| \leq \varepsilon) \wedge (|\Omega(\delta, s[Y]) \cap S| \leq \varepsilon)$, whereas ε is a integer constant and δ is a real constant. We say, that a set of segments S has (ε, δ)-*low directional density,* if any segment $s_i \in S$ has (ε, δ)-low directional density. We will call the constant δ mentioned above the *directional constant.*

3 The BSP construction method

3.1 Preliminary

One of generally applicable techniques for creating BSP trees are so-called *free cuts* (see figure 2). They make use of the following idea: 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.

We will use generalization of this idea into so-called ε-*free cuts,* (see figure 3) which can cut only constant number (ε) of other segments in our algorithm. For algorithm's intentions, we suppose that any segment has (ε,δ)-low directional density (i.e. there is δ-directional vicinity for all segment, such that each is crossed by at most ε of segments).

The proper algorithm will execute a recursive splitting of a set of segments S in the plane. In the case, there is any ε-free segments $s \in S$, we select a line l containing this segment. Otherwise, let $A = \{s_i \cap l | s_i \in S\}$ and $B = \{\Omega(\delta, s_i) \cap l | s_i \in S\}$ for an appropriate δ. We choose splitting line l in the way, that the following holds: either $A = \emptyset$ and $B \neq \emptyset$, or $A \neq \emptyset$ and $|A|/|B| < \varepsilon$ where ε is a constant.

The second condition denotes the case, where we cannot apply ε-free cuts. Hence we choose a cut bounding a sufficient number of segments, which prepare the terms for repeated application of ε-free cuts.

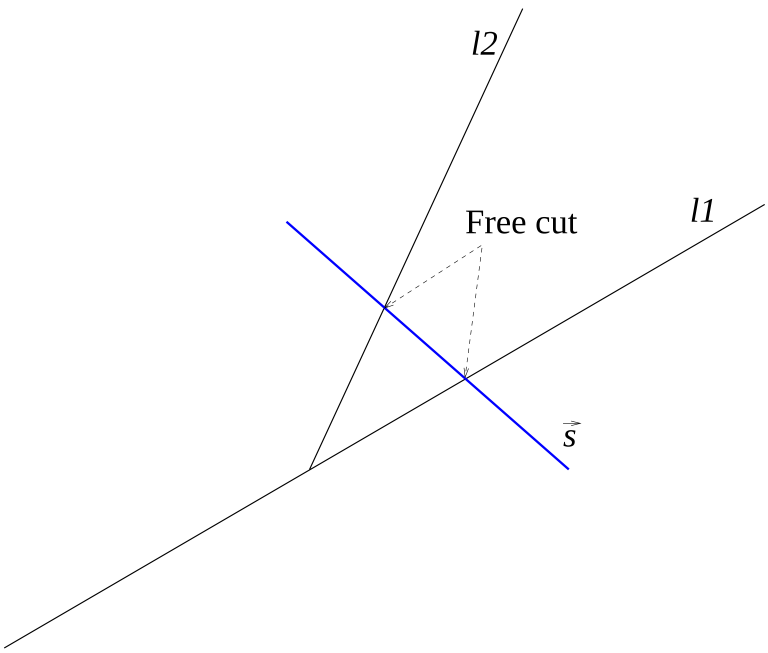

Figure 2: Free cut

3.2 Correctness of approach

Lemma 1: *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: $\exists(p) : |p \cap S| \leq \alpha|p \cap B|$.

Proof: We can suppose without lost of generality, that the vector v is parallel with axis y (otherwise, we can rotate this scene into that position). Let l be an arbitrary line perpendicular to vector v in the way, that any segment $s \in S$ or $b \in B$ lie above this line (so in the halfspace l^+). We project all segments from sets S and B to v onto this line. Let us suppose, no two endpoints of this segments project into single point on l (we will solve this

