Binary Space Partitions for Fat Rectangles

Pankaj K. Agarwal[*] Edward F. Grove[†] T. M. Murali[‡] Jeffrey Scott Vitter[§]

Abstract

We consider the practical problem of constructing binary space partitions (BSPs) for a set S of n orthogonal, non-intersecting, two-dimensional rectangles in $\mathbb{R}^3$ such that the aspect ratio of each rectangle in S is at most α, for some constant $\alpha \geq 1$. We present an $n2^{O(\sqrt{\log n})}$-time algorithm to build a binary space partition of size $n2^{O(\sqrt{\log n})}$ for S. We also show that if m of the n rectangles in S have aspect ratios greater than α, we can construct a BSP of size $n\sqrt{m}2^{O(\sqrt{\log n})}$ for S in $n\sqrt{m}2^{O(\sqrt{\log n})}$ time. The constants of proportionality in the big-oh terms are linear in $\log \alpha$. We extend these results to cases in which the input contains non-orthogonal or intersecting objects.

1. Introduction

How to render a set of opaque or partially transparent objects in $\mathbb{R}^3$ in a visually realistic way is a fundamental problem in computer graphics [12, 22]. A central component of this problem is *hidden-surface removal*: given a set of objects, a viewpoint, and an image plane, compute the scene visible from the viewpoint on the image plane. Because of its importance, the hidden-surface removal problem has been studied extensively in both the computer graphics and the computational geometry communities [11, 12]. One of the conceptually simplest solutions to this problem is the z-buffer algorithm [6, 12]. This algorithm sequentially processes the objects; and for each object it updates the pixels of the image plane covered by the object, based on the distance information stored in the z-buffer. A very fast hidden-surface removal algorithm can be obtained by implementing the z-buffer in hardware. However, the cost of a hardware z-buffer is very high. Only special-purpose and costly graphics engines contain fast z-buffers, and z-buffers implemented in software are generally inefficient. Even when fast hardware z-buffers are present, they are not fast enough to handle the huge models (containing hundreds of millions of polygons) that often have to be displayed in real time. As a result, other methods have to be developed either to "cull away" a large subset of invisible polygons so as to decrease the rendering load on the z-buffer (when models are large; e.g., see [23]) or to completely solve the hidden-surface removal problem (when there are very slow or no z-buffers).

One technique to handle both of these problems is the *binary space partition* (BSP) introduced by Fuchs et al. [14]. They used the BSP to implement the so-called "painter's algorithm" for hidden-surface removal, which draws the objects to be displayed on the screen in a back-to-front order (in which no object is occluded by any object earlier in the order). In general, it is not possible to find a back-to-front order from a given viewpoint for an arbitrary set of objects. By fragmenting the objects, the BSP ensures that from *any* viewpoint a back-to-front order can be determined for the fragments.

Informally, a BSP for a set of objects is a tree each of whose nodes is associated with a convex region of space. The regions associated with the leaves of the tree form a convex decomposition of space, and the interior of each region does not intersect any object. The fragments created by the BSP are stored at appropriate nodes of the BSP. Given a viewpoint p, the back-to-front order is determined by a suitable traversal of the BSP. For each node v of the BSP, the objects in one of v's subtrees are separated from those in v's other subtree by a hyperplane. The viewpoint p will lie in one of the regions bounded by the hyperplane at v. The traversal recursively visits first the child of v cor-

[*] 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. Address: Box 90129, Department of Computer Science, Duke University, Durham, NC 27708–0129. Email: pankaj@cs.duke.edu

[†] Support was provided by Army Research Office grant DAAH04–93–G–0076. This work was partially done when the author was at Duke University. Address: Max-Planck-Institut für Informatik, Im Stadtwald, 66 Saarbrücken, Germany. Email: eddie@math.uri.edu

[‡] 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. Address: Box 90129, Department of Computer Science, Duke University, Durham, NC 27708–0129. Email: tmax@cs.duke.edu

[§] 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. Address: Box 90129, Department of Computer Science, Duke University, Durham, NC 27708–0129. Email: jsv@cs.duke.edu responding to the halfspace not containing p and then the other child of v. The efficiency of the traversal, and thus of the hidden-surface removal algorithm, depends upon the size of the BSP.

The BSP has subsequently proven to be a versatile data structure with applications in many other problems that arise in practice—global illumination [5], shadow generation [7, 8, 9], ray tracing [19], visibility problems [3, 23], solid geometry [17, 18, 24], robotics [4], and approximation algorithms for network flows and surface simplification [2, 16].

Although several simple heuristics have been developed for constructing BSPs of reasonable sizes [3, 13, 14, 23, 24], provable bounds were first obtained by Paterson and Yao. They show that a BSP of size $O(n^2)$ can be constructed for n disjoint triangles in $\mathbb{R}^3$, which is optimal in the worst case [20]. But in graphics-related applications, many common environments like buildings are composed largely of orthogonal rectangles. Moreover, many graphics algorithms approximate non-orthogonal objects by their orthogonal bounding boxes and work with the bounding boxes [12]. In another paper, Paterson and Yao show that a BSP of size $O(n\sqrt{n})$ exists for n non-intersecting, orthogonal rectangles in $\mathbb{R}^3$ [21]. This bound is optimal in the worst case.

In all known lower bound examples of orthogonal rectangles in $\mathbb{R}^3$ requiring BSPs of size $\Omega(n\sqrt{n})$, most of the rectangles are "thin." For example, Paterson and Yao's lower bound proof uses a configuration of $\Theta(n)$ orthogonal rectangles, arranged in a $\sqrt{n} \times \sqrt{n} \times \sqrt{n}$ grid, for which any BSP has size $\Omega(n\sqrt{n})$. All rectangles in their construction have aspect ratio $\Omega(\sqrt{n})$. Such configurations of thin rectangles rarely occur in practice. Many real databases consist mainly of "fat" rectangles, i.e., the aspect ratios of these rectangles are bounded by a constant. An examination of four datasets—the Sitterson Hall, the Orange United Methodist Church Fellowship Hall, and the Sitterson Hall Lobby databases from the University of North Carolina at Chapel Hill and the model of Soda Hall from the University of California at Berkeley—shows that most of the rectangles in these models have aspect ratio less than 30.

