Binary Plane Partitions for Disjoint Line Segments

Csaba D. Tóth*

Abstract

A binary space partition (BSP) for a set of disjoint objects in Euclidean space is a recursive decomposition, where each step partitions the space (and some of the objects) along a hyperplane and recurses on the objects clipped in each of the two open halfspaces. The size of a BSP is defined as the number of resulting fragments of the input objects. It is shown that every set of n disjoint line segments in the plane admits a BSP of size $O(n \log n / \log\log n)$. This bound is best possible apart from the constant factor.

1 Introduction

A *binary space partition* (*BSP*) for a set of objects is a simple hierarchical decomposition of the space into convex faces. Given a finite set of disjoint $(d-1)$-dimensional objects in $\mathbb{R}^d$, a BSP partitions the space (and some of the input objects) along a hyperplane and recurses on the objects clipped in each nonempty open half-space. An *auto-partition* is a special type of BSP, where every partition step is done along the supporting hyperplane of one of the input objects. The partition steps of a BSP can be stored in a binary tree data structure, called *BSP tree*, where each node corresponds to a subproblem and each nonleaf node stores a partition hyperplane [11].

BSPs were introduced in the computer graphics community [16, 24, 25] for maintaining the back-to-front order of the *fragments* of the input objects, and for rendering polygonal scenes efficiently with z-buffering. Because of their simplicity, BSPs have found a variety of applications in solid modeling, shadow generation, motion planning, approximate range searching, and network design problems [2, 3, 6, 9, 12, 14, 20, 21], to name a few examples.

The most important parameter of a BSP is the number of fragments it partitions the input objects into, this is called the *size* of the BSP. It is an obvious lower bound for the size of the corresponding BSP tree data structure, and it is also an asymptotic upper bound if every fragment has bounded description complexity (e.g., line segments in the plane or axis-aligned boxes in $\mathbb{R}^d$), and the number of objects monotonically decreases in each subproblem (i.e., there are no "redundant" cuts).

Theoretical research focused on finding the minimum size of a BSP for certain types of objects in $\mathbb{R}^d$. Paterson and Yao [22] proved that every set of n disjoint line segments in the plane admits a BSP of size $O(n \log n)$. They also showed that by partitioning along the input segments in a random order (combined with *free cuts*, defined below) produces an auto-partition of $O(n \log n)$ expected size

Paterson and Yao [23] also proved that n disjoint line segments with only two distinct directions (e.g., axis-parallel segments) admit a BSP of size $O(n)$. This was later generalized: n disjoint line segments with k, $1 \le k \le n$, distinct directions admit a BSP of size $O(n \log k)$ and an auto-partition of size $O(nk)$ [28]. A set of n disjoint segments also admits a BSP of size $O(n)$ if their lengths differ by no more than a constant factor [10]. However, there are sets of n disjoint line segments for which

*Department of Computer Science, Tufts University, and Department of Mathematics and Statistics, University of Calgary. E-mail: `cdtoth@cs.tufts.edu`. Supported in part by NSERC grant RGPIN 35586.

any BSP has $\Omega(n \log n / \log \log n)$ size [26]. Here we show that this lower bound is best possible apart from the constant factor.

Theorem 1. *Every set of n disjoint line segments in the plane admits a BSP of size $O(n \log n / \log \log n)$.*

The proof is constructive and leads to a deterministic algorithm for constructing a BSP tree in $O(n \text{ polylog } n)$ time. Our partition algorithm does not guarantee that *each* line segment is fragmented into $O(\log n / \log \log n)$ pieces. It relies on a charging scheme where each event that "a partition line cuts an input segment" is charged to one of the input segments, and each segment is charged $O(\log n / \log \log n)$ events. The BSP we present is not necessarily an auto-partition, since the partition lines are necessarily spanned by a segment in the corresponding subproblem. With some extra work, however, we can also construct an *auto-partition* of size $O(n \log n / \log \log n)$.

Theorem 2. *Every set of n disjoint line segments in the plane admits a auto-partition of size $O(n \log n / \log \log n)$.*

Organization. We present a key lemma (Lemma 3) and a basic building block of our partition algorithms in Section 2.1. We prove Lemma 3 in Section 3. Then we prove Theorem 1 by repeatedly applying Lemma 3 in Section 4. We adjust these methods to produce an *auto-partition* of size $O(n \log n / \log \log n)$ in Section 5. We describe how to implement our BSP algorithm for a set of n disjoint line segments in $O(n \text{ polylog } n)$ time (in the real RAM model of computation) in Section 6. We conclude with some open problems in Section 7.

Related results. Paterson and Yao [22] showed that n disjoint $(d-1)$-dimensional simplices in $\mathbb{R}^d$, $d \geq 2$, admit an auto-partition of size $O(n^{d-1})$, and this bound is best possible apart from the constant factor. This is also the best bound for BSPs for $d = 3$ (for disjoint triangles in $\mathbb{R}^3$). However, there is no set of disjoint objects in $\mathbb{R}^d$ is known, for any $d \in \mathbb{N}$, that requires a super-quadratic BSP.

Paterson and Yao [23] showed that any set of n axis-aligned line segments in $\mathbb{R}^d$ admits a BSP of size $O(n^{d/(d-1)})$ for $d > 2$, and this bound is best possible. They also gave a tight $O(n^{3/2})$ bound on the BSP size for n disjoint axis-aligned rectangles in $\mathbb{R}^3$. Dumitrescu *et al.* [15] proved upper and lower bounds for the BSP size of disjoint k-dimensional axis-aligned boxes in $\mathbb{R}^d$, for all $1 < k < d$. Their upper bound of $O(n^{d/(d-k)})$ is tight for $1 \leq k < d/2$. They also proved a tight bound of $O(n^{5/3})$ for n disjoint axis-aligned 2-rectangles in $\mathbb{R}^4$. Agarwal *et al.* [1] gave an upper bound of $n 2^{O(\sqrt{\log n})}$ for the BSP size of n disjoint axis-aligned *fat* rectangles in $\mathbb{R}^3$ (in a set of fat rectangles the aspect ratio is bounded by a constant). This upper bound was later improved to $O(n \log^8 n)$ [27]. Hershberger *et al.* [18] gave a tight bound of $O(n^{4/3})$ for the BSP size of n axis-aligned boxes that tile $\mathbb{R}^3$. De Berg [7, 10] showed that there is an $O(n)$ size BSP for the boundary of disjoint fat polyhedra in any dimensions $\mathbb{R}^d$; this result was extended to a slightly more general class of *uncluttered* scenes [8].

2 Preliminaries

The size of a BSP for n disjoint line segments in the plane is the number of fragments that the input segments are partitioned into. Instead of counting fragments, we will keep track of the number of *events* that a partitioning line crosses an input segment, in other words, the events that a fragment is *cut* into two fragments. Since initially there are n line segments and each event increases the number of fragments by one, it is enough to show that the number of cuts is $O(n \log n / \log \log n)$.

At each node of a BSP tree, we maintain a convex region, called a *cell*, that contains all segments of the corresponding subproblem. The cell C_0 at the root is the entire plane, a suitable bounding box, or the convex hull of all input segments. At every nonleaf node v in the BSP tree, a partition line decomposes the convex cell C into two cells, which correspond to the two children of v. The cells at any level of the BSP tree correspond to a decomposition of C_0 into convex subcell. In the remainder of this paper, we assume that each problem is a pair (S, C), where S is a set of disjoint line segments in a convex cell C.

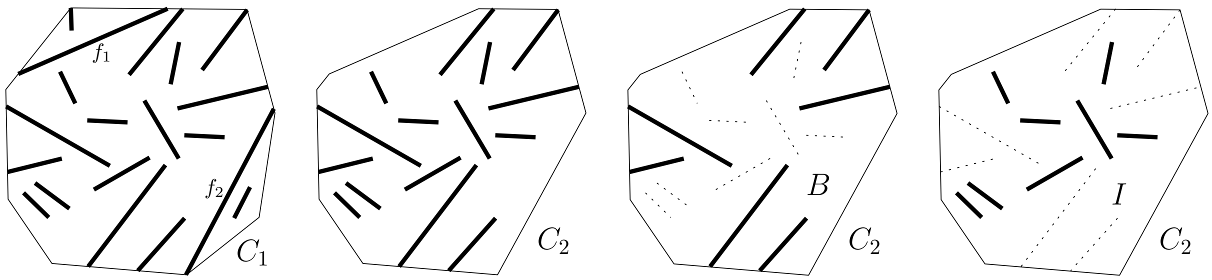


Figure 1: (a) A set of disjoint line segments, including two free segments, f_1 and f_2, in a cell C_1 (b) A set of disjoint line segments, with no free segment, clipped in a cell C_2. (c) The boundary segments in C_2. (d) The interior segments in C_2.

Let S be a set of n disjoint open line segments lying in a convex cell C. Denote by ∂C the boundary of C. We distinguish three types of segments in a problem (S, C). A segment $s \in S$ is

- a *free segment* if both endpoints lie on ∂C (Fig. 1(a));
- a *boundary segment* if one endpoint lies on ∂C (called *outer endpoint*) and the other endpoint lies in the interior of C (called *inner endpoint*) (Fig. 1(bc));
- an *interior segment* if s lies in the interior of C (Fig. 1(bd)).

We denote the sets of free, boundary, and interior segments by F, B, and I, respectively. We can split the problem into two subproblems along a free segment $f \in F$. A partition along a free segment is called a *free cut*: it does not increase the total number of fragments and it partitions the problem into two strictly smaller subproblems (since f does not belong to either open halfplane). In our algorithms, we always partition a problem along any possible free segment, and so we may assume that $F = \emptyset$, hence $S = B \cup I$.

After performing the partition steps up to a certain depth of a BSP tree, we have a decomposition of the plane into convex cells. Each fragment of an input segment $s \in S$ lies either in the interior or on the boundary of some convex cells. A fragment on the boundaries of cells is not part of any subproblem. A fragment that traverses a cell is a free segment and will not be further partitioned. So every surviving fragment of s is incident to an endpoint of s, and it is either an interior segment (if the entire segment lies in the interior of a cell) or a boundary segment (if exactly one endpoint lies in the interior of the cell). Our BSP for S is constructed by the repeated application of algorithm $\texttt{SubBSP}(B, C, k)$ presented in the following lemma.

Lemma 3. *Let S be a finite set of disjoint line segments in a convex cell C. For every k, $1 \le k \le |B|$, there is a binary plane partition $\texttt{SubBSP}(B, C, k)$ such that*

- *every boundary segment is cut at most $O(1)$ times;*
- *every interior segment is cut at most $O(k)$ times;*
- *every cell produced by $\texttt{SubBSP}(B, C, k)$ intersects less than $|B|/k$ segments in B.*

Notice that the *interior* segments in cell C are not part of the input of $\texttt{SubBSP}(B, C, k)$. When constructing $\texttt{SubBSP}(B, C, k)$, we can ignore the interior segments. We establish the property that $\texttt{SubBSP}(B, C, k)$ cuts every interior segment $O(k)$ times based on the fact that every interior segment is disjoint from the boundary segments.

2.1 BSPs along a conformal path

The basic building block of $\texttt{SubBSP}(B, C, k)$ is a recursive plane partition along the edges of a simple path. Consider a finite set B of disjoint boundary segments in a convex cell C. Direct every boundary segment $b \in B$ from its outer endpoint to its inner endpoint. Let $\overrightarrow{b}$ be the ray starting from the outer endpoint of b and containing b; and let $\bar{b}$ be the directed line segment along $\overrightarrow{b}$ from the outer endpoint of b to the first intersection point with another boundary segment or with ∂C.

