Cylindrical Static and Kinetic Binary Space Partitions

Pankaj K. Agarwal* Leonidas J. Guibas† T. M. Murali‡ Jeffrey Scott Vitter§

Abstract

We describe the first known algorithm for efficiently maintaining a Binary Space Partition (BSP) for n continuously moving segments in the plane. Under reasonable assumptions on the motion, we show that the total number of times the BSP changes is $O(n^2)$, and that we can update the BSP in $O(\log n)$ expected time per change. We also consider the problem of constructing a BSP for n triangles in $\mathbb{R}^3$. We present a randomized algorithm that constructs a BSP of expected size $O(n^2)$ in $O(n^2 \log^2 n)$ expected time. We also describe a deterministic algorithm that constructs a BSP of size $O((n+k)\log n)$ and height $O(\log n)$ in $O((n+k)\log^2 n)$ time, where k is the number of intersection points between the edges of the projections of the triangles onto the xy-plane.

1 Introduction

The Binary Space Partition (BSP, also known as BSP tree), originally proposed by Schumacker et al. [26] and further refined by Fuchs et al. [16], is a hierarchical partitioning of space widely used in several areas, including computer graphics (global illumination [7], shadow generation [10, 11], visibility determination [4, 28], and ray tracing [21]), solid modeling [22, 20, 29], geometric data repair [19], robotics [5], network design [18], and surface simplification [3]. Key to the BSP's success is that it serves both as a model for an object (or a set of objects) and as a data structure for querying the object.

Informally, a BSP $\mathcal{B}$ for a set of objects is a binary tree, where each node v is associated with a convex region Δ_v. The regions associated with the children of v are obtained by splitting Δ_v with a hyperplane. If v is a leaf of $\mathcal{B}$, then the interior of Δ_v does not intersect any object. The regions associated with the leaves of the tree form a convex decomposition of space. The faces of the decomposition induced by the leaves intersect the objects and divide them into fragments; these fragments are stored at appropriate nodes of the BSP. The efficiency of BSP-based algorithms depends on the number of nodes in the tree and on the height of the tree. As a result, several algorithms for constructing BSPs of small size and/or small height have been proposed; see [4, 8, 16, 24, 25, 28, 29].

In this paper, we study *cylindrical* BSPs in which all the cuts that do not contain any input objects are made by hyperplanes parallel to the same fixed direction. We address two problems. The first problem can be formulated as follows: Let S be a set of n interior-disjoint segments in the plane, each moving along a continuous path. We want to maintain the BSP for S as the segments in S move. We assume that the segments move in such a way that they never intersect, except possibly at their endpoints. Most of the work to date deals with constructing a BSP for a set of "static" segments, which do not move. Paterson and Yao propose a randomized algorithm that constructs a BSP of expected $O(n \log n)$ size for a set of n segments in the plane [24]. They also propose a deterministic algorithm, based on a divide-and-conquer approach, that constructs a BSP of size $O(n \log n)$ in $O(n \log n)$ time [24]. Both of these algorithms are not "robust," in the sense that a small motion of one of the segments may cause many changes in the tree, or may cause non-local changes. Therefore, they are ill-suited for maintaining a BSP for a set of moving segments.

There have been a few attempts to update BSPs when the objects defining them move. Naylor describes a method to implement dynamic changes in a BSP, where the static objects are represented by a balanced BSP (computed in a preprocessing stage), and then the moving objects are inserted at each time step into the static tree [23]. Using the same assumption that moving objects are known *a priori*, Torres proposes the augmentation of BSPs with additional separating planes, which may localize the updates needed for deletion and re-insertion of moving objects in a BSP [30]. This approach does try to exploit (by introducing additional planes) the spatial coherence of the dynamic changes in the tree. Chrysanthou suggests a more general approach, which does not make any distinction between static and moving objects [12]. By keeping additional information about topological adjacencies in the tree, the algorithm performs insertions and deletions of a node in a more localized way. But all these prior efforts boil down to deleting moving objects from their earlier positions and re-inserting them in their current positions after some time interval has elapsed. Such approaches suffer from the fundamental problem that it is

---

*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: Center for Geometric Computing, Department of Computer Science, Duke University, Box 90129, Durham, NC 27708–0129. Email: pankaj@cs.duke.edu

†Support was provided in part by National Science Foundation grant CCR–9623851 and by US Army MURI grant 5–23542–A. Address: Graphics Laboratory, Computer Science Department, Gates Building 3B, Room 374, Stanford University, Stanford, CA 94305. Email: guibas@cs.stanford.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: Center for Geometric Computing, Department of Computer Science, Duke University, Box 90129, 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: Center for Geometric Computing, Department of Computer Science, Duke University, Box 90129, Durham, NC 27708–0129. Email: jsv@cs.duke.edu

very difficult to know how to choose the correct time interval size: if the interval is too small, then the BSP does not in fact change combinatorially, and the deletion/re-insertion is just wasted computation; if it is too big, then important intermediate events can be missed, which affect applications that use the tree.

Our algorithm, instead, treats the BSP as a *kinetic data structure*, as defined by Basch et al. [6]. We view the equations of the cuts made at the nodes of the BSP and the edges and faces of the subdivision induced by the BSP as functions of time. The cuts and the edges and faces of the subdivision change continuously with time. However, "combinatorial" changes in the BSP and in the subdivision (we precisely define this notion later) occur only at certain times. We explicitly take advantage of the continuity of the motion of the objects involved so as to generate updates to the BSP only when actual events cause the BSP to change combinatorially.

In Section 3, we describe a randomized kinetic algorithm for maintaining a BSP for moving segments in the plane. We assume that the segment motions are *oblivious* to the random bits used by the algorithm. Following Basch et al. [6], we assume that each moving segment has a posted flight plan that gives full or partial information about its current motion. Whenever a flight plan changes (possibly due to an external agent), our algorithm is notified and it updates a global event queue to reflect the change. We first derive a randomized algorithm for computing a BSP for a set of static segments, which combines ideas from Paterson and Yao's randomized and deterministic algorithms, but is also robust, in the sense described earlier. The "combinatorial structure" of the BSP constructed by this algorithm changes only when the x-coordinates of a pair of segment endpoints, among a certain subset of $O(n)$ pairs, become equal. We show that under the above assumption on the segment motions, the BSP can be updated in $O(\log n)$ expected time at each such event. We also show that if k of the segments of S move along "pseudo-algebraic" paths, and the remaining segments of S are stationary, then the expected number of changes in the BSP is $O(kn \log n)$. As far as we know, this is the first nontrivial algorithm for maintaining a BSP for moving segments in the plane.

Next, we study the problem of computing a BSP for a set S of n interior-disjoint triangles in $\mathbb{R}^3$. Paterson and Yao [24] describe a randomized incremental algorithm that constructs a BSP of expected size $O(n^2)$ in time $O(n^3)$. They also show that their algorithm can be made deterministic without affecting its asymptotic running time. It has been an open problem whether a BSP for n triangles in $\mathbb{R}^3$ can be constructed in near-quadratic time. Sub-quadratic bounds are known for special cases [2, 14, 25]. However, none of these approaches lead to a near-quadratic algorithm for triangles in $\mathbb{R}^3$. We present a randomized algorithm (in Section 4) that constructs a BSP for S of expected size $O(n^2)$ in $O(n^2 \log^2 n)$ time.

The bottleneck in analyzing the expected running time of the Paterson-Yao algorithm is that no nontrivial bound is known on the number of vertices in the convex subdivision of $\mathbb{R}^3$ induced by the BSP constructed by the algorithm. Known techniques for analyzing randomized algorithms, such as the Clarkson-Shor framework [13] or backwards analysis [27], cannot be used to obtain a near-quadratic bound on the size of the convex subdivision corresponding to the BSP constructed by the Paterson-Yao algorithm, since the BSP constructed by the algorithm depends on the order in which triangles are added.

