Multi-way Space Partitioning Trees

Christian A. Duncan

Department of Computer Science, University of Miami,
duncan@cs.miami.edu,
http://www.cs.miami.edu/~duncan

Abstract. In this paper, we introduce a new data structure, the multi-way space partitioning (MSP) tree similar in nature to the standard binary space partitioning (BSP) tree. Unlike the super-linear space requirement for BSP trees, we show that for any set of disjoint line segments in the plane there exists a linear-size MSP tree completely partitioning the set. Since our structure is a deviation from the standard BSP tree construction, we also describe an application of our algorithm. We prove that the well-known Painter's algorithm can be adapted quite easily to use our structure to run in $O(n)$ time. More importantly, the constant factor behind our tree size is extremely small, having size less than $4n$.

1 Introduction

Problems in geometry often involve processing sets of objects in the plane or in a higher dimensional space. Generally, these objects are processed by recursively partitioning the space into subspaces. A common approach to partitioning the set involves constructing a binary space partitioning (BSP) tree on the objects. The operation is quite straightforward. We take the initial input and determine in some manner a hyperplane that divides the region. We then partition the space into two subspaces, corresponding to the two half-spaces defined by the hyperplane. The set of objects is also partitioned by the hyperplane, sometimes fragmenting individual objects. The process is then repeated for each subspace and the set of (fragmented) objects until each subspace (cell) contains only one fragment of an object. This requires the assumption that the objects are disjoint; otherwise, we cannot guarantee that every cell subspace contains only one fragment of an object. The final tree represents a decomposition of the space into cells. Each node of the tree stores the hyperplane splitting that subspace and each leaf represents a cell in the decomposition containing at most one fragmented object. For more detailed information see, for example, [9].

In computer graphics, one often wishes to draw multiple objects onto the screen. A common problem with this is ensuring that objects do not obstruct other objects that should appear in front of them. One solves this problem by doing some form of *hidden surface removal*. There are several approaches to solving this problem including the *painter's algorithm* [11]. Like a painter, one attempts to draw objects in a back-to-front order to guarantee that an object is drawn *after* all objects behind it are drawn and thus appears in front of all

F. Dehne, J.-R. Sack, M. Smid (Eds.): WADS 2003, LNCS 2748, pp. 219–230, 2003.

of them. Fuchs et al. [12] popularized the use of BSP trees by applying them to the *painter's algorithm*. Since then BSP trees have been successfully applied to numerous other application areas including shadow generation [4,5], solid modeling [13,15,19], visibility [3,17,18], and ray tracing [14].

The size of the BSP tree, bounded by the number of times each object is partitioned, greatly affects the overall efficiency of these applications. Paterson and Yao [15] showed some of the first efficient bounds on the size of the binary space partition tree. In particular, they showed that a BSP tree of size $O(n \log n)$ can be constructed in the plane and an $O(n^2)$-sized tree can be constructed in $\mathbb{R}^3$, which they prove to be optimal in the worst-case. Recently, Tóth [20] proved that there exist sets of line segments in the plane for which any BSP tree must have at least $\Omega(n \log n / \log \log n)$ size.

By making reasonable and practical assumptions on the object set, improved bounds have been established, see [6,10,16,21]. For example, Paterson and Yao [16] show that a linear-size BSP tree exists when the objects are orthogonal line segments in the plane. Tóth [21] shows a bound of $O(kn)$ when the number of distinct line segment orientations is k. In [6], de Berg et al. show that in the plane a linear size BSP tree exists on sets of fat objects, on sets of line segments where the ratio between the longest and shortest segment is bounded by a constant, and on sets of homothetic objects, that is objects of identical shape but of varying sizes. Our approach is very similar to theirs but with a different aim.

The research in higher-dimensional space is also quite rich but is not the focus of this paper [1,2,7,8,15,16]. We do feel that extending this structure to $\mathbb{R}^3$ is a natural next step for this data structure.

1.1 Our Results

This paper focuses on partitioning a set of n disjoint line segments in the plane. We introduce a new data structure, the multi-way space partitioning (MSP) tree. Unlike standard binary partitioning schemes, MSP trees are produced by partitioning regions into several sub-regions using a spirally shaped cut as described in the next section. We show that for any set of disjoint line segments in the plane there exists a linear-size MSP tree on the set. Unlike previous results on linear-size BSP trees, our segments have no constraints other than being disjoint. More importantly, the constant factors behind our techniques are extremely small. In fact, we show that the constructed tree has size less than $4n$.