Definition 4. *Given a set B of boundary segments in a cell C, a simple directed polygonal path $\beta = (u_0, u_1, \dots, u_t)$ for some $t \in \mathbb{N}$ is* conformal *if*

- *for every $j = 1, 2, \dots, t$, there is a boundary segment $b_j \in B$ such that $u_{j-1}u_j \subset \bar{b}_j$;*
- *the portions of segments $\bar{b}_j$ between the outer endpoint of b_j and point u_j have disjoint relative interiors. (See Fig. 2.)*

The algorithm $\texttt{ChainBSP}(\beta)$ below successively partitions the plane along the supporting lines of the edges of a conformal path β in *reverse* order (Fig. 2(ab)). Its input is conformal for an underlying problem (S, C). However, $\texttt{ChainBSP}(\beta)$ will be a subroutine of a larger BSP algorithm, and we assume that when we call $\texttt{ChainBSP}(\beta)$, the cell C (in which β is conformal) may have been decomposed into convex subcells and β does not necessarily lie in a single subcell.

Algorithm 1. $\texttt{ChainBSP}(\beta)$

Input: a conformal path $\beta = (u_0, u_1, \dots, u_t)$ such that for $j = 1, 2, \dots, t$, there is a boundary segment $b_j \in B$ with $u_{j-1}u_j \subset \bar{b}_j$

For $j = 0, 1, \dots, t-1$ do:

- *partition every cell that intersects the line segment between the outer endpoint of b_{t-j} and point u_{t-j} by the supporting line of b_{t-j}.*

Proposition 5. *For a conformal path $\beta = (u_0, u_1, \dots, u_t)$, $\texttt{ChainBSP}(\beta)$ cuts every boundary segment at most once. Specifically, only the first step, a partition along the supporting line of $u_{t-1}u_t$, may cut boundary segments.*

Proof. In the first step of $\texttt{ChainBSP}(\beta)$, the partition along the supporting line of b_t may cut other boundary segments. In any subsequent step, a partition along the supporting line of b_{t-j}, $j = 1, 2, \dots, t-1$, cuts only those segments in S that cross the part of $\bar{b}_{t-j}$ between the inner endpoint of b_{t-j} and point v_{t-j}. Since that $\bar{b}_{t-j}$ does not cross any boundary segment, these partition steps do not cut boundary segments. □

However, $\texttt{ChainBSP}(\beta)$ for a conformal path $\beta = (u_0, u_1, \dots, u_t)$, may cut interior segments up to t times: For example if (1) β is a zig-zag path, which alternately turns left and right (Fig. 2(a)); or if (2) for $j = 1, 2, \dots, t$, the part of $\bar{b}_j$ between the inner endpoint of b_j and u_j crosses an interior segment (Fig. 2(b)).

In Subsection 3.6, we will "simplify" a conformal path β to another conformal path δ, and show that $\texttt{ChainBSP}(\delta)$ cuts *every* input segment $O(1)$ times. We will also see the simplified path δ preserves some important properties of β.

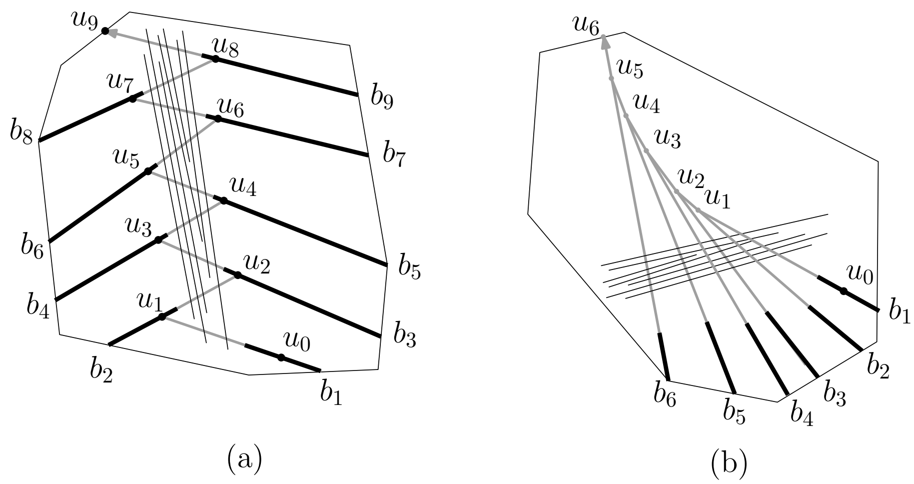


Figure 2: ChainBSP(β) for a conformal path $\beta = (u_0, u_1, \ldots, u_t)$ may cut some interior segments up to t times.

3 Proof of Lemma 3

Let B be a set of m boundary segments in a convex cell C, and let k be an integer, $1 \leq k \leq m$. In this section, we construct algorithm SubBSP(B, C, k) described in Lemma 3. The construction of SubBSP(B, C, k) is composed of several steps. In Subsection 3.1, we construct polygonal paths α_i, each starting from the outer endpoint of a boundary segment (these paths are pairwise non-crossing but may partially overlap). In Subsection 3.2, we select a subset of these paths that decompose cell C into faces, each of which intersects at most $m/(2k) - 1$ boundary segments. The union of the selected paths may consist of several connected components. We reduce the problem to a single connected component in Subsection 3.3. We decompose the union of the paths α_i into a collection of non-overlapping conformal paths β_i in Subsection 3.5, and simplify each path to conformal paths δ_i in Subsection 3.6. We construct algorithm SubBSP(B, C, k) as a concatenation of subroutines ChainBSP(δ_i) for the simplified paths δ_i.

3.1 Polygonal paths

Label the boundary segments arbitrarily as $B = \{s_1, s_2, \ldots, s_m\}$. We successively extend every boundary segment s_i along $\overrightarrow{s}_i$ until the extension hits another segment, the boundary of C, or a previous extension.

Algorithm 2. ConvexPartition(B)
For $i = 1$ to m, do:

- *Let* $\mathrm{ext}(s_i)$ *be the line segment along* $\overrightarrow{s}_i$ *between the inner endpoint of* s_i *and the first point along* $\overrightarrow{s}_i$ *that lies in* $(\bigcup_{j=1}^{m} s_j) \cup (\bigcup_{j=1}^{i-1} \mathrm{ext}(s_j)) \cup \partial C$.

We say that $\mathrm{ext}(s_i)$ is the *extension* of segment s_i; and the union $\hat{s}_i = s_i \cup \mathrm{ext}(s_i)$ is an *extended segment* of s_i for $i = 1, 2, \ldots m$. It is clear that $s_i \subset \hat{s}_i \subseteq \overline{s}_i$. By construction, the relative interiors of the extended segments $\hat{s}_i$, $s_i \in B$, are pairwise disjoint. Direct each $\hat{s}_i$ from the outer endpoint of s_i (tail) to its other endpoint (head). The head of each $\hat{s}_i$ lies either in the relative interior of another extended segment $\hat{s_i}$ or on ∂C. For each $s_i \in B$, we construct a path α_i that follows the directions of the extended segments (refer to Fig. 3(b)). The following algorithm computes the vertices of α_i.

Algorithm 3. `PathBuilder`(i, R)
Let the first vertex of α_i be the outer endpoint of s_i, and let $s := \hat{s}_i$.
Until the head of s lies along ∂C or in the relative interior of a previous edge of α_i.

- *Append the head of segment s to α_i,*
- *Let $b \in B$ be the boundary segment such that the head of s lies in the relative interior of $\hat{b}$, and set $s := \hat{b}$.*

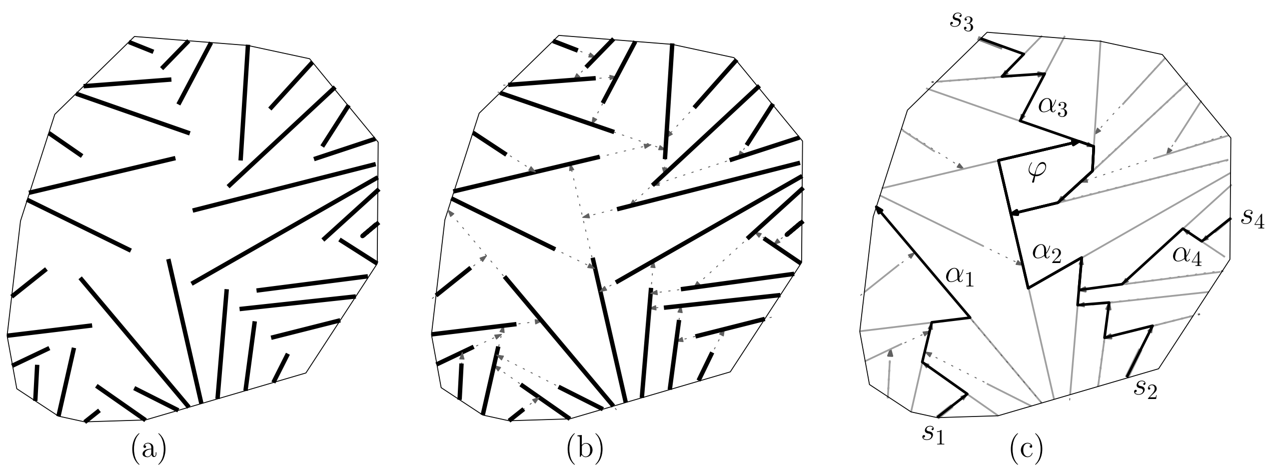


Figure 3: (a) A set of boundary segments (interior segments are not displayed). (b) The boundary segments are successfully extended in an arbitrary order. (c) Paths α_i, for segments s_i, $1 \leq i \leq 4$.

It is clear that each α_i fully contains segment s_i, and its only possible self-intersection is at its last vertex. The paths α_i are pairwise noncrossing, but they may overlap with each other. We prove a few structural properties of the α_i's.

Proposition 6. *If a path α_i, $1 \leq i \leq m$, terminates at a point q in the interior of C, then α_i is composed of a conformal path α'_i from the outer endpoint of s_i to q, and a directed* convex *cycle φ_i.*

Proof. The portion α'_i of path α_i from the outer endpoint of s_i to point q is simple by construction. Path α_i terminates at a point q in the interior of C only if α'_i has already passed through q, and so $\alpha \setminus \alpha_i$ is a simple cycle. Let $\varphi_i = \alpha \setminus \alpha'_i$. It remains to prove that cycle φ_i is convex. Let

$$R_i = \bigcup \{\hat{s}_j : \alpha_i \cap \hat{s}_j \neq \emptyset\}$$

denote the union all extended boundary segments visited by path α_i Note that R_i is the union of line segments $\hat{b}_j$ whose endpoints lie either on ∂C or in the relative interior of another segment in R_i. It follows that R_i decomposes C into convex faces. Cycle φ_i is the boundary of one of these faces, and so it is convex. □

In order to handle all cases uniformly, we define φ_i for *every* path α_i. If a path α_i terminates at a point $\tau_i \in \partial C$, then we let $\varphi_i = \tau_i$ be a degenerate convex cycle, otherwise φ_i is a (nondegenerate) convex cycle lying in the interior of C.

Proposition 7. *For every $1 \leq i < j \leq m$, if paths α_i and α_j intersect, then $\varphi_i = \varphi_j$.*

Proof. Suppose α_i and α_j, $i < j$, intersect. Let q be the first point along α_i that is part of α_j. If $q \in \alpha'_j$, then α_i follows α'_j from q to the cycle φ_j and then follows φ_j, making a full turn around φ_j. If $q \in \varphi_j$, then α_i follows φ_j, making a full turn around φ_j. In both cases, we have $\varphi_i = \varphi_j$. $\square$

Proposition 8. *Let $\psi = (u_0, u_1, \ldots, u_t)$ be a subpath of α'_i for some $1 \le i \le m$. Then the ray $\overrightarrow{u_{t-1}u_t}$ cannot cross path ψ.*

Proof. Assume to the contrary that ray $\overrightarrow{u_{t-1}u_t}$ crosses path ψ. Let x be their first intersection point along $\overrightarrow{u_{t-1}u_t}$. Let D be the simple polygon bounded by the portion of ψ from x to u_t and by segment $u_t x$. It lies in the interior of C.