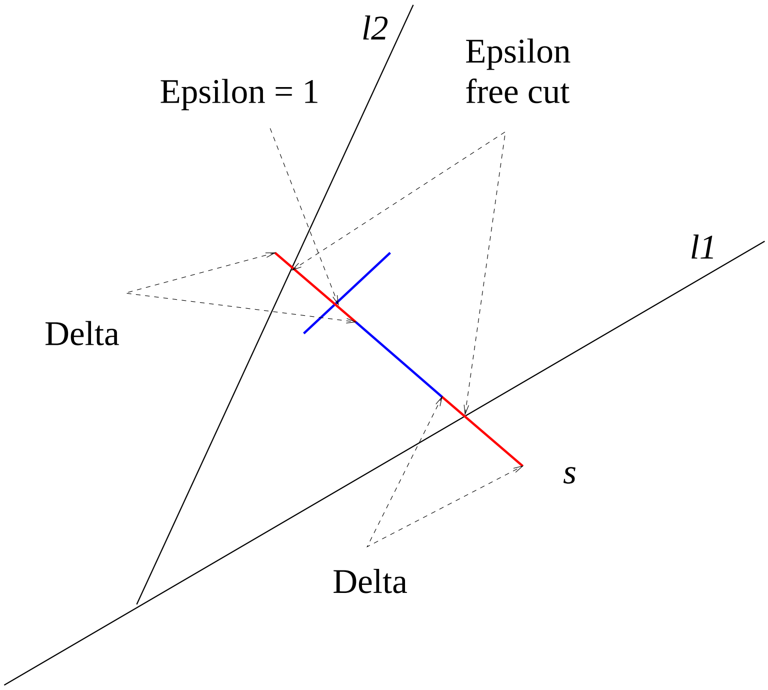

Figure 3: Epsilon-free cut

special case later). The projected endpoints $(X_1, ...X_{4n+2k})$ split the line l into $4n+2k+1$ fragments $l_1, ..., l_{4n+2k+1}$ (two halflines and $4n+2k-1$ segments). Furthermore, let l_i^k is a line perpendicular to l and crossing the middle of fragment l_i. We assign two parameters to any fragment l_i: $l_i^s = |\{s|s \cap l_i^k \neq 0; s \in S\}|$ and $l_i^b = |\{b|b \cap l_i^k \neq 0; b \in B\}|$. The parameter l_i^s represents the number of segments $s_i \in S$, which lie upward l_i and the parameter l_i^b represents the number of segments $b_i \in B$, which lie upward l_i (see Fig. 4).

Let us assume, there is not any line p parallel to v so, that $|p(v) \cap S| \leq \alpha |p(v) \cap B|$. Then it holds for all segments l_i, that $l_i^s > \alpha l_i^b \Rightarrow l_i^s |l_i| > \alpha l_i^b |l_i|$ and it shows

$$\sum_{i=1}^{4n+2k-1} l_i^s |l_i| > \alpha \sum_{i=1}^{4n+2k-1} l_i^b |l_i| \tag{1}$$

Now, we have to be aware of the fact that any segment l_i is a part of projection of l_i^s segments from the set S and of l_i^b segments from the set B. Hence, the sum $\sum_{i=1}^{4n+2k-1} l_i^s |l_i|$ is sum of lengths of all projected segments from the set

