Decomposing the Complement of the Union of Cubes in Three Dimensions [*]

Pankaj K. Agarwal[†] Micha Sharir[‡] Alex Steiger[§]

Abstract

Let $\mathcal{C}$ be a set of n axis-aligned cubes of arbitrary sizes in $\mathbb{R}^3$ in general position. Let $\mathcal{U} := \mathcal{U}(\mathcal{C})$ be their union, and let κ be the number of vertices on $\partial\mathcal{U}$; κ can vary between $O(1)$ and $O(n^2)$. We show that $\mathrm{cl}(\mathbb{R}^3 \setminus \mathcal{U})$ can be decomposed into $O(\kappa \log^4 n)$ axis-aligned boxes with pairwise-disjoint interiors. Given a boundary representation of $\mathcal{U}$, such a decomposition can be computed in $O(n \log^2 n + \kappa \log^6 n)$ time. We also show that a decomposition of size $O(\sigma \log^4 n + \kappa \log^2 n)$, where σ is the number of input cubes that appear on $\partial\mathcal{U}$, can be computed in $O(n \log^2 n + \sigma \log^8 n + \kappa \log^6 n)$ time. The complexity and runtime bounds improve to $O(n \log n)$ if all cubes in $\mathcal{C}$ are congruent.

1 Introduction

Decomposing the common exterior of a set of geometric objects is an important problem in motion planning [19] and solid modeling [13, 18]. In this paper we study a natural instance of this problem in which each object is an axis-aligned cube in $\mathbb{R}^3$. Formally, let $\mathcal{C} := \{C_1, \ldots, C_n\}$ be a set of n axis-aligned cubes in $\mathbb{R}^3$ in *general position*. By this we mean that no two vertices of any pair of distinct cubes have the same x-, y-, or z-coordinate.

Let $\mathcal{U} := \mathcal{U}(\mathcal{C})$ denote their union, and let $\mathcal{K} := \mathrm{cl}(\mathbb{R}^3 \setminus \mathcal{U})$ denote the closure of its complement. Denote by κ the *complexity* of $\mathcal{U}$, which we measure by the number of vertices of $\partial\mathcal{U}$; the number of edges and vertices of $\partial\mathcal{U}$ is proportional to the number of its vertices. The value of κ can be anywhere between $O(1)$ and $O(n^2)$, and it is $\Theta(n^2)$ in the worst case; see, e.g., [6]. However, when the cubes of $\mathcal{C}$ are all congruent, $\kappa = O(n)$; again, see [6]. If the sizes of the cubes are chosen randomly from an arbitrary probability distribution, the expected value of κ is $O(n \log^2 n)$ [1]. The question we study is whether $\mathcal{K}$ can be partitioned into a collection of axis-aligned boxes with pairwise-disjoint interiors, so that the number of boxes depends almost linearly on κ, and do so by a procedure with comparable running time.

Besides its application in motion planning, the problem of decomposing $\mathcal{K}$ also arises naturally in divide-and-conquer approaches to problems that involve the construction of the union of such a collection of cubes. We are not aware of any previous near-linear-time algorithm for constructing the union of axis-aligned congruent cubes in $\mathbb{R}^3$, or an output-sensitive algorithm for computing the union of axis-aligned cubes of arbitrary sizes. A potential approach for computing $\mathcal{U}(\mathcal{C})$ is the geometric divide-and-conquer approach based on random sampling. Typically, one draws a random sample $\mathcal{R}$ of $r \ll n$ cubes from $\mathcal{C}$, constructs the union $\mathcal{U}(\mathcal{R})$ of $\mathcal{R}$, and decomposes (the closure of) its complement $\mathcal{K}(\mathcal{R})$ into a small number of boxes with pairwise-disjoint interiors. One then constructs the union of the (unsampled) cubes of $\mathcal{C}$ within each box B of the decomposition, exploiting the property that, with high probability, every such box B is crossed by only $O\left(\frac{n}{r}\log r\right)$ cubes of $\mathcal{C}$ [17] (see also [16]). This yields a simple and effective divide-and-conquer mechanism, whose complexity crucially depends on the (expected) number of boxes B in the decomposition. See below for a further elaboration of this approach.

Background. Motivated by applications in various fields (e.g., physical simulation, computer graphics, robotics), decomposing a complex geometric region into simply-shaped regions, such as simplices or boxes, has been a central problem in computational geometry for more than four decades. For example, there has been extensive work on triangulating a polygonal region in 2D or a polyhedral region in 3D [5, 7, 14]. In this line of work the region that needs to be decomposed is given explicitly. However, in many applications, the

[*]Work on this paper by P.A. and A.S. was partially supported by NSF grants IIS-18-14493 and CCF-20-07556. Work on this paper by M.S. was supported by Grant 260/18 from the Israel Science Foundation, by Grant G-1367-407.6/2016 from the German-Israeli Foundation for Scientific Research and Development, and by the Blavatnik Research Fund in Computer Science at Tel Aviv University.

[†]Department of Computer Science, Duke University, Durham, NC 27708-0129, USA; pankaj@cs.duke.edu.

[‡]School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel; michas@tau.ac.il.

[§]Department of Computer Science, Duke University, Durham, NC 27708-0129, USA; asteiger@cs.duke.edu.

region to be decomposed is specified implicitly, e.g., as the arrangement of a set of geometric objects or as the common exterior of a set of geometric regions — our problem of decomposing $\mathcal{K}$ is an instance of the latter. The latter setting, as mentioned above, also arises in the context of collision-free motion planning [19]. In either case, the combinatorial complexity of the region ($\mathcal{K}$ in our case) and the complexity of decomposition thereof may differ significantly (see, e.g., [9]), making the decomposition task an even harder problem. As an indiciation of this phenomenon, Schwartz and Sharir [23] described a general decomposition scheme based on the so-called *cylindrical algebraic decomposition* of Collins [11], but it leads to a decomposition with too many pieces.

A widely popular approach to decompose a region of complex shape into simpler regions, which is more parsimonious than the cylindrical algebraic decomposition, is the "vertical decomposition;" see, e.g., [8, 22]. In our context, it will decompose $\mathcal{K}$ into axis-aligned boxes. However, the size of the vertical decomposition of κ could be $\Omega(n^2)$ even if $\kappa = O(n)$. The known algorithms for triangulating non-convex polyhedra into simplices also produce a triangulation whose size may be quadratic in the complexity of the polyhedron [10], and the known lower bounds show that one cannot hope to do better [9]. The construction in [9] actually gives a set of pairwise-disjoint prisms in $\mathbb{R}^3$ such that any convex decomposition of the common exterior has quadratic size. Paterson and Yao [21] construct a set of n pairwise-disjoint axis-aligned boxes in $\mathbb{R}^3$ such that any decomposition of their common exterior into boxes (or any convex decomposition) has size $\Omega(n^{3/2})$; note that $\kappa = O(n)$ in this case. So the only hope to obtain a decomposition of $\mathcal{K}$ in our setting into roughly κ boxes is to exploit the geometry of axis-aligned cubes.

Another common technique called *binary space partition* (BSP), which divides the space hierarchically into convex regions using local cuts by planes [3, 15, 20, 21, 24], is a possible approach to decompose $\mathcal{K}$ into axis-aligned boxes, but its worst-case complexity can be $O(\kappa^2)$. This can be improved, using the technique of [21], in which, for a given set $\mathcal{R}$ of n pairwise-disjoint axis-aligned rectangles in $\mathbb{R}^3$, the space can be partitioned hierarchically into $O(n^{3/2})$ boxes so that no rectangle of $\mathcal{R}$ intersects the interior of any box [21]. By decomposing $\partial\mathcal{U}$ into $O(\kappa)$ rectangles and using the result just mentioned, $\mathcal{K}$ can be decomposed into $O(\kappa^{3/2})$ axis-aligned boxes with pairwise-disjoint interiors, still a far cry from our desired bound which is nearly linear in κ. Agarwal *et al.* [3] and Tóth [24] have shown that a BSP of near-linear size can be constructed if the rectangles in $\mathcal{R}$ are *fat*, i.e., they have bounded *aspect ratio* (the ratio between its largest and smallest edge lengths). Unfortunately, the faces of $\partial\mathcal{U}$ need not have bounded aspect ratio, so it is not possible to decompose $\partial\mathcal{U}$ into $O(\kappa)$ fat rectangles and apply the results of [3, 24] directly. Nevertheless, by exploiting the properties of cubes, we obtain a much simpler decomposition scheme with the desired bound on the size of the decomposition.