Let π be the portion of α_i from u_t to the last vertex of α_i. We claim that π lies inside polygon D. Suppose, to the contrary, that path π crosses ∂D, and let y be the first point where π leaves D. Since both π and ψ are part of α_i, we have $y \in u_t x$. Since $\pi \subset \alpha_i$, there is a boundary segment $s \in B$ such that $y \in \hat{s}$ and $\hat{s}$ enters the exterior of D at y. Segment $\hat{s}$ comes from ∂C, and so it has to enter D before it leaves D at y. Hence $\hat{s}$ crosses path ψ: a contradiction, which proves the claim.

If π remains inside D, then the convex cycle $\varphi_i \subset \pi$ also lies inside D. Now let y be the closest vertex of φ_i to the supporting line of $u_t x$, and let e be the directed edge of φ incident to y. Since $\pi \subset \alpha_i$, there is a boundary segment $s \in B$ such that $e \subset \hat{s}$ and $\hat{s}$ has the same direction as φ. Therefore, the part of $\hat{s}$ between the outer endpoint of s and y has to cross path ψ. A contradiction, which proves that $\overrightarrow{u_{t-1}u_t}$ cannot cross path ψ. $\square$

3.2 Selecting paths

Let $R = \bigcup_{i=1}^{m} \alpha_i$ be the union of *all* paths α_i, $i = 1, 2, \ldots, m$. For every subset $P \subseteq \{1, 2, \ldots, m\}$, let $R_P = \bigcup_{i \in P} \alpha_i$. The diagram R_P decomposes cell C into a set F_P of *faces*, which are the connected components of $C \setminus R_P$ (see Fig. 3(c)). Two faces are *adjacent* if their boundaries intersect; they are *adjacent along the boundary* if they are both incident to an outer endpoint of some boundary segment s_i, $i \in P$. We define a *dual graph* G_P, where the nodes correspond to the faces in F_P, and two nodes are adjacent if and only if the corresponding faces are adjacent along the boundary. Note that G_P is not necessarily connected: every face of F_P in the interior of C corresponds to an isolated node in G_P.

Proposition 9. *Let $P \subset \{1, 2, \ldots, m\}$ with $i \notin P$, and let $P' = P \cup \{i\}$. Then $F_{P'}$ can be constructed from F_P by splitting the face in F_P incident to the outer endpoint of s_i into two faces.*

Proof. Path α_i starts from the outer endpoint of s_i, this point is disjoint from R_P. First assume that α_i is disjoint from R_P. It is either a simple path connecting two points on ∂C, or a composition of a convex loop φ_i in the interior of of C and a simple path α'_i between ∂C and φ. In either case, the insertion of α_i splits a face of F_P into two faces. Next assume that α_i intersects R_P. If it intersects some path α_j, $j \in P$, then $\varphi_i = \varphi_j$, and so the part of α_i that is disjoint from R_P is a simple path connecting two points on the boundary of a face in F_P. Again, the insertion of α_i splits a face of F_P into two faces. $\square$

Corollary 1. *The subdivision F_P has $|P| + 1$ faces.*

Proof. We can construct the diagram R_P by successively inserting the paths α_i, $i \in P$. By Proposition 9, the insertion of each path α_i, $i \in P$, increases the number of faces by one. $\square$

A path α_i does not properly cross any boundary segment. If α_i intersects a boundary segment s_j, then it follows s_j to the head of $\hat{s}_j$. So every boundary segment intersects (the interior of) at most one face in F_P. The following algorithm selects a set $P \subset \{1, 2, \ldots, m\}$ by elimination, for a given integer $k \in \mathbb{N}$, such that each face in F_P intersects at most $m/(2k) - 1$ boundary segments.

Algorithm 4. `PathSelector`(B, C, k)
Let $P := \{1, 2, \ldots, m\}$.
For $i = 1$ *to* m, *do*

- *If the faces of* F_P *incident to the outer endpoint of* s_i *jointly intersect at most* $m/(2k) - 2$ *boundary segments, then let* $P := P \setminus \{i\}$.

Output P.

Proposition 10. *Let* $P =$ `PathSelector`(B, C, k).

- *Every face* $f \in F_P$ *intersects at most* $m/(2k) - 1$ *boundary segments.*
- *If* $f_1, f_2 \in F_P$, $f_1 \neq f_2$, *are adjacent along the boundary, then they jointly intersect at least* $m/(2k) - 1$ *boundary segments.*

Proof. Initially, we have $P = \{1, 2, \ldots, m\}$, and no face in F_P intersects any boundary segment. Whenever index i is removed from P, two faces are merged into one face, which intersects at most $m/(2k) - 1$ boundary segments (the boundary segments intersecting the two faces and segment s_i).

Let f_1' and f_2' be the faces in F_P incident to the boundary endpoint of s_i at step i of Algorithm `PathSelector`(B, C, k). By Proposition 9, the removal of index i from P would merge faces f_1' and f_2' into a single face, which would also intersect the segment s_i. If index i is not removed from P, then the faces f_1' and f_2' jointly intersect at least $m/(2k) - 1$ boundary segments at that time. Even if `PathSelector`(B, C, k) later merges f_1 and f_2 with other faces, the two faces f_1 and f_2 incident to the outer endpoint of s_i jointly intersect at least $m/(2k) - 1$ boundary segments. □

3.3 Dual graphs of connected components

We define *another* dual graph, this time on the connected components of R_P (refer to Fig. 4). Let H_P be a graph whose nodes correspond to the *connected components* of R_P, two nodes are connected by an edge if and only if the corresponding components are adjacent to the same face in F_P.

The boundary of a face in F_P consists of portions of some connected components of R_P and portions of ∂C. If a face $f \in F_P$ is adjacent to h components of R_P, then ∂f contains h connected portions of ∂C. A chord between two arbitrary points in two distinct portions of ∂C along ∂f separates some components of R_P. It follows that graph H_P is a tree, since it is connected and every edge is a bridge.

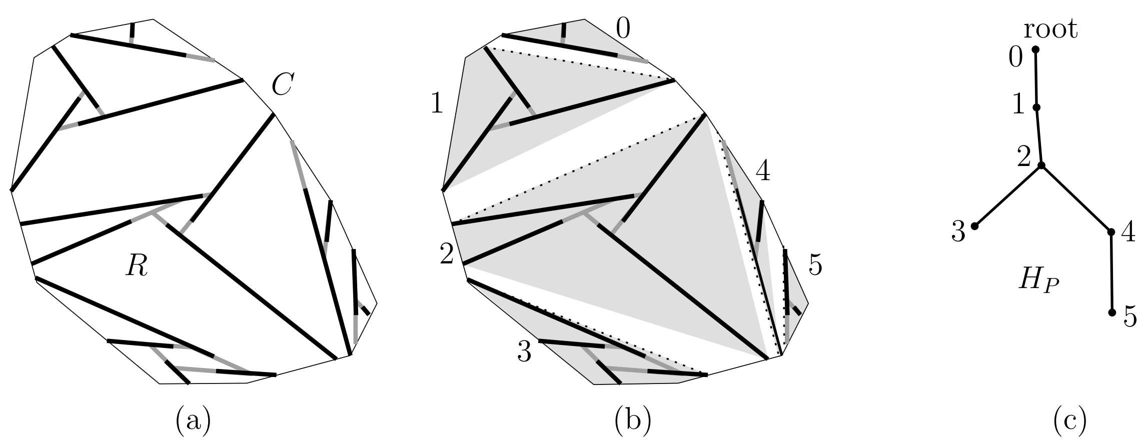


Figure 4: (a) The components of R in a cell C. (b) The convex hull of each component of R (light gray), and chords of ∂C separating each component from the root (dotted lines). (c) The dual graph H_P.

For any set $P \subseteq \{1, 2, \ldots, m\}$, each connected component of R_P is either a directed tree (consisting of paths α_i that all terminate at the same point $\tau \in \partial C$), or a directed cycle and directed trees entering the cycle (consisting of paths α_i that all terminate in a convex cycle φ). Each connected component of $R = \bigcup_{i=1}^m \alpha_i$ contains at most one component of R_P.

Choose an arbitrary component $R_0 \subseteq R_P$, and let R_0 be the *root* in the graph H_P. For every component $R'_P \subset R_P$, $R'_P \neq R_0$, let R' be the unique component of R containing R'_P, and let $e(R')$ be the edge of $\mathrm{conv}(R')$ that separates R' from R_0 in C. The chords $e(R'_P)$ for the nonroot components $R'_P \subset R_P$ decompose C into convex regions, which we call *sectors*. Each sector contains exactly one component of R_P. We can separate the sectors by making a cut along $e(R'_P)$ for every component $R'_P \subset R_P$, $R'_P \neq R_0$.

The following lemma establishes a link between the dual graph G_P defined on the faces F_P of $C \setminus R_P$ and the dual graph H_P defined on the sectors of C.

Lemma 11. *Let $P =$ `PathSelector`(B, C, k), and let $Q_1, Q_2, \ldots, Q_\ell$ be connected components of R_P that correspond to a simple path in the dual graph H_P. Then $Q_1, Q_2, \ldots, Q_\ell$ jointly contain at most $10k$ paths α_i, $i \in P$.*

Proof. If $m \leq 10k$, then our proof if complete. Assume $m > 10k$. Suppose that $Q_1, Q_2, \ldots, Q_\ell$ contains $h_1, h_2, \ldots, h_\ell \in \mathbb{N}$ paths α_i, respectively, with $h = \sum_{j=1}^{\ell} h_j$. We need to show that $h \leq 10k$. Consider the faces in F_P that are adjacent to a component Q_j, for some $1 \leq j \leq \ell$. If Q_j is a directed tree, consisting of paths α_i that all terminate at the same point $\tau \in \partial C$, then Q_j is adjacent to $h_j + 1$ faces in F_P, and they span a path in G_P. If Q_j consists of paths α_i that all terminate in a convex cycle φ in the interior of C, then Q_j is adjacent to $h_j + 1$ faces in F_P, one of which is an isolated node and $h_j \geq 2$ faces span a cycle in G_P.

Since Q_j and Q_{j+1} are adjacent in H_P, there is one common face adjacent to both Q_j and Q_{j+1}. The faces adjacent to two consecutive components are distinct. The set of all faces of F_P adjacent to $Q_1, Q_2, \ldots, Q_\ell$ form a chain of paths and cycles in G_P, such that every two consecutive paths or cycles share a distinct node. Let G'_P denote this subgraph of G_P. It has $(\sum_{j=1}^{\ell} h_j) - (\ell - 1) \geq \lceil h/2 \rceil$ nodes. It contains a matching that covers at least half of its nodes, at least $\lceil h/4 \rceil$ nodes in G'_P.

Apply Proposition 10 for each of the $\lceil h/4 \rceil$ disjoint pairs of faces in the matching in G'_P. Then the faces corresponding to G'_P intersect at least $\lceil h/4 \rceil \cdot (m/(2k) - 1) \leq hm/(10k)$ boundary segments. There are m boundary segments in total. This gives $hm/(10k) \leq m$ and $h \leq 10k$, as required. $\square$

3.4 Reduction to a single component of R_P

In Section 3.2, we have selected a set of indices $P \subseteq \{1, 2, \ldots, m\}$ such that every face in F_P intersects at most $m/(2k) - 1$ boundary segments. The diagram $R_P = \bigcup_{i \in P} \alpha_i$ may consist of several connected components. In this section, we separate the component of R_P from each other, and reduce to a single component. Recall that each component of R'_P lies in a unique sector of cell C. For a single component $R'_P \subseteq R_P$, we will prove the following lemma.

