Linear Binary Space Partitions and the Hierarchy of Object Classes

Petr Tobola* Karel Nechvíle*

Abstract

We consider the problem of constructing binary space partitions for the set $\mathcal{P}$ of d-dimensional objects in d-dimensional space. There are several classes of objects defined for such settings that support the design of effective algorithms. We extend the existing de Berg hierarchy of classes [4] by defining new classes based on old ones and we show the desirability of such an extension. Moreover we propose a new algorithm that works on generalized Λ-low-density scenes [11] (defined in this paper) and provides BSP trees of linear size. The tree can be constructed in $O(n \log^2 n)$ time and space, where n is the number of objects. Moreover, we can trade-off between size and balance of the BSP tree fairly simply.

1 Introduction

In the past, much attention has been paid to the development of algorithms whose goal is to construct the smallest possible BSP trees. Initially, several heuristics were developed (for example [5, 8]) but these can create a tree of excessive size under unfavourable circumstances ($\Omega(n^2)$ in the plane and $\Omega(n^3)$ in $\mathbb{R}^3$ space). The first provable bounds were obtained by Paterson and Yao [7, 6]. They showed [7] that the optimal size of BSP is $\Theta(n^2)$ in $\mathbb{R}^3$ space and $O(n \log n)$ in a plane in the worst case. The next result by these authors [6] was an optimal sized BSP algorithm for the set of orthogonal objects with $\Theta(n^{3/2})$ in $\mathbb{R}^3$ space in the worst case and $\Theta(n)$ in a plane in the worst case.

It was observed that, in practice, many scenes behave reasonably and enable efficient processing. Several attempts to specify object sets of such scenes have been made. Recently, de Berg et al. [3] have investigated the common properties of the scenes where effective algorithms can be used. Most real-world scenes fall into this category. A hierarchy of the known object classes has been composed and matched with realistic input models. This can simplify the design of algorithms that are provably efficient.

Although the de Berg hierarchy includes a large number of scenes, there is still a wide spectrum of simple and potentially practical scenes remaining that do not match it. We will outline a few particular examples of such scenes and then we will try to highlight their significant common features.

A wooden fence is a line of many long and thin planks. This scene does not fit within the de Berg hierarchy because the planks are located tightly one after another. Similar results can be obtained if we consider a school of long and thin fish, radiator fins, suits in a wardrobe, books in a library, a cluster of fluorescent lamps, or a thick forest.

*Support was provided by the grants GACR 201/98/K041, Grant Agency of Czech Republic and MSM 143300004, Ministry of Education, Czech Republic.

It seems that the real world scenes frequently contain groups of similar objects that usually appear in (isolated) clusters and these objects usually differ only in their position (and possibly in scale ratio). Moreover, it is very probable that the described similarity holds for many objects that lie near to each other. The correctness of this presupposition has been shown by results of practical implementation. We will be describing the practical results precisely in a forthcoming paper.

Although the picket fence does not form a set of fat objects, we can transform it to the set of fat objects by simple linear scale transformation. In the following, we extend some object classes of the existing de Berg hierarchy by including linear transformation.

This paper is organized as follows. In Section 2, we give some essential definitions needed in the rest of the paper. Section 3 presents the BSP construction method and description of the algorithm. Section 4 includes concluding remarks. Due to space limitations, all proofs are deferred to the full paper.

2 Preliminary

We introduce some ideas which will be used in the following sections. In particular we need the definitions of *simple* and *extended* rectangular neighborhood and a useful tool – the Cutting Lemma.

Definition 2.1: Let p be an object in $\mathbb{R}^d$. The *bounding box* of p (further bb(p)) is the smallest axis-aligned box that completely contains p.

We now define some special neighborhoods of axis-aligned hyperrectangles in d-dimensional space.

Definition (Simple/Extended neighborhood) 2.2: Let $r, r_n \subseteq \mathbb{R}^d$ be d-rectangles, $r \subset r_n$. Assume that r and r_n are axis-aligned. We call the difference $\Omega(r, r_n) = (r_n - r)$ the *rectangular neighborhood* of r.