It is natural to ask whether BSPs of near-linear size can be constructed if most of the rectangles are "fat." We call a rectangle *fat* if its aspect ratio (the ratio of the longer side to the shorter side) is at most α, for a fixed constant $\alpha \geq 1$. A rectangle is said to be *thin* if its aspect ratio is greater than α. In this paper, we consider the following problem:

> *Given a set S of n non-intersecting, orthogonal, two-dimensional rectangles in $\mathbb{R}^3$, of which m are thin and the remaining $n-m$ are fat, construct a BSP for S.*

We first show how to construct a BSP of size $n2^{O(\sqrt{\log n})}$ for n fat rectangles in $\mathbb{R}^3$ (i.e., when $m = 0$). We then show that if $m > 0$, a BSP of size $n\sqrt{m}2^{O(\sqrt{\log n})}$ can be built. This bound comes close to the lower bound of $\Omega(n\sqrt{m})$.

We finally prove two important extensions to these results. We show that an $np2^{O(\sqrt{\log n})}$-size BSP exists if p of the n input objects are non-orthogonal. Unlike in the case of orthogonal objects, fatness does not help in reducing the worst-case size of BSPs for non-orthogonal objects. In particular, there exists a set of n fat triangles in $\mathbb{R}^3$ for which any BSP has $\Omega(n^2)$ size. However, non-orthogonal objects can be approximated by orthogonal bounding boxes. The resulting bounding boxes might intersect each other. Motivated by this observation, we also consider the problem where n fat rectangles contain k intersecting pairs of rectangles, and we show that we can construct a BSP of size $(n+k)\sqrt{k}2^{O(\sqrt{\log n})}$. There is a lower bound of $\Omega(n + k\sqrt{k})$ on the size of such a BSP.

In all cases, the constant of proportionality in the big-oh terms is linear in $\log \alpha$, where α is the maximum aspect ratio of the fat rectangles. Our algorithms to construct these BSPs run in time proportional to the size of the BSPs they build, except in the case of non-orthogonal objects, when the running time exceeds the size by a factor of p. Experiments demonstrate that our algorithms work well in practice and construct BSPs of near-linear size when most of the rectangles are fat, and perform better than Paterson and Yao's algorithm for orthogonal rectangles [1].

As far as we are aware, ours is the first work to consider BSPs for the practical and common case of two-dimensional, fat polygons in $\mathbb{R}^3$. de Berg considers a weaker model, the case of fat polyhedra in $\mathbb{R}^3$ (a polyhedron is said to be fat if its volume is at least a constant fraction of the volume of the smallest sphere enclosing it), although his results extend to higher dimensions [10].

One of the main ingredients of our algorithm is an $O(n \log n)$-size BSP for a set of n fat rectangles that are "long" with respect to a box B, i.e., none of the vertices of the rectangles lie in the interior of B. To prove this result, we crucially use the fatness of the rectangles. We can use this procedure to construct a BSP of size $O(n^{4/3})$ for fat rectangles. The algorithm repeatedly applies cuts that bisect the set of vertices of rectangles in the input set S until all sub-problems have long rectangles and the total size of the sub-problems is $O(n^{4/3})$, at which point we can invoke the algorithm for long rectangles. We improve the size of the BSP to $n2^{O(\sqrt{\log n})}$ by simultaneously simulating the algorithm for long rectangles and partitioning the vertices of rectangles in S in a clever manner.

The rest of the paper is organized as follows: Section 2 gives some preliminary definitions. In Section 3, we show how to build an $O(n \log n)$-size BSP for n long rectangles. Then we show how to construct a BSP of size $O(n^{4/3})$ in Section 4. Sections 5 and 6 present and analyze a better algorithm that constructs a BSP of size $n2^{O(\sqrt{\log n})}$. We extend this result in Section 7 to construct BSPs in cases when some objects in the input are (i) thin, (ii) non-orthogonal, or (iii) intersecting. We conclude in Section 8 with some open problems.

Due to lack of space, we defer many proofs to the full version of the paper.

2. Geometric preliminaries

A *binary space partition* $\mathcal{B}$ for a set S of pairwise-disjoint, $(d-1)$-dimensional, polyhedral objects in $\mathbb{R}^d$ is a tree recursively defined as follows: Each node v in $\mathcal{B}$ represents a convex region $\mathcal{R}_v$ and a set of objects $S_v = \{s \cap \mathcal{R}_v \mid s \in S\}$ 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 $\{s \cap H_v \mid s \in S_v\}$, 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 the set of objects $S_v^- = \{s \cap H_v^- \mid s \in S_v\}$ and the right subtree of v is a BSP for the set of objects $S_v^+ = \{s \cap H_v^+ \mid s \in S_v\}$. The size of $\mathcal{B}$ is the number of nodes in $\mathcal{B}$.

Suppose v is a node of $\mathcal{B}$. In all our algorithms, the region $\mathcal{R}_v$ associated with v is a *box* (rectangular parallelepiped). We say that a rectangle r is *long* with respect to $\mathcal{R}_v$ if none of the vertices of r lie in the interior of $\mathcal{R}_v$. Otherwise, r is said to be *short*. A long rectangle is said to be *free* if all its edges lie on the boundary of $\mathcal{R}_v$; otherwise it is *non-free*. A *free cut* is a cutting plane that divides S into two non-empty sets and does not cross any rectangle in S. Note that the plane containing a free rectangle is a free cut. Free cuts will be very useful in preventing excessive fragmentation of the objects in S.

We will often focus on a box B and construct a BSP for the rectangles intersecting it. 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. For a set of points P, let P_B be the subset of P lying in the interior of B.

Although a BSP is a tree, we will often discuss just how to partition the region represented by a node into two convex regions. We will not explicitly detail the associated construction of the actual tree itself.

