Practical Techniques for Constructing Binary Space Partitions for Orthogonal Rectangles

Pankaj K. Agarwal* T. M. Murali† Jeffrey Scott Vitter‡
Center for Geometric Computing
Department of Computer Science, Duke University
Box 90129, Durham, NC 27708-0129
Email: {pankaj,tmax,jsv}@cs.duke.edu

Abstract

We present the first systematic comparison of the performance of algorithms that construct Binary Space Partitions for orthogonal rectangles in $\mathbb{R}^3$. We compare known algorithms with our implementation of a variant of a recent algorithm of Agarwal et al. [1]. We show via an empirical study that their algorithm constructs BSPs of near-linear size in practice and performs better than most of the other algorithms in the literature.

1 Introduction

The Binary Space Partition (BSP) [3, 5] is a versatile and popular data structure, with applications in many problems—hidden-surface removal, global illumination, shadow generation, solid geometry, geometric data repair, ray tracing, network design, and surface simplification; see [1] for a detailed list of references.

The efficiency of most BSP-based algorithms depends on the size and/or the depth of the BSP (we formally define the size of a BSP later). Therefore, several algorithms have been developed to construct BSPs of small size and depth; see [1] for a list of references.

Recently, Agarwal et al. [1] developed an algorithm that constructs a BSP for orthogonal rectangles in $\mathbb{R}^3$ when most rectangles have aspect ratio bounded by a constant. We have implemented their algorithm to study its performance on "real" data sets. We have also systematically compared its performance to that of various existing algorithms. We show that the algorithm of Agarwal et al. is indeed practical: it constructs a BSP of near-linear size on real data sets (the size varies between 1.5 and 1.8 times the number of input rectangles). This algorithm performs better than not only Paterson and Yao's algorithm [4] but also most heuristics described in the literature [2, 3, 7].

To compare the different algorithms, we measure the size of the BSP each algorithm constructs and the total number of pieces into which the rectangles are partitioned by the BSP. The size measures the storage needed for the BSP.

*Support was provided by National Science Foundation research grant CCR–93–01259, by Army Research Office MURI grant DAAH04–96–1–0013, by a Sloan fellowship, by a National Science Foundation NYI award and matching funds from Xerox Corp, and by a grant from the U.S.-Israeli Binational Science Foundation.

†This author is affiliated with Brown University. Support was provided in part by National Science Foundation research grant CCR–9522047 and by Army Research Office MURI grant DAAH04–96–1–0013.

‡Support was provided in part by National Science Foundation research grant CCR–9522047, by Army Research Office grant DAAH04–93–G–0076, and by Army Research Office MURI grant DAAH04–96–1–0013.

2 Geometric Preliminaries

A *binary space partition* $\mathcal{B}$ for a set S of pairwise-disjoint rectangles in $\mathbb{R}^3$ is a tree defined as follows: Each node v in $\mathcal{B}$ represents a box (rectangular parallelepiped) $\mathcal{R}_v$ and a set of rectangles $S_v = \{s \cap \mathcal{R}_v \mid s \in S\}$ that intersect $\mathcal{R}_v$. The box associated with the root is $\mathbb{R}^3$ itself. If S_v is empty, then node v is a leaf of $\mathcal{B}$. Otherwise, we partition $\mathcal{R}_v$ into two boxes by an orthogonal *cutting plane* H_v. At v, we store the equation of H_v and $\{s \cap H_v \mid s \in S_v\}$, the subset of rectangles 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 boxes 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 the set of rectangles $S_v^- = \{s \cap H_v^- \mid s \in S_v\}$ and the right subtree of v is a BSP for the set of rectangles $S_v^+ = \{s \cap H_v^+ \mid s \in S_v\}$. The size of $\mathcal{B}$ is the sum of the number of interior nodes in $\mathcal{B}$ and the total number of rectangles stored at all the nodes in $\mathcal{B}$.[1]

Given a set of rectangles R, let $R_B = \{s \cap B \mid s \in R\}$ be the set of rectangles obtained by clipping the rectangles in R within B. We say that a rectangle in S_B is *free* if none of its edges lies in the interior of B; otherwise it is *non-free*. A *free cut* is a cutting plane that does not cross any rectangle in S and that either divides S into two non-empty sets or contains a rectangle in S.