Let $\Omega(r, r_n)$ be a rectangular neighborhood of r, $i \in \{1, ..., d\}$, $x_i^-(r)$ and $x_i^-(r_n)$ be the minimal coordinates of r and r_n with respect to the i-th coordinate axis, $w_i^- = |x_i^-(r_n) - x_i^-(r)|$. Analogously, we can define $x_i^+(r)$, $x_i^+(r_n)$ and w_i^+. We define the *width* of the neighborhood to be the set $\{w_i^k | i \in \{1, ..., d\}, k \in \{+, -\}\}$.

The rectangle r is determined by its edge vectors $\vec{e}_1, \ldots, \vec{e}_d$ and a reference point. The neighborhood is determined by its width. Let f be an array of d functions $f = \langle f_1, ..., f_d \rangle$, $f_i = f_i(\delta, r)$ and δ be a parameter. We will take into account only the special case when $w_i^- = w_i^+$ (i.e. the rectangle r is centered in r_n). Then we define the $\Omega(\delta, f, r)$-neighborhood to be the rectangular neighborhood of r with $w_i^- = w_i^+ = |f_i(\delta, r)\vec{e}_i|$.

1. If $f_i = \delta$; $i \in \{1, ..., d\}$, then we call the neighborhood $\Omega(\delta, f, r)$ a *simple neighborhood* and denote it $\Omega_s(\delta, r)$.
2. Let j be the coordinate with maximal edge length $|\vec{e_j}|$. If $f_i = \frac{|\vec{e_j}|}{|\vec{e_i}|}(1 + 2\delta)$, $i \in \{1, ..., d\}$, then we call the neighborhood $\Omega(\delta, f, r)$ an *extended neighborhood* and denote it $\Omega_e(\delta, r)$.

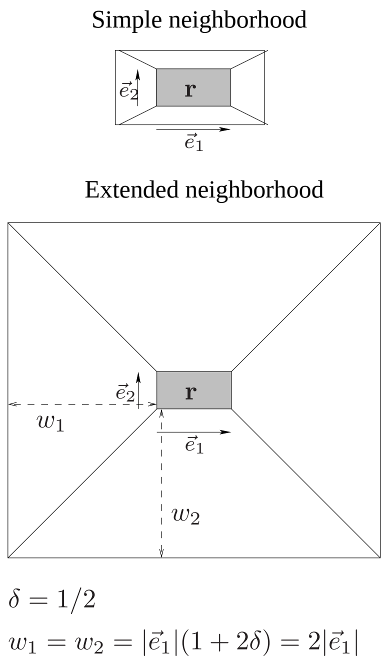

Figure 1: Simple and Extended hyperrectangle neighborhood.

We have already mentioned the de Berg hierarchy of object classes. The basic classes of that hierarchy are *Fatness*, *Low-density*, *Clutteredness* and *Simple-cover complexity*. The exact definition can be found in [4]. We now define the extension of that hierarchy which is based on the scaling idea mentioned above.

In the following text, we are using slightly incorrect notation for the scene-region intersection. Let $\mathcal{P}$ be a scene (i.e. set of objects) and R be a region. The intersection $\mathcal{P} \cap R$ means the set of objects $\{q_i | q_i = p_i \cap R, p_i \in \mathcal{P}\}$.

Definition 2.3: Let $\mathcal{C}$ be an object class and $\mathcal{P} \in \mathcal{C}$ be a set of objects (a scene). We call the class $\mathcal{C}$ *local* iff for any set of objects $\mathcal{P} \in \mathcal{C}$ and any convex region R holds that $(\mathcal{P} \cap R) \in \mathcal{C}$.

Locality markedly simplifies work with particular scenes. For example, the classes *Low density* and *Uncluttered scenes* are local. The classes *SCC* and *Guarding scenes* are not, which makes them more difficult to analyze.

Definition 2.4: Let $\mathcal{C}$ be a local class of objects. We define the extension of $\mathcal{C}$ called *locally-balanceable-*$\mathcal{C}$ (LB-$\mathcal{C}$) as follows:

Let $\mathcal{P} = \{p_1, ..., p_n\}$ be a scene (i.e. set of objects), δ be a constant, R_i be the minimal convex region containing $\Omega_e(\delta, bb(p_i))$. We say that $\mathcal{P} \in$ LB-$\mathcal{C}$ iff $\forall (p_i \in \mathcal{P})$: the intersection $R_i \cap \mathcal{P}$ can be reduced by linear scale transformations along some coordinate axes to a scene in $\mathcal{C}$.