We now state two preliminary lemmas that we will use in Sections 3 and 5. The first lemma characterizes a set of rectangles that are long with respect to a box and belong to one class. The second lemma applies to two classes of long rectangles.

Lemma 1 *Let C be a box, P a set of points in the interior of C, and R a set of rectangles long with respect to C. If the rectangles in R_C belong to one class,*

(i) *there exists a face g of the box C that contains one of the edges of each rectangle in R_C.*

(ii) *let V be the set of vertices of the rectangles in R_C that lie in the interior of g. In $O(|R_C| + |P|)$ time, we can find a plane h that partitions C into two boxes C_1 and C_2 such that $(|V \cap C_i| + a|P \cap C_i|) \leq (|V| + a|P|)/2$, for $i = 1, 2$.*

Lemma 2 *Let C be a box, P a set of points in the interior of C, and R a set of long rectangles with respect to C such that the rectangles in R_C belong to two classes. We can find two parallel free cuts h_1 and h_2 in $O(|R_C| + |P|)$ time that partition C into three boxes C_1, C_2, and C_3 such that either*

(i) *$|R_{C_i}| + a|P \cap C_i| \leq 2(|R_C| + a|P|)/3$, for all $1 \leq i \leq 3$, or*

(ii) *there is some $1 \leq i \leq 3$ such that $|R_{C_i}| + a|P \cap C_i| \geq 2(|R_C| + a|P|)/3$ and all rectangles in R_{C_i} belong to the same class.*

3. BSPs for long fat rectangles

Let S be a set of fat rectangles. Assume that all the rectangles in S are long with respect to a box B. In this section, we show how to build a BSP for S_B, the set of rectangles clipped within B. The box B 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 face of B lies on the yz-plane). See Figure 1.

A rectangle s 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. A long rectangle belongs to at least one of these three classes; a non-free rectangle belongs to a unique class. See Figure 1 for examples of rectangles belonging to different classes.

In general, S_B can have all three classes of rectangles. We first exploit the fatness of the rectangles to prove that whenever all three classes are present in S_B, a small number of cuts can divide B into boxes each of which has only two classes of rectangles. Then we describe an algorithm that constructs a BSP when all the rectangles belong to only two classes.

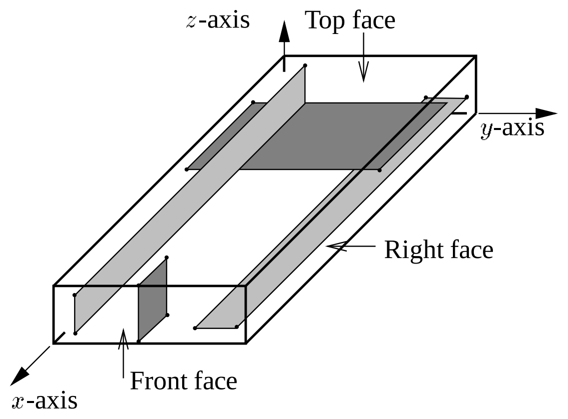

Figure 1. Different classes of rectangles.

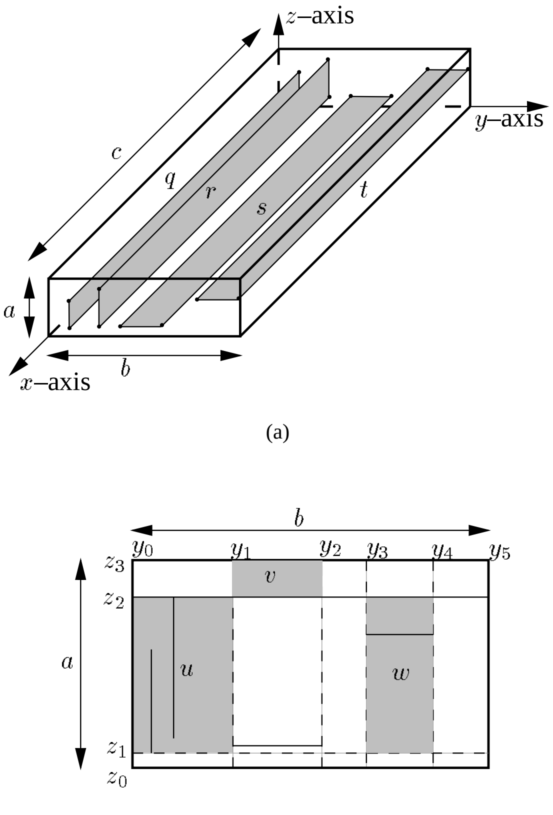

Figure 2. (a) Rectangles belonging to the sets R and T. (b) The back face of B; dashed lines are intersections of the back face with the α-cuts.

3.1. Reducing three classes to two classes

Let B and S_B be as defined earlier. Assume, without loss of generality, that the longest edge of B is parallel to the x-axis. The rectangles in S_B 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 box B) and the set T of rectangles that are horizontal (and parallel to the top face of box B). See Figure 2(a). Let e be the edge of B that lies on the z-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 $\bar{R} = \bigcup_{r \in R} \bar{r}$. 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 $\bar{R}$. (Note that $k-1$ may be less than $2|R|$, as in Figure 2(b), if some of the projected segments overlap.) Similarly, for each rectangle t in the set T, we define $\tilde{t}$ to be the projection of t on the y-axis, and $\tilde{T} = \bigcup_{t \in T} \tilde{t}$. Let $y_1 < y_2 < \cdots < y_{l-1}$ be the y-coordinates of the vertices of intervals in $\tilde{T}$ defined in the same way as $z_1, z_2, \cdots, z_{k-1}$.

We divide B into kl boxes by drawing the planes $z = z_i$ for $1 \le i < k$ and the planes $y = y_j$ for $1 \le j < l$. See Figure 2(b). This decomposition of B into kl boxes can easily be constructed in a tree-like fashion by performing $(k-1)(l-1)$ cuts. We refer to these cuts as *α-cuts*. If any resulting box has a free rectangle (such as t in Figure 2(b)), we divide that box into two boxes by applying the free cut along the free rectangle. Let $\mathcal{C}$ be the set of boxes into which B is partitioned in this manner. We can prove the following theorem about the decomposition of B into $\mathcal{C}$. This is the only place in the whole algorithm where we use the fatness of the rectangles in S.