Lemma 12. *Let R'_P be a connected component of R_P containing $h \in \mathbb{N}$ paths α_i. Let $C' \subseteq C$ be the sector of C that contains R'_P. There is a BSP algorithm `CompBSP`(B, C', R'_P) such that*

(i) *every boundary segment lying in C' is cut $O(1)$ times;*

(ii) *every interior segment is cut at most $O(h)$ times;*

(iii) *every cell produced by `CompBSP`(B, C', R'_P) intersects less than m/k boundary segments.*

We can now compose the partition algorithm `SubBSP`(B, C, k) from `CompBSP`(B, C', R'_P).

Algorithm 5. $\texttt{SubBSP}(B, C, k)$
1. *for every nonleaf component $R'_P \subseteq R_P$ in H_P, partition along chord $e(R'_P)$.*
2. *For every component $R'_P \subseteq R_P$, call $\texttt{CompBSP}(B, C', R'_P)$*

Proof of Lemma 3: The chords $e(R'_P)$ for all components R'_P of R_P decompose C into convex sectors, each of which contains a unique component of R_P. Consider a boundary segment $b \in B$. Since the chords between sectors do not cross any boundary segment, b lies in a unique sector. Assume that b lies in a sector C' containing component $R'_P \subseteq R_P$. By Proposition 16, $\texttt{CompBSP}(B, C', R'_P)$ cuts b at most $O(1)$ times.

Consider an interior segment $s \in S$. Assume that s intersects the sectors $C_1, C_2, \ldots, C_\ell$ containing components $Q_1, Q_2, \ldots, Q_\ell$ of R_P, respectively. These sectors correspond to a simple path in the dual graph H_P. By Lemma 11, they jointly contain at most $10k$ paths α_i, $i \in P$. By Proposition 16, if Q_j contains h_j paths α_i, $i \in P$, then $\texttt{CompBSP}(B, C_j, Q_j)$ cuts s at most $O(h_j)$ times. So s is cut $\sum_{i=1}^{q} O(h_j) = O(k)$ times.

By Lemma 12, every cell produced by $\texttt{CompBSP}(B, C', R'_P)$ intersects less than m/k boundary segments. □

3.5 Processing one components of R_P

Consider a connected component R'_P of R_P. Let R' be the component of R containing R'_P. We may assume (by relabeling the paths α_i if necessary) that R'_P is the union of the paths $\alpha_1, \alpha_2, \ldots, \alpha_h$; and R' is the union of the paths $\alpha_1, \alpha_2, \ldots, \alpha_r$ for $1 \leq h \leq r \leq m$. Recall that $R'_P = \bigcup_{i=1}^{h} \alpha_i$ is a collection of directed trees entering a directed cycle φ, where φ is either a point on ∂C or a convex cycle in the interior of C.

Decomposing R'_P into non-overlapping paths. Every vertex in R'_P has out-degree one. Let $Q \subset R'_P$ be a set of all vertices of degree at least three (empty dots in Fig 5(b)). The deletion of all points in Q decomposes R'_P into conformal paths. Denote by Γ the set of these paths, and let $g = |\Gamma|$. By construction, if two paths in Γ intersect, then they intersect only at their first or last vertex. We show that $h \leq g \leq 2h + 1$. The paths α_i, $i = 1, 2, \ldots, h$, start from distinct points on ∂C, hence $h \leq g$. If R'_P is a tree with h leafs, rooted at $\tau \in \partial C$, then R'_P decomposes into $2h - 1$ paths. If R'_P contains a cycle φ in the interior of C, then R'_P decomposes into at most $2h + 1$ paths.

Label the elements of Γ as follows: If R'_P is acyclic then it terminates at a point $\tau \in \partial C$, otherwise let τ be an arbitrary point in Q along the convex cycle φ. Traverse R'_P in reverse direction starting from τ. At every point $q \in Q$, descend first along the path which is collinear with the (unique) out-going edge at q. Label the directed paths by $\beta_1, \beta_2, \ldots, \beta_g$ in the order in which they are traversed.

The paths β_i, $i = 1, 2, \ldots, g$, are conformal. Since the union of the paths is the component $R'_P \subseteq R_P$, we can prove an additional property for each $\beta \in \Gamma$:

Proposition 13. *Let $(u_0, u_1, \ldots, u_t)$ be a sub-path of a path $\beta \in \Gamma$ such that it makes a right (resp., left) turn at every internal vertex. Then the vertices $u_0, u_1 \ldots, u_t$ are in convex position.*

Proof. If β is part of the convex cycle φ, then the vertices $u_0, u_1, \ldots u_t$ are in convex position. If β is disjoint from φ, then the vertices $u_0, u_1, \ldots, u_t$ are in convex position by Proposition 8. □

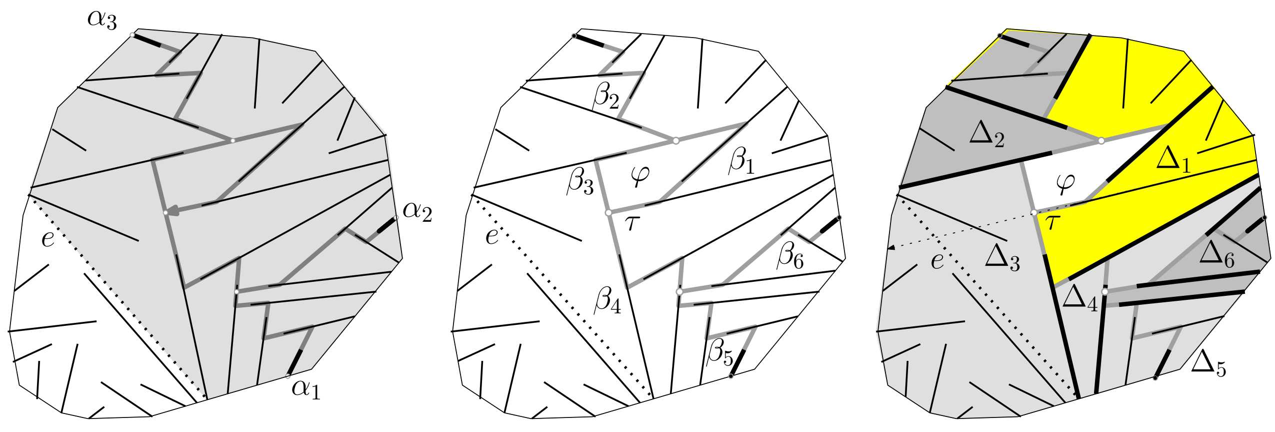


Figure 5: (a) A component R'_P of R_P, its convex hull $\mathrm{conv}(R'_P)$, and a chord $e = e(R'_P)$. (b) R'_P is decomposed into non-overlapping simple paths β_i, $i = 1, 2, \ldots, 7$. (c) Regions Δ_i for $i = 1, 2, \ldots, 7$.

3.6 Path simplification

For every path $\beta \in \Gamma$, we compute a simplified path δ. If β is a straight line segment, then let $\delta = \beta$. Suppose that β has at least three vertices. The directed path $\beta = (u_0, u_1, \ldots, u_t)$, $t \geq 2$, makes either a left or right turn at every internal vertex. In the remainder of this subsection, we assume that the last turn is a *right* turn (the case that it is a *left* turn is analogous). Let $\gamma = (u_\ell, u_{\ell+1}, \ldots, u_t)$ be the maximal suffix path of β that makes right turns only. That is, β makes the last left turn of at u_ℓ, or it makes no left turn at all. Since $\gamma \subseteq \beta$, it is a conformal path. By Proposition 13, the vertices of γ are in convex position.

The following algorithm simplifies γ. In each step, it reduces the number of vertices by one. More precisely, it replaces a subpath (a_0, a_1, a_2, a_3) by (a_0, x, a_3), where x is the intersection point of the supporting line of a_0a_1 and a_2a_3, whenever the resulting path is still conformal.

Algorithm 6. `Simplify`(B, C, γ)*:*

Input: a set B of boundary segments in a convex cell C, and a convex *conformal path $\gamma = (u_\ell, u_{\ell+1}, \ldots, u_t)$ which makes right turns only.*

Set $\delta := \gamma$.

While δ has four consecutive vertices (a_0, a_1, a_2, a_3) such that there are boundary segments $b_1, b_3 \in B$ with $a_0a_1 \subset \overline{b}_1$ and $a_2a_3 \subset \overline{b}_3$; and the intersection point x of the supporting line of b_1 and b_3 lies in the part of $\overline{b}_3$ between the outer endpoint of b_3 and a_2, do:

- *replace the subpath (a_0, a_1, a_2, a_3) by (a_0, x, a_3) in δ.*

Output: δ.

We can now prove the main properties of the simplified path δ.

Proposition 14. *The first (resp., last) vertex of both γ and $\delta =$*`Simplify`(B, C, γ) *is u_ℓ (resp., u_t). The supporting lines of first (resp., last) edge of γ and δ are the same.*

Proof. Initially algorithm `Simplify`(B, C, γ) sets $\delta = \gamma$. The algorithm does not change the first and the last vertex of the path. They also do not change the supporting line of the first and the last edge. □

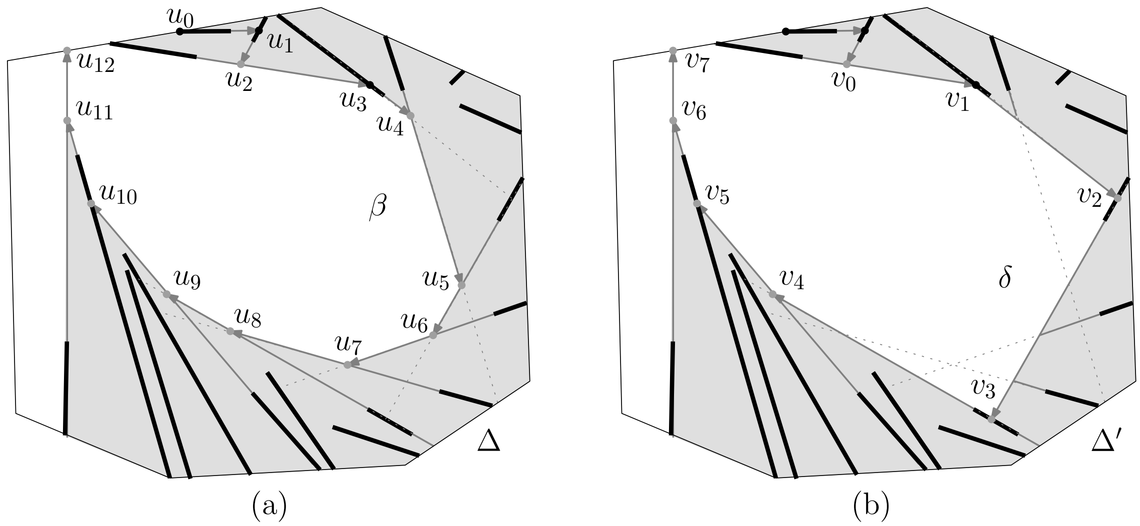


Figure 6: (a) A conformal path $\beta = (u_0, u_1, \ldots, u_{12})$. The vertices of $\gamma = (u_2, u_3, \ldots, u_{12})$ are in convex position. (b) The simplified path $\delta =$`Simplify`$(B, C, \gamma) = (v_0, v_1, \ldots, v_7)$.

For a directed line segment s, let s^- (resp., s^+) denote the closed halfplane bounded by the supporting line of s, lying on left left (resp., right) of s.