Our results. The main result of the paper is an efficient algorithm that decomposes $\mathcal{K}$ into $O(\kappa\,\mathrm{polylog}(n))$ boxes, whose running time is comparable to the size of the decomposition. Our algorithm takes the cubes in $\mathcal{C}$ as input, as well as the faces of $\partial\mathcal{U}$, where each face is represented by a sequence of edges on each connected component of its boundary. By the general-position assumption, every face of $\partial\mathcal{U}$ lies on a face of a distinct cube in $\mathcal{C}$, which is an important property to have for our algorithm and analysis. Our first result is the following:

Theorem 1.1. *Let $\mathcal{C}$ be a set of n axis-aligned cubes in $\mathbb{R}^3$, and let κ be the number of vertices on $\partial\mathcal{U}(\mathcal{C})$. The complement of $\mathcal{U}(\mathcal{C})$ can be decomposed into $O(\kappa \log^4 n)$ axis-aligned boxes with pairwise-disjoint interiors. Given a boundary representation of $\mathcal{U}(\mathcal{C})$, such a decomposition can be constructed in $O(n \log^2 n + \kappa \log^6 n)$ time.*

By further exploiting the structure at hand, we show that a slightly smaller decomposition can be computed at the cost of a slightly higher runtime:

Theorem 1.2. *Let $\mathcal{C}$ be a set of n axis-aligned cubes in $\mathbb{R}^3$, let κ be the number of vertices on $\partial\mathcal{U}(\mathcal{C})$, and let $\sigma \le \min\{n, \kappa\}$ be the number of cubes in $\mathcal{C}$ that appear on $\partial\mathcal{U}$. The complement of $\mathcal{U}(\mathcal{C})$ can be decomposed into $O(\sigma \log^4 n + \kappa \log^2 n)$ axis-aligned boxes with pairwise-disjoint interiors. Given a boundary representation of $\mathcal{U}(\mathcal{C})$, such a decomposition can be constructed in $O(n \log^2 n + \sigma \log^8 n + \kappa \log^6 n)$ time.*

We remark that our algorithm can be extended to degenerate configurations of cubes using symbolic perturbation (also known as simulation of simplicity [12]), but the running time will be proportional to the union-size of the perturbed configuration, which may be much larger than the original κ depending on how the combinatorial complexity of the union is defined for degenerate configurations.

We observe that a *fat* box B, namely a box with a bounded aspect ratio can be decomposed into a family $\mathcal{C}_B$ of $O(1)$ possibly overlapping cubes such that $\mathcal{U}(\mathcal{C}_B) = B$, so our algorithm also extends to a set of fat boxes. There is a technicality that the cubes in $\mathcal{C}_B$ are not in general position but if the input boxes are in general position, then symbolic perturbation will increase the union complexity by a constant factor. Therefore we obtain the following:

Corollary 1.3. *Let $\mathcal{B}$ be a set of n fat boxes in $\mathbb{R}^3$ in general position so that the aspect ratio of every box is bounded by a constant α. Then the complement of $\mathcal{U}(\mathcal{B})$ can be decomposed into $O(\kappa \log^4 n)$ or $O(\sigma \log^4 n + \kappa \log^2 n)$ boxes in time $O(n \log^2 n + \kappa \log^6 n)$ or $O(n \log^2 n + \sigma \log^8 n + \kappa \log^6 n)$, respectively, where κ is the number of vertices on $\partial\mathcal{U}(\mathcal{B})$ and $\sigma \leq \min\{n, \kappa\}$ is the number of boxes in $\mathcal{B}$ that appear on $\partial\mathcal{U}$.*

Agarwal *et al.* [1] have shown that if the sizes of the cubes in $\mathcal{C}$ are chosen from an arbitrary probability distribution (but their centers are fixed), then the expected complexity of $\mathcal{U}(\mathcal{C})$ is $O(n \log^2 n)$. However, their technique does not give an efficient algorithm for computing $\mathcal{U}(\mathcal{C})$. Using Theorem 1.1, we obtain the following:

Corollary 1.4. *Let $\mathcal{C}$ be a set of n axis-aligned cubes in $\mathbb{R}^3$ whose sizes are chosen from an arbitrary probability distribution (but their centers are fixed). Assuming a boundary representation of $\mathcal{U}(\mathcal{C})$ is given, the complement of $\mathcal{U}(\mathcal{C})$ can be decomposed into $O(n \log^6 n)$ expected number of axis-aligned boxes with pairwise-disjoint interiors, by an algorithm with expected running time $O(n \log^8 n)$.*

The expected bounds in the result above are only with respect to the random sizes of the cubes; the algorithm remains deterministic.

We note that the random-sampling approach does not lead to an output-sensitive algorithm for computing $\mathcal{U}(\mathcal{C})$ if the sizes of the cubes of $\mathcal{C}$ are chosen by an adversary, because the expected complexity of $\mathcal{U}(\mathcal{R})$ for a random subset $\mathcal{R} \subseteq \mathcal{C}$ of cubes can be quite large compared to that of $\mathcal{U}(\mathcal{C})$.

If the cubes in $\mathcal{U}$ are congruent, then we obtain the following improved result:

Theorem 1.5. *Let $\mathcal{C}$ be a set of n axis-aligned congruent cubes in $\mathbb{R}^3$. The complement of $\mathcal{U}(\mathcal{C})$ can be decomposed into $O(n \log n)$ axis-aligned boxes with pairwise-disjoint interiors, in $O(n \log n)$ time.*

Analogous to Corollary 1.3, we obtain the following:

Corollary 1.6. *Let $\mathcal{B}$ be a set of n boxes in $\mathbb{R}^3$ in general position so that the aspect ratio of every box is bounded by a constant $\alpha \geq 1$ and the ratio of the largest to the smallest size box is bounded by a constant β. Then the complement of $\mathcal{U}(\mathcal{B})$ can be decomposed into $O(n \log n)$ boxes in time $O(n \log n)$.*

By plugging the bounds in Theorems 1.1 and 1.5 into the randomized divide-and-conquer framework mentioned earlier in this section, we obtain the following:

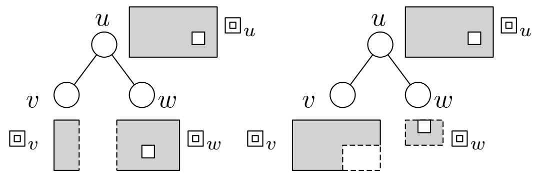

Figure 1. Two planar renderings of BBD subtrees with identical regions $\boxdot_u$ at the root nodes. On the left, $\boxdot_u$ is partitioned into $\boxdot_v, \boxdot_w$ by a splitting plane. On the right, $\boxdot_u$ is partitioned by a splitting box.

Corollary 1.7. *Let $\mathcal{C}$ be a set of n axis-aligned cubes in $\mathbb{R}^3$. If the cubes are congruent or if their sizes are chosen randomly from an arbitrary probability distribution (but their centers are fixed), $\mathcal{U}(\mathcal{C})$ can be computed in $O(n^{1+\varepsilon})$ time for any arbitrarily small constant $\varepsilon > 0$.*

At a high level, our decomposition techniques are inspired by the BSP construction schemes described in [3, 21, 24], but their implementation exploits the geometry of cubes and attains significantly improved performance bounds.

Roadmap of the paper. We begin by giving a brief overview of balanced-box decomposition trees [2], which is a key component of our algorithm for cubes of different sizes. Next, we describe our first algorithm (given in Theorem 1.1) for cubes of arbitrary sizes in Section 3, and our second algorithm (given in Theorem 1.2) in Section 4. Finally, we describe the more efficient algorithm for congruent cubes (given in Theorem 1.5) in Section 5. The algorithms are quite simple; only the notations and analyses are somewhat involved.

2 Balanced-Box Decomposition (BBD) Trees

Let $P \subseteq \mathbb{R}^3$ be a set of n points. Introduced by Arya *et al.* [2], the *BBD tree* $\mathcal{T}$ for P is a binary tree that represents a hierarchical decomposition of P. Each node u of $\mathcal{T}$ is associated with a region $\boxdot_u$, which is the set-theoretic difference $\square_u^O \setminus \square_u^I$ of a pair of axis-aligned boxes: an *outer box* $\square_u^O$ and a (potentially empty) *inner box* $\square_u^I \subseteq \square_u^O$. If u is not a leaf, then u is also associated with either a single *splitting plane* h_u or a *splitting box* $\square_u^S$, where neither of which cross the boundary of $\square_u^I$. Furthermore, if u has a splitting box $\square_u^S$ and $\square_u^I \neq \varnothing$, then $\square_u^I \subseteq \square_u^S \subseteq \square_u^O$. The splitting planes and boxes partition $\boxdot_u$ into the two sub-regions $\boxdot_v$ and $\boxdot_w$ associated with its two respective children, v and w. See Figure 1. Any leaf u of $\mathcal{T}$ has $|P \cap \boxdot_u| \leq 1$. The height of $\mathcal{T}$ is $O(\log n)$ and $\mathcal{T}$ can be constructed in $O(n \log n)$ time [2]. See Appendix A and the original paper [2] for more details on BBD trees.