Our algorithm is a variant of the Paterson-Yao algorithm. We construct the BSP for S in such a way that there is a close relationship between the BSP and the planar arrangement of lines supporting the edges of the xy-projections of the triangles in S. We use results from ε-net theory [17] and on arrangements of lines [15] to bound the expected number of vertices in the convex subdivision of $\mathbb{R}^3$ induced by the BSP and the expected running time of the algorithm.

Finally, we present a deterministic algorithm (Section 5) for constructing a BSP for a set S of n triangles in $\mathbb{R}^3$. If k is the number of intersection points of the xy-projections of the edges of triangles in S, then the algorithm constructs a BSP of size $O((n+k)\log n)$ in time $O((n+k)\log^2 n)$; if $k \ll n^2$, the deterministic algorithm constructs a much smaller BSP than do Paterson and Yao's and our randomized algorithm. Another nice property of our deterministic algorithm is that the height of the BSP it constructs is $O(\log n)$, which is useful for ray-shooting queries, for example. It was an open problem whether BSPs of near-quadratic size and $O(\log n)$ height could be constructed for n triangles in $\mathbb{R}^3$. The height of the BSP constructed by the randomized algorithms (both ours and the one by Paterson and Yao) can be $\Omega(n)$, e.g., when S is the set of faces of a convex polytope. Due to lack of space, we omit many proofs and details from this abstract.

2 Definitions

A *binary space partition* $\mathcal{B}$ for a set S of convex $(d-1)$-polytopes in $\mathbb{R}^d$ with pairwise-disjoint interiors is a tree defined as follows: Each node v in $\mathcal{B}$ is associated with a convex d-polytope Δ_v and a set of $(d-1)$-polytopes $S_v = \{s \cap \Delta_v \mid s \in S\}$. The polytope 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 Δ_v into two convex polytopes by a *cutting hyperplane* H_v. At v, we store the equation of H_v and the set $\{s \mid s \subseteq H_v, s \in S_v\}$ of polytopes in S_v that lie in H_v. If we let H_v^+ be the positive halfspace and H_v^- be the negative halfspace bounded by H_v, the polytopes associated with the left and right children of v are $\Delta_v \cap H_v^-$ and $\Delta_v \cap H_v^+$, respectively. The left subtree of v is a BSP for $S_v^- = \{s \cap H_v^- \mid s \in S_v\}$ and the right subtree is for $S_v^+ = \{s \cap H_v^+ \mid s \in S_v\}$. The size of $\mathcal{B}$ is the sum of the number of nodes in $\mathcal{B}$ and the total number of polytopes stored at all the nodes in $\mathcal{B}$.

At a node v of $\mathcal{B}$, the cutting hyperplane H_v may support a polytope $s \in S$ such that $H_v \cap \Delta_v \subseteq s$, i.e., the portion of H_v that lies in the interior of Δ_v is contained in s. Such a cutting hyperplane will be referred to as a *free* cut. Free cuts will be critical in keeping the size of $\mathcal{B}$ small by preventing excessive fragmentation of the polytopes in S.

For our purposes, S is either a set of n segments in the plane or a set of n triangles in $\mathbb{R}^3$. A unifying feature of all the BSPs constructed by our algorithms is that the region Δ_v associated with each node v is a *cylindrical* cell in the sense that Δ_v may contain top and bottom faces that are contained in objects belonging to S, but all other faces are vertical. In the plane, Δ_v is a trapezoid; in $\mathbb{R}^3$, Δ_v may have large complexity, as it may contain many vertical faces.

3 Kinetic Algorithm for Segments

Let S be a set of n non-intersecting segments in the plane. We first describe a randomized algorithm for computing a BSP $\mathcal{B}$ for S when the segments in S are stationary, and then explain how to maintain $\mathcal{B}$ as each segment in S moves along a continuous path.

Our algorithm makes two types of cuts: a *vertical* cut through an endpoint of a segment and an *edge* cut along a segment. Edge cuts are always contained totally within input segments; therefore, they are free cuts. Each face in the planar subdivision induced by $\mathcal{B}$ is a trapezoid; the left and right boundaries of a trapezoid are bounded by vertical cuts, and the top and bottom edges are bounded by edge cuts. At an interior node v of $\mathcal{B}$, if Δ_v is split by a vertical cut through an endpoint p, we store p at v; the left (resp., right) subtree of v corresponds to the BSP for the trapezoid lying to the left (resp., to the right) of the cut; if Δ_v is split by an edge cut along a segment s, we store s at v, and the left (resp., right) subtree of v corresponds to the BSP for the trapezoid lying below (resp., above) of the cut.

For a node v in $\mathcal{B}$, the *combinatorial structure* of the trapezoid Δ_v is the 4-tuple $(\lambda_v, \rho_v, \tau_v, \beta_v)$, where λ_v (resp., ρ_v) is the endpoint of a segment in S so that the vertical line passing it contains the left (resp., right) edge of Δ_v, and τ_v (resp., β_v) is the segment in S containing the top (resp., bottom) edge of Δ_v. The *combinatorial structure* of $\mathcal{B}$ is a binary tree, each of whose nodes v is associated with the set of segments S_v. We will use the combinatorial structure of the BSP crucially in our kinetic algorithm.

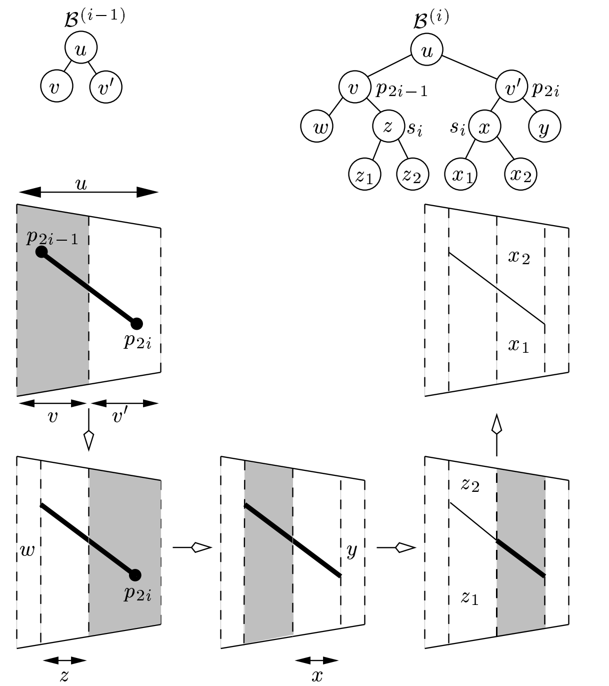

Figure 1: The BSP $\mathcal{B}^{(i-1)}$, the sequence of cuts made in the ith stage, and BSP $\mathcal{B}^{(i)}$. At each step, the shaded trapezoids are split. Portions of s_i that lie in the interior of a trapezoid are drawn using thick lines.