Proposition 15. *If γ is a convex conformal path, then $\delta =$*`Simplify`*(B, C, γ) is also a convex conformal path.*

Proof. If γ has two vertices, then $\delta = \gamma$, and there is nothing to prove. Suppose that $\gamma = (u_\ell, u_{\ell+1}, \ldots, u_t)$ has at least three vertices. Initially algorithm `Simplify`(B, C, γ) sets $\delta = \gamma$, so δ is conformal. Every step of the algorithm preserves the property that δ is conformal. Since γ makes a *right* turn at internal vertices, we have $b_j \subset b^-_{j+1}$ for every $j = \ell, \ell+1, \ldots, t-1$. Moreover, if $\overline{b}_j$ intersects $\overline{b}_{j'}$, $\ell \leq j < j' \leq t$, then $b_j \subset b^-_{j'}$. These properties are also preserved in every step of `Simplify`(B, C, γ), and so the vertices of the output δ are also in convex position. $\square$

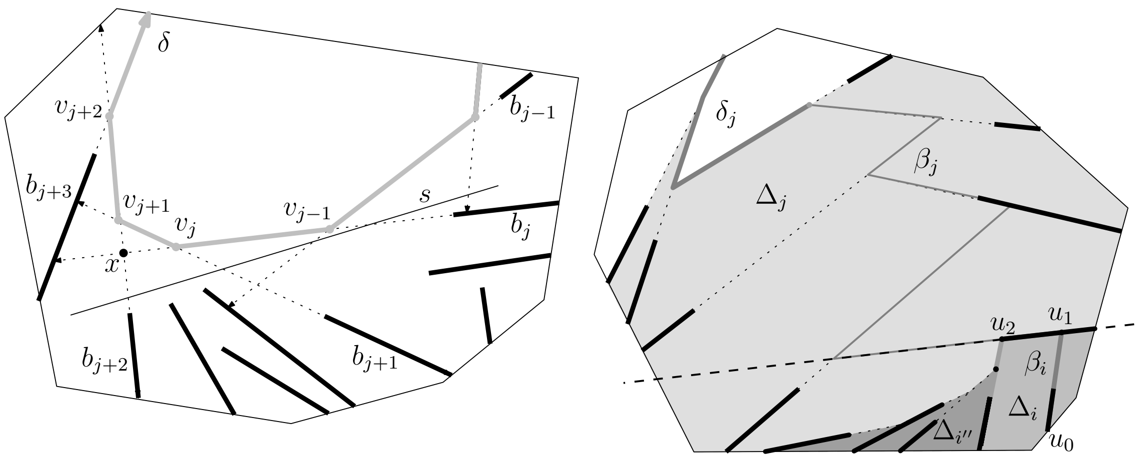
Proposition 16. *If γ is a convex conformal path and $\delta =$*`Simplify`*(B, C, γ), then* `ChainBSP`*(δ) cuts every interior segment $O(1)$ times.*

Proof. If γ has two vertices, then $\delta = \gamma$, and `ChainBSP`(δ) partition along only one line. Consider a conformal convex path γ with at least three vertices. By Proposition 15, δ is a convex conformal path. Let $\delta = (v_0, v_1, \ldots, v_z)$ and let $b_j \in B$ denote the boundary segment such that $v_{j-1}v_j \subset \overline{b}_j$, for $j = 1, 2, \ldots, z$. Denote by $\hat{b}_j$ the part of $\overline{b}_j$ between the outer endpoint of b_j and v_{j-1} (Fig. 7(a)).

Algorithm `ChainBSP`(δ) cuts every segment that crosses the supporting line of b_z, the convex path δ, or segments $\hat{b}_j$ for $j = 1, 2, \ldots, z$. Let $s \in I$ be an interior segment. The supporting line of b_z and the convex path δ jointly cut s at most 3 times, into at most 4 subsegments. Let $s' \subseteq s$ be one of these subsegments. It is enough to show that s' crosses $O(1)$ segments $\hat{b}_j$.

Path δ and segments $\hat{b}_j$ decompose C into regions, each of which is adjacent to exactly two segments $\hat{b}_j$, and all but one are adjacent to two consecutive segments $\hat{b}_j$ and $\hat{b}_{j+1}$. It is enough to show that s' crosses at most two consecutive segments $\hat{b}_j$.

Suppose, by contradiction, that s' crosses three consecutive segments $\hat{b}_j$, $\hat{b}_{j+1}$, and $\hat{b}_{j+2}$. Since γ makes a right turn at v_{j-1} and v_j, we have $\hat{b}_{j+1} \subset b^+_j$ and $\hat{b}_{j+2} \subset b^+_{j+1}$. Therefore, s' crosses both



(a) (b)

Figure 7: (a) Path δ; an internal segment $s \in I$ crosses three consecutive segments $\hat{b}_j$, $\hat{b}_{j+1}$, and $\hat{b}_{j+2}$. (b) The last edge of $\beta_i = (u_0, u_1, u_2)$ is collinear with the first edge of β_j. Domain $\Delta_{i''}$ is in Δ_j but disjoint from Δ_i.

$\hat{b}_{j+1}$ and $\hat{b}_{j+2}$ in the halfplane b_j^+; and the supporting line of b_j intersects $\hat{b}_{j+2}$. Denote by x this intersection point.

Note that $\hat{b}_j$ cannot intersect $\hat{b}_{j+2}$, otherwise $\texttt{Simplify}(B, C, \gamma)$ would have simplified the subpath $(v_{j-1}, v_j, v_{j+1}, v_{j+2})$ to (v_{j-1}, x, v_{j+2}) (Fig. 7(a)). Hence, $\bar{b}_j$ starts from the outer endpoint of b_j, it extends beyond v_j but ends before reaching x. So some boundary segment intersects segment $v_j x$. This boundary segment is either b_{j+2} (which may extend beyond x) or it separates $\hat{b}_{j+1}$ from $\hat{b}_{j+2}$ in the halfplane b_j^+. Since s' is disjoint from all boundary segments, it cannot reach $\hat{b}_{j+2}$. A contradiction, which completes our proof. $\square$

The union of all simplified paths $\bigcup_{i=1}^{g} \delta_i$ decomposes sector C' into a set F'_P of *faces*, which are the connected components of $C \setminus \bigcup_{i=1}^{g} \delta_i$.

Proposition 17. *Every (open) face $f \in F_P$ intersects at most $m/(2k) - 1$ boundary segments.*

Proof. The paths $\gamma_i \in \Gamma$ and $\delta_i = \texttt{Simplify}(B, C, \gamma_i)$ have the same endpoints (Proposition 14), so if $\delta_i \neq \gamma_i$, then they bound a nonempty region in the plane. Since both of them are conformal, they do not cross any boundary segment, and so the interior of the region bounded by γ_i and δ_i is disjoint from boundary segments. The faces in F_P, bounded by $\bigcup_{i=1}^{g} \gamma_i$, and they intersect at most $m/(2k) - 1$ boundary segments, and so the faces in F'_P, bounded by $\bigcup_{i=1}^{g} \delta_i$, also intersect at most $m/(2k) - 1$ boundary segments. $\square$

3.7 Proof of Lemma 12

We can now present the BSP algorithm $\texttt{CompBSP}(B, C', R'_P)$.

Algorithm 7. $\texttt{CompBSP}(B, C', R'_P)$*:*

For $i = 0, 1, 2, \ldots, g$, apply $\texttt{ChainBSP}(\delta_i)$.

We will show that $\texttt{CompBSP}(B, C', R'_P)$ satisfies properties (i)–(iii) in Lemma 12. For every path $\beta_i \in \Gamma$, we define a closed polygonal domains Δ_i. If β_i is a line segment, then let $\Delta_i = \beta_i$ (a degenerate domain). If β_i has at least three vertices, then assume that it makes a *right* turn at its last internal vertex (the case that it makes a left turn is analogous). Hence, γ_i and the simplified path δ_i make right turns only. Let $\delta_i = (v_0, v_1, \ldots, v_z)$, such that $v_0v_1 \subset \hat{b}_1$, and point v_z lies in the relative interior of the extended boundary segment $\hat{b}_{z+1}$. Let $\partial_i C$ denote the counterclockwise portion of ∂C between the outer endpoints of b_1 and b_{z+1}. Let Δ_i be the polygonal domain bounded by segment $\hat{b}_1$, path δ_i, segment $\overline{b}_{t+1}$, and $\partial_i C$ (Fig. 6). We observe a few immediate consequences of the definition.

Proposition 18. *For every path $\beta_i \in \Gamma$, Δ_i fully contains every boundary segment incident to $\partial_i C$; and it does not intersect any other boundary segment.* $\square$

Proposition 19.

(i) *For $1 \leq i < j \leq g$, the domains Δ_i and Δ_j are either interior disjoint or we have $\Delta_j \subsetneq \Delta_i$.*

(ii) *If $\Delta_j \subsetneq \Delta_i$, then $j > i$.*

Proof. Part (i) follows from the fact that the extended boundary segments have pairwise disjoint relative interiors. Part (ii) follows from the ordering of the paths γ_i and γ_j in Γ. $\square$

For every $i = 1, 2, \ldots, g$, let

$$D_i = \Delta_i \setminus \left(\bigcup_{i<j} \Delta_j \right)$$

be the part of Δ_i remaining after removing all domains Δ_j nested in Δ_i.

Proposition 20. *Let $\beta_i = (u_0, u_1, \ldots, u_t) \in \Gamma$, and $\gamma_i = (u_\ell, u_{\ell+1}, \ldots, u_t)$. Then $(u_0, u_1, \ldots, u_\ell) \subset D_i$.*

Proof. Path δ_i is on the boundary of Δ_i by definition. It is the simplification of $\gamma_i = (u_\ell, u_{\ell+1}, \ldots, u_t)$, with $u_\ell = v_0$ and $u_\ell u_{\ell+1} \subseteq v_0v_1$ (Proposition 14). Assume w.l.o.g. that γ_i makes right turns only. Path β does not cross any path δ_j, $j = 1, 2, \ldots g$. Since β makes a *left* turn at u_ℓ, its initial portion $(u_1, u_2, \ldots, u_\ell)$ lies in Δ_i. Its relative interior is disjoint any domain Δ_j, $\Delta_j \subseteq \Delta_i$, and so it must remain in D_i. $\square$

Proposition 21. *All boundary segments that intersect a cell produced by* `CompBSP`(B, C', R'_P)

- *either intersect one face of F'_P; or*
- *intersect two adjacent faces of F'_P or the boundary between those faces.*

Proof. `CompBSP`(B, C', R'_P) makes cuts along the portion of $\partial \Delta_i$ lying in the interior of C. So every cell produced by `CompBSP`(B, C', R'_P) is contained in a face of the arrangement of the boundaries $\partial \Delta_i$, for all $i = 1, 2, \ldots, g$. If a face in this arrangement lies outside of all domains Δ_i, then it lies in a face of F'_P. Otherwise it lies in a domain D_i for some $i = 1, 2, \ldots, g$. By Proposition 20, the only path in Γ that possibly intersects the interior of D_i is γ_i, and so every domain D_i is covered by at most two faces of F'_P (lying on two sides of path γ_i). $\square$

For $i = 1, 2, \ldots, g$, let $\hat{c}_i$ denote the extended boundary segment that contains the last edge of β_i. By Proposition 14, $\hat{c}_i$ contains the last edge of δ_i. By Proposition 5, the only partition step of `ChainBSP`(δ_i) that possibly cuts boundary segments is the partition along the supporting line of $\hat{c}_i$. Therefore, the only partition steps in `CompBSP`(B, C', R'_P) that possibly cut boundary segments are the partitions along the supporting lines of $\hat{c}_i$, for $i = 1, 2, \ldots, g$. We show next that every boundary segment is cut in $O(1)$ of these steps.

Proposition 22. *Suppose that the last vertex of $\beta_i \in \Gamma$ is the first vertex of $\beta_j \in \Gamma$, and the last edge of β_i is collinear with the first edge of β_j. Then the partition along the supporting line of $\hat{c}_i$ does not cut any boundary segment lying in any domain $\Delta_{i'} \subsetneq \Delta_j$.*

Proof. In the ordering of the paths in Γ, we have $j < i$. Assume first that β_j is part of the convex cycle φ. Then $\gamma_j = \beta_j$, and the last edge of γ_j is collinear with the last edge of δ_i. ChainBSP(δ_j) is performed before ChainBSP(δ_i), and so the cut along $\hat{c}_i$ is restricted to the first edge of δ_j, it does not cut any domain $\Delta_{i'}$ nested in Δ_j.