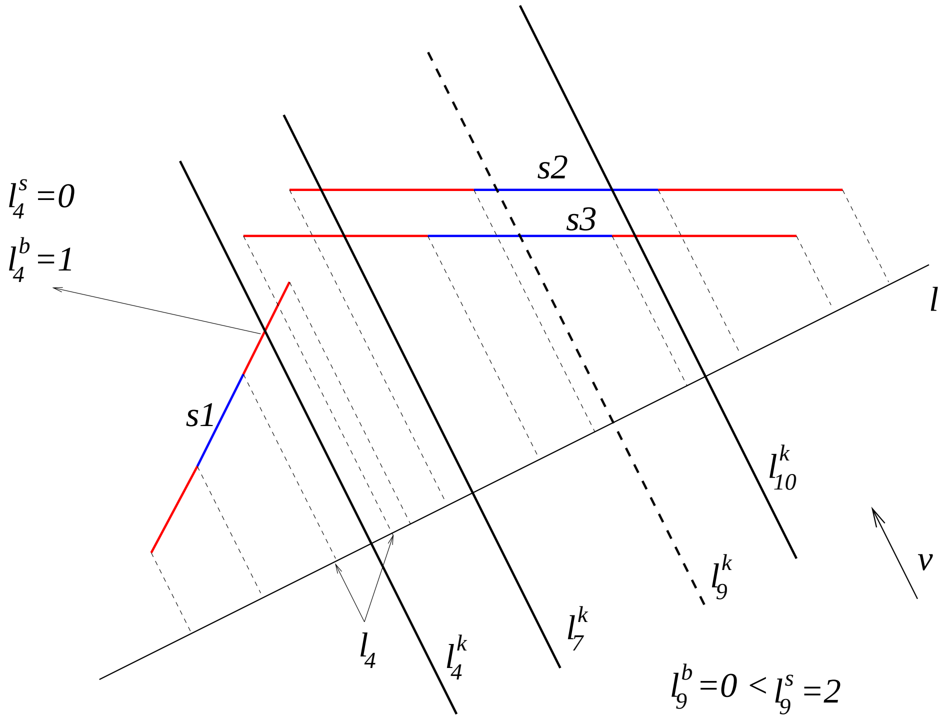

Figure 4: Projection

S onto line l

$$\sum_{i=1}^{4n+2k-1} l_i^s |l_i| = \sum_{i=1}^{n} |Pr(s_i)| \tag{2}$$

and the sum $\sum_{i=1}^{4n+2k-1} l_i^b |l_i|$ is sum of lengths of all projected segments from set B onto line l.

$$\sum_{i=1}^{4n+2k-1} l_i^b |l_i| = \sum_{i=1}^{n+k} |Pr(b)| \geq \sum_{i=1}^{n} |Pr(b_{\sigma(i)})| \tag{3}$$

Hereafter, assumption of parallelism suggests, that the ratio between the length of any pair of segments s_i and $b_{\sigma(i)}$ remains constant after projection. Thus $\beta|s_i| = |Pr(s_i)|$ and $\beta|b_{\sigma(i)}| = |Pr(b_{\sigma(i)})|$, $\beta \in \langle 0, 1\rangle$. Because of $|s_i| \leq \alpha|b_{\sigma(i)}|$, it holds that

$$|Pr(s_i)| \leq \alpha |Pr(b_{\sigma(i)})|$$

$$\sum_{i=1}^{n} |Pr(s_i)| \leq \alpha \sum_{i=1}^{n} |Pr(b_{\sigma(i)})|$$

and in fine from 2 a 3

$$\sum_{i=1}^{4n+2k-1} l_i^s|l_i| \leq \alpha \sum_{i=1}^{4n+2k-1} l_i^b|l_i|$$

But this is a contradiction to 1.

Now we return to the case of several distinguish points projected into the same place on l. The Lemma assumption $\exists(s_i)|s_i \nparallel u$ ensures, that there is at least one segment $s_i \in S$ with the property $|Pr(s_i)| \neq 0$. The rest can be solved simply by lexicographic ordering. Because of zero distance between such points, no change occurs. □

Note: In the whole following text, we will consider the condition of Lemma 1 $\exists(s_i)|s_i \nparallel u$ to be automatically true. In the opposite case this is the question of a set of parallel segments and we can create the BSP tree (subtree) by an arbitrary auto-partition[3] in linear time.

In the next Lemma we will show the existence of linear BSP trees for sets of segments with (ε, δ)-low directional density. We will split the plane by appropriate lines, which will go through δ-directional vicinity of segments $s_i \in S$. Thus we will keep under control the number of cuts of segments from the set S. In this way, we will convert the segments $s_i \in S$ into ε-free segments and these will be eliminated by constant number of cuts.