For our purposes, it is convenient to introduce the notation Σ_u, for each node u of $\mathcal{T}$, to be the set that

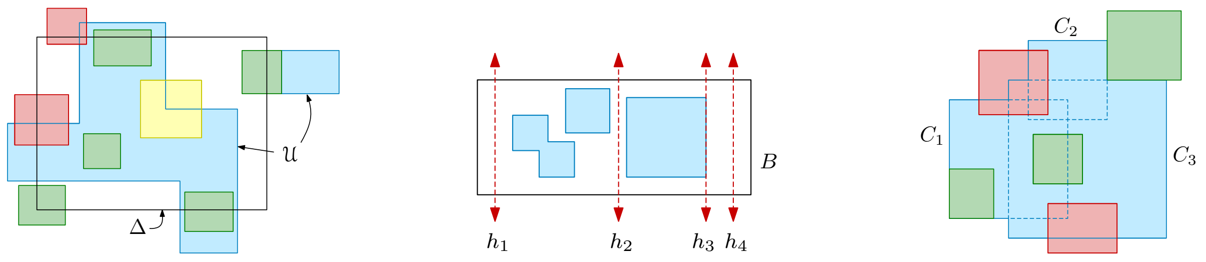

Figure 2. In each 2D illustration, $\mathcal{U}$ is depicted in blue. (left) The green boxes are passive and compatible with Δ, the yellow box is active and compatible with Δ, and the red boxes are active but not compatible with h. (middle) A box B that is compatible with h_1 and h_4 but not compatible with h_2 or h_3. (right) A union of boxes $\mathcal{C} = \{C_1, C_2, C_3\}$ such that only the green boxes are void of $\mathcal{C}$; e.g., the bottom red box is void of $\{C_1, C_2\}$ but not of $\{C_3\}$.

contains either the single splitting plane h_u at u, or the axis-aligned planes that support the (at most 6) faces of the splitting box $\square_u^S$ at u but not the faces of $\square_u^O$. We refer to Σ_u as the set of *separating planes* at node u.

We prove the following property of BBD trees (*cf.* Appendix A for the proof), which is crucial for our application.

Lemma 2.1. *Let u be a node of a BBD tree $\mathcal{T}$ for a point set $P \subseteq \mathbb{R}^3$. There is a set H_u of at most 24 planes that induces a subdivision of $\boxdot_u$ into $O(1)$ axis-aligned boxes such that any axis-aligned cube C that intersects $\boxdot_u$ but none its vertices lie in the interior of $\boxdot_u$ contains an edge of each box that it intersects.*

3 Decomposing the Complement of the Union for Cubes of Arbitrary Sizes

Let $\mathcal{C}, \mathcal{U}, \mathcal{K}$ be as defined at the beginning of the Introduction. Let $\square$ be a cube containing $\mathcal{U}$. We trivially partition the exterior of $\square$ into $O(1)$ boxes, so we focus on partitioning $\mathcal{K} \cap \square$ into $O(\kappa \log^4 n)$ boxes.

The algorithm maintains a partition $\mathcal{B}$ of $\square$ into axis-aligned boxes with pairwise-disjoint interiors. It successively refines the partition until the interior of each box in $\mathcal{B}$ fully lies in $\mathcal{U}$ or in $\mathcal{K}$. Each step of the algorithm picks a box B of $\mathcal{B}$ whose interior intersects $\partial\mathcal{U}$ and *splits* B by a carefully chosen axis-aligned plane h, so that each of the resulting two boxes lies in one of the two halfspaces bounded by h; we refer to the rectangle $B \cap h$ as a *cut*, which splits B into two boxes. For an axis-aligned plane $h : x_i = a$, we use h^- (resp. h^+ to denote the closed halfspace $x_i \leq a$ (resp. $x_i \geq a$). When this refinement process terminates, we return the subset of boxes of $\mathcal{B}$ that lie in $\mathcal{K}$. We begin by a few preliminaries (Section 3.1), then describe the algorithm (Sections 3.2 and 3.3), followed by the analysis of the algorithm (Section 3.4).

3.1 Preliminaries A box $B \in \mathcal{B}$ is called *active* if $\text{int}(B) \cap \partial\mathcal{U} \neq \varnothing$, and *passive* otherwise. Passive boxes are not partitioned further and belong to the final decomposition $\mathcal{B}$. For a 3D region Δ (in our case Δ will be a box or an annular region lying between two nested boxes), let $\mathcal{B}_\Delta \subseteq \mathcal{B}$ be the set of boxes that intersect Δ. Let $\mathcal{A}_\Delta \subseteq \mathcal{B}_\Delta$ be the subset of active boxes B that intersect $\partial\mathcal{U}$ inside Δ, i.e., $\text{int}(B) \cap \partial\mathcal{U} \cap \text{int}(\Delta) \neq \varnothing$; $\mathcal{B}_\Delta \setminus \mathcal{A}_\Delta$ may contain active boxes B for which $\text{int}(B) \cap \partial\mathcal{U} \subseteq B \setminus \Delta$. A box B is *compatible* with Δ if $\text{int}(B) \cap \partial\mathcal{U} \subseteq \Delta$, and a subset $\mathcal{Z} \subseteq \mathcal{B}$ is *compatible* with Δ if every box of $\mathcal{Z}$ is compatible with Δ. See Figure 2 (left).

Abusing the notation a little, we say that B is *compatible* with a plane h if B is compatible with one of the halfspaces bounded by h, i.e., $\text{int}(B) \cap \partial\mathcal{U}$ lies in one of the two open halfspaces bounded by h; if B intersects h, then B does not intersect $\partial\mathcal{U}$ in one of the two open halfspaces. A subset $\mathcal{Z}$ is *compatible* with h if every box in $\mathcal{Z}$ is compatible with h. We describe in Section 3.3 a procedure GlobalCut$(\mathcal{Z}, h)$ that refines the boxes in $\mathcal{Z}$ to make them compatible with h. See Figure 2 (middle) for an illustration of these notions.

Let $X \subseteq \mathcal{C}$ be a subset of input cubes. Let $\partial\mathcal{U}_X$ denote the portion of $\partial\mathcal{U}$ that appears on the faces of cubes in X. A box $B \in \mathcal{B}$ is called *void* of X if $\text{int}(B) \cap \partial\mathcal{U}_X = \varnothing$, i.e., none of the cubes in X appear on $\partial\mathcal{U}$ inside B. For a subset $\mathcal{Z} \subseteq \mathcal{B}$, $\mathcal{Z}$ is *void* of X if every box in $\mathcal{Z}$ is void of X. See Figure 2 (right).

3.2 Overall algorithm We now describe the overall algorithm. Let V be the set of vertices of the input cubes; $|V| = 8n$. We construct a BBD tree $\mathcal{T}$ on V with $\square$ as the region associated with the root of $\mathcal{T}$. Recall that each node u of $\mathcal{T}$ is associated with an annular region $\boxdot_u$ lying between two nested boxes $\square_u^O$ and $\square_u^I$ (where the latter box may be empty) with $\boxdot_u := \text{cl}(\square_u^O \setminus \square_u^I)$. A cube $C \in \mathcal{C}$ intersecting $\boxdot_u$ is called *short* at u if at

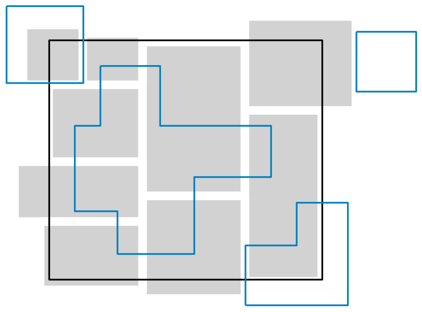
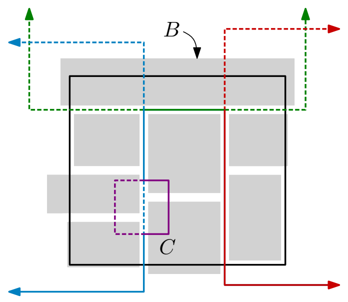

Figure 3. In both figures, a 2D box $\boxdot_u$ (black) and boxes of $\mathcal{A}_u$ (grey) are shown; for better visualization, the boxes are slightly shrunk towards their centers. (left) An example of invariant (I1): $\mathcal{A}_u$ is compatible with $\boxdot_u$. $\partial\mathcal{U}$ is shown in blue. (right) An example of invariant (I2): $\mathcal{A}_u$ is void of the three long (partially depicted) squares $\mathcal{L}_u$, but not of $\{C\}$, where $C \in \mathcal{S}_u$. The solid portions of the square boundaries are part of $\partial\mathcal{U}$, whereas the dashed portions lie in $\text{int}(\mathcal{U})$. Note that the box $B \in \mathcal{A}_u$ is indeed void of $\mathcal{L}_u$, although dashed portions of the long squares intersect its interior.