Lemma 3 *The set of boxes $\mathcal{C}$ formed by the above process satisfies the following properties:*

(i) *Each box C in $\mathcal{C}$ has only two classes of rectangles,*

(ii) *There are at most $26\lfloor\alpha\rfloor^2 n$ boxes in $\mathcal{C}$, and*

(iii) $\sum_{C \in \mathcal{C}} |S_C| \le 18\lfloor\alpha\rfloor n$.

Proof: Let z_0 and z_k be the endpoints of e, the edge of the box B that lies on the z-axis. Similarly, define y_0 and y_l to be the endpoints of the edge of B that lies on the y-axis.

Claim (i) Let C be a box in $\mathcal{C}$. If C does not contain a rectangle from $T \cup R$, the proof is trivial since the rectangles in T and R together constitute the front class. Suppose C contains rectangles from the set R. Rectangles in R belong to the front class and are parallel to the right face of B. We claim that C cannot have any rectangles from the right class. To see this claim, consider an edge of C parallel to the z-axis. The endpoints of this edge have z-coordinates z_i and z_{i+1}, for some $0 \le i < k$. Since C

contains a rectangle from R, the interval $z_i z_{i+1}$ must be covered by projections of rectangles from R onto the z-axis. Any rectangle from the right class inside C must intersect one of the rectangles whose projections cover $z_i z_{i+1}$. That cannot happen since the rectangles in S do not intersect each other. A similar proof shows that if C contains rectangles from T, then C is free of rectangles in the top class.

Claim (ii) We first show that that both k and l are at most $2\lfloor\alpha\rfloor + 3$. Let a (respectively, b, c) denote the length of the edges of B parallel to the z-axis (respectively, y-axis, x-axis). By assumption, $a, b \leq c$. Let r be a rectangle from R with dimensions z and x, where $z \leq x$. Consider $\bar{r}$, the projection of r onto the z-axis. Suppose that $\bar{r} \subseteq z_i z_{i+1}$, for some $0 < i < k-1$, i.e., $\bar{r}$ lies in the interior of the edge e of B lying on the z-axis. Since r is a rectangle in the front class and is parallel to the right face of B, we know that $z \leq a \leq c = x$. If $\hat{r}$, the rectangle supporting r in the set S has dimensions $\hat{z}$ and $\hat{x}$, where $\hat{z} \leq \hat{x} \leq \alpha\hat{z}$, we have $z = \hat{z}$ and $x \leq \hat{x}$. (If i is 0 or $k-1$, we cannot claim that $z = \hat{z}$; in these cases, it is possible that z is much less than $\hat{z}$.) See Figure 3. We see that

$$a \leq c = x \leq \hat{x} \leq \alpha\hat{z} = \alpha z.$$

It follows that the length of $\bar{r}$, and hence the length of $z_i z_{i+1}$, is at least a/α. Since every alternate interval $z_i z_{i+1}, 0 < i < k-1$ contains $\bar{s}$ for at least one rectangle s in R, k is at most $2\lfloor\alpha\rfloor + 3$. In a similar manner, l is also at most $2\lfloor\alpha\rfloor + 3$.

This implies that B is divided into at most $kl \leq (2\lfloor\alpha\rfloor + 3)^2$ boxes by the planes $z = z_i$, $1 \leq i \leq k-1$ and the planes $y = y_j$, $1 \leq j \leq l-1$. Each such box C can contain at most n rectangles. Hence, at most n free cuts can be made inside C. The free cuts can divide C into at most $n+1$ boxes. This implies that the set $\mathcal{C}$ has at most at most $26\lfloor\alpha\rfloor^2 n$ boxes.

Claim (iii) Each rectangle r in S_B is cut into at most kl pieces. The edges of these pieces form an arrangement on r. Each face of the arrangement is one of the at most kl rectangles that r is partitioned into. Only $2(k+l-1)$ faces of the arrangement have an edge on the boundary of r. All other faces can be used as free cuts. Hence, after all possible free cuts are used in the boxes into which B is divided by the kl cuts, only $2(k+l-1)$ pieces of each rectangle in S_B survive. This proves that $\sum_{C\in\mathcal{C}} |S_C| \leq 18\lfloor\alpha\rfloor n$.

□

3.2. BSPs for two classes of long rectangles

Let C be one of the boxes into which B is partitioned in Section 3.1. We now present an algorithm for constructing a BSP for the set of clipped rectangles S_C, which has only two classes of long rectangles. We recursively apply the following steps to each of the boxes produced by the algorithm until no box contains a rectangle.

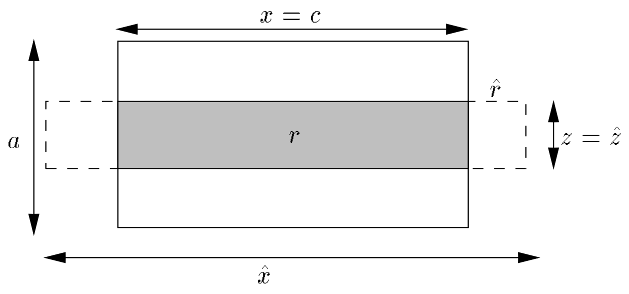

Figure 3. Projections of $\hat{r}$ (the dashed rectangle), $r = \hat{r} \cap B$ (the shaded rectangle), and the right face of B onto the zx-plane.

1. If S_C has a free rectangle, we use the free cut containing that rectangle to split C into two boxes.

2. If S_C has two classes of rectangles, we use Lemma 2 (with $R = S$ and $P = \emptyset$) to split C into at most three boxes using two parallel free cuts.

3. If S_C has only one class of rectangles, we split C into two by a plane as suggested by Lemma 1 (with $R = S$ and $P = \emptyset$).