Lemma 2: *Let S be a set of segments with (ε, δ)-low directional density. Then there is a BSP tree of linear size for this set.*

Proof: At first, we build up an auxiliary set of segments B in the following way: we create two segments $b_{i_1} \in B$; $b_{i_1} = \Omega(\delta, s_i[X])$ and $b_{i_2} \in B$. $b_{i_2} = \Omega(\delta, s_i[Y])$ to each segment $s_i \in S$. By the pre-condition of Lemma2, the set S has (ε, δ)-low directional density and since it holds, that $\forall b_{i_j} \in B$: $|s_i| \leq 1/\delta|b_{i_j}|$ where $1/\delta$ is a constant. It follows from the definition, that $n = |S| \leq |B| = 2n$ and $s_i \parallel b_{i_j}$. The assumptions of Lemma 1 are satisfied for the sets S and B and thus we can find such line p, that $|p \cap S| \leq 1/\delta|p \cap B|$.

The BSP construction algorithm proceeds with loops consisting of two sections. In the first section we process all ε-free segments and dispose the

[3] i.e. any partition line contains a segment (fragment of segment) from S

one from S. In the second section we split the set S into two portions by a line p according to Lemma 1, provided that S is not empty.

Section 1:

while There are any ε-free segment in S **do**
  begin
    Pick an arbitrary $s \in S$ ε-free segment up;
    Destinate the line p containing s;
    Eliminate all segments $b \in B | b \cap p \neq \emptyset$ from the set B;
    Use p as the splitting line for the set $S \cup B$;
  end;

Section 2:

if The set S is not empty **then**
  begin
    Select a line p according Lemma 1;
    Eliminate all segments $b \in B | b \cap p \neq \emptyset$ from the set B;
    Use p as the splitting line for the set $S \cup B$;
  end;

Now we must certify, that the assumptions of Lemma 1 are satisfied when entering the section 2 of the algorithm.

All ε-free segments are discarded in the section 1. We can observe, that there are exactly two segments $b_{i_j} \in B$ belonging to segment $s_i \in S$ at the start of this algorithm. Any segment $b \in B$ is discarded in both sections of the algorithm only in the case, when the one is crossed by (or contained in) a BSP splitting line. The segment s_i becomes clearly ε-free if both segments b_{i_j} associated with s_i have been discarded. Nevertheless, before entering the section 2 all ε-free segments have been discarded. Since we can associate any segment s_i in the second section of the algorithm with at least one segment $b_{i_j} \in B$ and thus $|S| \leq |B|$. In the whole algorithm no changes of length of any segment b_{i_j} occur. Only a segment 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 1 are apparently true.

It remains to prove, that given algorithm really creates a linear size BSP tree for a set of segments S. It is clear, that the nodes of resultant BSP tree contain only segments (or fragments of segments) from set S, because any segment b_{i_j} is discarded whenever a common point with the splitting line p occurs. Hereafter, there are two kinds of partition used in the algorithm:

- The partitioning in the first section of the algorithm uses the ε-free cuts (i.e. the splitting line contains an ε-free segment $s \in S$). That cut can cross at most 2ε other segments and can be applied only n times. (If we split a segment s into two parts, we have to apply this cut twice. But in this case, the cut can cross at most ε another lines).

- The partitioning in the second section of the algorithm uses the splitting line from Lemma 1. Since, 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. From here, the statement holds that the number of crossed segments of the set $S \leq 1/\delta(2n)$, because the number of segments in the set B is at most $2n$.

Let us denote the total number of splitting of all segments Γ. From the previous text results:

$$\Gamma \leq 2n\varepsilon + 2n/\delta = 2n(\varepsilon + 1/\delta). \tag{4}$$

Now we use the fact, that one splitting of any segment adds only one new segment and ε and δ are constants. Hence, the number of segments in the BSP tree is at most $O(n)$ and the resultant BSP tree has linear size. □

It is not quite clear now how to construct the linear BSP tree for a set of segments with low directional density, even though the Lemma 1 provides a proof of existence of such trees. The problem is that we don't know the constants ε and δ which are needed to construct the BSP tree.

The first opportunity is a direct computation of constants δ and ε. Here we draft the computing process.

Let us denote $\Theta = \{\varepsilon_1 = 1, ..., \varepsilon_n = n\}$ the set of all possible values of ε for the set of segments S. We will compute responding $\delta_i \in \Delta$ for any $\varepsilon_i \in \Theta$. If we will use *an arrangement* (for example in [O'Rou95]), the computation is not very complicated.