The proposed BSP algorithm exploits properties of the *λ-low-density* scenes which extend to the *LB-λ-low-density* scenes.

Finally, we give the cornerstone of the algorithm - the Cutting Lemma. It was introduced by Tobola and Nechvíle in [10] and it is essential for designing the algorithm and for proving the bounds on the size of resulting BSP trees. The Cutting lemma concerns two sets of segments in a plane. The simplified idea of the Cutting lemma is as follows.

Let us have two identical sets of segments S and B in the plane. Then we shift the whole set B in an arbitrary direction. It seems probable that for any direction we can select a line l cutting at least as many segments from the set B as segments from the set S. The Cutting lemma generalizes and proves this idea.

Proposition (Cutting lemma) 2.5: *Let S, B be non-empty sets of segments in the plane which fulfil the following conditions:*

1. $n = |S| \leq |B| = n + k; k \geq 0$ *is an integer.*
2. *There is an injective mapping $\sigma : I \to J$, where $I = \{1, ..., n\}, J = \{1, ..., n+k\}$, and a real constant α such, that the following is true 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 $b_{\sigma(i)} \in B$.*

Then for any non-zero vector v such that $\exists (s_i) : s_i \nparallel v$ there is a line p parallel to v, such that $|p \cap S| \leq \alpha |p \cap B| \geq 1$, where $|p \cap S|$ is the number of intersections between p and segments from S, $|p \cap B|$ is the number of intersections between p and segments from B.

3 The BSP Construction

We now are going to describe the algorithm for creating a linear BSP tree. The algorithm exploits a set of hyperrectangles to form a BSP tree. The set of the hyperrectangles $R = \{r_1, ..., r_n\}$ is a set of bounding boxes of the original objects $\mathcal{P} = \{p_1, ..., p_n\}$ with constant complexity in $\mathcal{R}^d$ space. Initially, we build up d pairs of auxiliary sets of segments B_i, $S_i | i \in \{1, ..., d\}$ as follows:

Let us project the set $\{r | r \in R\}$ of original hyperrectangles onto the i-th coordinate axis (further i-axis). The set of segments S_i contains the projected rectangles: $S_i = \{s | s = Proj_i(r), r \in R\}$. 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 can be simply solved by lexicographical ordering of the points of the original hyperrectangles. Each endpoint of $s \in S_i$ can be considered as a projection of the unique point p_{max} (p_{min}) $\in r$ maximal (minimal) in the standard lexicographical ordering.

The $\Omega_s(\delta, r)$ neighborhood belonging to r is a part of a hyperrectangle enclosing r. Let us project the set $\{\Omega_s(\delta, r) | r \in R\}$ onto the i-axis. We get the set of segments. Let us split each segment $Proj_i(\Omega_s(\delta, r))$ into two parts by subtraction of the $Proj_i(r)$ from it. We obtain two segments: b_1 with lower coordinates and b_2 with higher coordinates associated with the segment s, as you can see in Figure 2. It follows from the definition of $\Omega_s(\delta, r)$ that $|b_1| = |b_2| = \delta|s|$. Moreover, the segments b_1, b_2 form $\Omega_s(\delta, s)$ neighborhood of the segment s. The set B_i is a unification of all segments b_1 and b_2 generated by the set $\{\Omega_s(\delta, r)|r \in R\}$. The degenerate cases are treated by lexicographical ordering as well.

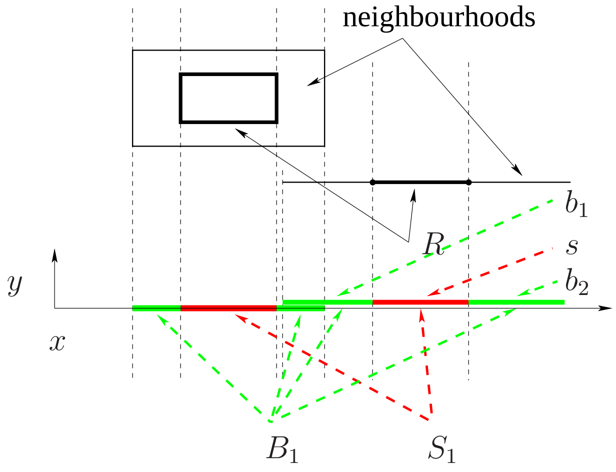

Figure 2: The sets S_1 and B_1.