Next assume that β_j is not part of the convex cycle φ. Denote by $b_1 \in B$ be the boundary segment whose extension contains the first edge of γ_i. The path $\hat{b}_1 \cup \gamma_i$ makes right turns only, and by Proposition 8 its vertices are in convex position. Therefore, the supporting line of $\hat{c}_i$ does not cross $\hat{b}_1 \cup \gamma_i$, and so it does not cut any boundary segment lying in Δ_i.

Let $\Delta_{i''}$ be a domain contained in Δ_j but interior-disjoint from Δ_i (Fig. 7(b)). By the ordering of the paths in Γ, we have $j < i < i''$. Suppose that the supporting line of $\hat{c}_i$ crosses $\Delta_{i''}$, and let x be the first point where $\overrightarrow{c}_i$ enters $\Delta_{i''}$. Denote by $b_2 \in B$ the boundary segment whose extension contains x. Path α_2, which starts from the boundary segment of b_2, passes through x and it contains path β_i (since the paths in Γ intersect only at their endpoints and $i < i''$). By Proposition 8, applied to the portion of path α_2 from the outer endpoint of b_2 to the last vertex of β_i, the supporting line of $\hat{c}_i$ cannot cross α_2. A contradiction, which implies that supporting line of $\hat{c}_i$ does not cross $\Delta_{i''}$. □

Proposition 23. *Assume that a boundary segment $s \in B$ is cut while algorithm* CompBSP(B, C', R'_P) *runs subroutine* ChainBSP(δ_i)*, for some $1 \le i \le g$. Then*

- *either $i = 1$; or*
- *the last edge of δ_i is collinear with the first edge of some β_j, $j < i$, and $s \subset D_j$.*

Proof. If $i = 1$, then there is nothing left to prove. Assume $i \neq 1$. By Proposition 5, ChainBSP(δ_i) can cut s only in a partition along the supporting line of $\hat{c}_i$. The last vertex of β_i is the the last vertex of at least another path $\beta_{i'} \in \Gamma$, and it is the first vertex of a path β_j, with $j < \min(i, i')$. If the last edge of β_i and the first edge of β_j are not collinear, then $j < i' < i$ by the ordering of the paths in Γ. Algorithm CompBSP(B, C', R'_P) calls ChainBSP($\delta_{i'}$) before ChainBSP(δ_i). Therefore, the cut along the last edge of β_i does not extend beyond the last vertex of β_i, and so it does not cross any boundary segment. We conclude that the last edge of β_i is the first edge of β_j.

Since ChainBSP(δ_j) is called before ChainBSP(δ_i), the partition along the supporting line of the last edge of δ_i can cut only those segments that lie in Δ_j. By Proposition 22, it does not cut boundary segments in any other domain $\Delta_{i''}$ nested in Δ_j. Hence, segment s must lie in D_j. □

Proof of Lemma 12: By Proposition 5, the only step of ChainBSP(δ_i) that possibly cuts boundary segments is the partition along the supporting line of $\hat{c}_i$. By Proposition 23, the partition along the supporting line of $\hat{c}_i$ can cut only those boundary segments that lie in D_i, for $i = 2, 3, \ldots, g$. Every boundary segment lies in at most one domain D_i. So every boundary segment may be cut once by the partition along the supporting line of $\hat{c}_1$; and once by the partition along the supporting line of $\hat{c}_i$, for at most one of $i = 2, 3, \ldots, g$. This proves part (i).

R'_P is the union of $g \le 2h + 1$ nonoverlapping paths β_i, $1 = 1, 2, \ldots, g$. Every path β_i was simplified to a path δ_i. By Proposition 16, each ChainBSP(δ_i) cuts an interior segment $O(1)$ times. Hence, CompBSP(B, C', R'_P) cuts every interior segment $O(h)$ times. This proves (ii).

Consider a cell $C'' \subseteq C'$ produced by CompBSP($B, C'R'_P$). By Proposition 21, the boundary segment that intersect cell C'' intersect one or two faces in F_P, and the boundary between those faces. By Proposition 10, each face of F_P intersects at most $m/(2k) - 1$ boundary segments. So C''

intersects at most $2(m/(2k)-1)+1 = m/k-1$ boundary segment: the segments intersecting two adjacent faces in F_P, and at most one boundary segment along the common boundary of the two faces. This proves part (iii). $\square$

4 Proof of Theorem 1

In this section, we apply $\texttt{SubBSP}(B, C, k)$ repeatedly to construct a BSP of size $O(n \log n / \log\log n)$ for n disjoint segments in the plane. The input of algorithm $\texttt{BSP}(S, C, k)$ below is a set S of $n \geq 1$ of disjoint line segments lying in a convex cell C, and k is an integer. For a convex cell C', denote by $S(C')$ the set of the segments in S clipped in C', that is, $S(C') = \{s \cap C' : s \in S, s \cap C' \neq \emptyset\}$.

Algorithm 8. $\texttt{BSP}(S, C, k)$

1. *If* $|B| > 0$,

   then call $\texttt{SubBSP}(B_S, C, \min(|B_S|, k))$,

   else partition C *along the supporting line of an arbitrary segment* $s \in S$.

2. *For each cell* C' *produced in step 1 with* $S(C') \neq \emptyset$, *call* $\texttt{BSP}(C', S(C'), k)$.

Lemma 24. *For a set* S_0 *of* n *disjoint line segments in a convex cell* C_0, $\texttt{BSP}\Big(S_0, C_0, \Big\lceil \frac{\log n}{\log\log n}\Big\rceil\Big)$ *is a BSP for* S, *and it partitions* S *into* $O(n \log n / \log\log n)$ *fragments.*

Proof. Let $k = \lceil \log n / \log\log n \rceil$. We may assume w.l.o.g. that $k \geq 4$. Let T be the tree of recursion corresponding to algorithm $\texttt{BSP}(S_0, C_0, k)$, where each node v corresponds to a subproblem (S_v, C_v) for which we either call $\texttt{BSP}(S_v, C_v, k)$ or partition along an arbitrary segment in S_v.

Consider a segment $s \in S_0$. If s lies in the interior of C_0, then there is a step $v \in T$ in the recursion where s is first cut into fragments. By Lemma 3, s it cut at most $O(k) = O(\log n / \log\log n)$ times at this step. If a fragment of s appears in any subsequent level of the recursion, then it is a boundary segment whose inner endpoint is an endpoint of s. That is, at most two fragments of s occurs at any level (all other fragments are free in their respective subproblems, and are not fragmented any further). Since the surviving fragments of s may get shorter at each level of the recursion, we use the *endpoint* of s for identifying which input segment it belongs to.

Motivation for a charging scheme. By Lemma 3, a boundary segment is cut at most $O(1)$ times in each level of the recursion It is enough to show that every boundary segment (identified by an endpoint of an input segment) survives *on average* $O(\log n / \log\log n)$ levels of the recursion. The intuition for this is the following: If $|B| = n$ and $I = \emptyset$, then the number of boundary segments in the subproblems decreases by a factor of k. Hence after $\log_k n = O(\log n / \log\log n)$ levels of the recursion, we have no more boundary segments, and the BSP is complete. Unfortunately, we cannot assume $I = \emptyset$: Each recursion step may cut some interior segments whose two extremal fragments may become boundary segments in the subproblems.

In the remainder of the proof, we introduce a charging scheme that charges each event that *a partition line crosses a segment in* B_S to an endpoint of an input segment $s \in S_0$. It is enough to show that every segment endpoint is charged $O(k)$ times.

The charging scheme. Let $V(S_0)$ be the set of the $2n$ endpoints of the n input segments. In a top-down traversal of the recursion tree T, we will construct a collection $\mathcal{A}$ of pairwise disjoint subsets of $V(S_0)$. We assign each subproblem (S, C) to a subset $A \in \mathcal{A}$ such that $|B_S| \leq 4|A|$.

$\texttt{SubBSP}(S, C, k)$ performs $O(|B_S|)$ cuts on the segments in B_S. We charge these $O(|B_S|)$ cuts to the endpoints in A, such that each endpoint in A is charged at most $O(1)$ times. Finally we will show that each segment endpoint in A is charged at most $O(k)$ times.

For each subproblem (S, C), we recursively define the *status* of each boundary segment $s \in B_S$ as *juvenile*, *active*, or *retired*. Along the way, we also select the pairwise disjoint sets in $\mathcal{A}$, which are the endpoints of the *active* segments in certain subproblems. Initially, all segments in B_{S_0} are active in the initial problem (S_0, C_0); let the first subset $A_0 \in \mathcal{A}$ contain all inner endpoints of the segments in B_{S_0}, and the initial problem (S_0, C_0) is assigned to $A_0 \in \mathcal{A}$. If the status of every segment of B_S is already defined in a problem (S, C), then we can define the status of each segment in $B_{S'}$, where S' is a child of S, as follows: Recall that every segment $s' \in B_{S'}$ is the part of some segment $s \in B_S \cup I_S$ adjacent to an inner endpoint of s. If $s' \in B_{S'}$ comes from a segment $s \in I_S$, then it *tentatively* becomes *juvenile*. If $s' \in B_{S'}$ comes from a segment $s \in B_S$, then it *tentatively* receives the same status as s has in the subproblem (S, C).

- If fewer than half of the segments in $B_{S'}$ are tentatively *juvenile*, then (i) each segment in $B_{S'}$ takes its tentative status (*juvenile*, *active*, or *retired*) and (ii) the subproblem (S', C') is assigned to the same set in $\mathcal{A}$ as its parent (S, C).
- If at least half of the segments in $B_{S'}$ are tentatively juvenile, then (i) all tentatively retired or active segments in $B_{S'}$ become *retired*, (ii) all tentatively juvenile segments in $B_{S'}$ become *active*, (iii) we create a new set $A \in \mathcal{A}$ containing the inner endpoints of all tentatively juvenile (i.e., active) segments in $B_{S'}$, and (iv) the subproblem (S', C') is assigned to this new set $A \in \mathcal{A}$.

Each segment is charged at most $O(k)$ times. When a set $A \in \mathcal{A}$ is created at a problem (S, C), then A is a set of inner endpoints of the *active* segments, and all other segments in B_S are retired. In any subproblem assigned to A, every active or retired segment is part of a segment in B_S. Hence, at any level of the recursion, the total number of active segments in all subproblems assigned to A is at most $|A|$. In problem (S, C), there are at most $|A|$ retired segments in B_S. Therefore, at any level of the recursion, the total number of retired segments in all subproblems assigned to A is at most $|A|$. In each subproblem assigned to A, there are fewer juvenile segments than retired and active segments together. Therefore, at any level of the recursion, the total number of juvenile segments in all subproblems assigned to A is less than $2|A|$. Altogether, at any level of the recursion, there are fewer than $4|A|$ segments in $B_{S'}$ over all subproblems (S', C') assigned to A. It is enough to show that subproblems from at most $O(k)$ different levels of the recursion are assigned to A.