3 The Agarwal et al. Algorithm

In this section, we describe a variant of the algorithm of Agarwal et al. [1] that we have implemented. In our implementation, we have modified their algorithm slightly in order to improve its performance. We call this algorithm Rounds.

A box B in $\mathbb{R}^3$ has six faces—*top, bottom, front, back, right,* and *left*. We assume, without loss of generality, that the back, bottom, left corner of B is the origin (i.e., the back

[1] We do not store the leaves of $\mathcal{B}$ explicitly since all the information about a leaf is captured by its parent and the cutting plane at the parent.

face of B lies on the yz-plane). We say that a rectangle r in S_B is *long* with respect to a box B if none of the vertices of r lie in the interior of B. Otherwise, r is said to be *short*. We can partition long rectangles into three classes: a rectangle s that is long with respect to B belongs to the *top class* if two parallel edges of s are contained in the top and bottom faces of B. We similarly define the *front* and *right* classes. For a set of points P, let P_B be the subset of P lying in the interior of B.

The algorithm proceeds in rounds. At the beginning of the ith round, where $i > 0$, the algorithm has a top subtree $\mathcal{B}_i$ of the BSP for S. Let Q_i be the set of boxes associated with the leaves of $\mathcal{B}_i$ containing at least one rectangle. The initial tree $\mathcal{B}_1$ consists of one node and Q_1 consists of one box that contains all the input rectangles. If Q_i is empty, we are done. Otherwise, in the ith round, for each box $B \in Q_i$, we construct a top subtree $\mathcal{T}_B$ of the BSP for the set S_B and attach it to the corresponding leaf of $\mathcal{B}_i$. This gives us the new top subtree $\mathcal{B}_{i+1}$. Thus, it suffices to describe how to build the tree $\mathcal{T}_B$ on a box B during a round.

Let $F \subseteq S_B$ be the set of rectangles that are long with respect to B. Set $f = |F|$, and let k be the number of vertices of rectangles in S_B that lie in the interior of B (note that each such vertex is a vertex of an original rectangle in S). As in the Agarwal et al. algorithm, we choose a parameter $a = 2^{\sqrt{\log(f+k)}}$, which remains fixed throughout the round. In a round, we partition B using a sequence of cuts in two stages, the separating stage and the dividing stage. The separating stage divides B into a set of boxes $\mathcal{C}$ such that for each box $C \in \mathcal{C}$, F_C contains only two classes of rectangles. The dividing stage further refines each such box C until a new round is to be started in the resulting boxes. We now describe each stage in detail.

Separating Stage: Assume without loss of generality that the longest edge of B is parallel to the x-axis. The rectangles in F that belong to the front class can be partitioned into two subsets: the set R of rectangles that are vertical (and parallel to the right face of B) and the set T of rectangles that are horizontal (and parallel to the top face of B). Let e be the edge of B that lies on the z-axis and e' be the edge of B that lies on the y-axis. The intersection of each rectangle in R with the back face of B is a segment parallel to the z-axis. Let $\bar{r}$ denote the projection of this segment onto the z-axis, and let $\bar{R} = \{\bar{r} \mid r \in R\}$. Let $z_1 < z_2 < \cdots < z_{k-1}$ be the endpoints of intervals in $\bar{R}$ that lie in the interior of e but not in the interior of any interval of $\bar{R}$. Similarly, for each rectangle t in the set T, we define $\bar{t}$ to be the projection of t onto the y-axis, and $\bar{T} = \{\bar{t} \mid t \in T\}$. Let $y_1 < y_2 < \cdots < y_{l-1}$ be the endpoints of intervals in $\bar{T}$ that lie in the interior of e' but not in the interior of any interval of $\bar{T}$.

We divide B into kl boxes by drawing the planes $z = z_i$ for $1 \leq i < k$ and the planes $y = y_j$ for $1 \leq j < l$. We refer to these cuts as α-*cuts*. Let $\mathcal{C}$ be the set of boxes into which B is partitioned in this manner.