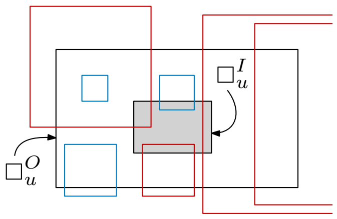

Figure 4. A 2D illustration of red long and blue short cubes (here squares) that intersect an annular region $\boxdot_u$.

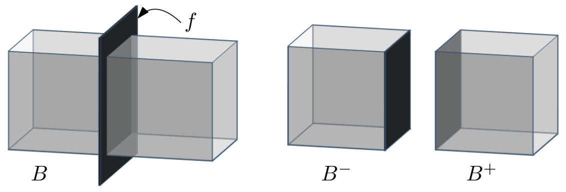

Figure 5. An illustration of a box B with a free cut defined by some face f of $\partial\mathcal{U}$, where $B^- := B \cap \text{span}(f)^-$ and $B^+ := B \cap \text{span}(f)^+$.

least one of the vertices of C lies inside $\boxdot_u$, and *long* otherwise. Note that vertices of a long cube C might lie in the inner box $\square_u^I$ of $\boxdot_u$. See Figure 4.

Let $\mathcal{S}_u$ (resp. $\mathcal{L}_u$) be the subset of cubes in $\mathcal{C}$ that are short (resp. long) at u. Let $\mathcal{C}_u := \mathcal{L}_u \setminus \mathcal{L}_{p(u)}$, where $p(u)$ is the parent of u, be the set of cubes that are long at u but short at $p(u)$ (if u is the root, we have $\mathcal{C}_u = \mathcal{L}_u = \varnothing$). If a cube $C \in \mathcal{L}_u$ contains $\boxdot_u$, then $\boxdot_u \subseteq \mathcal{U}$ and no refinement of $\boxdot_u$ is needed. Similarly if $\mathcal{L}_u \cup \mathcal{S}_u = \varnothing$, then $\boxdot_u \subseteq \mathcal{K}$ and there is no need to refine $\boxdot_u$. So assume $\partial\mathcal{U}$ intersects $\boxdot_u$. Set $\mathcal{B}_u := \mathcal{B}_{\boxdot_u}$ and $\mathcal{A}_u := \mathcal{A}_{\boxdot_u}$.

A box B admits a *free cut* if there is a face f of $\partial\mathcal{U}$ that intersects the interior of B and the edges of ∂f do not, i.e., $f \cap B = \text{span}(f) \cap B$ where $\text{span}(f)$ is the plane that contains f. Since $\text{span}(f) \cap B \subseteq f$, such a cut does not cross any other face of $\partial\mathcal{U}$ and $f \cap B$ does not lie in the interior of any box after B is split by this cut. Therefore it is desirable to split a box by a free cut whenever it admits one. See Figure 5. This notion is similar to the one used in the construction of binary space partitions [20].

The algorithm visits the nodes of $\mathcal{T}$ in a top-down manner, i.e., performs a pre-order traversal of $\mathcal{T}$, and successively refines $\mathcal{B}$. Initially $\mathcal{B}$ consists of a single box, namely $\square$ itself. A node u of $\mathcal{T}$ is marked *processed* immediately after executing the steps (i)–(iv) at u, as detailed below, and before proceeding recursively to the subtrees rooted at the children of u. The algorithm maintains the following three invariants:

(I1) When the algorithm arrives at a node u of $\mathcal{T}$, $\mathcal{A}_u$ is compatible with $\boxdot_u$. That is, for any box B of $\mathcal{A}_u$, $\text{int}(B) \cap \partial\mathcal{U} \subseteq \boxdot_u$. See Figure 3 (left).

(I2) When the algorithm finishes processing a node u of $\mathcal{T}$ in the sense defined above, $\mathcal{A}_u$ is void of $\mathcal{L}_u$. If u is a leaf, then $\mathcal{A}_u$ is void of $\mathcal{S}_u$ as well, which implies that $\mathcal{A}_u$ is void of $\mathcal{C}$ (and hence $\partial\mathcal{U}$ does not intersect the interior of any box in $\mathcal{A}_u$). See Figure 3 (right).

(I3) None of the boxes in $\mathcal{B}$ admit a free cut.

Assuming invariant (I2) holds after the algorithm completes the traversal of $\mathcal{T}$, the final set of boxes in $\mathcal{B}$ forms the desired subdivision of $\square$ because the regions associated with the leaves of $\mathcal{T}$ partition $\square$, and $\mathcal{A}_z$, for each leaf z, is void of $\mathcal{C}$.

Next, we describe the steps taken by the algorithm at each node of $\mathcal{T}$, to maintain the invariants (I1)–(I3), as it traverses $\mathcal{T}$.

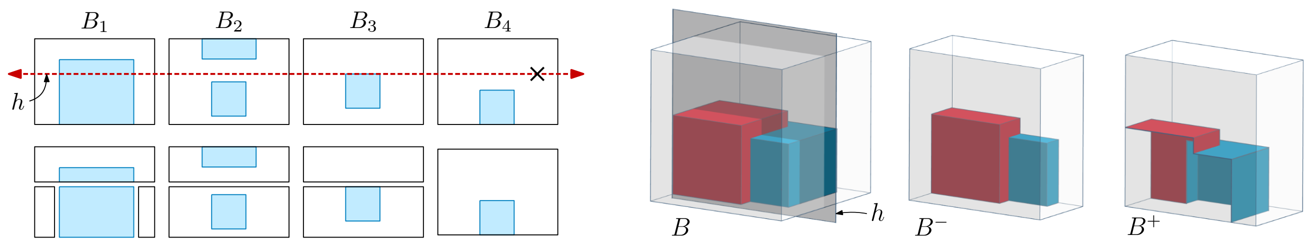

Figure 6. (left) A 2D view of boxes $B_1, B_2, B_3, B_4 \in \mathcal{A}_\Delta$ before (top) and after (bottom) the call GlobalCut(Δ, h), where $\mathcal{U} \cap \Delta$ is depicted in blue. Boxes B_1, B_2, B_3 are split by h during the call, but not B_4. After the call, sub-box of B_1 admitted free cuts supporting each face of $\mathcal{U}$ and was split by them. The boundaries of the sub-boxes are shown slightly shrunk towards their centers for better visualization. (right) An illustration of a box $B \in \mathcal{A}_\Delta$ split by plane h during GlobalCut(Δ, h): $\mathrm{int}(B) \cap \mathcal{U}$ is defined by two input cubes, one red and one blue. The portions of $\partial\mathcal{U}$ that lie strictly in the interior of the boxes are shown.

Suppose the algorithm has reached a node u of $\mathcal{T}$. Let $H_u := \{h_1, \ldots, h_r\}$ be the set of at most 24 planes obtained by applying Lemma 2.1 to u, and let Ξ_u be the subdivision of $\boxdot_u$ consisting of $O(1)$ boxes induced by H_u. By Lemma 2.1, if a long cube $C \in \mathcal{C}_u$ intersects a box $R \in \Xi_u$, C contains an edge of R. The algorithm performs the following steps at the node u:

(i) For each $h_i \in H_u$, we call the procedure GlobalCut$(\boxdot_u, h_i)$ to refine $\mathcal{A}_u$ so that it becomes compatible with h_i. By construction, after this step, for all boxes $R \in \Xi_u$, $\mathcal{A}_R$ is compatible with R, i.e., for each box $B \in \mathcal{A}_R$, $\mathrm{int}(B) \cap \partial\mathcal{U} \subseteq R$.

(ii) Fix a box $R \in \Xi_u$ and an edge $e \in R$. Let $\mathcal{C}_{R,e} \subseteq \mathcal{C}_u$ be the set of long cubes that intersect R and contain the edge e. We call the procedure Staircase$(R, e, \mathcal{C}_{R,e})$ to ensure that $\mathcal{A}_R$ becomes void of $\mathcal{C}_{R,e}$. We repeat this procedure for all edges e of R and for all boxes $R \in \Xi_u$.