3.1 The static algorithm

We now describe our static algorithm. We choose a random permutation $\langle s_1, s_2, \ldots, s_n\rangle$ of S. We say that s_i has a *higher priority* than s_j if $i < j$. We add the segments in decreasing order of priority and maintain a BSP for the segments added so far. Let $S^{(i)} = \{s_1, s_2, \ldots, s_i\}$ be the set of the first i segments in the permutation. Our algorithm works in stages. At the beginning of the ith stage, where $i > 0$, we have a BSP $\mathcal{B}^{(i-1)}$ for $S^{(i-1)}$; $\mathcal{B}^{(0)}$ consists of a single node v, where Δ_v is the entire plane. In the ith stage, we add s_i and compute a BSP $\mathcal{B}^{(i)}$ for $S^{(i)}$ as follows: Suppose p_{2i-1} and p_{2i} are the left and right endpoints of s_i, respectively. Let Δ_v be the trapezoid containing p_{2i-1} in the planar subdivision induced by $\mathcal{B}^{(i-1)}$. We store p_{2i-1} at v and create two children w and z of v. We partition Δ_v into two trapezoids by drawing a vertical segment through p_{2i-1}; Δ_w and Δ_z are the trapezoids lying to the left and right of the vertical line, respectively. The combinatorial structures of w and z are $(\lambda_v, p_{2i-1}, \tau_v, \beta_v)$ and $(p_{2i-1}, \rho_v, \tau_v, \beta_v)$, respectively. We then perform a similar step for p_{2i}. Finally, for each trapezoid Δ_x that intersects s_i, we split Δ_x into two trapezoids Δ_{x_1} and Δ_{x_2} by making an edge cut along s_i. We store s_i at x, create two children x_1 and x_2 of x, and associate Δ_{x_1} (resp., Δ_{x_2}) with x_1 (resp., x_2). If Δ_{x_1} is above s_i and Δ_{x_2} is below s_i, then the combinatorial structure of x_1 and x_2 are $(\lambda_x, \rho_x, \tau_x, s_i)$ and $(\lambda_x, \rho_x, s_i, \beta_x)$, respectively. The resulting tree is the BSP $\mathcal{B}^{(i)}$ for $S^{(i)}$. See Figure 1 for an example of constructing $\mathcal{B}^{(i)}$ from $\mathcal{B}^{(i-1)}$.

The vertical segment drawn upwards (resp., downwards) from an endpoint p_i will be referred to as the *upper* (resp., *lower*) *thread* of p_i. The segment containing the other endpoint of a thread is called the *stopper* of that thread. Note that the priority of the stopper of a thread of p_i is higher than that of the segment containing p_i.

This completes the description of our algorithm. Note that once we fix the permutation, the algorithm is deterministic and constructs a unique BSP. Using an analysis similar to Paterson and Yao's [24], we can prove the following lemmas:

Lemma 3.1 *Let p be an endpoint of a segment in S. The expected number of segments crossed by the threads of p is $O(\log n)$.*

Theorem 3.2 *The expected size of the BSP constructed by the above algorithm is $O(n \log n)$, and the expected height of the BSP is $O(\log n)$.*

3.2 The kinetic algorithm

We now describe how to maintain the static BSP as the segments in S move continuously. The position of a segment s_i with endpoints p_{2i-1} and p_{2i} can be specified by a point $(x, y, \tan(\theta/2)) \in \mathbb{R}^3$, where (x, y) denotes the position of p_{2i-1} in the plane and θ denotes the angle that the ray along the direction $\overrightarrow{p_{2i-1}p_{2i}}$ makes with the $(+x)$-axis, in the counterclockwise direction. Let $s_i(t)$ denote the segment s_i at time t, and let $S(t)$ denote the set S at time t. We assume that we choose a random permutation π of S once in the very beginning (at $t = 0$), and that π does not change with time. Let $\mathcal{B}(t)$ denote the BSP of $S(t)$ constructed by the static algorithm, using π as the permutation to decide the priority of the segments. We describe an algorithm that updates the BSP under the following assumption:

($\star$) *There is no correlation between the motion of the segments in S and their priorities. Therefore, the chosen permutation π always behaves like a random permutation, and Lemma 3.1 and Theorem 3.2 hold at all times.*

We parameterize the motion of the segments by time and use t to denote time. For a given time instant t, we will use t^- and t^+ to denote the time instants $t-\varepsilon$ and $t+\varepsilon$, respectively, where $\varepsilon > 0$ is an arbitrarily small constant. As the segments in S move continuously, the equations of the cuts associated with the nodes of $\mathcal{B}$ also change. At the same time, the edges and vertices of the trapezoids in the subdivision of the plane induced by $\mathcal{B}$ also move. However, the combinatorial structure of a trapezoid is unchanged until two edges of the subdivision collide, at which point a trapezoid Δ_v, where v is a node in $\mathcal{B}$, shrinks to a vertical segment. Since the segments of S are not allowed to intersect, the top and bottom edges of Δ_v cannot meet, so the combinatorial structure of Δ_v can change only when the left and right edges of Δ_v become identical. Similarly, the combinatorial structure of $\mathcal{B}$ changes when the set S_v changes for some node $v \in \mathcal{B}$. Since the segments in S are disjoint, the set S_v changes when the endpoint of a segment in S_v lies on the left or right edge of Δ_v. Then we can show that there is a node $w \in \mathcal{B}$ such that Δ_w shrinks to a vertical segment.

A trapezoid Δ_v shrinks to a segment only when the x-coordinates of λ_v and ρ_v become equal, although not all such instances change the combinatorial structure of $\mathcal{B}$. At each instant t when a trapezoid Δ_v shrinks to a segment, either λ_v or ρ_v moves from a trapezoid Δ_w (at t^-) to an adjacent trapezoid Δ_x (at t^+), causing a change in the set of segments S_w or S_x. Hence, the combinatorial structure of $\mathcal{B}(t)$ also changes at t. Let $\mathcal{B}(t^-)$ and $\mathcal{B}(t^+)$ denote the trees at t^- and t^+, respectively. We would like to ensure that the edit distance between $\mathcal{B}(t^-)$ and $\mathcal{B}(t^+)$ is small,[1] and that $\mathcal{B}(t^+)$ can be obtained from $\mathcal{B}(t^-)$ in time proportional to the edit distance between them.

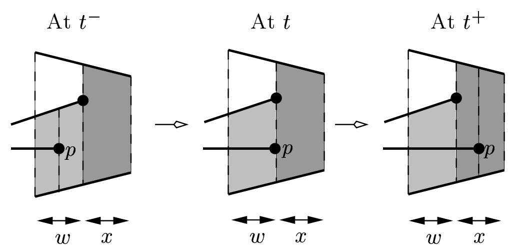

Figure 2: Endpoint p moves from Δ_w at t^- to Δ_x at t^+. The set S_x changes at time instant t.

A node v of $\mathcal{B}(t)$ is called *transient* if Δ_v does not contain any endpoint in its interior and a vertical cut was made at the parent $p(v)$ of v; Δ_v is called a transient trapezoid. See Figure 3. Using the above assumption and Lemma 3.1, we can prove that transient nodes have the following properties:

1. No proper ancestor or proper descendant of a transient node is a transient node.

2. The number of transient nodes in $\mathcal{B}(t)$ is $O(n)$.

3. Only edge cuts are made at the descendants of a transient node v (including v itself). The left and right edges of the trapezoids associated with all the descendants of v are portions of the left and right edges of Δ_v.

4. The expected number of descendants of a transient node is $O(\log n)$.

[1]The *edit distance* between $\mathcal{B}(t_1)$ and $\mathcal{B}(t_2)$ is the minimum number of insertions of nodes, deletions of nodes, and pointer changes required to obtain $\mathcal{B}(t_2)$ from $\mathcal{B}(t_1)$.

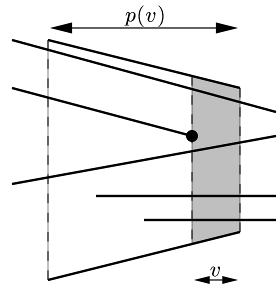

Figure 3: The shaded trapezoid Δ_v is transient.

Recall that for a node v in $\mathcal{B}(t)$, λ_v (resp., ρ_v) denotes the endpoint whose thread contains the left (resp., right) boundary of Δ_v. The following lemma characterizes the time instances t at which the combinatorial structure of $\mathcal{B}$ changes. Let $\Gamma(t) = \{(\lambda_v, \rho_v) \mid v \text{ is a transient node at time } t\}$; the properties of transient nodes imply that $|\Gamma(t)| = O(n)$.