The total number of $s \in S = \cup_{i\in\{1,...,d\}} S_i$ segments is dn and the total number of $b \in B = B \cup_{\{1,...,d\}} B_i$ segments is $2dn$, where d is the dimension of space.

Definition 3.1: We call the segment $s \in S_i^{'}$ *bounded* iff both segments b_1, b_2 associated with s have been eliminated from the set $B_i^{'}$ (see the algorithm below). We call the hyperrectangle r *bounded* iff all segments s associated with r are *bounded*.

Algorithm

```
if The set $\bigcup_{j=1}^{d} S_j$ is not empty then
  begin
(1) Eliminate all segments $\{s \in S_j|\ j \in \{1, ..., d\}$
      $\wedge$ s belongs to a bounded hyperrectangle$\}$
      from the set $S_j$;
(2) if We cannot select an $i$ such that a line $l$
      fulfills the Cutting lemma conditions
      for the sets $S_i$, $B_i$ then
        // $\forall(l) : \frac{|l\cap B|}{|l\cap S|} \leq \delta$
        Select a line $l$ with maximal value of $\frac{|l\cap B|}{|l\cap S|}$;
(3) else Select an appropriate line $l$
      according to the Cutting lemma;
(4) Select a hyperplane $p$ that contains the line $l$.
      The hyperplane $p$ should be perpendicular
      to the $i$ axis;
(5) Eliminate all segments $b \in B_j|b \cap p \neq \emptyset$
      from the sets $B_j|j \in \{1, ..., d\}$;
(6) Use $p$ as the splitting hyperplane for the set $R$
      onto subregion $R^{p^-}$ and $R^{p^+}$;
(7) Split the sets $S_j \cup B_j|j \in \{1, ..., d\}$ using $p$
      so that the resulting sets $S_j^{p^-}$ and $S_j^{p^+}$
      correspond to $R^{p^-}$ and $R^{p^+}$ (see fig. 3);
(8) recurse on the sets $S_j^{p^-}, B_j^{p^-}|j \in \{1, ..., d\}$ and $R^{p^-}$;
(9) recurse on the sets $S_j^{p^+}, B_j^{p^+}|j \in \{1, ..., d\}$ and $R^{p^+}$;
  end
else
(10) Apply autopartition on the resulting set;
```

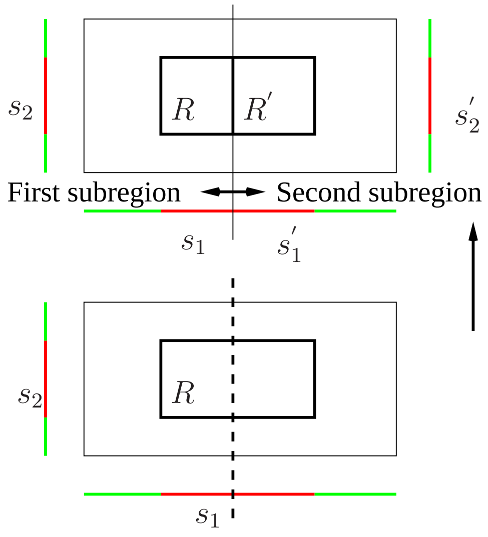

Figure 3: The rectangle R and sets S and B are split into two parts.

Lemma 3.2: *Let $\mathcal{P} \subset \mathbb{R}^d$ be the λ-low density scene, d be the constant. The proposed algorithm provides a linear BSP tree for the scene $\mathcal{P}$.*

Lemma 3.3: *If the above algorithm works for a local class $\mathcal{C}$, then it also works for the class LB-$\mathcal{C}$.*

Corollary 3.4: *The above algorithm works for the class LB-low-density.*

The method presented above provides us with a pseudo algorithm for creating a linear BSP tree. If we opt for a brute force approach in implementation of the algorithm, it could behave very inefficiently and it could violate the space and time bounds given above.

The main difficulty is to find the splitting line according to the Cutting lemma. We suggest two ways to achieve the $O(n \log^2 n)$ time bound.