(iii) If u is a leaf, then we also ensure that $\mathcal{A}_u$ is void of $\mathcal{S}_u$. If $\mathcal{S}_u = \varnothing$, there is nothing to do. Otherwise $\boxdot_u$ contains one vertex, say, ξ, of one short cube C and $\mathcal{S}_u = \{C\}$. Let g_1, g_2, g_3 be the three planes supporting the faces of C that contain ξ; no other face of C intersects $\mathrm{int}(\boxdot_u)$. We call GlobalCut$(\boxdot_u, g_i)$, $i = 1, 2, 3$, to ensure that $\mathcal{A}_u$ is compatible with g_i. This step ensures that $\mathcal{A}_\Delta$ is void of $\mathcal{S}_u$, which implies that $\mathcal{B}_u$ is void of $\mathcal{C}$.

(iv) If u is an interior node with v and w as its children, then let Σ_u be the set of at most 6 separating planes at u. For each $\sigma \in \Sigma_u$, we call GlobalCut$(\boxdot_u, \sigma)$ to ensure that $\mathcal{A}_u$ is compatible with each $\sigma \in \Sigma_u$, which in turn ensures that $\mathcal{A}_u$ becomes compatible with $\boxdot_v$ and $\boxdot_w$.

This completes the description of the (non-recursive) processing of a node u of $\mathcal{T}$. If u is an interior node, the algorithm recursively visits the two children of u (in a preorder fashion).

3.3 The two procedures

We now describe the two subroutines called by the main algorithm.

The GlobalCut procedure. Given an annular region (or box) Δ and a plane h, GlobalCut(Δ, h) ensures that $\mathcal{A}_\Delta$ is compatible with h, i.e., for each box $B \in \mathcal{A}_\Delta$, $\mathrm{int}(B) \cap \partial\mathcal{U}$ lies in only one of the two open halfspaces bounded by h. As a result, no face of $\partial\mathcal{U}$ that lies on h intersects the interior of any box in $\mathcal{A}_\Delta$ afterwards, i.e., $\mathrm{int}(B) \cap \partial\mathcal{U} \cap h = \varnothing$ for any box $B \in \mathcal{A}_\Delta$.

We visit each box $B \in \mathcal{A}_\Delta$ one by one and perform the following steps. If $\mathrm{int}(B) \cap \partial\mathcal{U} \cap h = \varnothing$ and B is compatible with h, leave B as it is. Otherwise, we divide B into two boxes $B^- := B \cap h^-$ and $B^+ := B \cap h^+$ by splitting B by h. See Figure 6. If the box B^- (or B^+) admits a free cut, we split it by the free cut. We perform this step repeatedly until the resulting boxes have no free cuts.

We note that if $B \cap \partial\mathcal{U}$ lies in one of the open halfspaces bounded by h, then GlobalCut does not split B even if h intersects its interior. See box B_4 in Figure 6 (left). This simple rule is crucial in keeping the size of the decomposition small.

The Staircase procedure. Given a box Δ, where $\mathcal{A}_\Delta$ is compatible with Δ, an edge e of Δ, and a set X of cubes, each of which intersects Δ and contains e (and thus it is long at Δ), Staircase(Δ, e, X) refines $\mathcal{A}_\Delta$ so that it becomes void of X. Recall that for any call Staircase(Δ, e, x) made during step (ii) of the algorithm, Δ is indeed a box, not an annular region.

If $\Delta = \varnothing$ or $X = \varnothing$, Δ is trivially void of X, and the procedure terminates. So assume that both $\Delta \neq \varnothing$ and $X \neq \varnothing$. We assume that each cube C of X appears on $\partial\mathcal{U}(X) \cap \mathrm{int}(\Delta)$ because otherwise we can simply ignore C in the present invocation of the

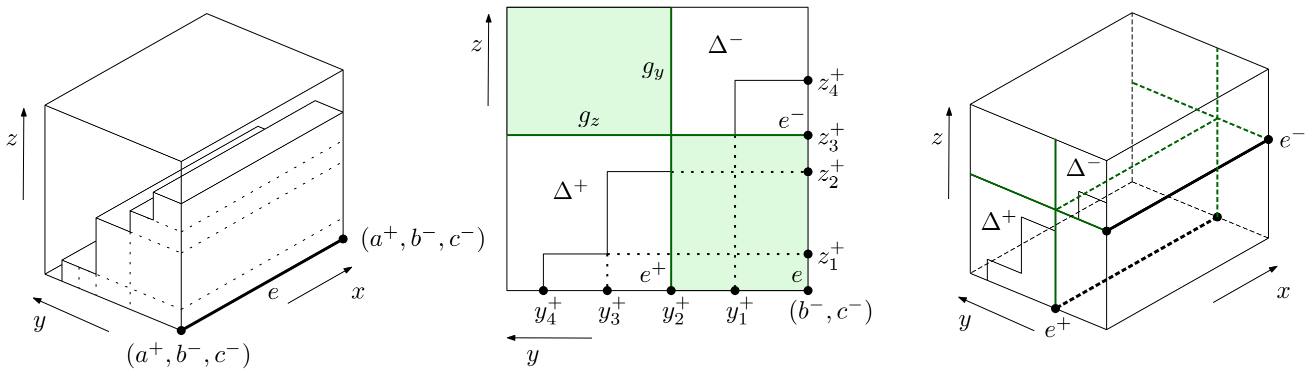

Figure 7. An illustration of STAIRCASE(Δ, e, X).

procedure (removing C does not alter $\partial\mathcal{U}(X) \cap \text{int}(\Delta)$). Suppose that $\Delta = [a^-, a^+] \times [b^-, b^+] \times [c^-, c^+]$ and $e = [a^-, a^+] \times \{b^-\} \times \{c^-\}$ for concreteness. For a cube $C_i \in X$, let $[y_i^-, y_i^+]$ (resp. $[z_i^-, z_i^+]$) be its projection on the y-axis (resp. z-axis). Let $C_1, C_2, \ldots, C_r$ be the cubes in X sorted in increasing order of their upper y-coordinates, i.e., $y_1^+ < y_2^+ < \cdots < y_r^+$. As we only consider cubes in X that appear on $\partial\mathcal{U}(X)$ inside Δ, we have $z_1^+ > z_2^+ > \cdots > z_r^+$. Let g_y be the plane $y = y^+_{\lceil r/2 \rceil}$ and g_z be the plane $z = z^+_{\lceil r/2 \rceil}$. We partition Δ into four boxes by the planes g_y and g_z. See Figure 7. The box lying in the quadrant $g_y^- \cap g_z^-$ lies inside $\mathcal{U}$ and the box lying in the quadrant $g_y^+ \cap g_z^+$ is disjoint from X, so neither of these two boxes need to be processed further at this invocation of STAIRCASE. We first make $\mathcal{A}_\Delta$ void of $\{C_{\lceil r/2 \rceil}\}$, and then solve the problem recursively in the two remaining boxes that lie in the quadrants $g_y^- \cap g_z^+$ and $g_y^+ \cap g_z^-$, as follows.

We first call GLOBALCUT(Δ, g_y) and GLOBALCUT(Δ, g_z) to ensure $\mathcal{A}_\Delta$ is compatible with both g_y and g_z, and hence no face of $\partial\mathcal{U}$ on g_y or g_z intersects the interior of any box in $\mathcal{A}_\Delta$; in particular, $\mathcal{A}_\Delta$ is void of $\{C_{\lceil r/2 \rceil}\}$. Note that if $r \leq 2$, we could have $y^+_{\lceil r/2 \rceil} \geq b^+$ or $z^+_{\lceil r/2 \rceil} \geq c^+$, in which cases $\mathcal{A}_\Delta$ is already compatible with g_y or g_z, and so there is no need to call GLOBALCUT(Δ, g_y) or GLOBALCUT(Δ, g_z), respectively.

Let Δ^- (resp. Δ^+) be the box $\Delta^- := \Delta \cap g_y^- \cap g_z^+$ (resp. $\Delta^+ := \Delta \cap g_y^+ \cap g_z^-$), let

$$e^- := [a^-, a^+] \times \{b^-\} \times \{z^+_{\lceil r/2 \rceil}\}, \text{ and}$$
$$e^+ := [a^-, a^+] \times \{y^+_{\lceil r/2 \rceil}\} \times \{c^-\},$$

and let $X^- := \{C_1, \ldots, C_{\lceil r/2 \rceil - 1}\}$ and $X^+ := \{C_{\lceil r/2 \rceil + 1}, \ldots, C_r\}$. By construction, $\partial\mathcal{U}(X^-) \cap \text{int}(\Delta) \subseteq \Delta^-$ and $\partial\mathcal{U}(X^+) \cap \text{int}(\Delta) \subseteq \Delta^+$. See Figure 7 again. We recursively call STAIRCASE(Δ^-, e^-, X^-) and STAIRCASE(Δ^+, e^+, X^+) to ensure that $\mathcal{A}_{\Delta^-}$ and $\mathcal{A}_{\Delta^+}$, and thus $\mathcal{A}_\Delta$, become void of X. (Note that, indeed, immediately before the recursive calls, Δ^- (resp. Δ^+) is void of $X^+ \cup \{C_{\lceil r/2 \rceil}\}$ (resp. $X^- \cup \{C_{\lceil r/2 \rceil}\}$), $\mathcal{A}_{\Delta^-}$ (resp. $\mathcal{A}_{\Delta^+}$) is compatible with Δ^- (resp. Δ^+), and each box $B \in X^-$ (resp. $B \in X^+$) contains the edge e^- (resp. e^+) of Δ^- (resp. Δ^+).)