Lemma 3.3 *For any time instant t, $\mathcal{B}(t^-)$ and $\mathcal{B}(t^+)$ have different combinatorial structures if and only if there exists a transient node v in $\mathcal{B}(t^-)$ such that the x-coordinates of the endpoints λ_v and ρ_v become equal at time t.*

Intuitively, transient nodes are the highest nodes in $\mathcal{B}(t)$ whose combinatorial structure can change next. If a trapezoid contains an endpoint in its interior, it cannot shrink to a segment; and if an edge cut is made at the parent $p(v)$ of a node v, then $\Delta_{p(v)}$ also shrinks to a segment whenever Δ_v shrinks to a segment and $\Delta_{p(v)}$ does not contain an endpoint. Hence, it suffices to keep track of transient nodes to determine all the combinatorial changes in $\mathcal{B}(t)$. To this end, we maintain the set $\Gamma(t)$. For each pair (λ_v, ρ_v) in $\Gamma(t)$, we compute the time at which the x-coordinates of λ_v and ρ_v coincide, and store these time values in a global priority queue. We refer to these values of time as *critical events*.

We will prove that if the combinatorial structure of $\mathcal{B}$ changes at time t, then we can obtain $\mathcal{B}(t^+)$ from $\mathcal{B}(t^-)$ in $O(\log n)$ expected time. We will also show that at each event point, the expected number of changes in the global event queue is $O(\log n)$.

In order to expedite the updating of $\mathcal{B}$, we store some additional information with the nodes in $\mathcal{B}$: At each node v of $\mathcal{B}$, we store the number c_v of endpoints lying in the interior of Δ_v (c_v helps determine the new transient trapezoids at t^+); and for each endpoint p_j, we maintain the list T_j (resp., B_j) of segments that its upper (resp., lower) thread crosses; T_j (resp., B_j) is sorted in the $(+y)$-direction (resp., $(-y)$-direction). As the endpoints move, these lists will be used to compute new stoppers of threads. Although we do not really need these lists, they simplify the description of the update procedure without affecting its expected running time.

We now describe the procedure for updating the tree at each critical event. Recall that at each such instant t, an endpoint p of a segment in S moves from a trapezoid Δ_w (at t^-) to an adjacent trapezoid Δ_x (at t^+). See Figure 2. At time t^-, the vertical cut made through p divides Δ_w into two trapezoids. At time t, one of these two trapezoids shrinks to a segment, and at t^+, a new trapezoid appears inside Δ_x. These changes inside Δ_w and Δ_x cause combinatorial changes in other trapezoids. We exemplify these ideas below. Let $\mathcal{B}^- = \mathcal{B}(t^-)$ and $\mathcal{B}^+ = \mathcal{B}(t^+)$. For a node $z \in \mathcal{B}^-$, let $\mathcal{B}_z^-$ denote the subtree of $\mathcal{B}^-$ rooted at z; define $\mathcal{B}_z^+$ similarly.

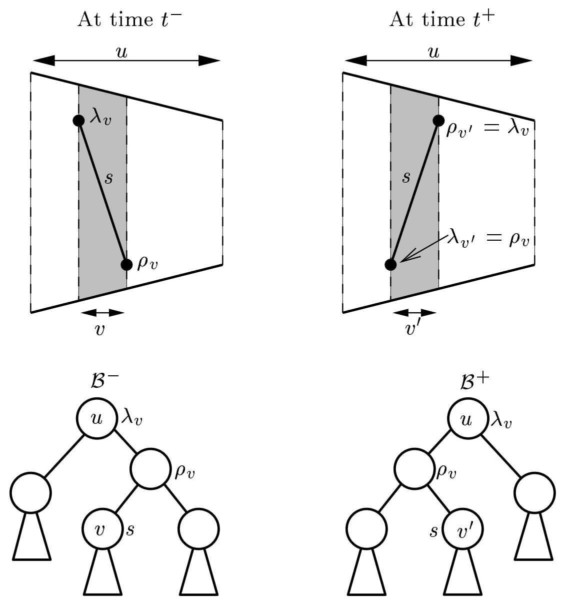

Figure 4: The case when λ_v and ρ_v belong to the same segment.

Suppose v is a transient node in at time $\mathcal{B}^-$. Hence, the x-coordinates of the endpoints λ_v and ρ_v become equal at time t. Suppose λ_v and ρ_v are endpoints of the same segment $s \in S$; s is vertical at time t. See Figure 4. Let u be v's grandparent in $\mathcal{B}^-$; the trapezoid Δ_u contains s. At time t^+, a new trapezoid v' with combinatorial structure $(\rho_v, \lambda_v, \tau_v, \beta_v)$ appears inside u. We obtain $\mathcal{B}^+$ by simply swapping the left and right subtrees of u in $\mathcal{B}^-$ and replacing v with v'.

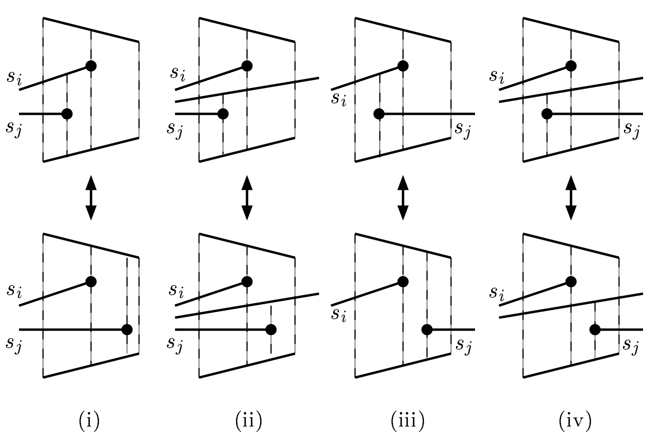

Figure 5: Some possible changes in the combinatorial structure of the trapezoids in the subdivision induced by the BSP.

Now suppose λ_v and ρ_v are endpoints of different segments. Figure 5 shows some of the ways in which the combinatorial structure of trapezoids can change at a critical event; the other cases can be reduced to these cases by taking a reflection with respect to one of the two axes, by going backward in time, or by doing both. Assume that $\rho_v = p_{2i}$ and $\lambda_v = p_{2j}$ are the right endpoints of the segments s_i and s_j, respectively, that s_i lies above s_j, that the priority of s_i is higher than that of s_j, and that the x-coordinate of p_{2j} is less than the x-coordinate of p_{2i} at t^-; see Figures 5(i) and 5(ii). We now describe how we update $\mathcal{B}(t)$ for this case; we omit the other cases from this abstract, since they can be handled in a similar manner. Let u and w be the nodes in $\mathcal{B}^-$ at which the vertical cuts through p_{2i} and p_{2j}, respectively, were made. Then, by our assumptions, v is the right child of w, and w lies in the left subtree of u. Let u_L be the left child of u, and let w_L be the left child of w. The segments τ_v and β_v supporting the top and bottom edges of Δ_v, respectively, are the stoppers of the top and bottom threads of p_{2j}; obviously, τ_v does not occur earlier than s_i in the random permutation π used to construct $\mathcal{B}$, and τ_v does not lie above s_i. There are two cases to consider depending on whether $\tau_v = s_i$.

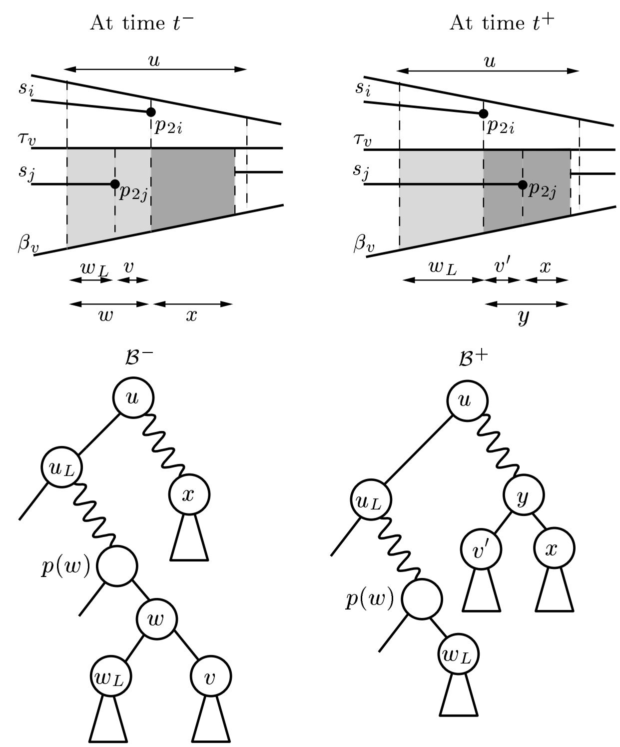

Figure 6: The case $\tau_v \neq s_i$: arrows on the bottom mark the horizontal extents of the shaded trapezoids.