We first analyze the algorithm for two classes of long rectangles. The BSP produced has the following structure: If Step 3 is executed at a node v, then Step 2 is not invoked at any descendant of v. In view of Lemma 2, repeated execution of Steps 1 or 2 on S_C constructs in $O(|S_C| \log |S_C|)$ time a top subtree $\mathcal{T}_C$ of the BSP with $O(|S_C|)$ nodes such that each leaf in $\mathcal{T}_C$ has only one class of rectangles and the total number of rectangles in all the leaves is at most $|S_C|$. At each leaf v of the tree $\mathcal{T}_C$, recursive invocations of Steps 1 and 3 build a BSP of size $O(|S_v| \log |S_v|)$ in $O(|S_v| \log |S_v|)$ time (see [20] for details). Since $\sum_v S_v \leq |S_C|$, where the sum is taken over all leaves v of $\mathcal{T}_C$, the total size of the BSP constructed inside C is $O(|S_C| \log |S_C|)$.

We now analyze the overall algorithm for long rectangles. The algorithm first applies the α-cuts to the rectangles in S_B, as described in Section 3.1. Consider the set of boxes $\mathcal{C}$ produced by the α-cuts. Each of the boxes in $\mathcal{C}$ contains only two classes of rectangles (by Lemma 3(i)). In view of the above discussion, for each box $C \in \mathcal{C}$, we can construct a BSP for S_C of size $O(|S_C| \log |S_C|)$ in time $O(|S_C| \log |S_C|)$.

Lemma 3(ii) and 3(iii) imply that the total size of the BSP is $O(n) + \sum_{C \in \mathcal{C}} O(|S_C| \log |S_C|) = O(n \log n)$. The BSP can be built in the same time. We can now state the following theorem.

Theorem 1 *Let S be a set of n fat rectangles and B a box so that all rectangles in S are long with respect to B. Then an $O(n \log n)$-size BSP for the clipped rectangles S_B can be constructed in $O(n \log n)$ time. The constants of proportionality in the big-oh terms are linear in α^2, where α is the maximum aspect ratio of the rectangles in S.*

Remark: In our algorithm for two classes of long rectangles, by using in Step 3 above the algorithm of Paterson and Yao for constructing linear-size BSPs for orthogonal segments in the plane [21], rather than their $O(n \log n)$ algorithm for arbitrarily-oriented segments in the plane [20], we can improve the size of the BSP to linear. This improvement implies that we can construct linear-size BSPs for long rectangles. We will not need this improved result below, except in Section 4.

4. BSPs of size $O(n^{4/3})$

In this section, we present a simple algorithm that constructs a BSP of size $O(n^{4/3})$ for n fat rectangles. We then use the intuition gained from the $O(n^{4/3})$ algorithm to develop an improved BSP algorithm in Section 5. We analyze the improved algorithm in Section 6.

We need a definition before describing the algorithm. A *bisecting cut* is an orthogonal cut that divides B into two boxes and bisects the set of vertices of rectangles in S that lie in the interior of B.

The algorithm for fat rectangles proceeds in phases. A *phase* is a sequence of three bisecting cuts, with exactly one cut perpendicular to each of the three orthogonal directions. After each phase, if a box contains a free rectangle, we use the corresponding free cut to further divide the box into two. We begin the first phase with a box enclosing all the rectangles with at most $4n$ vertices in its interior (since there are n rectangles in S each with four vertices) and continue executing phases of bisecting cuts until each node has no vertex in its interior. At termination, each node contains only long rectangles. We then invoke the algorithm for long rectangles to construct a BSP in each of these nodes.

The crux of the analysis of the size of the BSP produced by this algorithm is counting how many pieces one rectangle can split into when subjected to a specified number of phases. To this effect, we use the following result due to Paterson and Yao [21].

Lemma 4 (Paterson-Yao) *A rectangle that has been subjected to d phases of cuts (with free cuts used whenever possible) is divided into $O(2^d)$ rectangles.*

Theorem 2 *A BSP in $\mathbb{R}^3$ of size $O(n^{4/3})$ can be constructed for n fat orthogonal rectangles. The constant of proportionality in the big-oh term is linear in α^2, where α is the maximum aspect ratio of the input rectangles.*

Proof: If a box B has k vertices in its interior, one phase of cuts partitions B into boxes each of which has at most $k/8$ vertices in its interior. Since we start with n rectangles that have at most $4n$ vertices, the number of phases executed by the above algorithm is at most $\lceil (\log n)/3 + 2/3 \rceil$. Lemma 4 now implies that the total number of rectangles formed once all the phases are executed is $O(n \times 2^{\lceil (\log n)/3 + 2/3 \rceil}) = O(n^{4/3})$. At this stage, all nodes have only long rectangles. Hence, Theorem 1 and the remark at the end of Section 3 imply that constructing a BSP in each of these nodes increases the total size of the BSP only by a constant factor. This proves the theorem. $\square$

5. The improved algorithm

The algorithm proceeds in rounds. Each round simulates a few steps of the algorithm for long rectangles as well as partitions the vertices of the rectangles in S into a small number of sets of approximately equal size. 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. Our algorithm maintains the invariant that for each box $B \in Q_i$, all long rectangles in S_B are non-free. 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 in S_B that are long with respect to B. Set $f = |F|$ and k to 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 the input set S). By assumption, all rectangles in F are non-free. We choose a parameter a, which remains fixed throughout the round. We pick $a = 2^{\sqrt{\log(f+k)}}$ to optimize the size of the BSP that the algorithm creates (see Section 6). We now describe the ith round in detail.

If $k = 0$, i.e., if all rectangles in S_B are long, we apply the algorithm described in Section 3 to construct a BSP for S_B. Otherwise, we perform a sequence of cuts in two stages that partition B as follows:

Separating Stage: We apply the α-cuts, as described in Section 3.1. We make these cuts with respect to the rectangles in F, i.e., we consider only those rectangles of S_B that are long with respect to B. In each box so formed, if there is a free rectangle, we apply the free cut along that rectangle. Let $\mathcal{C}$ be the resulting set of boxes.