3.4 Analysis

In this subsection, we prove the correctness of the algorithm, bound the size of the subdivision that it produces, and analyze its running time. We first introduce two concepts that will be useful for the analysis.

History tree. We note that the GLOBALCUT procedure is the only procedure that refines the subdivision $\mathcal{B}$ (the main procedure and STAIRCASE refine $\mathcal{B}$ only through calls to GLOBALCUT) by subdividing a box of $\mathcal{B}$ into two boxes B^-, B^+ by a cut (which is a rectangle of the form $B \cap h$ for some axis-aligned plane h); see Figure 6. Let $\mathcal{B}_0, \mathcal{B}_1, \ldots, \mathcal{B}_F$ be the sequence of subdivisions that arise during the execution of the algorithm, so that $\mathcal{B}_0 = \{\square\}$, $\mathcal{B}_{i+1}$ is obtained from $\mathcal{B}_i$ by splitting a box of $\mathcal{B}_i$ into two boxes, and $\mathcal{B}_F$ is the final subdivision. We define a binary tree $\mathsf{H} := (\mathsf{V}, \mathsf{E})$, which we refer to as the *history tree* of the algorithm. V is the set of boxes that appear in at least one $\mathcal{B}_i$. If a box B was split into two boxes B_1, B_2 by a cut, we add the edges (B, B_1) and (B, B_2) to H, making B_1 and B_2 the children of B. The leaves of H are the set of boxes in the final subdivision $\mathcal{B}_F$.

Fragments. A *fragment* is a maximal connected portion of a face of $\partial\mathcal{U}$ that is contained in the interior of a box of some $\mathcal{B}_i$. See Figure 8 for an illustration. Fix a face f of $\partial\mathcal{U}$. The face f itself is a fragment because it is the unique maximal connected portion of f lying in the interior of the initial box $\square$ of $\mathcal{B}_0$. Let φ be a fragment of f lying in the interior of a box B of $\mathcal{B}$. As the algorithm progresses, φ is repeatedly divided into smaller fragments by cuts, until it is "trapped" by a cut

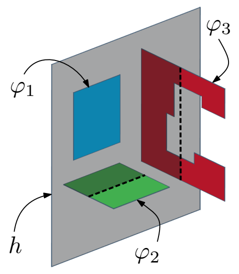

Figure 8. Examples of fragments that are alive in a common box (not shown) that is cut by plane h. φ_1 becomes eternal, φ_2 and φ_3 die, and the maximally connected portions of φ_2, φ_3 in h^-, h^+ are newly created fragments — two for φ_2 and three for φ_3.

that contains φ (i.e., as B is split by the plane supporting φ), and then φ stops being a fragment and will not be divided further anymore — see below. If φ does not touch ∂f, then φ is rectangular and corresponds to a free cut, so the box that contains φ is split by this cut as soon as φ materializes, and φ is never subdivided again.

If α is the highest node of H at which φ appears, we say that φ is *created* at α (every original face of $\partial\mathcal{U}$ is created at the root). Fragment φ continues to appear at nodes along a (unique) path starting from α in H until one of the following two events occurs at a descendant β of α, for which φ is still a fragment in the box β:

(i) The box β is split by a cut orthogonal to φ that divides φ into multiple fragments, each of which is created in one of the two children of β. In this case we say that φ *dies* at node β.

(ii) The box β is split by a cut supporting φ so that φ no longer appears in the interior of any box of $\mathcal{B}$, and is no longer further divided. In this case, we say that φ becomes *eternal* at β; officially, φ is no longer a fragment, but we regard it as staying alive. See φ_1 in Figure 8.

We note that while an eternal fragment φ is not further divided, a box B whose boundary contains (a portion of) φ may be split by a cut orthogonal to φ. Although this cut may intersect φ, it cannot cross φ — it terminates at φ and does not subdivide φ. Since the leaves of H are void of $\mathcal{C}$, each fragment either dies or becomes eternal during the execution of the algorithm. Let Φ_F denote the set of eternal fragments when the algorithm terminates, and let Γ be the set of vertices of fragments in Φ_F. Set $\mu \coloneqq |\Phi_F|$ and $\nu \coloneqq |\Gamma|$. At most four fragments share any vertex in Γ, so we have $\mu \leq 4\nu$. Since the multiplicity of any element in Γ is at most four, with a slight abuse of notation, we will use Γ to denote both the set and the multiset of vertices of fragments in Φ_F. We will prove in Lemma 3.2 that the size of the final subdivision, $\mathcal{B}_F$, is $O(\mu)$, and then bound ν by $O(\kappa \log^4 n)$ in Corollary 3.8. Together, these bounds imply the stated upper bound for $|\mathcal{B}_F|$.

Proof of correctness. We now prove the correctness of the algorithm.

LEMMA 3.1. *The algorithm maintains invariants (I1), (I2), (I3).*

Proof. Free cuts are created by the GLOBALCUT procedure, which is the only procedure that refines $\mathcal{B}$. Since GLOBALCUT splits boxes by free cuts as soon as they appear, (I3) is maintained — none of the boxes of $\mathcal{B}$ admit a free cut.

Next, we prove that invariant (I1), namely that upon reaching a node u of $\mathcal{T}$, $\mathcal{A}_u$ is compatible with $\boxdot_u$, holds, by induction on the depth of the nodes v of $\mathcal{T}$. It is trivially true initially at the root of $\mathcal{T}$ because $\square$ contains $\partial\mathcal{U}$. Suppose the algorithm arrives at a node v. By induction hypothesis, (I1) holds at $p(v)$, so if (I1) is not true at v, there is a box $B \in \mathcal{A}_v$ such that B intersects one of the separating planes $\sigma \in \Sigma_{p(v)}$ and $\operatorname{int}(B) \cap \partial\mathcal{U}$ lies in both open halfspaces bounded by σ, i.e., B is not compatible with σ. However, this is impossible because step (iv) of the algorithm at $p(v)$ calls GLOBALCUT$(\boxdot_{p(v)}, \sigma)$ with all planes σ in $\Sigma_{p(v)}$, which would have made $\mathcal{A}_{p(v)}$, and thus $\mathcal{A}_v$, compatible with $\boxdot_v$. Hence (I1) holds at v too.

We now prove that (I2), namely that upon finishing processing a node u of $\mathcal{T}$, $\mathcal{A}_u$ is void of $\mathcal{L}_u$ (and of $\mathcal{S}_u$ when u is a leaf), holds by induction on the depth of the nodes v of $\mathcal{T}$. The invariant is trivially true at the root u of $\mathcal{T}$ because $\mathcal{L}_u = \varnothing$. Suppose the algorithm has processed a node v of $\mathcal{T}$. Since $p(v)$ is processed before v, by induction hypothesis, $\mathcal{A}_{p(v)}$ was void of $\mathcal{L}_{p(v)}$ when the processing of v began. Hence, $\mathcal{A}_v$ is void of $\mathcal{L}_{p(v)}$, so it suffices to prove that $\mathcal{A}_v$ is void of $\mathcal{C}_v = \mathcal{L}_v \setminus \mathcal{L}_{p(v)}$. Since (I1) holds, for any box $B \in \mathcal{A}_v$, $B \setminus \boxdot_v$ is void of $\mathcal{C}_v$, so we only focus on portions that lie inside $\boxdot_v$. Let Ξ be the subdivision of $\boxdot_u$ provided by Lemma 2.1.

It suffices to prove that $\mathcal{A}_\Delta$ is void of $\mathcal{C}_v$ for each $\Delta \in \Xi_v$. Step (i) ensures that for each $\Delta \in \Xi_v$, $\mathcal{A}_\Delta$ is compatible with Δ. Let $C \in \mathcal{C}_v$ be a (long) cube that intersects the interior of a box $\Delta \in \Xi_v$. By Lemma 2.1, C contains one of the edges of Δ, say, e. Then the call to STAIRCASE$(\Delta, e, \mathcal{C}_{\Delta,e})$ makes Δ void of C. Hence, after the procedure STAIRCASE$(\Delta, e, \mathcal{C}_{\Delta,e})$ is invoked for all edges of Δ, $\mathcal{A}_\Delta$ is void of $\mathcal{C}_v$. After repeating this step for all boxes $\Delta \in \Xi_v$, $\mathcal{A}_v$ becomes void of $\mathcal{C}_v$. Finally, if v is a leaf then, in addition to the argument just given, step (iii) of the algorithm ensures that $\mathcal{A}_v$ is void of $\mathcal{S}_v$ as well.