We extend any segment $s_i \in S$ to infinite line $l_i \in L$. The arrangement of L can be constructed in $O(n^2)$ time. Let $\delta_{i,j}$ denote the length of max. δ-directional vicinity for the segment s_i crossed by at most j segments from the set S and $\delta_j = min\{\delta i, j | i \in \{1, ..., n\}\}$. The set of $\delta_{i,j} | i, j \in \{1, ..., n\}$ can be determined by going through the arrangement in quadratic time and any δ_j can be computed by simple comparing the $\delta_{i,j} | i \in \{1, ..., n\}$ in linear time. It follows from the denotation, that the number of segments crossing the δ_j-directional vicinity is at most j (i.e. $\varepsilon_j = j$). Finally, we compute the most appropriate values of ε and δ from the last equation of Lemma 2. But the overall time complexity of this computation is unfortunately $\Theta(n^2)$.

Lemma 3: *Let the condition of (ε, δ)-low directional density holds for almost all segments from any set S with the exception of constant number of segments. Then there is a linear BSP tree for the set S.*

Proof: The proof of this Lemma is similar to the proof of Lemma 2. □

The second opportunity of computing the BSP tree for any set of segments is to suppose that the property of low directional density holds for the most of practical scenes of segments. In this case, we choose a directional constant δ and suppose, that the δ-directional vicinity of almost all segments is crossed by at most constant number of segments.

Once we have got the constant δ (by estimation or by computation), we can start the BSP tree construction algorithm.

3.3 The BSP tree construction algorithm

Let us suppose we have an appropriate directional constant δ for any scene. Next we describe, how to implement the algorithm effectively. The principle of the algorithm results from the Lemma 2. It uses the method of *tandem search* as it is described in [Berg93]. We start with rough sketch of the algorithm and then we describe some difficult steps in detail. Finally we determine the time complexity of the algorithm.

{Initialisation}
function CreateTree(*SegmentList*, δ) of Tree;
    begin
(1) Create two auxiliary segments b_{i_1} a b_{i_2} for any segment $s_i \in SegmentList$ as its δ-vicinity;
    Insert all segments b_{i_j} into the set *SegmentList*;
    {The segments s_i will be marked as S-segments

and segments b_{i_j} will be marked as B-segments}

(2) Let P be the set of all endpoints of segments $s \in SegmentList$.
Create a dynamic balanced binary tree $\mathcal{T}(P)$ lexicographically ordered by x-coordinates of points $p \in P$;

(3) Create dynamic convex hull $\mathcal{CH}(P)$;

(4) *Tree*:=BSP(*SegmentList*,$\mathcal{T}$,$\mathcal{CH}$);

end

{The function creating BSP}

function BSP(*SegmentList*,$\mathcal{T}$,$\mathcal{CH}$) of Tree;

begin

(5) **if** Card(*SegmentList*) > 0 **then**

begin

(6) **if** There exists any ε-free segments **then**

begin

(6.a) Select any ε-free segment s;

l:= line containing s;

end;

(6.b) **else** l:= FindPartitionLine(*SegmentList*, $\mathcal{T}$);

(7) Split the scene into the next sets:

SegmentList_r:= $(SegmentList - b|b \in B, b \cap l \neq \emptyset) \cap l^+$

SegmentList_l:= $(SegmentList - b|b \in B, b \cap l \neq \emptyset) \cap l^-$

SegmentList_c:= $(SegmentList - b|b \in B, b \cap l \neq \emptyset) \cap l$;

Create the structures $\mathcal{T}(P_r)$, $\mathcal{T}(P_l)$,

$\mathcal{CH}(P_r)$ and $\mathcal{CH}(P_l)$ for the sets P_r a P_l;

(8) Root.right:= BSP(*SegmentList_r*, $\mathcal{T}(P_r)$, $\mathcal{CH}(P_r)$);

Root.left:= BSP(*SegmentList_l*, $\mathcal{T}(P_l)$, $\mathcal{CH}(P_l)$);

Root.list:= *SegmentList_c*;

return Root;

end;

(9) **else return** nil;

end;

Now we explain the individual steps of the algorithm.