Dividing Stage: We refine each box C in $\mathcal{C}$ by applying cuts, similar to the ones made in Section 3.2, as described below. We recursively invoke the dividing stage on each box that C is partitioned into. Let k_C denote the number of vertices of rectangles in S_C that lie in the interior of C. The set F_C is the set of rectangles in F that are clipped within C.

1. If C has any free rectangle, we use the free cut containing that rectangle to split C into two boxes.
2. If $|F_C| + ak_C \leq (f + ak)/2\sqrt{a}$, we do nothing.
3. If the rectangles in F_C belong to two classes, let P_C denote the set of vertices of the rectangles in S_C that lie in the interior of C. We apply two parallel free cuts h_1 and h_2 that satisfy Lemma 2, with $R = F$ and $P = P_C$.
4. If the rectangles in F_C belong to just one class, we apply one cut h using Lemma 1, with $R = F$ and $P = P_C$.

The cuts introduced during the dividing stage can be made in a tree-like fashion. At the end of the dividing stage, we have a set of boxes so that for each box D in this set, S_D does not contain any free rectangle and $|F_D| + ak_D \leq (f + ak)/2\sqrt{a}$. Notice that as we apply cuts in C and in the resulting boxes, rectangles that are short with respect to C may become long with respect to the new boxes. We ignore these new long rectangles until the next round, except when they induce a free cut.

6. Analysis of the improved algorithm

We now analyze the size of the BSP constructed by the algorithm and the time complexity of the algorithm. In a round, the algorithm constructs a top subtree $\mathcal{T}_B$ on a box B for the set of clipped rectangles S_B. Recall that F is the set of rectangles long with respect to B. For a node C in $\mathcal{T}_B$, let $\mathcal{T}_C$ be the subtree of $\mathcal{T}_B$ rooted at C, ϕ_C the number of long rectangles in F_C, and n_C the number of long rectangles in $S_C \setminus F_C$.

For a box D corresponding to a leaf of $\mathcal{T}_B$, let f_D be the number of long rectangles in S_D. Note that f_D counts both the "old" long rectangles in F_D (pieces of rectangles that were long with respect to B) and the "new" long rectangles in $S_D \setminus F_D$ (pieces of rectangles that were short with respect to B, but became long with respect to D due to the cuts made during the round) ; $f_D = \phi_D + n_D$.

Lemma 5 *For a box D associated with a leaf of $\mathcal{T}_B$, we have*
$$f_D + ak_D \leq (f + ak)/\sqrt{a}.$$

Proof: We know that n_D is at most k (since a rectangle in $S_D \setminus F_D$ must be a piece of a rectangle short with respect to B, and there are at most k such short rectangles). Hence,
$$\begin{aligned} f_D + ak_D &\leq \phi_D + n_D + ak_D \\ &\leq \phi_D + k + ak_D. \end{aligned}$$

Since $\phi_D + ak_D \leq (f + ak)/2\sqrt{a}$, the lemma follows. □

For a box C in $\mathcal{T}_B$, we use the notation L_C to denote the set of leaves in $\mathcal{T}_C$.

Lemma 6 *Let C be a box associated with a node in $\mathcal{T}_B$. If all rectangles in F_C belong to one class, then*
$$\sum_{D \in L_C} f_D \leq 2\phi_C + 2\,(n_C + k_C) \left\lceil \frac{\phi_C + ak_C}{\mu} \right\rceil,$$
where $\mu = (f + ak)/2\sqrt{a}$.

Lemma 7 *Let C be a box associated with a node in $\mathcal{T}_B$. If all rectangles in F_C belong to two classes, then*
$$\sum_{D \in L_C} f_D \leq 2\phi_C + 3\,(n_C + k_C) \left(\frac{\phi_C + ak_C}{\mu} \right)^3,$$
where $\mu = (f + ak)/2\sqrt{a}$.

Lemma 8 *The tree $\mathcal{T}_B$ constructed on box B in a round has the following properties:*

(i) $\sum_{D \in L_B} k_D \leq k$,

(ii) $\sum_{D \in L_B} f_D \leq O(f + a^{3/2}k)$, *and*

(iii) $|L_B| = O(f \log a + a^{3/2}k)$.

Proof: The bound on $\sum_{D \in L_B} k_D$ follows, since each vertex in the interior of S_B lies in the interior of at most one box of L_B. Next, we will use Lemmas 6 and 7 to prove a bound on $\sum_{D \in L_B} f_D$. A similar argument will prove the bound on $|L_B|$.

Let $\mathcal{C}$ be the set of boxes into which B is partitioned by the separating stage. See Figure 4. Let C be a box in $\mathcal{C}$. Since all rectangles in F_C belong to at most two classes, Lemmas 6 and 7 imply that
$$\sum_{D \in L_C} f_D = 2\phi_C + 3\,(n_C + k_C) \left(\frac{\phi_C + ak_C}{\mu} \right)^3, \quad (1)$$

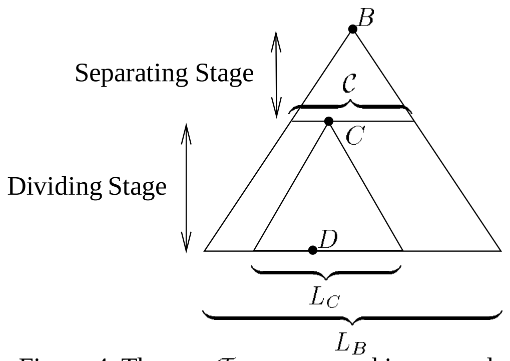

Figure 4. The tree $\mathcal{T}_B$ constructed in a round.

where $\mu = (f + ak)/2\sqrt{a}$.