Putting it all together, we conclude that the algorithm maintains the invariants (I1)–(I3). $\square$

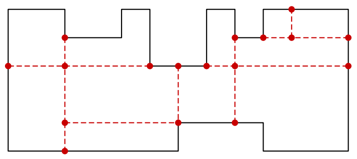

Figure 9. A 2D view of the subdivision Π_f (red) of a face f of $\partial\mathcal{U}$.

The decomposition size. The following sequence of lemmas bound the size of $\mathcal{B}_F$.

LEMMA 3.2. *The size of the final subdivision $\mathcal{B}_F$ is at most 2μ, where μ is the total number of fragments that are alive (and eternal) at the end of the algorithm.*

Proof. The size of $\mathcal{B}_F$ is the same as the number of leaves in the history tree H. Let B be a leaf of H, and let B' be the parent of B in H, so B' was split into two boxes, B and, say, $\overline{B}$, by a plane h made at B'. We claim that h supported a live fragment φ lying in $\mathrm{int}(B')$. Suppose h did not support a fragment inside B', i.e., either h divided an alive fragment in B' into two fragments or $h \cap \mathrm{int}(B') \cap \partial\mathcal{U} = \varnothing$ (see Figure 6). In either case, both $\mathrm{int}(B)$ and $\mathrm{int}(\overline{B})$ intersect $\partial\mathcal{U}$, which contradicts the assumption that B is a leaf, so our claim is true. We charge the leaf B to the fragment $\varphi \in \Phi_F$, which becomes eternal after the node B' is processed and does not appear in the interior of any descendant of B'. Hence, φ is charged at most twice, namely, once for each child of B' that is a leaf. Furthermore, $\varphi \in \Phi_F$. Hence, $|\mathcal{B}_F| \le 2\mu$. $\square$

The next four lemmas bound the value of ν, the number of fragment vertices. Specifically, we prove that a face f of $\partial\mathcal{U}$ with κ_f vertices contains $O(\kappa_f \log^4 n)$ fragment vertices. To prove this bound, we consider the evolving subdivision $\widetilde{\Pi}_f$ of f induced by the cuts that cross f throughout the execution of the algorithm and monitor how $\widetilde{\Pi}_f$ evolves over time. In particular, whenever a box B with $f \cap \mathrm{int}(B) \neq \varnothing$ is split by an axis-aligned plane h crossing f, each segment (connected component) of $f \cap (B \cap h)$ *creates* a new edge of $\widetilde{\Pi}_f$. (Cuts that intersect f but do not cross it do not subdivide f.) The endpoints of a new edge lie on the existing edges of $\widetilde{\Pi}_f$ (possibly on edges of f), become new vertices of $\widetilde{\Pi}_f$, and subdivide those existing edges.

Let Π_f denote the final subdivision of f when the algorithm terminates. The faces of Π_f are the eternal fragments of Φ_F that lie on f. See Figure 9. By definition, every edge of Π_f is a portion of a segment γ corresponding to some cut used during the algorithm that crosses f. Many edges may lie on such a segment γ as subsequent segments whose endpoints lie on γ may have subdivided γ. We say that an edge of Π_f was *created* when its corresponding segment γ was created in $\widetilde{\Pi}_f$ during the execution of the algorithm. Additionally, we label an edge γ of Π_f with a node v of the BBD tree $\mathcal{T}$ if the call to GLOBALCUT that created γ was executed while processing v.

We call an axis-parallel segment contained in a face of $\partial\mathcal{U}$ a *mast* (as in [24]).

LEMMA 3.3. *Let γ be a mast lying on a face f of $\partial\mathcal{U}$. Let $\mathcal{E}_{\Delta,h}$ be the set of edges of Π_f that were created by cuts made by some single call to* GLOBALCUT(Δ, h). *If γ is parallel to h, then γ crosses no edge in $\mathcal{E}_{\Delta,h}$; otherwise γ crosses at most one edge of $\mathcal{E}_{\Delta,h}$.*

Proof. By definition, none of the free cuts made by GLOBALCUT(Δ, h) cross any face of $\partial\mathcal{U}$, so they do not create any edges of $\mathcal{E}_{\Delta,h}$. Hence, all the edges of $\mathcal{E}_{\Delta,h}$ created by GLOBALCUT(Δ, h) lie on the line $\ell := h \cap \mathrm{span}(f)$. If γ is parallel to h, then γ is parallel to ℓ so γ crosses no edges in $\mathcal{E}_{\Delta,h}$. Otherwise γ crosses ℓ at most once, so it crosses at most one edge of $\mathcal{E}_{\Delta,h}$. $\square$

LEMMA 3.4. *Let γ be a mast lying on a face f of $\partial\mathcal{U}$. Let $\mathcal{E}_{\Delta,e,X}$ be the set of edges of Π_f that were created by cuts made by a single call of* STAIRCASE(Δ, e, X). *Then γ crosses $O(\log|X|)$ edges of $\mathcal{E}_{\Delta,e,X}$.*

Proof. Following the notation in the description of the STAIRCASE procedure, assume that e is parallel to the x-axis, let g_y and g_z be the two "median" planes for which the procedure called GLOBALCUT(Δ, g_y) and GLOBALCUT(Δ, g_z), and let Δ^- and Δ^+ be the two sub-boxes of Δ for which the STAIRCASE procedure was called recursively, as STAIRCASE(Δ^-, e^-, X^-) and STAIRCASE(Δ^+, e^+, X^+).

We first note that if γ is parallel to e, then γ does not cross any edge of $\mathcal{E}_{\Delta,e,X}$ because all cutting planes with which GLOBALCUT is called inside STAIRCASE(Δ, e, X), including recursive calls, are parallel to e and thus to γ. By Lemma 3.3, none of the edges of $\mathcal{E}_{\Delta,e,X}$ (which lie in these cutting planes) are crossed by γ. So assume γ is orthogonal to e, say, γ is parallel to the y-axis; a symmetric argument holds if γ is parallel to the z-axis.

Since γ is parallel to the y-axis, it is not crossed by the plane g_z. By Lemma 3.3, γ crosses no edge created by GLOBALCUT(Δ, g_z) and at most one edge created by GLOBALCUT(Δ, g_y). See Figure 10. Since γ misses g_z, γ also misses Δ^+ (resp. Δ^-) if it lies in the halfspace g_z^+ (resp. g_z^-). If γ misses Δ^+ (resp. Δ^-), it is only crossed by the edges of $\mathcal{E}_{\Delta,e,X}$ that are created by the recursive call STAIRCASE(Δ^-, e^-, X^-) (resp. STAIRCASE(Δ^+, e^+, X^+)). Using the fact that

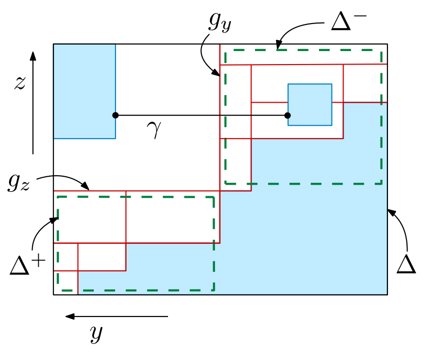

Figure 10. A 2D view of the edges $\mathcal{E}_{\Delta,e,X}$ (red) created in subdivision Π_f on $\Delta \cap \mathrm{span}(f)$ after calling STAIRCASE(Δ, e, X). The shaded regions (blue) form the cross section $\mathcal{U} \cap \Delta$. The boundaries of Δ^-, Δ^+ in the initial call are shown slightly shrunk towards their centers for better visualization.