Since our structure is a deviation from the standard BSP tree construction, we also describe an application of our algorithm. More specifically, we prove that the painter's algorithm can quite easily be adapted to use our structure to run in $O(n)$ time. We accomplish this by creating a visibility ordering of the cells from a viewpoint v. That is, for any two cells, c_i and c_j, if any line segment from v to a point in c_i intersects c_j then c_j comes before c_i in the ordering. Since many other applications using BSP trees rely on some form of a visibility ordering on the various cell regions, our algorithm should easily adapt to other applications.

2 Multi-way Space Partitioning

For the remainder of this paper, we shall assume we are working exclusively with objects which are disjoint line segments in the plane. Multi-way space partitioning trees store information in a fashion very similar to BSP trees. At each node in the tree, rather than restricting partitioning to a single hyperplane, we also allow a spiral cut to partition the region into multiple disjoint sub-regions. Since every region produced will be convex, when we refer to a region we specifically mean a convex region. As with the BSP tree, every segment that intersects a sub-region is propagated downwards. In some cases, line segments may be split by the (spiral) cut and belong to multiple sub-regions. A leaf in the tree is created when a region contains a single segment. To minimize the size of the MSP tree, we wish to reduce the number of times any segment is divided by cuts. In our construction algorithm, we shall bound the number of times a segment is split to at most three; thus proving a size of less than $4n$. Before we can proceed with the construction and proof of the tree size, we must first define the spiral cut in detail (see Figure 1).

Definition 1. *A* **spiral cut of size** $k \geq 3$ *is a cyclic set of rays* $C = \{c_0, \ldots, c_{k-1}\}$ *such that,*

1. c_i *intersects* c_j *if and only if* $j \equiv i \pm 1 \bmod k$*; only neighboring rays intersect.*
2. c_i *and* c_{i+1} *intersect only at the endpoint of* c_{i+1} *(*mod$k$*, of course).*

Let p_i *be the endpoint of ray* c_i *lying on ray* c_{i-1}*. Let the* **center line segment** l_i *be the segment, lying on* c_i*, formed by the endpoints* p_i *and* p_{i+1}*. Let the* **exterior ray** c'_i *be the ray formed by removing* l_i *from* c_i*. Note that* c'_i *has endpoint* p_{i+1}*.*

Define the **arm region** R_i *to be the V-shaped region lying between the two rays* c_i *and the* c'_{i-1} *defined by* p_i*. Define the* **center region** R_k *to be the convex hull of the set of endpoints* $p_i, i \in \{0, \ldots, k\}$*, which consists of the set of center line segments.*

A point p *lies to the* **right of ray** c_i *if the angle formed from* c_i *to the ray starting at* p_i *passing through* p *is in the range* $(0, \pi)$*. Similarly, a point* p *lies to the* **left of ray** c_i *if the angle is negative. In addition, a ray* c_{i+1} *is to the* **right** *(***left***) of ray* c_i *if any point on* c_{i+1} *is to the right (left) of* c_i*.*

A spiral cut is oriented **clockwise** *(***counterclockwise***) if every consecutive ray is to the right (left) of its previous ray. That is, if* c_{i+1} *is to the right of* c_i *for all* $c_i \in C$*.*

Because the rays are cyclically ordered and only intersect neighboring rays, every *turn* must be in the same direction. Therefore, there are only two types of spiral cuts, clockwise and counterclockwise. As described above, a spiral cut of size k divides the region into $k + 1$ convex sub-regions. There are k sub-regions, $R_0, \ldots, R_{k-1}$, each associated with an arm region of the spiral, and one sub-region R_k in the center of the spiral (see Figure 1).

There are several properties that we can establish that will prove useful in our evaluation of the MSP tree.

Property 1. If the spiral cut, C, is clockwise (counterclockwise), then any point p in the center region R_k lies to the right (left) of every ray $c_i \in C$. For a clockwise spiral cut, let p be any point in an arm region, say R_0. Point p lies to the left of c_0 and the right of c_{k-1}. In addition, there exists a ray c_m such that p lies to the left of all rays c_i for $0 \leq i \leq m$ and to the right of all rays c_i for $m < i \leq k-1$. That is, traversing the cycle from c_0 around to c_{k-1}, divides the cycle into two continuous sequences those with p on the left and those with p on the right. For counterclockwise spiral cuts, the reverse directions apply.