Case (i): $\tau_v \neq s_i$. See Figure 6. Let x be the highest node in the right subtree of u (in $\mathcal{B}^-$) with the combinatorial structure $(p_{2i}, p_r, \tau_v, \beta_v)$, for some endpoint p_r. Then Δ_w and Δ_x share a common edge in $\mathcal{B}(t^-)$. At time t^+, as p_{2j} leaves the trapezoid Δ_w and enters Δ_x, Δ_{w_L} expands to Δ_w, Δ_v disappears, and a new trapezoid $\Delta_{v'} = (p_{2i}, p_{2j}, \tau_v, \beta_v)$ appears (i..e, Δ_x is split at t^+ into two trapezoids: $\Delta_{v'}$ and the portion of Δ_x lying to the right of the cut through p_{2j}.) Moreover, $\Delta_{v'}$ (at time t^+) is intersected by s_j and by the set of segments intersecting Δ_v (at time t^-). At time t^-, Δ_w is split by a vertical cut through p_{2j} and Δ_{w_L} is split by an edge cut along s_j, while at time t^+, Δ_w is split by an edge cut along s_j. Therefore $\mathcal{B}_w^+$ is the same as $\mathcal{B}_{w_L}^-$.

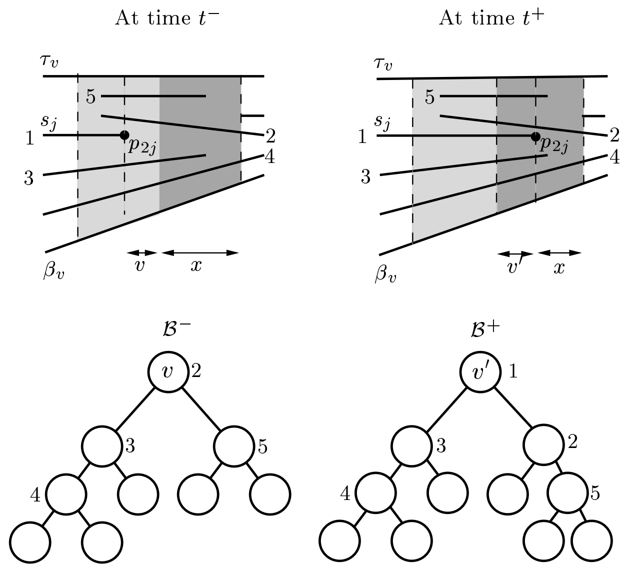

Figure 7: The edge cuts made in $\mathcal{B}_v^-$ and $\mathcal{B}_{v'}^+$. Segments crossing Δ_v and $\Delta_{v'}$ are labelled with their priorities. The label next to a node in $\mathcal{B}^-$ and $\mathcal{B}^+$ is the priority of the segment containing the edge cut made at that node.

To obtain $\mathcal{B}^+$, we delete the node w from $\mathcal{B}^-$, and if w was a left (resp., right) child of its parent, we make w_L the new left (resp., right) child of $p(w)$. We then construct $\mathcal{B}_{v'}^+$ by making edge cuts through the segments intersecting v' in decreasing order of priority (see Figure 7). and attach it to a descendant of the right child of u as follows: We create a node y with combinatorial structure $(p_{2i}, p_r, \tau_v, \beta_v)$. The cut associated with y is the vertical cut through p_{2j} (with τ_v and β_v as its stoppers). The left and right subtrees of y in $\mathcal{B}^+$ are $\mathcal{B}_{v'}^+$ and $\mathcal{B}_x^-$, respectively. If x is the left (resp. right) child of its parent $p(x)$ in $\mathcal{B}^-$, then y is the new left (resp. right) child of $p(x)$.

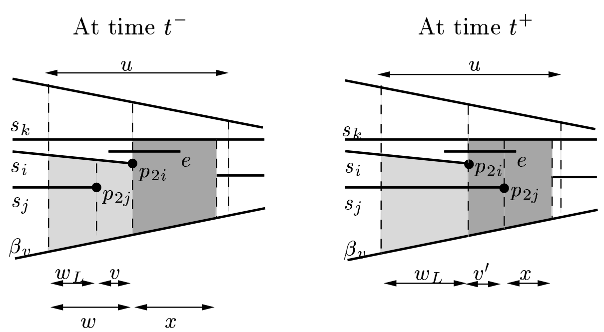

Figure 8: The case $\tau_v = s_i$: arrows on the bottom mark the horizontal extents of the shaded trapezoids. The segment e is intersected by T_{2j} at t^+ but not at t^-.

Case (ii): $\tau_v = s_i$. See Figure 8. In this case, at time t, the stopper of the upper thread of p_{2j} switches from s_i to a segment lying above s_i. The new stopper is the first segment s_k in T_{2i}, the set of segments intersected by the top thread of p_{2i}, whose priority is higher than that of s_j. All segments that appear before s_k in T_{2i} are crossed by the top thread of p_{2j} at t^+. The node x now corresponds to the highest node in the right subtree of u (in $\mathcal{B}^-$) whose combinatorial structure is $(p_{2i}, p_r, s_k, \beta_v)$, for some endpoint p_r. The new trapezoid $\Delta_{v'}$ that appears at t^+ has combinatorial structure $(p_{2i}, p_{2j}, s_k, \beta_v)$. The set $S_{v'}$ of segments crossing $\Delta_{v'}$ at t^+ is s_j, the set of segments in T_{2j} and the segments in T_{2i} stored before s_k at t^-.

We first find s_k by traversing the list T_{2i}. $\mathcal{B}^+$ is constructed in the same way as in the previous case. We again construct the subtree $\mathcal{B}_{v'}^+$ by adding the segments in $S_{v'}$ in decreasing order of their priority.

Finally, in both cases, we insert s_j into B_{2i}, the list of segments in S crossed by the lower thread of p_{2i}, and update T_{2j}. For a node z, c_z, the number of endpoints lying in the interior of Δ_z, changes only if z lies along the paths from u to the nodes $p(w)$ and y. For such a node z, if $c_z = 0$ and if $\Delta_{p(z)}$ is split by a vertical cut, we add (λ_z, ρ_z) to the list $\Gamma(t^+)$. On the other hand, if $c_z \neq 0$ but z is transient at t^- (z must be an ancestor of x in $\mathcal{B}^-$), we delete (λ_v, ρ_z) from $\Gamma(t^+)$. Using Lemma 3.1 and assumption ($\star$), we can show that the expected time spent in these steps is $O(\log n)$. We thus obtain the main result of this section:

Theorem 3.4 *At each event point, $\mathcal{B}(t)$ can be updated in $O(\log n)$ expected time.*

Note that this theorem makes our BSP a kinetic data structure that is responsive, efficient, local, and compact, in the sense defined by Basch et al. [6].