Dividing Stage: We refine each box C in $\mathcal{C}$ by applying cuts as described below. Let V_C be the set of vertices of rectangles in S_C that lie in the interior of C. We recursively invoke the following steps until $|F_C| + 2a|V_C| < (f + ak)/a$ and S_C does not contain any free rectangles.

1. If C has any free rectangle, we use the free cut containing that rectangle to split C into two boxes.
2. If the rectangles in F_C belong to two classes, we make
   (i) either one cut that partitions C into two boxes C_1 and C_2 so that for $i = 1, 2$ $|R_{C_i}| + w|V_{C_i}| \leq 2(|R_C| + w|V_C|)/3$, or
   (ii) at most two parallel cuts that divide C into three boxes C_1, C_2, and C_3 such that $|R_{C_2}| + w|V_{C_2}| \geq (|R_C| + w|V_C|)/3$ and such that all rectangles in R_{C_2} belong to the same class (these two cuts are unique).
3. If F_C has only one class of rectangles, let g be the face of C that contains exactly one of the edges of each rectangle in R_C. Let P be the set of those vertices of the rectangles in R_C that lie in the interior of g. We use a plane that is orthogonal to g to partition C into two boxes C_1 and C_2 so that $|P \cap C_i| + w|V_{C_i}| \leq 2(|P| + w|V_C|)/3$, for $i = 1, 2$.

In Steps 2i and 3, if there are many planes that satisfy the conditions on the cuts, we use the plane that intersects the smallest number of rectangles in S_C.

4 Other Algorithms

In this section we discuss our implementation of some algorithms available in the literature for constructing BSPs. All the algorithms work on the same basic principle: examine all the planes containing the rectangles in S_B and determine how "good" each plane is. Split B using the "best" plane and recurse. Our implementation refines the original descriptions of these algorithms in two respects: (i) At a node B, we first check whether S_B contains a free rectangle; if it does, we apply the free cut containing that rectangle.[2] (ii) If there is more than one "best" plane, we choose one of these planes based on some simple criteria. To complete the description of each technique, it suffices to describe how it measures how "good" a candidate plane is.

For a plane π, let f_π denote the number of rectangles in S_B intersected by π, f_π^+ the number of rectangles in S_B completely lying in the positive halfspace defined by π, and f_π^- the number of rectangles in S_B lying completely in the negative halfspace defined by π. We also define the *occlusion factor* α_π to be the ratio of the total area of the rectangles in S_B lying in π to the area of π (when π is clipped within B), the *balance* β_π to be the ratio $\min\{f_\pi^+, f_\pi^-\}/\max\{f_\pi^+, f_\pi^-\}$ between the number of polygons that lie completely in each halfspace defined by π, and σ_π to be the *split factor* of π, which is the fraction of rectangles that π intersects, i.e., $\sigma_\pi = f_\pi/|S_B|$. We now discuss how each algorithm measures how good a plane is.

ThibaultNaylor: We discuss two of the three heuristics that Thibault and Naylor [7] present (the third performed poorly in our experiments). Below, w is a positive weight that can be changed to tune the performance of the heuristics.

1. Pick a plane the minimizes the function $|f_\pi^+ - f_\pi^-| + wf_\pi$.
2. Maximize the measure $f_\pi^+ \cdot f_\pi^- - wf_\pi$.

[2] Only Paterson and Yao's algorithm [4] originally incorporated the notion of free cuts.

In our experiments, we use $w = 8$, as suggested by Thibault and Naylor [7].

Airey: Airey [2] proposes a measure function that is a linear combination of a plane's occlusion factor, its balance, and its split factor: $0.5\alpha_\pi + 0.3\beta_\pi + 0.2\sigma_\pi$.

Teller: Let $0 \leq \tau \leq 1$ be a real number. Teller [6] chooses the plane with the maximum occlusion factor α_π, provided $\alpha_\pi \geq \tau$. If there is no such plane, he chooses the plane with the minimum value of f_π. We use the value $\tau = 0.5$ in our implementation, as suggested by Teller.