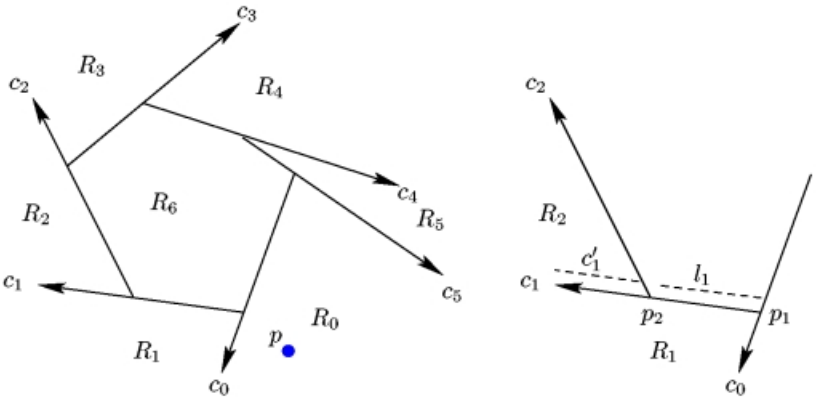

Fig. 1. An example of a clockwise spiral cut $C = \{c_0, c_1, c_2, c_3, c_4, c_5\}$ forming. 6 arm regions and the center region. The point $p \in R_0$ lies to the left of c_0 and c_1 but to the right of all other rays.

2.1 Construction

Assume we are given an initial set of segments S. The general construction algorithm is quite simple, start with an initial bounding region of the segment endpoints. For every region R, if there is only one segment of S in the region, nothing needs to be done. Otherwise, find an appropriate halfplane cut or spiral cut. Then, divide the region into sub-regions $R_0, R_1, \ldots R_k$ which become child regions of R. The line segments associated with the cut are stored in the current node and all remaining line segments in R are then propagated into the appropriate (possibly multiple) sub-regions. Finally, repeat on each of the sub-regions.

What remains to be shown is how to determine an appropriate cut. We do this by classifying our segments into two categories: rooted and unrooted segments (see Figure 2). For any convex region R, a *rooted segment* of R is a segment which intersects both the interior and boundary of R. Similarly, an *unrooted segment* of R is a segment which intersects the interior of R but not its boundary. By this definition unrooted segments of R must lie completely inside the region.

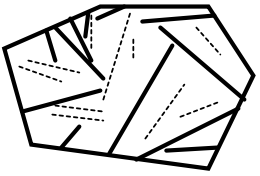

Fig. 2. An example of rooted (solid) and unrooted (dashed) segments in a convex region.

For any region R, let $\mathcal{S}(R)$ represent the set of all segments of S lying (partially) inside R. Let $\mathcal{R}(R) \subseteq \mathcal{S}(R)$ represent the set of rooted segments of S in R and let $\mathcal{U}(R) = \mathcal{S}(R) - \mathcal{R}(R)$ represent the set of unrooted segments of S in R. For any two regions R_1 and R_2 if there exists a segment $s \in S$ such that $s \in \mathcal{U}(R_1)$ and $s \in \mathcal{U}(R_2)$ then $R_1 \subseteq R_2$ or $R_2 \subseteq R_1$. This means that R_1 and R_2 must lie on the same path from the root node to a leaf in the MSP tree. In addition, if $s \in \mathcal{U}(R_1)$ and $s \in \mathcal{R}(R_2)$ then $R_2 \subset R_1$. That is, R_2 must be a descendant of R_1 in the tree.

Let us now see how we can exploit these rooted and unrooted segments. In [6], de Berg et al. show that if a region contains *only* rooted segments then a BSP tree of linear size can be constructed from it. Of course, the challenge is in guaranteeing that this situation occurs. As a result, they first made numerous cuts to partition the initial region R into sub-regions such that every segment was cut at least once but also not too many times. Their result relied on the assumption that the ratio between the longest segment and the shortest segment was some constant value.

We take a somewhat different approach to this problem. We do not mind having unrooted segments in our region and actually ignore them until they are first intersected by a dividing cut, after which they become rooted segments and remain so until they are selected as part of a cut. In our construction, we guarantee that rooted segments are *never* divided by a partitioning cut. That is, only unrooted segments will be cut. This situation can only occur once per segment in S. Let us now see how to find an appropriate partitioning cut.