The steps (1) – (4) represent the initial part of the algorithm. They are essentially simple. In step (1), we create the set of auxiliary segments B

as in the Lemma 2[4] and in steps (2) and (3) we create the data structures necessary for implementation of the tandem search. In the step (4) the function creating a BSP tree is called.

The recursive function generating the BSP tree through steps (5) – (9) follows.

In step (5), we check, whether continuation of recursive partitioning makes a sense. In case it does not, we will return to the step (9).

In step (6), we must choose one of two options on splitting line l. We will check whether the list U_ε of ε-free segments is empty.

(6.a) If the U_ε is not empty, we choose an arbitrary U_ε-free segment s and discard the one from U_ε. We choose a line containing the segment s as the splitting line l.

(6.b) In the case the list U_ε is empty, we call the function FindPartition-Line. This function selects a splitting line according Lemma1. The function is based on the following idea:

We use a horizontal sweep line l' and sweep the scene from left to right. An elementary change of number of segments crossed by l' occurs only in case, the line passes through an endpoint of a segment $u \in SegmentList$. As the x-coordinates are stored in the tree $\mathcal{T}$, we can do the sweep effectively. The number of S-segments and B-segments crossed by the sweep line l' during the sweep will be maintained with help of two variables $S.number$ and $B.number$. According to the Lemma 1, the case of $S.number/B.number \geq \delta$ must arise at least once.

It looks like we could simply select a first line l which fulfils the condition to be the splitting line. But if we proceeded with the proposed algorithm, we could go through all endpoints of the majority of both sets arisen by the recursive splitting of the set P. Such proceeding causes a bad time complexity of the algorithm in the worst case. In practice it would be enough to go through the smaller of both sets. Hence we use the tandem search technique now. We use two sweep lines (l'_1 a l'_2) instead of one sweep line. Both lines are vertical and they proceed alternately (over one endpoint). The l'_1 proceeds from the left border of the scene to the right and the l'_1 proceeds from the right border of the scene to the left. In such way, we make only the number of steps linear in relation to the size of the smaller of the two subsets (from

[4]we create two segments $b_{i_1} \in B$; $b_{i_1} = \Omega(\delta, s_i[X])$ and $b_{i_2} \in B$. $b_{i_2} = \Omega(\delta, s_i[Y])$ to each segment $s_i \in S$.

beginning the function to locating the first line l satisfying the condition of Lemma 1).

In step (7), we repeatedly use the tandem search technique, because it is enough to find the smaller of the two parts, which the set P will be divided into. The larger set contains remaining elements of P.

Thus, we need to find effectively two points for tandem search. The first point lies on the right of the splitting line l and the second point lies on the left of the splitting line l. Because the line l can occur in general position (we use the ε-free cuts), we must choose a more sophisticated solution than in (6.b). For that purpose we use the dynamic convex hull $\mathcal{CH}$. We will find two points lying in the farthest distance from l (p_r to the right of l and p_l to the left of l) by means of the $\mathcal{CH}$. (The fact that the points p_r and p_l are farthest from l is not significant. It is only important that the points lie on the opposite sides of the splitting line l and that we can find them effectively.)

Now we show how to process the point p_r. (The point p_l can be processed similarly. Only in the case of event 3 we will not process this point.) Let u be a segment containing the point p_r. We eliminate the points p_1 a p_2 of the segment u from $\mathcal{CH}$ at first. Hereafter, the one of the next four cases may occur:

1. The line l contains the segment u.

   If the segment u is a S-segment, we add it to the *SegmentList_c*.

2. Both endpoints of u lie in the same halfplane designated by the line l.

   In this case we add the points p_1 and p_2 to the list $aux.P_r$ of points to the right of l. (A similar list $aux.P_l$ is created for the points to the left of l.) The segment u is added to the *SegmentList_r*.