We say that the trajectories followed by a set of segments are *pseudo-algebraic* if the segments move so that each pair of endpoints exchanges y-order only $O(1)$ times. A special case of pseudo-algebraic trajectories is when all the trajectories of the endpoints are constant-degree polynomials. If the trajectories of k of the segments in S are pseudo-algebraic and the remaining segments are stationary, then the total number of event points is $O(kn)$. We spend $O(\log n)$ expected time to maintain $\mathcal{B}(t)$ at each event point. Hence, we obtain the following corollary to Theorem 3.4:

Corollary 3.5 *Let S be a set of n segments in the plane, and let $G \subseteq S$ be a set of k segments. Suppose each segment of G moves along a pseudo-algebraic trajectory and the remaining segments of S are stationary, the total expected time spent in maintaining $\mathcal{B}$ is $O(kn \log n)$.*

4 BSPs for Triangles: A Randomized Algorithm

In this section we describe a randomized algorithm for constructing a BSP $\mathcal{B}$ of expected size $O(n^2)$ for a set of n triangles in $\mathbb{R}^3$. The expected running time of the algorithm is $O(n^2 \log^2 n)$. We describe the algorithm in Section 4.1 and analyze its performance in Section 4.2.

4.1 Algorithm

For an object s in $\mathbb{R}^3$, let s^* denote the xy-projection of s. Let E be the set of edges of the triangles in S, and let E^* denote the set $\{e^* \mid e \in E\}$. Let L be the set of lines in the xy-plane supporting the edges in E^*. We choose a random permutation $\langle \ell_1, \ell_2, \ldots, \ell_{3n} \rangle$ of L, and add the lines one-by-one in this order to compute $\mathcal{B}$. Let $L^{(i)} = \{\ell_1, \ell_2, \ldots \ell_i\}$. The algorithm works in $3n$ stages. The ith stage adds ℓ_i and constructs a top subtree $\mathcal{B}^{(i)}$ of $\mathcal{B}$ by refining the leaves of $\mathcal{B}^{(i-1)}$; $\mathcal{B}^{(0)}$ consists of one node (corresponding to $\mathbb{R}^3$) and $\mathcal{B}^{(3n)}$ is $\mathcal{B}$. As usual, we have a convex polytope Δ_v

associated with each node v of $\mathcal{B}^{(i)}$; Δ_v is a cylindrical cell, bounded by a set of vertical faces (i.e., faces parallel to the z-axis) and, possibly, top and bottom faces. If the top or bottom face of Δ_v exists, it is contained in a triangle of S. If v is a leaf of $\mathcal{B}^{(i)}$ and no triangle in S intersects the interior of Δ_v, i.e., $S_v = \emptyset$, then v is a leaf of $\mathcal{B}$ and we do not refine it further. Otherwise, we partition Δ_v into two cylindrical cells; these two cells are leaves of $\mathcal{B}^{(i+1)}$.

Before describing the ith stage of the algorithm in detail, we explain the structure of $\mathcal{B}^{(i)}$. We need a few definitions first. We say that a leaf v of $\mathcal{B}^{(i)}$ (or the cell Δ_v) is *active* if a triangle in S intersects the interior of Δ_v (i.e. $S_v \neq \emptyset$); similarly, we say that a face f in the line arrangement $\mathcal{A}(L^{(i)})$ is *active* if the xy-projection of some edge of a triangle in S intersects the interior of f. For each active leaf v in $\mathcal{B}^{(i)}$, Δ_v satisfies the following properties:

(P1) If a triangle $s \in S$ intersects the interior of Δ_v, then the boundary of s also intersects the interior of Δ_v.

(P2) The cell Δ_v is a vertical section of the cylinder $\{(p,z) \mid p \in f, z \in \mathbb{R}\}$, for exactly one active face f of $\mathcal{A}(L^{(i)})$; the vertical section may be truncated by triangles of S at the top and bottom. See Figure 9.

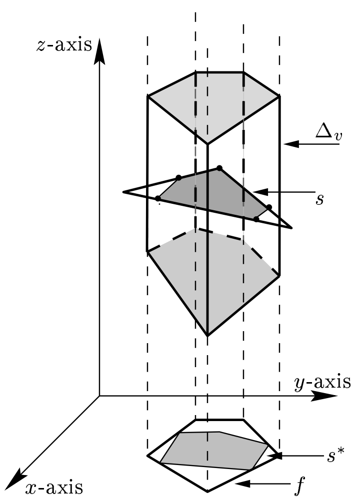

Figure 9: An active face f and an active cell $\Delta_v \in \mathbf{\Delta}(f)$ that is a vertical section of the cylinder erected on f. The boundary of triangle s intersects Δ_v and the boundary of s^* intersects f.

In order to execute each stage efficiently, we maintain the following additional information:

(i) For each active cell Δ in $\mathcal{B}^{(i)}$, we store the set $S_\Delta \subseteq S$ of triangles that intersect the interior of Δ.

(ii) We maintain the arrangement $\mathcal{A}(L^{(i)})$ as a planar graph; see [15]. For each active face f in $\mathcal{A}(L^{(i)})$, we maintain the list $\mathbf{\Delta}(f)$ of those active cells Δ in $\mathcal{B}^{(i)}$ that lie inside the cylinder $\{(p,z) \mid p \in f, z \in \mathbb{R}^3\}$. Note that by Properties P(1) and P(2), a face $f \in \mathcal{A}(L^{(i)})$ is active if and only if $\mathbf{\Delta}(f) \neq \emptyset$.

In the ith stage, we make a cut along the vertical plane supporting ℓ_i, and then make cuts contained in triangles of S as follows: Let h_i be the vertical plane supporting ℓ_i and let h_i^+ (resp., h_i^-) be the positive (resp., negative) halfspace supported by h_i.

1. We trace ℓ_i through the faces of $\mathcal{A}(L^{(i-1)})$. For each face $f \in \mathcal{A}(L^{(i-1)})$ intersected by ℓ_i, we use ℓ_i to split f into two faces f^+ and f^-. Let Δ be an active cell in $\mathbf{\Delta}(f)$. We partition Δ into two cells $\Delta^+ = \Delta \cap h_i^+$ and $\Delta^- = \Delta \cap h_i^-$.

2. We then compute the set $S_{\Delta^+} \subseteq S_\Delta$ of triangles that intersect the interior of Δ^+. We also compute the set $F_{\Delta^+} \subseteq S_{\Delta^+}$ of triangles whose boundaries do not cross Δ^+. Similarly, we compute the sets S_{Δ^-} and F_{Δ^-} for Δ^-.

3. We split Δ^+ into a set Ψ of $|F_{\Delta^+}| + 1$ cells by making free cuts along each of the triangles in F_{Δ^+}. For each triangle $s \in S_{\Delta^+} \setminus F_{\Delta^+}$, we perform a binary search to determine the unique cell in $\Delta' \in \Psi$ that intersects s and add s to $S_{\Delta'}$. (Δ' is unique because the triangles in S are pairwise disjoint.) Finally, for each cell $\Delta' \in \Psi$, we add Δ' to the set $\mathbf{\Delta}(f^+)$ if $S_{\Delta'} \neq \emptyset$. Next, we perform a similar procedure for Δ^-.

The resulting tree is $\mathcal{B}^{(i)}$. It is easily seen that $\mathcal{B}^{(i)}$ satisfies properties (P1) and (P2). Note that after the three lines supporting the xy-projections of the edges of a triangles $s \in S$ have been processed, s does not intersect the interior of any cell.

Remark: The free cuts made in Step 3 are crucial in keeping the size of the BSP quadratic. Instead, if we simply erect vertical planes as we do in the algorithm, and make cuts along a triangle $s \in S$ only when all three lines supporting the xy-projections of s's edges have been added, then there are instances of input triangles for which our algorithm will construct a BSP of $\Omega(n^3)$ size regardless of the initial random permutation.

4.2 Analysis of the algorithm

We first bound the expected size of $\mathcal{B}$. The cuts made in the algorithm partition each triangle in S into a number of pieces. We can show that the size of $\mathcal{B}$ is boundedby the total number of pieces into which the triangles in S are split by the cuts made in the algorithm. To bound the expected total number of pieces, we count the number of new pieces created in the ith stage, and sum the result over all stages. In the ith stage, we count the number of new pieces into which a triangle s in S is partitioned by the cuts made in the ith stage, and sum the resulting bound over all triangles in S.