2.2 Finding a Spiral or Hyperplane Cut

Let us assume we are given some region R. For this subsection, we will completely ignore unrooted segments. Therefore, when we refer to a segment s we always mean a rooted segment $s \in \mathcal{R}(R)$.

Although not necessary, observe that if a rooted segment intersects the boundary of R in two locations then we can choose this segment as a valid partitioning cut. Therefore, for simplicity, we assume that no segment in R intersects the boundary of R more than once.

As in [6], we try to find either a single cut that partitions the region or else a cycle of segments that do. We do this by creating an ordered sequence on the

segments starting with an initial segment $s_0 \in \mathcal{R}(R)$. Let us extend s_0 into R until it either hits the boundary of R or another segment in $\mathcal{R}(R)$. Define this extension to be $\text{ext}(s_0)$. For clarity, note that the extension of s_0 includes s_0 itself. If $\text{ext}(s_0)$ does not intersect any other segment in $\mathcal{R}(R)$, then we take it as a partitioning cut. Otherwise, the extension hits another segment s_1. In this case, we take s_1 to be the next segment in our sequence. The rest of the sequence is completed in almost the same fashion.

Let us assume that the sequence found so far is $\{s_0, s_1, s_2, \ldots, s_i\}$. We then extend s_i until s_i hits either the boundary of R, a previous extension $\text{ext}(s_j)$ for $j < i$, or a new segment s_{i+1}. If it hits the boundary of R, then we can take s_i as a partitioning cut. If it intersects $\text{ext}(s_j)$, then we have completed our cycle, which is defined by the sequence $\mathcal{C}(R) = \{\text{ext}(s_j), \text{ext}(s_{j+1}), \ldots, \text{ext}(s_i)\}$. Otherwise, we repeat with the next segment in our sequence, s_{i+1}.

Since there are a bounded number of segments in $\mathcal{R}(R)$, the sequence must find either a single partition cut s or a cycle $\mathcal{C}$. If it finds a single partition cut s then we can simply divide the region R into two sub-regions by the line formed by s as usual. Otherwise, we use the cycle $\mathcal{C}$ to define a spiral cut.

Let $\text{ext}(s_i)$ and $\text{ext}(s_{i+1})$ be two successive extension segments on the cycle. By the construction of the cycle, $\text{ext}(s_i)$ has an endpoint p_i on $\text{ext}(s_{i+1})$. We, therefore, define the ray for c_i to be the ray starting at p_i and extending outward along $\text{ext}(s_i)$ (see Figure 3). To be consistent with the spiral cut notation, we must reverse the ordering of the cycle. That is, we want p_i to lie on c_{i-1} and *not* c_{i+1}. Also, except possibly for the initial extended segment, every extension $\text{ext}(s_i)$ is a subset of l_i, the center line segments forming the convex center region, R_k.

Since the initial region is convex and by the general construction of the cycle, this new cycle of rays defines a spiral cut. We can now proceed to use this spiral cut to partition our region into multiple regions and then repeat the process until the space is completely decomposed.

2.3 MSP Size

To complete our description of the tree, we only need to analyze its worst-case size. The size of the MSP tree produced by our construction algorithm depends only on the following two conditions:

1. At every stage no rooted segment is partitioned.
2. At every stage no unrooted segment is partitioned more than a constant, c, number of times.

If both of these conditions hold, the size of the tree is at most $(c+1)n$ since an unrooted segment once split is divided into rooted segments only and each ray of the spiral cut corresponds to one rooted segment.

Lemma 1. *Given a convex region R with a set of rooted segments $\mathcal{R}(R)$ and unrooted segments $\mathcal{U}(R)$, a partitioning cut or spiral cut can be found which divides R into sub-regions such that no segment in $\mathcal{R}(R)$ is intersected by the*