The boxes in $\mathcal{C}$ correspond to the leaves of a top subtree of $\mathcal{T}_B$. Therefore, the total number of long rectangles in the boxes associated with the leaves of $\mathcal{T}_B$ is

$$\sum_{D \in L_B} f_D = \sum_{C \in \mathcal{C}} \sum_{D \in L_C} f_D,$$

which by equation (1) is

$$\sum_{D \in L_B} f_D = \sum_{C \in \mathcal{C}} \left(2\phi_C + 3\left(n_C + k_C\right) \left(\frac{\phi_C + ak_C}{\mu} \right)^3 \right).$$

By an argument similar to the one used to prove Lemma 3, we have $\sum_{C \in \mathcal{C}} \phi_C = O(f)$ and $\sum_{C \in \mathcal{C}} n_C = O(k)$. We also know that $\sum_{C \in \mathcal{C}} k_C \leq k$. Therefore, we obtain

$$\begin{aligned} \sum_{D \in L_B} f_D &= O\left(f + k \left(\frac{f + ak}{\mu} \right)^3 \right) \\ &= O\left(f + a^{3/2}k \right). \end{aligned}$$

□

We now bound the size of the BSP constructed by the algorithm. Let $S(f, k)$ denote the maximum size of the BSP produced by the algorithm for a box that contains f long rectangles and k vertices in its interior. If $k = 0$, Theorem 1 implies that $S(f, k) = O(f \log f)$. For the case $k > 0$, by Lemma 8(iii), we construct the subtree $\mathcal{T}_B$ on B of size $O(f \log a + a^{3/2}k)$ in one round, and recursively construct subtrees for each box in the set of leaves L_B. Hence, when $k > 0$,

$$S(f, k) = \sum_{D \in L_B} S(f_D, k_D) + O(f \log a + a^{3/2}k),$$

where $\sum_D k_D \leq k$, $\sum_D f_D \leq O(f + a^{3/2}k)$, and $f_D + ak_D \leq (f + ak)/\sqrt{a}$ for every box D in L_B. The solution to this recurrence is

$$S(f, k) = (f + k)2^{O(\sqrt{\log(f+k)})},$$

where the constant of proportionality in the big-oh term is linear in $\log \alpha$. The intuition behind this solution is that each round increases the number of "old" long rectangles by at most a constant factor, while also creating $O(a^{3/2}k)$ "new" long rectangles. The depth of each round is $O(\log a)$. Choosing $a = 2^{\sqrt{\log(f+k)}}$ balances the total increase in the number of "old" rectangles (over all the rounds) and the total increase in the number of "new" rectangles.

Since all operations at a node can be performed in time linear in the number of rectangles at that node, the same bound can be obtained for the running time of the algorithm. Since $f \leq n$ and $k \leq 4n$ at the beginning of the first round, we obtain the following theorem.

Theorem 3 *Given a set S of n rectangles in $\mathbb{R}^3$ such that the aspect ratio of each rectangle in S is bounded by a constant $\alpha \geq 1$, we can construct a BSP of size $n2^{O(\sqrt{\log n})}$ for S in time $n2^{O(\sqrt{\log n})}$. The constants of proportionality in the big-oh terms are linear in $\log \alpha$.*

7. Extensions

In this section, we show how to modify the algorithm of Section 5 to handle the following three cases: (i) some of the rectangles are thin, (ii) some of the rectangles are non-orthogonal, and (iii) the input consists of intersecting fat rectangles.

7.1. Fat and thin rectangles

Let us assume that the input $S = F \cup T$ has n rectangles, consisting of m thin rectangles in T and $n - m$ fat rectangles in F.

Given a box B, let f be the number of long rectangles in F_B, let k be the number of vertices of rectangles in F_B that lie in the interior of B, and let t be the number of rectangles in T_B. The algorithm we use now is very similar to the algorithm for fat rectangles. We fix the parameter $a = 2^{\sqrt{\log(f+k)}}$.

1. If S_B contains a free rectangle, we use the corresponding free cut to split B into two boxes.

2. If $k = t = 0$, we use the algorithm for long rectangles to construct a BSP for B.

3. If $t \geq (f + k)$, we use the algorithm by Paterson and Yao for orthogonal rectangles in $\mathbb{R}^3$ to construct a BSP for B [21].

4. If $(f + k) > t$, we perform one round of the algorithm described in Section 5.

This algorithm is recursively invoked on all boxes that B is split into. Let $S(k,f,t)$ be the size of the BSP produced by this algorithm for a box with k vertices in its interior, f long rectangles in F_B, and t thin rectangles in T_B. Analyzing the algorithm's behavior as in Section 5, we can show that $S(k,f,t) = O(f \log f)$, when $k = t = 0, S(k,f,t) = O(t\sqrt{t})$, when $t \geq (f+k)$ (see [21] for details), and when $(f+k) > t$,

$$\begin{aligned} S(k,f,t) &= \sum_D S(k_D, f_D, t_D) \\ &+ O(f \log a + a^{3/2}k + a^{3/2}t) \end{aligned}$$

The solution to this recurrence is

$$S(k,f,t) = (f+k)2^{O(\sqrt{\log(f+k)})} + O(t\sqrt{t}),$$

where the constant of proportionality in the first big-oh term is linear in $\log \alpha$. The following theorem is immediate.

Theorem 4 *A BSP of size $n\sqrt{m}2^{O(\sqrt{\log n})}$ can be constructed in $n\sqrt{m}2^{O(\sqrt{\log n})}$ time for n rectangles in $\mathbb{R}^3$, of which m are thin. The constants of proportionality in the big-oh terms are linear in $\log \alpha$, where α is the maximum aspect ratio of the fat rectangles.*

There exists a set of m thin rectangles and $n - m$ fat rectangles in $\mathbb{R}^3$ for which any BSP has size $\Omega(n\sqrt{m})$.

7.2. Fat rectangles and non-orthogonal rectangles

Suppose p objects in the input are non-orthogonal and the rest are fat rectangles. The algorithm we use is very similar to the algorithm in Section 7.1, except in two places. In Step 1, we check whether we can make free cuts through the non-orthogonal objects too. In Step 3, if the number of non-orthogonal object at a node dominates the number of fat rectangles, we use Paterson and Yao's algorithm for triangles in $\mathbb{R}^3$ to construct a BSP of size quadratic in the number of objects in cubic time [20].