PatersonYao: For a box B, let s_x (resp., s_y, s_z) denote the number of edges of the rectangles in S_B that lie in the interior of B and are parallel to the x-axis (resp., y-axis, z-axis). We define the measure of B to be $\mu(B) = s_x s_y s_z$. We make a cut that is perpendicular to the smallest family of edges and divides B into two boxes, each with measure at most $\mu(B)/4$.

5 Experimental Results

We have implemented the above algorithms and run them on the following data sets containing orthogonal rectangles:[3]

1. the `Fifth` floor of Soda Hall containing 1677 rectangles,
2. the `Entire` Soda Hall model with 8690 rectangles,
3. the Orange United Methodist `Church` Fellowship Hall with 29988 rectangles,
4. the Sitterson Hall `Lobby` with 12207 rectangles, and
5. `Sitt`erson Hall containing 6002 rectangles.

We performed our experiments on a Sun SPARCstation 5 running SunOS 5.5.1 The table below displays the size of the BSP and the total number of times the rectangles are fragmented by the cuts made by the BSP.

	Fifth	Entire	Church	Lobby	Sitt.
#rectangles	1677	8690	29988	12207	6002
Size of the BSP					
Rounds	2744	14707	45427	22225	9060
Teller	2931	14950	33518	13911	7340
PatersonYao	3310	22468	56868	30712	20600
Airey	3585	24683	41270	21753	19841
ThibaultNaylor1	6092	32929	65313	25051	10836
ThibaultNaylor2	3235	20089	58175	23159	12192
Number of Fragments					
Rounds	113	741	838	475	312
Teller	301	1458	873	514	153
PatersonYao	449	5545	12517	9642	6428
Airey	675	7001	5494	5350	8307
ThibaultNaylor1	1868	10580	13797	3441	1324
ThibaultNaylor2	262	2859	6905	1760	1601

Examining this table, we note that, in general, the number of fragments and size of the BSP scale well with the size of the data set. For the Soda Hall data sets (`Fifth` and `Entire`), algorithm Rounds creates the smallest number of fragments and constructs the smallest BSP. For the other three sets, algorithm Teller performs best in terms of BSP size. However, there are some peculiarities in the table. For example, for the `Church` data set, Rounds creates a smaller number of fragments than Teller but constructs a larger BSP. We believe that this difference is explained by the fact that the 29998 rectangles in the `Church` model lie in a total of only 859 distinct planes. Since Teller makes cuts based on how much of a plane's area is covered by rectangles, it is reasonable to expect that the algorithm will "place" a lot of rectangles in cuts made close to the root of the BSP, thus leading to a BSP with a small number of nodes.

The time taken to construct the BSPs also scaled well with the size of the data sets. Rounds took 11 seconds to construct a BSP for the `Fifth` floor of Soda Hall and about 4.5 minutes for the `Church` data set. Typically, PatersonYao took about 15% less time than Rounds while the heuristics (Airey, ThibaultNaylor, and Teller) took 2-4 times as much time as Rounds to construct a BSP.

[3] We discarded all non-orthogonal polygons from these data sets. The number of such polygons was very small.

6 Conclusions

Our comparison indicates that Rounds and Teller are the best algorithms for constructing BSPs for orthogonal rectangles in $\mathbb{R}^3$. We plan to extend our algorithm to construct BSPs for triangles in $\mathbb{R}^3$. One algorithm we intend to implement is to enclose each triangle in an orthogonal bounding box and construct a BSP for the bounding boxes. We also plan to compare the different BSPs in terms of the time they take to answer standard visibility queries like ray shooting and line stabbing.

Acknowledgments We would like to thank Seth Teller for providing us with the Soda Hall data set created at the Department of Computer Science, University of California at Berkeley. We would also like to thank the Walkthrough Project, Department of Computer Science, University of North Carolina at Chapel Hill for providing us with the data sets for Sitterson Hall, the Orange United Methodist Church Fellowship Hall, and the Sitterson Hall Lobby.

References

[1] P. K. Agarwal, E. F. Grove, T. M. Murali, and J. S. Vitter, Binary space partitions for fat rectangles, *Proceedings of the 37th IEEE Annual Symposium on foundations of Computer Science (FOCS '96)*, October 1996.

[2] J. M. 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.

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

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

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

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

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