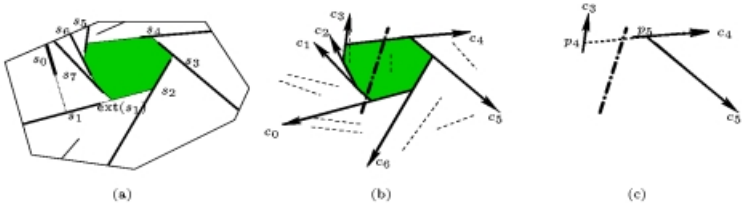

Fig. 3. (a) An example of finding a cycle of cuts. Here, s_0 is the initial cut and the cycle completes when s_7 intersects ext(s_1). Thus, the sequence is $\{s_1, s_2, \ldots, s_7\}$. (b) The resulting spiral cut, $\{c_0, \ldots, c_6\}$. This cycle is formed by the sequence of segments reversed. Observe how the unrooted segments intersect the spiral cut and in particular how the bold dashed unrooted segment is intersected the maximum three times. (c) Here, c_4 is the ray extending from p_4, c'_4 is the ray extending from p_5, l_4 is the segment between p_4 and p_5. The ext(s_4) is the dashed portion starting at p_4 and ending before p_5. Observe how the unrooted segment can intersect only the ext(s_4) if it intersects c_4.

cut except those that are part of the cut and no unrooted segment in $\mathcal{U}(R)$ is intersected by the cut more than three times.

Proof. We construct the sequence $\{s_0, s_1, \ldots s_k\}$ as described in the previous subsection. If we choose a segment s_i, as a partitioning cut, then by our construction it does not intersect any other rooted segment. Also, it can intersect an unrooted segment at most once.

Let us then assume that we have identified a spiral cut $\{c_0, c_1, c_2, \ldots, c_{k-1}\}$. Given the construction of the spiral cut itself, it is clear that no rooted segment that is not part of the cycle is intersected. So, all that is left to prove is that unrooted segments are intersected at most three times.

As described earlier, the rays of the spiral cut can be broken into two pieces, the portion of the ray forming the convex central region R_k and the arm regions R_i for $0 \leq i < k$. In particular, let us look at any three successive rays, say c_0, c_1, and c_2. Recall that p_1 is the endpoint of c_1 and p_2 is the endpoint of c_2. In addition, p_1 lies on c_0 and p_2 lies on c_1. Recall that the *center* line segment l_1 is defined to be the segment from p_1 to p_2 and that the exterior ray c'_1 is the ray extending from p_2 along c_1.

Now, let us look at an unrooted segment $s \in \mathcal{U}(R)$. We first look at the line segments l_i forming R_k. Because the region is convex, s can intersect at most two segments of the convex central region. Let us now look at the exterior ray portions. Recall that each extension, ext(s_i), except for $i = 0$, is a subset of the center line segment l_i. Since the portion of c_i lying inside R is exactly the union of s_i and ext(s_i) and, except for $i = 0$, ext(s_i) is a subset of the center line segment l_i, the portion of c'_i lying inside R is a subset of the segment s_i. Since all segments are disjoint and s is unrooted, s cannot intersect c'_i except for c'_0. As a result, the spiral cut intersects s at most three times (see Figure 3b). □

This lemma along with the construction of the multi-way space partitioning tree leads to the following theorem:

Theorem 1. *Given a set of n disjoint segments $S \subset \mathbb{R}^2$, a multi-way space partitioning tree T can be constructed on S such that $|T| < 4n$ in $O(n^3)$ time.*

Proof. The proof of correctness and size is straightforward from the construction and from Lemma 1. As for the running time, a straightforward analysis of the construction algorithm shows $O(n^2)$ time for finding a single spiral cut and hence the $O(n^3)$ overall time. □

This is most likely not the best one can do for an MSP tree construction. It seems possible to get the time down to near quadratic time. Although it may be difficult to develop an algorithm to compete with the $O(n \log n)$ construction time for a regular BSP tree, we should point out that the BSP tree created is not necessarily optimal and is typically created via a randomized construction.

3 Painter's Algorithm

To illustrate the vitality of the MSP tree, we now show how to apply this structure to the painter's algorithm. In a BSP tree, the traditional approach to solving the painter's algorithm is to traverse the tree in an ordered depth-first traversal. Assume we are given an initial view point, v. At any region R in the tree, we look at the partitioning cut. Ignoring the degenerate case where v lies on the cut itself, v must lie on one side or the other of the cutting line. Let R_1 be the sub-region of R lying on the same side of the line as v and let R_2 be the other sub-region. We then recursively process R_2 first, process the portion of the line segment in the region corresponding to the cutting line, and then process R_1. In this way, we guarantee that at any time a line segment s is drawn it will always be drawn *before* any line segment between s and v.