Assume that problem (S, C) is assigned to A, and one of its children (S', C') is also assigned to A. If $|B_S| > 0$, then $\texttt{BSP}(S, C, k)$ calls $\texttt{SubBSP}(B_S, C, k)$. By Lemma 3, parts of at most $|B_S|/k$ segments of B_S become elements of $B_{S'}$. If S' is still assigned to the same set $A \in \mathcal{A}$, then $|B_{S'}| \leq 2|B_S|/k$, since $B_{S'}$ has at most $|B_S|/k$ segments coming from B_S and at most the same number of segments coming from I_S. That is, the number of boundary segments in a subproblem decreases by a factor of at least $2/k$ in each step of the recursion. We have $|A| \leq n$, at the step where A is created, and we also have $|B_S| \leq n$ at that time. The cardinality of B_S can decrease by a factor of $2/k$ at most $\log_{k/2} n = \log n / \log(k/2) = O(\log n / \log\log n) = O(k)$ times. This proves that subproblems from at most $O(k)$ levels are assigned to any set $A \in \mathcal{A}$, and so each segment in A is charged at most $O(k)$ times. $\square$

5 Auto-partitions

We have presented a BSP of size $O(n \log n / \log \log n)$ for n disjoint line segments in the plane. In this section, we show how to adjust this algorithm to obtain an *auto-partition* of size $O(n \log n / \log \log n)$. Our BSP repeatedly called algorithm SubBSP(B, C, k), which separated the sectors of the input cell C along chords of ∂C and called ChainBSP(δ_i) for a sequence of carefully selected conformal paths δ_i. There are essentially two reasons why our BSP may not be an auto-partition: First, a chord of ∂C is typically not collinear with any input segment. Second, ChainBSP(δ) might partition a cell along a line spanned by a segment that is not present in that cell.

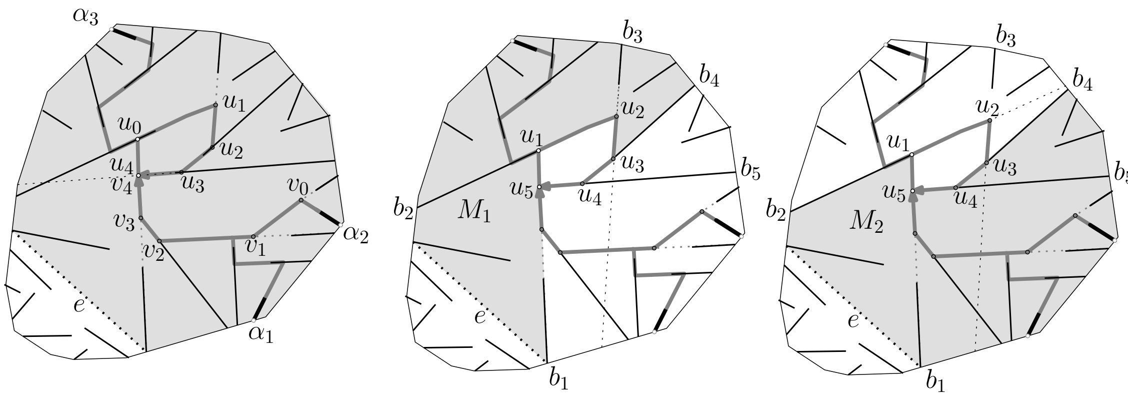


Figure 8: (a) Two conformal paths $\delta_1 = (u_0, u_1, \ldots, u_4)$ and $\delta_2 = (v_0, v_1, \ldots, v_4)$. ChainBSP$(\delta_1)$ is not an auto-partition. ChainBSP(δ_2) is an auto-partition. (b)–(c) Domains M_1 and M_2 both contain chord e.

Let $\delta = (v_0, v_1, \ldots, v_z)$ be a convex conformal path such that $v_{j-1}v_j \subset \overline{b}_j$ for a boundary segment $b_j \in B$. Recall that ChainBSP(δ) partitions every cell that intersects the line segment between the outer endpoint of b_j and point u_j by the supporting line of b_j, for $j = z, z-1, \ldots, 1$. We call these segments the *generator segments* of ChainBSP(δ). Every generator of ChainBSP(δ) contains an input segment. Clearly, ChainBSP(δ) is an auto-partition if no generator segment is cut before ChainBSP(δ) performs the partition along the supporting line of this generator. We can use ChainBSP(δ) as the basic building block of an *auto-partition* if the paths δ satisfy some simple conditions.

Proposition 25. *Let $\delta = (v_0, v_1, \ldots, v_z)$ be a convex conformal path. Assume that δ and the boundary segments b_j with $v_{j-1}v_j \subset \overline{b}_j$, for $j = 1, 2, \ldots, z$, lie in a single cell.* ChainBSP(δ) *is an auto-partition if the supporting line of b_z does not cross the part of $\overline{b}_1$ between the outer endpoint of b_1 and point v_1.*

Proof. After each partition step, all remaining generator segments of ChainBSP(δ) lie in a single cell. Every cell is partitioned along a boundary segment in that cell, and so ChainBSP(δ) is an auto-partition. □

Proposition 26. *Let $\gamma_i \in \Gamma$, and $\delta_i =$*Simplify(B, C, γ_i)*. Assume that $\delta_i = (v_0, v_1, \ldots, v_z)$ and the boundary segments b_j with $v_{j-1}v_j \subset \overline{b}_j$, for $j = 1, 2, \ldots, z$, lie in a single cell.* ChainBSP(δ_i) *is an auto-partition if*

- *γ_i is not part of the convex cycle φ, or*
- *γ_i is part of the convex cycle φ but the turning angle of γ_i at most* $180°$.

Proof. By Proposition 25, it is enough to check whether the supporting line of b_z crosses the portion of $\bar{b}_1$ between the outer endpoint of b_1 and v_1. The first part follows from Proposition 8. The second part is immediate. $\square$

In particular, if R_P has only one component (i.e., no cuts are made along chords of ∂C) and it is a tree (all paths α_i terminate in a point along ∂C), then $\texttt{SubBSP}(B, C, k)$ is an auto-partition. However, if R_P has several components or contains cycles, then we adjust $\texttt{SubBSP}(B, C, k)$ to obtain an auto-partition. The following strengthening of Lemma 3 combined with Section 4 readily implies Theorem 2.

Lemma 27. *Let S be a finite set of disjoint line segments in the plane. For every integer k, $1 \le k \le |B|$, there is an* auto-partition $\texttt{SubAuto}(B, C, k)$ *such that*

- *every boundary segment in B is cut at most $O(1)$ times* on average*;*
- *every interior segment in I is cut at most $O(k)$ times;*
- *every cell produced by $\texttt{SubAuto}(B, C, k)$ intersects at most $|B|/k$ segments in B.*

The proof of Lemma 27 is analogous to that of Lemma 3. As described in Subsections 3.1–3.3, we compute the extensions of all boundary segments, the paths α_i, a subset $P = \texttt{PathSelector}(B, C, k)$, and the dual graphs H_P (which is a tree). Compute chords of ∂C that partition C into convex sectors, each containing one component of R'_P. Even though we cannot make cuts along the chords, we process each sector separately, in a bottom-up traversal of the tree H_P. Lemma 12 is replaced by the following lemma, using auto-partitions.

Lemma 28. *Let $P = \texttt{PathSelector}(S, C, k)$. Let R'_P be a connected component of R_P containing $h \in \mathbb{N}$ paths α_i, and lying in sector C'. There is an auto-partition $\texttt{CompAuto}(B, C, R'_P)$ such that*

(i) *every boundary segment is cut at most $O(1)$ time* on average*;*

(ii) *every interior segment is cut at most $O(h)$ times;*

(iii) *every cell produced by $\texttt{CompAuto}(B, C, R'_P)$ intersects less than $|B|/k$ boundary segments lying in sector C';*

(iv) *no partition line cuts chord $e = e(R'_P)$, and the resulting cell containing e contains no boundary segment from sector C'.*

The last condition replaces the functionality of a cut along the chord $e(R'_P)$.

5.1 Processing an acyclic component of R_P

Let R'_P be a connected component of R_P in a sector $C' \subseteq C$, and let $e = e(R'_P)$ be the chord of ∂C, which is an edge of the convex hull of the unique component $R' \subset R$ which contains R'_P. Chord e lies in a convex face $f \in F_{\{1,2,\ldots,m\}}$. The two endpoints of e are the outer endpoints of two distinct boundary segments, say s_1 and s_2 (Fig. 9). The paths $\alpha_1, \alpha_2 \subset R'$ start from these endpoints and reach the convex cycle $\varphi \subset R'_P$. They follow the boundary of face f until they meet at some point $q \in \partial f$. Augment R'_P with the paths α_1 and α_2 (if they are not already included in R'_P). These additional paths will establish part (iv) of Lemma 28.

If R'_P is a tree terminating at a point $\tau \in \partial C$, then $\texttt{CompBSP}(B, C', R'_P)$ is an auto-partition. Let $\texttt{CompAuto}(B, C', R'_P)$ $=\texttt{CompBSP}(B, C', R'_P)$. Parts (i)–(iii) of Lemma 28 follow from Lemma 12, it remains to prove part (iv). When we decompose R'_P into a set Γ of non-overlapping paths, the portion of α_1 (resp., α_2) from ∂C to point q is the union of some paths in Γ (Fig. 9). These conformal paths

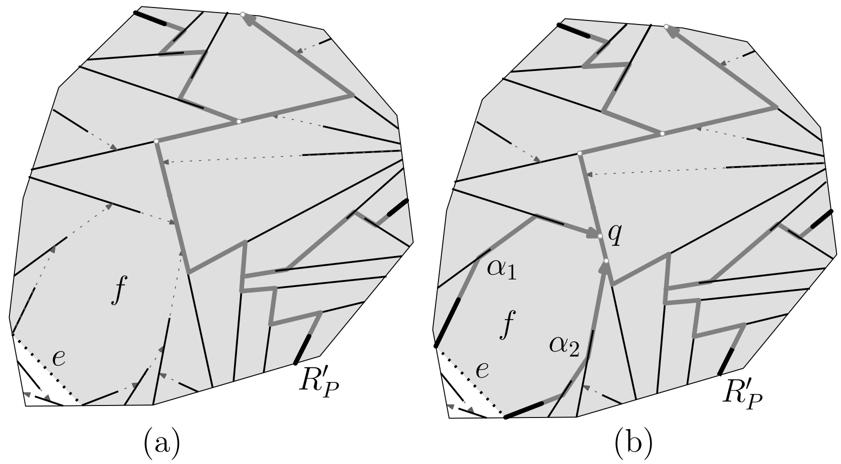


Figure 9: The convex face $f \in F_{\{1,2,\ldots,m\}}$ containing chord e, and the paths α_1 and α_2.

are already convex (since they lie on the boundary of a convex face f), so they are not truncated only simplified. So $\texttt{CompBSP}(B, C', R'_P)$ produces a subcell that contains f but does not intersect any boundary segment from sector C'. By Proposition 23, the interior of face f and chord e are not cut by any $\texttt{ChainBSP}(\delta_i)$ during $\texttt{CompBSP}(B, C', R'_P)$.

5.2 Processing a cyclic component of R_P

Now consider the case that component $R'_P \subseteq R_P$ contains a cycle φ lying in the interior of sector C'. Similar to the previous case, we augment R'_P with the paths α_1 and α_2 that start from the two endpoints of chord e.

We will partition R'_P into two *trees*, T_1 and T_2, such that each of $\texttt{CompBSP}(B, C', T_1)$ and $\texttt{CompBSP}(B, C', T_2)$ is an auto-partition (however performing both of them successively may not be an auto-partition). The domains M_1 and M_2, associated to the two trees, will jointly cover all boundary segments. For $i = 1, 2$, $\texttt{CompBSP}(B, C', T_i)$ partitions Δ_i into cells that each intersect at most $|B|/k$ boundary segment. We perform $\texttt{CompBSP}(B, C', T_i)$ for the tree where Δ_i contains at least half of the boundary segments from sector C'. We recursively call $\texttt{SubAuto}(B'', C'', k)$ in the resulting cells C'' that still intersect more that $|B|/k$ boundary segments from sector C'. At each level of this recursion, every remaining boundary segment is cut at most $O(1)$ times. So throughout this recursion, a boundary segment is cut on average at most $O(\sum_{i=0}^{\infty}(\frac{1}{2})^i) = O(1)$ times. If sector C' contains $O(m')$ boundary segments, then $\texttt{CompBSP}(B, C', T_i)$ cuts an interior segment $O(m'/(m/k)) = O(km'/m$ times. Since at most half of the boundary segments survive each recursive call, altogether an interior segment is cut $O(\frac{k}{m}\sum_{i=0}^{\infty}(\frac{1}{2}m')^i) = O(km'/m)$ times in sector C'.