Let ν_s be the total number of new pieces into which a triangle $s \in S$ is partitioned by the cuts made in the ith stage. Note that these cuts are contained in the vertical plane containing ℓ_i. For $1 \le k \le i$, let λ_k be the intersection of the vertical plane through ℓ_k with the triangle s, and let $\Lambda = \{\lambda_k, 1 \le k \le i\}$. To calculate ν_s, consider the line arrangement $\mathcal{A}(\Lambda)$ on s. We call a face of $\mathcal{A}(\Lambda)$ a *boundary* face if it is adjacent to an edge of s; otherwise, it is an *interior* face. Recall that we partition a cell Δ_v, for a leaf $v \in \mathcal{B}^{(i)}$, only if Δ_v is active. Property (P1) implies that the cuts made in the ith stage do not intersect the interior of any interior face of $\mathcal{A}(\Lambda)$, since such a face cannot intersect the interior of any active cell Δ_v. Hence, ν_s is the number of boundary faces of $\mathcal{A}(\Lambda)$ that are intersected by λ_i.

For $1 \leq k \leq i$, let $\mu(\Lambda, k)$ denote the number of boundary faces in the arrangement $\mathcal{A}(\Lambda \setminus \{\lambda_k\})$ that are intersected by λ_k. Observe that the sum $\sum_{1 \leq k \leq i} \mu(\Lambda, k)$ equals the total number of edges bounding the boundary faces of $\mathcal{A}(\Lambda)$. Each such edge lies in the zone (in $\mathcal{A}(\Lambda)$) of one of the edges of s. Hence, by the Zone Theorem [9, 15],

$$\sum_{1 \leq k \leq i} \mu(\mathcal{A}(\Lambda), k) = O(i).$$

Since ℓ_i is chosen randomly from the set $L^{(i)}$, λ_i can be any of the lines $\lambda_1, \lambda_2, \ldots, \lambda_i$ with equal probability. Therefore, the expected value $E\left[\nu_s\right]$ of ν_s is

$$E\left[\nu_s\right] = \frac{1}{i} \sum_{1 \leq k \leq i} \mu(\mathcal{A}(\Lambda), k) = O(1).$$

Hence, the total number of pieces created in the ith stage is $O(n)$. Summing over i, we find that the total number of pieces into which the triangles in S are partitioned into over the entire algorithm is $O(n^2)$. The following lemma is immediate:

Lemma 4.1 *The expected number of nodes in the BSP constructed by the above algorithm is $O(n^2)$.*

For an active face f in $\mathcal{A}(L^{(i-1)})$, let k_f be the number of projected edges in E^* that intersect the interior of f. By Property 1, if a triangle $s \in S$ intersects the interior of a cell $\Delta \in \boldsymbol{\Delta}(f)$, then the boundary of s also intersects the interior of Δ. Therefore, an edge of s^* intersects the interior of f. Further, s cannot intersect the interior of two different cells in $\boldsymbol{\Delta}(f)$; otherwise, s intersects a non-vertical face g of one of those cells, which is impossible, since each such face g is contained in a triangle in S and the triangles in S are pairwise disjoint. As a result, we have

$$\sum_{\Delta \in \boldsymbol{\Delta}(f)} |S_\Delta| \leq k_f. \tag{4.1}$$

We now analyze the expected running time of the algorithm. We first count the time spent during the ith stage in processing the line ℓ_i. In Step 1, we spend $O(i)$ time to trace ℓ_i through $\mathcal{A}(L^{(i-1)})$. When processing an active face f of $\mathcal{A}(L^{(i-1)})$ that is intersected by ℓ_i, for each cell $\Delta \in \boldsymbol{\Delta}(f)$, we spend $|S_\Delta|$ time in Step 2. In Step 3, we can find the cell in the set Ψ that intersects a triangle $s \in S_{\Delta^+} \setminus F_{\Delta^+}$ in $O(\log |F_{\Delta^+}|)$ time by performing a binary search. Hence, the total time spent in Step 3 for the face f is $O(|S_\Delta| \log |S_\Delta|)$. Thus, (4.1) implies that the total time spent in processing f is

$$\sum_{\Delta \in \boldsymbol{\Delta}(f)} O(|S_\Delta| \log |S_\Delta|) = O(k_f \log k_f).$$

Let Z be the set of all active faces of $\mathcal{A}(L^{(i-1)})$ that are intersected by ℓ_i. The total time spent in processing ℓ_i is $\sum_{f \in Z} O(k_f \log k_f)$. Using the Zone Theorem [9, 15] and the theory of random-sampling [13, 17], we can show that the expected value of $\sum_{f \in Z} k_f$ is $O(n \log n)$, which implies the following theorem:

Theorem 4.2 *Let S be a set of n non-intersecting triangles in $\mathbb{R}^3$. We can compute a BSP for S of expected size $O(n^2)$ in expected time $O(n^2 \log^2 n)$ time.*

Remark: Using a similar argument, we can also prove that the expected value of the total number of vertices of the nodes of $\mathcal{B}$ is $O(n^2)$. The height of $\mathcal{B}$ can be $\Omega(n)$, e.g., if the triangles in S form a convex polytope.

5 BSPs for Triangles: A Deterministic Algorithm

In this section we describe a deterministic algorithm for computing a BSP for a set S of n triangles in $\mathbb{R}^3$. As in the previous section, let E denote the set of edges of triangles in S, and let $E^* = \{e^* \mid e \in E\}$ be the set of xy-projections of the edges in E. Let k be the number of intersection points in E^*. Our algorithm constructs in $O((n+k)\log^2 n)$ time a BSP $\mathcal{B}$ of size $O((n+k)\log n)$. As in the previous section, each node v of $\mathcal{B}$ is associated with a cylindrical cell Δ_v, but the top and bottom faces of Δ_v are now trapezoids. Let Δ_v^* denote the xy-projection of the top (or bottom) face of Δ_v; two of the edges of Δ_v^* will be parallel to the y-axis. Let E_v^* be the set of segments in E^* that intersect the interior of Δ_v^* and are clipped within Δ_v^*. A segment $\gamma \in E_v^*$ is called *anchored* if its endpoints lie on the two parallel edges of Δ_v^* and if it does not intersect any other edge of E_v^*. The anchored edges in E_v^* can be linearly ordered from top to bottom (since they are disjoint). Let A_v be the set of anchored edges in E_v^*. Let $F_v \subseteq S_v$ the set of all free triangles in S_v. Recall that a triangle $s \in S_v$ is free with respect to Δ_v if no edge s intersects the interior of Δ_v. Since Δ_v is a cylindrical cell, the triangles in F_v can be sorted by their heights.

The algorithm constructs $\mathcal{B}$ in a top-down fashion by maintaining a top subtree of $\mathcal{B}$. A leaf v of the subtree is *active* if $S_v \neq \emptyset$. We store the set of all active leaves of the current subtree in a list. For each active leaf v, we maintain three sets: F_v, A_v, and S_v. Note that the set E_v^* can be easily computed from S_v. At each step, we choose an active leaf v and compute the cutting plane h_v that is used to split Δ_v into two cells Δ_z and Δ_w. Once h_v is chosen, we can compute Δ_z and Δ_w and the sets associated with them in $O(|S_v| + |F_v|)$ time. If S_w (resp., S_z) is nonempty, we mark w (resp., z) as being active.