To see the corresponding approach to traversing the MSP tree, let us generalize the depth-first search. Recall at a region R, we visit all the sub-regions on the opposing side of the cutting line to v and then all sub-regions on the same side as v. Let R_1 be a sub-region of R visited in the search. The ultimate goal is to guarantee that, for any point $p \in R_1$, the line segment $\overline{pv}$ intersects only sub-regions that have not been visited already. Now, let R have multiple sub-regions $R_0, R_1, \ldots, R_k$ rather than just two. We still wish to construct an ordering on the sub-regions such that the following property holds:

- Let p_i be any point in R_i. The line segment $\overline{p_i v}$ does not intersect any region R_j with $j < i$ in our ordering.

Notice if this property holds, then we can traverse each sub-region recursively as before and guarantee that no line segment s is drawn after a line segment appearing between v and s.

3.1 Spiral Ordering

Unfortunately, given a spiral cut, we cannot actually guarantee that such an ordering of the sub-regions always exists from any viewpoint v. However, when

processing a scene one also considers a viewing direction and a viewing plane onto which to project the scene. In this sense, we assume that one has a view line v_p that passes through v and defines a particular viewing half-plane V. Therefore, all line segments **behind** the viewer can in fact be ignored.

Adding a view line does in fact enable us to create an ordering. This particular point will only arise in one specific case. In addition, for applications such as shadow generation requiring full processing of the scene, observe that we may perform the process *twice* using the same view line with opposing normals.

To compute the order using a spiral, it is somewhat easier to describe how to compute the *reverse* ordering. After creating this ordering, we can simply reverse the ordering to get the desired result. Let us define the following ordering on a spiral cut:

Definition 2. *Given a view point v, a viewing half-plane V, and a spiral cut $\{c_0, c_1, \ldots, c_{k-1}\}$. Let $R_0, R_1, \ldots, R_k$ be the sub-regions produced by the cut. A* **visible ordering** *$o(x)$ represents a permutation of the sub-regions such that,*

- *for any point $p_i \in R_i \cap V$, if the line segment $\overline{p_i v}$ intersects a region R_j, then $o(j) \leq o(i)$.*

Moreover, given any ordering, we say a point $p_i \in R_i$ is **visible** *from v if the above condition holds for that point. We also say that v* **sees** *p_i.*

In other words, we visit regions in such a way that v can see every point in a region R_i by passing *only* through previously visited regions. Notice this is the reverse ordering of the painter's algorithm where we want the opposite condition that it only passes through regions that it has not yet visited. A simple flip of the ordering once generated produces the required ordering for the painter's algorithm.

Lemma 2. *Given a view point v, a viewing half-plane V, and a spiral cut $\{c_0, c_1, \ldots, c_{k-1}\}$. Let $R_0, R_1, \ldots, R_k$ be the sub-regions produced by the cut. There exists a visible ordering $o(x)$ on the spiral cut.*

Proof. Let R_i be the region containing the view point v itself. Let $o(i) \to 0$ be the first region in our ordering. Notice that since every region is convex and $v \in R_i$, any point $p \in R_i$ is visible from v.

Without loss of generality, assume that the spiral cut is a clockwise spiral. The argument is symmetrical for counterclockwise spirals. Let us now look at two different subcases. Recall that the spiral cut consists of two parts the center region and the arm regions.

Case 1: Let us first assume that R_i is an arm region. Without loss of generality assume that $R_i = R_0$. We will create our ordering in three stages. In the first stage, we add regions $R_1, R_2, \ldots, R_m$ for some m to be described shortly. We then add the center region R_k and finally we add the remaining regions $R_{k-1}, R_{k-2}, \ldots R_{m+1}$.

Let us begin with the first stage of ordering. Assume that we have partially created the ordering $o(0), o(1), \ldots, o(i)$ and let $R_i = R_{o(i)}$ be the last region

added. Recall that R_i is defined by the rays c_i and c'_{i-1}. Let us now look at the neighboring region R_{i+1} defined by the rays c_{i+1} and $c'_i \subset c_i$.

If v lies to the **left** of c_{i+1}, add R_{i+1} to the ordering. That is, let $o(i+1) \rightarrow i+1$. We claim that all points in R_{i+1} are visible from v. Let p be any point in R_{i+1}. Notice that p also lies to the left of c_{i+1}. Therefore the line segment $\overline{pv}$ cannot intersect ray c_{i+1} and must therefore intersect ray c_i. Let q be the point on this intersection or just slightly passed it. Notice that q lies inside R_i. By induction, q must be visible from v. Therefore, the line segment $\overline{qv}$ intersects only regions with ordering less than or equal to i. In addition, the line segment $\overline{pq}$ intersects only R_{i+1}. Therefore, the line segment $\overline{pv}$ intersects only regions with ordering less than or equal to $i+1$ and p is visible from v.