We now describe how to decompose R'_P into two trees T_1 and T_2. Suppose that φ makes *right* turns only (the case that is makes left turn only is analogous). The interior of φ is disjoint of boundary segments, we will cover the region between ∂C and φ by two polygonal domains, each of which is adjacent to chord e. Let $\varphi = (u_1, u_2, \ldots, u_t)$, and denote the extended boundary segments along φ by $\hat{b}_1, \hat{b}_2, \ldots, \hat{b}_t$ such that $u_{j-1}u_j \subset \hat{b}_j$ (where $u_{-1} = u_t$). The t boundary segments decompose ∂C into t arcs. We may assume w.l.o.g. that the arc between b_1 and b_2 contains chord e. Let ℓ, $1 < \ell < t$ be the largest index such that the supporting line of b_ℓ does not cross $\hat{b}_1$. It follows that the supporting line of $b_{\ell+1}$ crosses $\hat{b}_1$, and so the supporting line of b_1 does not cross $\hat{b}_\ell$ (Fig. 8(bc)).

Let M_1 be the polygonal domain bounded by $\hat{b}_1$, path $(u_1, u_2, \ldots, u_\ell)$, and $\hat{b}_{\ell+1}$; and let M_2 be the domain bounded by $\hat{b}_{\ell+1}$, path $(u_\ell, u_{\ell+1}, \ldots, u_1)$, and $\hat{b}_2$. Note that both domains contain chord

e, and every boundary segment lies in at least one of the two domains. Let T_1 and T_2 be the part of R'_P clipped in M_1 and M_2, respectively.

The first partition step of $\texttt{CompBSP}(B, C', T_1)$ is made along the supporting line of b_ℓ. The first partition step of $\texttt{CompBSP}(B, C', T_2)$ is made along the supporting line of b_2. Neither partition line crosses chord e, and both $\texttt{CompBSP}(B, C', T_1)$ and $\texttt{CompBSP}(B, C', T_2)$ are auto-partitions. We perform the one for which domain M_i contains more boundary segments.

6 Construction of a BSP in $O(n \text{ polylog } n)$ time

Our algorithm $\texttt{BSP}(S)$ is composed of repeated calls to $\texttt{SubBSP}(B, C, k)$, with $k = \lceil \log n / \log\log n \rceil$. The input of $\texttt{SubBSP}(B, C, k)$ is a set of boundary segment in a convex cell C. It returns a BSP tree, which stores the recursive cuts along partition lines that decompose C into convex subcells. $\texttt{SubBSP}(B, C, k)$ does not compute, however, how the interior segments are fragmented. If an interior segment is cut, then its fragments are free segments or boundary segments in the subproblems. For each resulting cell C', algorithm $\texttt{BSP}(S, C)$ calls $\texttt{SubBSP}(B_{S(C')}, C', k)$, so we need to compute how the interior segments are fragments.

While constructing a BSP tree for S, we maintain the set of boundary and interior segment for each cell in a data structure. For each call to $\texttt{SubBSP}(B, C, k)$, we extract the set B of boundary segments with respect to C from this data structure. When $\texttt{SubBSP}(B, C, k)$ returns a BSP tree for cell C, we record the effect of the binary cuts in this data structure. We use the data structure of Ishaque *et al.*[19] that supports so-called *ray shooting-and-insertion* queries among disjoint polygonal obstacles (in our case, line segments) in the plane. Each query is a point p on the line segment and a direction d_p; it reports the point q where the ray emitted by p in direction d_p hits the first obstacle (ray shooting) *and* inserts the segment pq as a new obstacle (segment insertion). For an input of n disjoint line segments, it uses $O(n \log n)$ preprocessing time, and it supports m ray shooting-and-insertion queries in $O((n+m)\log^2 n + m \log m)$ total time in the real RAM model of computation.

To partition a cell C along a line ℓ, shoot a ray along ℓ from one intersection point $\ell \cap \partial C$, and whenever a ray hits a segment $s \in S$, shoot a new ray from the opposite side of s in the same direction. A BSP that partitions n line segments into m fragments requires $O(m)$ ray shooting-and-insertion queries. We have $m = O(n \log n / \log\log n)$, and so the maintenance of the data structure requires $O(n \log^3 n / \log\log n)$ time. We can easily detect free segments, and perform any possible free cuts.

The input of $\texttt{SubBSP}(B, C, k)$ includes only a cell C and the set B of boundary segments with respect to C. The fragments of a segment $s \in S$ are involved in an average of $O(\log n / \log\log n)$ calls to $\texttt{SubBSP}(B, C, k)$. To prove that the total runtime is $O(n \log^3 / \log\log n)$, it is enough to show that $\texttt{SubBSP}(B, C, k)$ can be implemented in $O(|B| \log n)$ time.

Implementation of algorithm $\texttt{SubBSP}(B, C, k)$ in $O(|B| \log n)$ time. Let $m = |B|$, and assume w.l.o.g. that no input segment is vertical. The extensions of all boundary segments can be computed in $O(m \log m)$ time in two line sweeps: first in a left-to-right sweep, extend every boundary segment whose inner endpoint is the right endpoint; then in a right-to-left sweep, extend the boundary segments whose inner endpoint is the left endpoint. Whenever two extensions meet along the sweep line, one arbitrary extension ends and the other one continues.

It is straightforward to implement $\texttt{PathSelector}(B, C, k)$ in $O(m)$ time. We can detect the connected components of R_P in a simple traversal of R, in $O(m)$ time. Similarly, we can decompose the connected components $R'_P \subset R_P$ into non-overlapping conformal paths in Γ, and compute the convex conformal paths γ_i in $O(m)$ total time.

The algorithm $\texttt{Simplify}(B, C, \gamma)$ involves the segments $\bar{b}$, which stretch from the outer endpoint of b to the first intersection point with another boundary segment or with ∂C. The complexity of the full arrangement of these segments may be $\Theta(m^2)$. However, we can compute each of them in $O(\log m)$ time with a standard ray shooting data structure [13, 17] for the (weakly) simple polygon formed by C and all boundary segments in B. Once we have computed $\bar{b}$ for each $b \in B$, we can perform $\texttt{Simplify}(B, C, \gamma_i)$ for all $i = 1, 2, \dots g$, in $O(m)$ total time. Finally, $\texttt{SubBSP}(B, C, k)$ is a concatenation of binary cuts along segments in the simplified paths δ_i, of $O(m)$ total complexity. Over all, we can compute $\texttt{SubBSP}(B, C, k)$ in $O(m \log m)$ time.

7 Conclusion

We have shown that every set of n disjoint line segments in the plane admits a BSP and an auto-partition of size $O(n \log n / \log \log n)$. These bounds are the best possible. The *height* of a BSP is the height of the recursion tree. It is an important parameter for efficient manipulation of the BSP tree data structure. Arya [4] studied tradeoffs between the size and the height of a BSP for *axis-parallel* line segments.

Our algorithm produces, for n segments, a BSP whose height may be as large as $O(n)$ (e.g., if there are n boundary segments, and they are all adjacent to a convex cycle φ). It remains an open problem whether a BSP (or an auto-partition) BSP of size $O(n \log n / \log \log n)$ and height $O(\log n)$ exists for every set of n disjoint line segments in the plane.

References

[1] P. K. Agarwal, E. F. Grove, T. M. Murali, and J. S. Vitter, Binary space partitions for fat rectangles, *SIAM J. Comput.* **29** (2000), 1422–1448.

[2] S. Ar, B. Chazelle, and A. Tal, Self-customized BSP trees for collision detection, *Comput. Geom. Theory Appl.* **15** (1-3) (2000), 91–102.

[3] S. Ar, G. Montag, and A. Tal, Deferred, self-organizing BSP trees, *Comput. Graph. Forum* **21** (3) (2002), 269–278.

[4] S. Arya, Binary space partitions for axis-parallel line segments: size-height tradeoffs, *Inform. Proc. Letts.* **84** (4) (2002), 201–206.

[5] S. Arya, T. Malamatos, and D.M. Mount, Nearly optimal expected-case planar point location, in *Proc. 41st Sympos. Foundations of Comp. Sci.*, IEEE Press, 2000, pp. 208–218.

[6] T. Asano, M. de Berg, O. Cheong, L.J. Guibas, J. Snoeyink, and H. Tamaki, Spanning trees crossing few barriers, *Discrete Comput. Geom.* **30** (2003), 591–606.

[7] M. de Berg, Linear size binary space partitions for fat objects, in *Proc. 3rd European Sympos. Algorithms*, vol. 979 of LNCS, Springer, Berlin, 1995, pp. 252–263.

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

[9] M. de Berg, H. David, M. Katz, M. Overmars, A.F. van der Stappen, and J. Vleugels, Guarding scenes against invasive hypercubes, *Comput. Geom. Theory Appl.* **26** (2003), 99–117.

[10] M. de Berg, M. de Groot, and M. Overmars, New results on binary space partitions in the plane, *Comput. Geom. Theory Appl.* **8** (1997), 317–333.

[11] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*, 3rd edition, Springer, Berlin, 2008.

[12] M. de Berg and M. Streppel, Approximate range searching using binary space partitions, *Comput. Geom. Theory Appl.* **33** (3) (2006), 139–151.

[13] B. Chazelle, H. Edelsbrunner, M. Grigni, L. J. Guibas, J. Hershberger, M. Sharir, and J. Snoeyink, Ray shooting in polygons using geodesic triangulations, *Algorithmica* **12** (1994), 54–68.

[14] N. Chin and S. Feiner, Fast object-precision shadow generation for areal light sources using BSP trees, *Comput. Graph.* **25** (1992), 21–30.

[15] A. Dumitrescu, J. S. B. Mitchell, and M. Sharir, Binary space partitions for axis-parallel segments, rectangles, and hyperrectangles, *Discrete Comput. Geom.* **31** (2) (2004), 207–227.

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

[17] J. Hershberger and S. Suri, A pedestrian approach to ray shooting: Shoot a ray, take a walk, *J. Algorithms* **18** (3) (1995), 403–431.

[18] J. Hershberger, S. Suri and Cs. D. Tóth, Binary space partitions of orthogonal subdivisions, *SIAM J. Comput.* **34** (6) (2005), 1380–1397.

[19] M. Ishaque, B. Speckmann, and Cs. D. Tóth, Shooting permanent rays among disjoint polygons in the plane, in *Proc. 25th ACM Sympos. Comput. Geom.* ACM Press, 2009, these proceedings.

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

[21] B. Naylor, Constructing good partitioning trees, *Proc. Graphics Interface*, 1993, Canadian Human-Computer Communications Society, Toronto, pp. 181–191.

[22] 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.

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

[24] 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, San Antonio, TX, 1969.

[25] W. C. Thibault and B. F. Naylor, Set operations on polyhedra using binary space partitioning trees, *Comput. Graph.* **21** (4) (1987), 153–162, Proc. SIGGRAPH '87.

[26] Cs. D. Tóth, A note on binary plane partitions, *Discrete Comput. Geom.* **30** (2003), 3–16.

[27] Cs. D. Tóth, Binary space partition for axis-aligned fat rectangles, *SIAM J. Comput.* **38** (1) (2008), 429–447.

[28] Cs. D. Tóth, Binary space partition for line segments with a limited number of directions, *SIAM J. Comput.* **32** (2) (2003), 307–325.