Theorem 5 *A BSP of size $np2^{O(\sqrt{\log n})}$ can be constructed in $np^2 2^{O(\sqrt{\log n})}$ time for n objects in $\mathbb{R}^3$, of which p are non-orthogonal and the rest are fat rectangles. The constants of proportionality in the big-oh terms are linear in $\log \alpha$, where α is the maximum aspect ratio of the fat rectangles.*

7.3. Intersecting fat rectangles

We now consider the case when the n fat rectangles contain $k \leq \binom{n}{2}$ intersecting pairs. For each intersecting pair of rectangles, we break one of the rectangles in the pair into a constant number of smaller pieces such that the pieces do not intersect the other rectangle in the pair. This process creates a total of $n + O(k)$ rectangles. Some or all of the "new" $O(k)$ rectangles may be thin. We then use the algorithm of Section 7.1 to construct a BSP for the rectangles. The theorem below follows.

Theorem 6 *A BSP of size $(n+k)\sqrt{k}2^{O(\sqrt{\log n})}$ can be constructed in $(n+k)\sqrt{k}2^{O(\sqrt{\log n})}$ time for n rectangles in $\mathbb{R}^3$, which have k intersecting pairs of rectangles. The constants of proportionality in the big-oh terms are linear in $\log \alpha$, where α is the maximum aspect ratio of the fat rectangles.*

There exists a set of n rectangles in $\mathbb{R}^3$, containing k intersecting pairs, for which any BSP has size $\Omega(n + k\sqrt{k})$.

8. Conclusions

Since worst-case complexities for BSPs are very high ($\Omega(n^2)$ for n triangles in $\mathbb{R}^3$ and $\Omega(n^{3/2})$ for n orthogonal rectangles in $\mathbb{R}^3$) and all known examples that achieve the worst case use mainly skinny objects, we have made the natural assumption that objects are fat and have shown that this assumption allows smaller worst-case size of BSPs. We have implemented these algorithms. The practical results are very encouraging and are presented in a companion paper [1].

It seems very probable that BSPs of size smaller than $n2^{O(\sqrt{\log n})}$ can be built for n orthogonal rectangles of bounded aspect-ratio in $\mathbb{R}^3$. The only lower bound we have is the trivial $\Omega(n)$ bound. It would be interesting to see if algorithms can be developed to construct BSPs of optimal size. Similar improvements can be envisioned for Theorems 4, 5 and 6.

An even more challenging open problem is determining the right assumptions that should be made about the input objects and the graphics display hardware so that provably fast and *practically efficient* algorithms can be developed for doing hidden-surface elimination of these objects. A preliminary investigation into an improved model for graphics hardware has been made by Grove et al. [15].

Acknowledgments We would like to thank Seth Teller for providing us with the Soda Hall dataset 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 datasets 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. Practical methods for constructing binary space partitions for orthogonal objects. In preparation.

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

[3] J. M. Airey. *Increasing Update Rates in the Building Walkthrough System with Automatic Model-space Subdivision and Potentially Visible Set Calculations*. PhD thesis, University of North Carolina, Chapel Hill, 1990.

[4] C. Ballieux. Motion planning using binary space partitions. Technical report, Utrecht University, 1993.

[5] A. T. Campbell. *Modeling Global Diffuse Illumination for Image Synthesis*. Ph.D. thesis, University of Texas, Austin, Dec. 1991.

[6] E. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. PhD thesis, Computer Science Department, University of Utah, Salt Lake City, UT, December 1974.

[7] N. Chin. Near real-time object-precision shadow generation using BSP trees—master thesis. Technical Report CUCS-068-90, University of Columbia, 1990.

[8] N. Chin and S. Feiner. Near real-time shadow generation using bsp trees. In *Computer Graphics*, volume 23, pages 99–106, July 1989.

[9] N. Chin and S. Feiner. Fast object-precision shadow generation for areal light sources using BSP trees. In D. Zeltzer, editor, *Computer Graphics (1992 Symposium on Interactive 3D Graphics)*, volume 25, pages 21–30, Mar. 1992.

[10] M. de Berg. Linear size binary space partitions for fat objects. In *Algorithms – ESA'95*, 1995.

[11] S. E. Dorward. A survey of object-space hidden surface removal. *Internat. J. Comput. Geom. Appl.*, 4:325–362, 1994.

[12] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, Reading, MA, 1990.

[13] H. Fuchs, G. D. Abram, and E. D. Grant. Near-real-time shaded display of rigid objects. *Computer Graphics*, 10(3):65–72, July 1983.

[14] 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.

[15] E. F. Grove, T. M. Murali, and J. S. Vitter. The object complexity model for hidden-surface elimination. In *Proc. 7th Canad. Conf. Comput. Geom.*, pages 273–278, 1995.

[16] J. S. B. Mitchell. On maximum flows in polyhedral domains. *J. Comput. Syst. Sci.*, 40:88–123, 1990.

[17] B. Naylor. SCULPT an interactive solid modeling tool. In *Proceedings of Graphics Interface '90*, pages 138–148, May 1990.

[18] B. Naylor, J. Amanatides, and W. Thibault. Merging BSP trees yields polyhedral set operations. In F. Baskett, editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 115–124, Aug. 1990.

[19] B. Naylor and W. Thibault. Application of BSP trees to ray-tracing and CSG evaluation. Technical Report GIT-ICS 86/03, Georgia Institute of Tech., School of Information and Computer Science, Feb. 1986.

[20] 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.

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

[22] I. E. Sutherland, R. F. Sproull, and R. A. Shumacker. A characterization of ten hidden surface algorithms. *ACM Comput. Surv.*, 6:1–55, 1974.

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

[24] W. C. Thibault and B. F. Naylor. Set operations on polyhedra using binary space partitioning trees. In M. C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 153–162, July 1987.