3. The segment u is a S-segment divided by the line l into two parts.

   In this case we split the segment u into two parts in the point of intersection with l: the $u_r = u \cap l^+$ and $u_l = u \cap l^-$. The endpoints of u_r are added to $aux.P_r$ and the endpoints of u_l are added to $aux.P_l$. The segment u_r is added to the list *SegmentList_r* and the segment u_l is added to the *SegmentList_l*.

4. The segment u is a B-segment (for example b_{i_1}) divided by the line l onto two parts.

In this case a test is performed, whether there is a segment u' (contained in $SegmentList$) dual to the segment u. (We test, whether there is a segment $b_{i_2} \in SegmentList$ to the segment b_{i_1} in our example.) If the dual segment does not exist, we add the S-segment associated with u to the U_ε list. (Both segments associated with the S-segment have been eliminated (and thus crossed by a splitting line) from the $SegmentList$.)

We will continue this way, until the points on the right of l and on the left of l exists at the same time.

Let us suppose, there is not any point on the right of l yet. (The case of non-existence of points on the left of l is symmetrical again.) Since the set $aux.P_r$ contains endpoints of the smaller set into which P is split by l. The set $aux.P_l$ contains as many endpoints as $aux.P_r$ but they lie at the opposite halfspace designated by l. It remains to create the convex hulls $\mathcal{CH}(P_l)$ and $\mathcal{CH}(P_r)$ and the trees $\mathcal{T}(P_r)$ and $\mathcal{T}(P_l)$ yet. The convex hull $\mathcal{CH}(P_r)$ and the tree $\mathcal{T}(P_r)$ can be created from scratch, because they belong to the smaller of both set. The convex hull $\mathcal{CH}(P_l)$ is created by adding the points of the set $aux.P_l$ to the remainder of original convex hull $\mathcal{CH}$. The tree $\mathcal{T}(P_l)$ is created by recovering the points of the set $aux.P_l$ from the original tree $\mathcal{T}$.

In step (8) we compile the resultant tree by recursive calling of the procedure BSP.

Lemma 4: *The algorithm described above creates a linear BSP tree for a set of segments with low directional density and an appropriate directional constant δ.*

Proof: This statement results from the proof of Lemma 2, since the algorithm above works according to the principle of Lemma 2. □

Lemma 5: *The algorithm described above can be implemented for a set of segments with low directional density in the time $O(n \log^3 n)$.*

Proof: Let us have a look at the initialisation part of the algorithm at first. The step (1) can be processed simply in the linear time. The construction of the balanced binary tree $\mathcal{T}$ (the one allows update in $O(\log n)$ time and operations $succ(p)$ and $pred(p)$ of finding successor and predecessor of a leaf p in linear time) takes $O(n \log n)$. Furthermore, the dynamic data structure of convex hull $\mathcal{CH}$ allowing deletions and insertions have to be created. For this purpose we use the data structure of Overmars and Van Leeuwen [Overm81]. Updates in this structure take $O(\log^2 n)$ time and queries take $O(\log n)$ time. The preprocessing time is $O(n \log n)$.

The analysis of one recursion pass will be described in the following text. From aspect of time complexity, the steps (6.a) and (7) are essential. The rest can be processed in constant time.

In step (6.a) we make vertical sweep with help of two lines. We achieve at most $O(k)$ by the means of tandem search. The number k is the size of the smaller set, which the set $SegmentsSet$ will be split into. Furthermore, we spend only constant time in any step of the sweep. This holds by reason of constant time operations $succ(p)$ and $pred(p)$ on the tree $\mathcal{T}$ lists.

In step (7) we create the sets $aux.P_r$ and $aux.P_l$ at first. It is done by tandem search technique. The both sets have at most k elements, where k is the number of endpoints of the smaller set which is the set P split into. When any endpoint is treated the operations of query $(O(\log n))$ and update $(O(\log^2 n)$ have to be done. Hence, this takes $O(k \log^2 n)$ time in total.