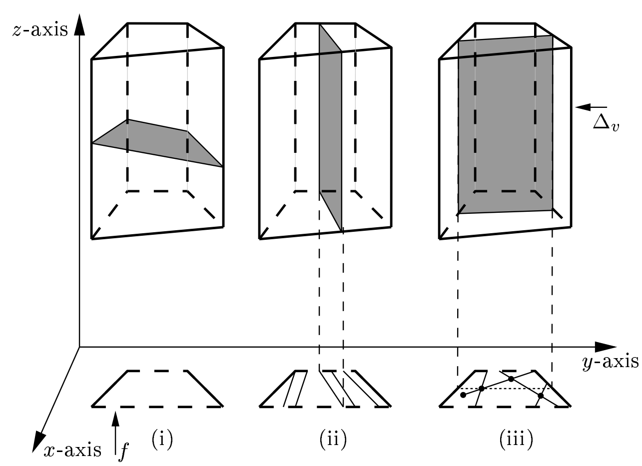

Figure 10: Cuts made in the deterministic algorithm: (i) Free cut, (ii) Cut parallel to the z-axis through an anchored edge, and (iii) Cut parallel to the yz-plane through a vertex of $\mathcal{A}(E_v^*)$

We choose the cutting plane h_v as follows:

1. If $F_v \neq \emptyset$, we choose the median triangle s of F_v and set h_v to be the plane supporting s. See Figure 10(i). Recall that the triangles in F_v are ordered by height. Since s does not intersect any triangle of S_v, each triangle of $S_v \setminus \{s\}$ belongs to either S_w or S_z. Moreover, $F_w, F_z \subseteq F_v$ and $A_w, A_z \subseteq A_v$.

2. If $F_v = \emptyset$, i.e., there is no free triangle in S_v, we use A_v to choose the cutting plane, as follows: If A_v contains at least one anchored segment, let γ be the middle anchored segment. Since the anchored segments can be totally ordered, the middle anchored segment is well defined. We choose the cutting plane h_v to be the vertical plane containing γ. See Figure 10(ii). Since h_v may intersect some triangles of S_v, a triangle of S_v may belong to both S_w and S_z. If a triangle $s \in S_v$ intersects h_v and if none of its edges intersect the interior of Δ_w (resp., Δ_z), then $s \in F_w$ (resp., $s \in F_z$).

3. Finally, if A_v is empty, let σ be a vertex in the arrangement $\mathcal{A}(E_v^*)$ with the median y-coordinate; σ is either an endpoint of a segment of E_v^* or an intersection point of two segments of E_v^*. We choose h_v to be the plane $x = x(\sigma)$, i.e., h_v is the plane parallel to the yz-plane and passing through σ. See Figure 10(iii). In this case too, h_v can intersect the triangles of S_v. If k_v is the number of intersection points of E_v^*, σ can be computed in $O(|S_v| + k_v)$ time, after some initial preprocessing. We can again compute the required sets for w and z in $O(|S_v|)$ time.

This completes the description of the algorithm. We now give the analysis. At each node v, since we choose the middle free triangle, middle anchored edge, or a middle vertex of $\mathcal{A}(E_v^*)$, we can show that the height of $\mathcal{B}$ is $O(\log^2 n)$. However, if we assign appropriate weights to each free triangle in F_v and to each anchored edge and choose a weighted median of the free triangles or the anchored edges, we can improve the height of $\mathcal{B}$ to $O(\log n)$; for example, see [24]. Following an analysis similar to the one given in [24], we can show that the number of nodes in $\mathcal{B}$ is $O((n+k)\log n)$, and that the running time of the algorithm is $O((n+k)\log^2 n)$. Hence, we obtain the following result:

Theorem 5.1 *Let S be a set of n triangles in the plane, and let k be the number of intersection points of the xy-projections of the edges of S. Then we can compute in $O((n+k)\log^2 n)$ time a BSP of size $O((n+k)\log n)$ for S.*

6 Conclusions

We have presented an efficient algorithm to maintain the BSP of moving segments in the plane. Currently, we do not know any non-trivial lower bounds for this problem. Recently, Agarwal et al. [1] have extended our result and developed an algorithm to maintain BSPs for moving triangles in $\mathbb{R}^3$.

We have also presented algorithms to construct BSPs for triangles in $\mathbb{R}^3$. The algorithms are (near-)optimal in the worst-case. No efficient algorithm is known for constructing a BSP of optimal or near-optimal size for triangles in $\mathbb{R}^3$. On the other hand, it is not known whether the problem is NP-hard.

References

[1] P. K. Agarwal, J. Erickson, and L. J. Guibas, Kinetic binary space partitions in $\mathbb{R}^3$, In preparation, 1997.

[2] 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.

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

[4] 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.

[5] C. Ballieux, Motion planning using binary space partitions, Tech. Rep. inf/src/93-25, Utrecht University, 1993.

[6] J. Basch, L. Guibas, and J. Hershberger, Data structures for mobile data, *Proc. 7th SIAM Symp. on Discr. Algorithms*, 1997, pp. 747–756.

[7] A. T. Campbell, *Modeling Global Diffuse Illumination for Image Synthesis*, Ph.D. Thesis, Dept. of Computer Sciences, University of Texas, Austin, 1991.

[8] T. Cassen, K. R. Subramanian, and Z. Michalewicz, Near-optimal construction of partitioning trees by evolutionary techniques, *Proc. of Graphics Interface '95*, 1995, pp. 263–271.

[9] B. Chazelle, L. J. Guibas, and D. T. Lee, The power of geometric duality, *BIT*, 25 (1985), 76–90.

[10] N. Chin and S. Feiner, Near real-time shadow generation using BSP trees, *Proc. SIGGRAPH '89*, *Comput. Graph.*, Vol. 23, ACM SIGGRAPH, 1989, pp. 99–106.

[11] N. Chin and S. Feiner, Fast object-precision shadow generation for areal light sources using BSP trees, *Comput. Graph. (Proc. 1992 Symp. on Interactive 3D Graphics)*, Vol. 25, 1992, pp. 21–30.

[12] Y. Chrysanthou, *Shadow Computation for 3D Interaction and Animation*, Ph.D. Thesis, Queen Mary and Westfield College, University of London, 1996.

[13] K. L. Clarkson and P. W. Shor, Applications of random sampling in computational geometry, II, *Discrete Comput. Geom.*, 4 (1989), 387–421.

[14] M. de Berg, Linear size binary space partitions for fat objects, *Proc. Third Europ. Symp. on Algorithms*, LNCS 979, Springer-Verlag, September 1995, pp. 252–263.

[15] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer-Verlag, Heidelberg, West Germany, 1987.

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

[17] D. Haussler and E. Welzl, Epsilon-nets and simplex range queries, *Discrete Comput. Geom.*, 2 (1987), 127–151.

[18] C. Mata and J. S. Mitchell, Approximation algorithms for geometric tour and network design problems, *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, 1995, pp. 360–369.

[19] T. M. Murali and T. A. Funkhouser, Consistent solid and boundary representations from arbitrary polygonal data, *To appear in Proc. 1997 Symp. on Interactive 3D Graphics*, 1997.

[20] B. Naylor, J. A. Amanatides, and W. Thibault, Merging BSP trees yields polyhedral set operations, *Comput. Graph.*, 24 (1990), 115–124. Proc. SIGGRAPH '90.

[21] 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, 1986.

[22] B. F. Naylor, SCULPT: an interactive solid modeling tool, *Proc. Graphics Interface '90*, 1990, pp. 138–148.

[23] B. F. Naylor, Interactive solid geometry via partitioning trees, *Proc. Graphics Interface '92*, 1992, pp. 11–18.

[24] M. S. Paterson and F. F. Yao, Efficient binary space partitions for hidden-surface removal and solid modeling, *Discrete Comput. Geom.*, 5 (1990), 485–503.

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

[26] R. A. Schumacker, R. Brand, M. Gilliland, and W. Sharp, Study for applying computer-generated images to visual simulation, Tech. Rep. AFHRL–TR–69–14, U.S. Air Force Human Resources Laboratory, 1969.

[27] R. Seidel, Backwards analysis of randomized geometric algorithms, in: *New Trends in Discrete and Computational Geometry* (J. Pach, ed.), *Algorithms and Combinatorics*, Vol. 10, Springer-Verlag, 1993, pp. 37–68.

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

[29] 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.

[30] E. Torres, Optimization of the binary space partition algorithm (BSP) for the visualization of dynamic scenes, *Eurographics '90*, North-Holland, 1990, pp. 507–518.