If v lies to the **right** of c_{i+1}, we are done with the first stage of our ordering, letting $m = i$.[1] We now add the center region R_k into our ordering. That is, let $o(i+1) \rightarrow k$. Again, we claim that all points in R_k are visible from v. Recall from Property 1 that v lies to the right of all rays from c_{m+1} to c_{k-1}, given that v lies in R_0. Let p be any point in R_k. Again, from Property 1 we know that p lies to the right of every ray in the cut. Let R_j be any region intersected by the line segment $\overline{pv}$. If R_j is R_k or R_0 we are done since they are already in the ordering. Otherwise, we know that since R_j is convex, $\overline{pv}$ must intersect the ray c_j. Since p is to the right of c_j as with all rays, this implies that v must lie to the left of c_j. But, that means that c_j cannot be part of c_{m+1} to c_{k-1}. R_j must be one of the regions already visited and so $j \in \{o(0), \ldots, o(m)\}$. Hence, p is visible from v.

We now enter the final stage of our ordering. We shall now add into the ordering the regions from R_{k-1} backwards to R_{m+1}. Let us assume that we have done so up to R_j. We claim that all points in R_j are visible from v. Let p be any point in R_j. Again look at the line segment $\overline{pv}$ and the first (starting from p) intersection point q with another region. This point must certainly lie on one of the two rays c_{j-1} or c_j. Since p is to the right of c_{j-1} (Property 1), if it intersects c_{j-1}, v must lie to the left of c_{j-1}. This means that R_{j-1} is already in the ordering and, as with previous arguments, q is visible from v and hence so is p. If it intersects c_j instead, then q lies either in R_k or R_{j+1}. But again in either case, since we added the center region already and are counting backwards now, both R_k and R_{j+1} are in the ordering. This implies that q is visible from v and so then is p.

Thus, we have constructed a visible ordering of the regions assuming p lies in one of the arm regions. We now need to prove the other case.

Case 2: Let v lie in the center region R_k. In this case, unfortunately, there is no region that is completely visible from v except for the center region. This is where the viewing half-plane V comes into play. Our arguments are almost identical to the above case except we now only look at points in V. For simplicity, let us assume that V is horizontal with an upward pointing normal.

[1] For the sake of simplicity, we are ignoring degenerate cases such as when v lies directly on the line defined by c_{i+1}.

Look at the ray from v going horizontal to the right and let R_i be the first new region hit by this ray. That is, R_i is the region directly to the right of v. Without loss of generality, we can let this region be R_{k-1}. We then add all regions into the ordering starting with the center region and counting backwards from the rightmost region, $R_k, R_{k-1}, R_{k-2}, \ldots, R_m$, where R_m is the last region visible, at least partially intersecting V. We first claim that all point in $R_{k-1} \cap V$ are visible from v. Let p be any point in $R_{k-1} \cap V$. Since p lies to the left of and v lies to the right of c_{k-1}, the line segment $\overline{pv}$ must intersect c_{k-1}. Let q be this intersection point. Since R_{k-1} is the first region to the right of v and p lies *above* the line defined by V, we know that q must actually lie on l_{k-1} or else R_0 would be seen first horizontally by v. This implies that q is seen from v and hence so is p. Let us now assume that we have constructed the ordering up to some region R_i. We claim that all points in $R_i \cap V$ are visible from v. Let p be any point in $R_i \cap V$. Once again from the sidedness of p and v, we know that the line segment $\overline{pv}$ must intersect c_i. Let q be this intersection point. Now, either q lies in $R_k \cap V$ or in $R_{i+1} \cap V$. In either case, both regions have been added to our ordering and so q is visible from v. Therefore, p must also be visible from v. By induction, our ordering is a proper visible ordering and we are done. □

The technique for calculating the ordering is quite straightforward. The algorithm must make one full scan to determine the sub-region containing v. Afterwards, it either marches along one direction, adds in the center region, and marches in the other direction or it adds in the center region first, finds the first region intersected by the viewing half-plane V and marches backwards along the list. In either case, the algorithm can be implemented in at most two scan passes. These observations and the fact that the MSP tree has linear size, leads to the following theorem:

Theorem 2. *Given an MSP tree constructed on a set of n line segments S in $\mathbb{R}^2$, one can perform the painter's algorithm on S in $O(n)$ time.*

4 Conclusion and Open Problems

In this paper, we have described a simple space-partitioning tree that can be constructed in linear size on any set of disjoint line segments in the plane. We hope to improve construction time and reduce the maximum degree for any single node from $O(n)$ to constant degree. More importantly, we would like to focus on a similar technique in $\mathbb{R}^3$ space where BSP trees are known to have very poor sizes. The question arises whether deviating from the standard notion of binary space partitions provides better performance, even in the average case. We feel that answering such a question would demonstrate the greatest promise for this new tree structure. The spiral cut as mentioned for the plane will not immediately translate into higher-dimensions, but we are hopeful that some other deviation from the standard cutting method may produce surprising results.

References

1. P. Agarwal, T. Murali, and J. Vitter. Practical techniques for constructing binary space partitions for orthogonal rectangles. In *Proc. of the 13th Symposium on Computational Geometry*, pages 382–384, New York, June 4–6 1997. ACM Press.
2. P. K. Agarwal, E. F. Grove, T. M. Murali, and J. S. Vitter. Binary space partitions for fat rectangles. *SIAM Journal on Computing*, 29(5):1422–1448, Oct. 2000.
3. J. M. Airey. *Increasing Update Rates in the Building Walkthrough System with Automatic Model-Space Subdivision and Potentially Visible Set Calculations*. PhD thesis, Dept. of CS, U. of North Carolina, July 1990. TR90-027.
4. N. Chin and S. Feiner. Near real-time shadow generation using BSP trees. *Computer Graphics (SIGGRAPH '90 Proceedings)*, 24(4):99–106, Aug. 1990.
5. N. Chin and S. Feiner. Fast object-precision shadow generation for areal light sources using BSP trees. *Computer Graphics (1992 Symposium on Interactive 3D Graphics)*, 25(4):21–30, Mar. 1992.
6. de Berg, de Groot, and Overmars. New results on binary space partitions in the plane. *CGTA: Computational Geometry: Theory and Applications*, 8, 1997.
7. M. de Berg. Linear size binary space partitions for fat objects. In *Algorithms—ESA '95, Third Annual European Symposium*, volume 979 of *Lecture Notes in Computer Science*, pages 252–263. Springer, 25–27 Sept. 1995.
8. M. de Berg and M. de Groot. Binary space partitions for sets of cubes. In *Abstracts 10th European Workshop Comput. Geom.*, pages 84–88, 1994.
9. M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry Algorithms and Applications*. Springer-Verlag, Berlin Heidelberg, 1997.
10. A. Dumitrescu, J. S. G. Mitchell, and M. Sharir. Binary space partitions for axis-parallel segments, rectangles, and hyperrectangles. In *Proceedings of the 17th annual symposium on Computational geometry*, pages 141–150. ACM Press, 2001.
11. J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, Reading, MA, 1990.
12. 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.
13. B. Naylor, J. A. Amanatides, and W. Thibault. Merging BSP trees yields polyhedral set operations. *Comp. Graph (SIGGRAPH '90).*, 24(4):115–124, Aug. 1990.
14. B. Naylor and W. Thibault. Application of BSP trees to ray-tracing and CGS evaluation. Technical Report GIT-ICS 86/03, Georgia Institute of Tech., School of Information and Computer Science, Feb. 1986.
15. 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.
16. M. S. Paterson and F. F. Yao. Optimal binary space partitions for orthogonal objects. *J. Algorithms*, 13:99–113, 1992.
17. S. J. Teller. *Visibility Computations in Densely Occluded Polyhedral Environments*. PhD thesis, Dept. of Computer Science, University of California, Berkeley, 1992.
18. S. J. Teller and C. H. Séquin. Visibility preprocessing for interactive walkthroughs. *Comput. Graph.*, 25(4):61–69, July 1991. Proc. SIGGRAPH '91.
19. W. C. Thibault and B. F. Naylor. Set operations on polyhedra using binary space partitioning trees. *Comput. Graph.*, 21(4):153–162, 1987. Proc. SIGGRAPH '87.
20. C. D. Tóth. A note on binary plane partitions. In *Proceedings of the seventeenth annual symposium on Computational geometry*, pages 151–156. ACM Press, 2001.
21. C. D. Tóth. Binary space partitions for line segments with a limited number of directions. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 465–471. ACM Press, 2002.