Furthermore, the update of data structures of the convex hull $\mathcal{CH}(P_r)$, $\mathcal{CH}(P_l)$ and the balanced binary tree $\mathcal{T}(P_r)$, $\mathcal{T}(P_l)$ is needed. The smaller member in any pair can be created from scratch from an appropriate aux-set which takes $O(k \log n)$ time. The higher member in any pair is created by adding (recovering) of k endpoints of an appropriate aux-set to (from) a remainder of the original data structure $\mathcal{CH}$ ($\mathcal{T}$). It can be done in $O(k \log^2 n)$ $(O(k \log n))$ time in total.

In conclusion, we have to determine the complexity of processing of endpoints belonging to splitted B-segments and endpoints added to the set $SegmentList\_c$. The processing of any such endpoint takes $O(\log^2 n)$ time (the most expensive operation, again, is the update of $\mathcal{CH}$). We deal with each such endpoint only once and the total number of the endpoints is linear. Thus the total time cost is $O(n \log^2 n)$.

The consequential time we spend in step (7) (without the time of operations mentioned in the paragraph above) is $O(k \log^2 n)$ altogether.

Now we use the issue of Lemma 3 that the consequential BSP tree has linear size. Hence, 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

$$T(n) \leq \max_{0 \leq k \leq cn/2} O(k \log^2 n) + T(r) + T(cn - r) | k \leq r$$

which implies that $T = O(n \log^3 n)$. So the running time of the algorithm is bounded by $T = O(n \log^3 n)$. $\square$

Let us summarise our results in the following theorem:

Theorem 6: *Given a set of n disjoint segments in the plane which satisfy the condition of low directional density and an appropriate directional constant δ, it is possible to construct a BSP tree of linear size in $O(n \log^3 n)$ time.*

4 Conclusion

We have proved the existence of linear BSP tree for sets of segments with low directional density in the plane. We have also presented an efficient algorithm for computing linear size binary space partitions for such sets of segments.

This work extends previous results achieved by Paterson and Yao [Pater90b], who proved existence of linear size BSPs for sets of orthogonal segments in the plane and [Berg94], who proved existence of the one for sets of segments with constant ratio between the lengths of the longest and shortest segment. However, the problem of existence of linear size BSP for any set of line segments still remains open. We hope that our technique can help to solve this general problem.

References

[Agarw96] 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.

[Airey90] John Milligan Airey. *Increasing Update Rates in the Building Walkthrough System with Automatic Model-space Subdivision and Potentially Visible Set Calculations.* PhD thesis, Dept. of Computer Science, University of North Carolina, Chapel Hill, 1990.

[Agarw97] 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.

[Agarw94] Pankaj K. Agarwal and Subhash Suri. Surface approximation and geometric partitions. In *Proc. 5th ACM-SIAM Sympos. Discrete Algorithms*, pages 24–33, 1994.

[Balli93] C. Ballieux. Motion planning using binary space partitions. Technical Report Inf/src/93-25, Utrecht University, 1993.

[Chin89] 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.

[Berg95] 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.

[Berg93] M. de Berg, M. de Groot, and M. Overmars. Perfect binary space partitions. In *Proc. 5th Canad. Conf. Comput. Geom.*, pages 109–114, 1993.

[Berg94] 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.

[Fuchs83] 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.

[Fuchs80] 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.

[Naylo90] 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.

[O'Rou95] Joseph O'Rourke. *Computational Geometry in C.* Cambridge University Press, Cambredge CB2 1RP, 1995.

[Overm81] M. H. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Syst. Sci.*, 23:166–204, 1981.

[Pater90a] 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.

[Pater90b] M. S. Paterson and F. F. Yao. Optimal binary space partitions for orthogonal objects. Research Report 158, Univ. Warwick, 1990.

[Schum69] 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.

[Telle92] S. J. Teller. *Visibility Computations in Densely Occluded Polyhedral Environments.* PhD thesis, Dept. of Computer Science, University of California, Berkeley, 1992.

[Thiba87] 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.

Publications in the FI MU Report Series are in general accessible
via WWW and anonymous FTP:

`http://www.fi.muni.cz/informatics/reports/`
`ftp ftp.fi.muni.cz (cd pub/reports)`

Copies may be also obtained by contacting:

Faculty of Informatics
Masaryk University
Botanická 68a
602 00 Brno
Czech Republic