The first way is to use the *tandem search* [2] technique. This technique leads to the $O(n \log^2 n)$ time and, furthermore, it works in $O(n)$ space. Unfortunately, this method tends to create maximally unbalanced BSP trees. Because the balance criterion is very important in practice, we suggest another method for solving this problem.

The second way is to use the segment trees discovered by Bentley [1]. We will maintain the set of segments B_i and S_i in a segment tree along with some extra data. Using these trees, we will be able to select the splitting plane effectively according to the Cutting lemma. This technique is described in detail in [9]. The advantage of this technique lies in the possibility of controlling the search of the cutting line l. Hence, we can easily trade-off the balance and size of the resulting BSP tree. The small drawback of this method is the $O(n \log^2 n)$ space requirement.

At the end it should be noted that the algorithm works well even if the non-overlapping condition is not satisfied. The resulting BSP tree is always correct.

Theorem 3.5: *Let $\mathcal{P}$ be a set of objects in $\mathbb{R}^d$, $P \in$ LB-λ-low-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 resulting tree.*

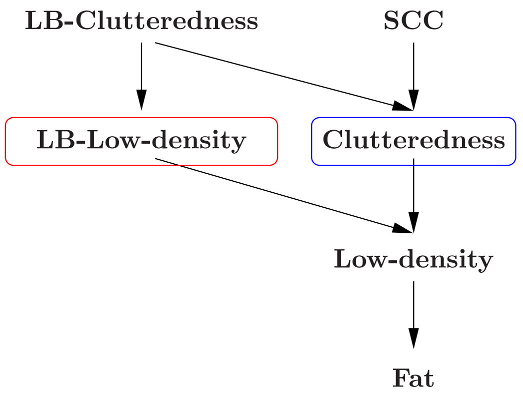

Figure 4: The hierarchy of recent object classes. The left box denotes the domain of our algorithm and the right box denotes domain of de Berg's algorithm.

4 Conclusion

In this paper, we have proposed the extension of the existing de Berg hierarchy of object classes by a set of *locally balanceable* (LB) classes. The full class hierarchy is depicted in Figure 4.

Moreover, we have presented a new BSP construction algorithm that works provably well for sets of objects with LB-λ-low density. In such cases it provides linear BSP trees and runs in $O(n \log^2 n)$ time and space. This method is very simple.

The de Berg's algorithm works for the uncluttered scenes which are more general than λ-low density scenes. However, our algorithm covers some scenes which fall outside the definition of uncluttered scenes. Moreover, the Cutting lemma provides our algorithm with certain input sensitivity.

We have implemented both the discussed algorithms. The practical comparisons show that the BSP trees of the de Berg algorithm have been almost two times greater on average.

The next step is to perform some other practical tests. Moreover, it would be interesting to explore the definition of LB-$\mathcal{C}$ scenes and to try to refine it further.

References

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

[2] M. de Berg, M. Overmars, and O. Schwarzkopf. Computing and verifying depth orders. In *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pages 138–145, 1992.

[3] Mark de Berg. Linear size binary space partitions for uncluttered scenes. *Algorithmica*, 28(3):353–366, 2000.

[4] Mark de Berg, Matthew J. Katz, A. Frank van der Stappen, and Jules Vleugels. Realistic input models for geometric algorithms. In *Symposium on Computational Geometry*, pages 294–303, 1997.

[5] 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.

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

[7] 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.

[8] S. Teller. *Visibility Computations in Densely Occluded Polyhedral Environments*. Ph.D. thesis, Computer Science Div., Univ. of California, Berkeley, 1992.

[9] Petr Tobola and K. Nechvile. Linear BSP trees for sets of hyperrectangles with low directional density. In V. Skala, editor, *WSCG 2001 Conference Proceedings*, pages 237–244, 2001.

[10] Petr Tobola and Karel Nechvile. Linear BSP tree in the plane for set of segments with low directional density. In V. Skala, editor, *WSCG'99 Conference Proceedings*, pages 297–304, 1999.

[11] A. van der Stappen and M. Overmars. Motion planning amidst fat obstacles. In *In Proc. 10th Annu. ACM Sympos. Comput. Geom.*, pages 31–40, 1994.