$|X^-|, |X^+| \le \lceil |X|/2 \rceil - 1 \le |X|/2$, a simple recursive argument shows that γ crosses $O(\log|X|)$ edges of $\mathcal{E}_{\Delta,e,X}$. □

LEMMA 3.5. *Let γ be a mast lying on a face f of $\partial\mathcal{U}$. For any node v of $\mathcal{T}$, γ crosses $O(\log n)$ edges of Π_f labeled v.*

Proof. Let $\mathcal{E}_v$ be the set of edges of Π_f that are labeled v. Consider the cuts made by the algorithm while processing v. Step (i) calls GLOBALCUT $O(1)$ times, and γ crosses at most one edge of $\mathcal{E}_v$ created by each call, by Lemma 3.3. These calls split γ into $O(1)$ segments, each of which lies in one of the boxes of the subdivision Ξ_v of $\boxdot_v$. Fix a box $R \in \Xi_v$ intersecting γ and let $\gamma_R \coloneqq \gamma \cap R$. For each edge e of R, by Lemma 3.4, γ_R crosses $O(\log|\mathcal{C}_{\mathcal{R},e}|) = O(\log n)$ edges of $\mathcal{E}_v$ that are created by STAIRCASE$(R, e, \mathcal{C}_{R,e})$. Summing over all $O(1)$ such calls, γ_R is crossed by $O(\log n)$ edges of $\mathcal{E}_v$ that are created at step (ii). Next, summing over the $O(1)$ boxes $R \in \Xi_v$, γ is crossed by $O(\log n)$ edges of $\mathcal{E}_v$ created at step (ii).

If v is a leaf, γ crosses at most three edges of $\mathcal{E}_v$ that are created at step (iii). Finally, if v is an interior node, γ crosses $O(1)$ edges of $\mathcal{E}_v$ that are created at step (iv).

Putting everything together, γ crosses $O(\log n)$ edges of $\mathcal{E}_v$. □

LEMMA 3.6. *A mast γ lying on a face f of $\partial\mathcal{U}$ crosses $O(\log^2 n)$ edges of Π_f.*

Proof. Let $C \in \mathcal{C}$ be the cube whose boundary contains the face f. For a node v of $\mathcal{T}$, let $i_v \le F$ be the index such that $\mathcal{B}_{i_v}$ is the subdivision immediately before the algorithm begins processing v. Let $\mathcal{A}_v^<$ be the set of active boxes B in $\mathcal{B}_{i_v}$; $\mathrm{int}(B) \cap \partial\mathcal{U} \cap \boxdot_v \ne \varnothing$ for all $B \in \mathcal{A}_v^<$. By invariant (I1), $\mathrm{int}(B) \cap \partial\mathcal{U} \subseteq \boxdot_v$ for all $B \in \mathcal{A}_v^<$.

By invariant (I2), if f lies in the interior of a box of $\mathcal{A}_v^<$ then C is short at $p(v)$. Since the interiors of the regions $\boxdot_u$ are pairwise-disjoint for all nodes at a fixed level of $\mathcal{T}$, there are at most 16 nodes v (twice the number of vertices of C) at any level of $\mathcal{T}$ for which f intersects the interior of a box of $\mathcal{A}_v^<$, i.e., f contains a fragment that is alive at node v and may be further subdivided. Hence, there are $O(\log n)$ nodes v of $\mathcal{T}$ for which $\mathcal{A}_v^<$ is not void of $\{C\}$, and thus the edges of Π_f have $O(\log n)$ distinct labels. By Lemma 3.5, γ crosses $O(\log n)$ edges of each label, so γ crosses $O(\log^2 n)$ edges of Π_f. □

To bound the number of vertices of Π_f, we construct a standard 2D vertical decomposition of the face f: Without loss of generality, assume that f is parallel to the xy-plane. From each vertex q of f, we draw a ray in the $(+y)$-direction or in the $(-y)$-direction within the interior of f until it hits another edge of f. (Only one of the two rays lies in the interior of f in the neighborhood of q.) The resulting subdivision $f^{||}$ of f consists of a set of $O(\kappa_f)$ axis-aligned rectangles with pairwise-disjoint interiors.[1] Consider any rectangle ρ of $f^{||}$. The x-edges of ρ are portions of ∂f but the y-edges may not lie in ∂f or may partially overlap with ∂f. Let Π_ρ be the subdivision of ρ induced by Π_f by clipping Π_f in the interior of ρ and adding $\partial\rho$ to it; see Figure 11(a). Each vertex of Π_f lying in ρ is a vertex of Π_ρ, so it suffices to bound the number of vertices of Π_ρ. If a vertex ξ of Π_f lies in the interior of ρ, ξ is a vertex of Π_ρ but if ξ lies on $\partial\rho$ then it might not be a vertex of Π_ρ (e.g. grey vertices in Figure 11(a)). However, ξ will be a vertex of $\Pi_{\rho'}$ for some other rectangle ρ' of $f^{||}$.

The following lemma, which is similar to Proposition 7 in [24], bounds the number of vertices of Π_ρ.

LEMMA 3.7. *For each rectangle ρ of $f^{||}$, Π_ρ contains $O(\log^4 n)$ vertices.*

Proof. We call a vertex of Π_ρ lying in $\mathrm{int}(\rho)$ an *inner* vertex, and *outer* otherwise. Since each edge ε of ρ is a mast and each outer vertex of Π_ρ on ε (except possibly for its corners) is formed by the intersection of ε with an edge of Π_f, Lemma 3.6 implies[2] that ε contains $O(\log^2 n)$ vertices of Π_ρ. Hence, Π_ρ has $O(\log^2 n)$ outer vertices.

[1]The subdivision of $f^{||}$ is constructed only for the analysis. It is not part of the algorithm.

[2]Since vertices of Π_f that are not vertices of Π_ρ might lie on ε, we should apply Lemma 3.6 to the segment ε' that is the slight translation of $\mathrm{int}(\varepsilon)$ into $\mathrm{int}(\rho)$; the number of edges of Π_f crossed by ε' correspond to the outer vertices on ε, which is what we want to bound here.

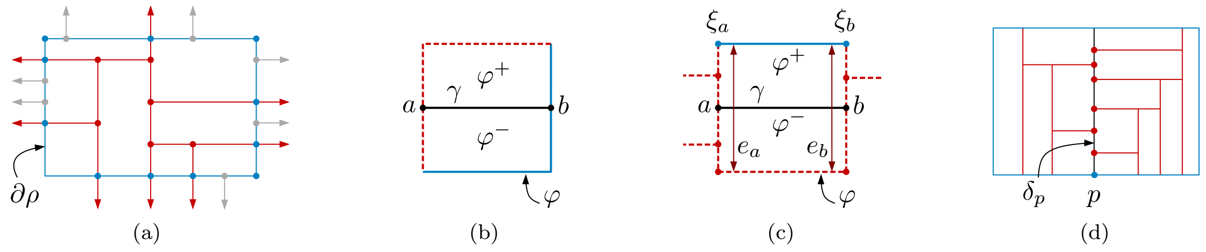

Figure 11. (a) An illustration of Π_f clipped within ρ is shown in red, where the outer vertices, inner vertices, and vertices of Π_f that are on edges of $\partial\rho$ but not in Π_ρ are depicted in blue, red, and grey, respectively. (b) An exposed face φ of $\widetilde{\Pi}$ is split by segment γ, where the solid blue edges (resp. dashed red edges) lie on $\partial\rho$ (resp. in int(ρ)). b is an outer vertex, which a charges, being an inner vertex. (c) Similar to (b), except that a and b are both inner vertices and a (resp. b) charges the outer vertex ξ_a (resp. ξ_b). Edges e_a, e_b of ρ are shown slightly shifted inwards for better visualization. (d) An illustration of mast δ_p, all of whose incident inner vertices (red) charge the outer vertex p.

See Figure 11(a). We carefully charge each inner vertex to an outer vertex so that each outer vertex is charged by $O(\log^2 n)$ inner vertices. This would imply that the number of inner vertices in Π_ρ is $O(\log^4 n)$, as claimed. We now describe the charging scheme and argue that each outer vertex is indeed charged only $O(\log^2 n)$ times.

To analyze the charging of the inner vertices of Π_ρ, instead of viewing Π_ρ as a static subdivision, we monitor the evolution of Π_ρ as the algorithm progresses and the subdivision is being refined. Let $\widetilde{\Pi}$ denote this dynamic subdivision of ρ. Initially, $\widetilde{\Pi} = \rho$, and $\widetilde{\Pi} = \Pi_\rho$ when the algorithm terminates. A face φ of $\widetilde{\Pi}$ lying completely in the interior of ρ is a face of Π_f that lies completely in the interior of f, and thus φ corresponds to a free cut in some box containing φ. Since the algorithm splits boxes by free cuts as soon as they appear, φ becomes eternal (and is never further refined).

We call a face φ of $\widetilde{\Pi}$ *exposed* if at least one of its edges lies on $\partial\rho$. At each step, $\widetilde{\Pi}$ is refined by splitting an exposed face φ of $\widetilde{\Pi}$ into two rectangles φ^-, φ^+ by a segment γ. The endpoints of γ, denoted by a and b, create new vertices of $\widetilde{\Pi}$. If an endpoint of γ is an outer vertex, it is already acccounted for in the sense that we have already bounded the number of outer vertices ever created by $O(\log^2 n)$, so assume that at least one of a and b is an inner vertex. There are two cases to consider. The first case is when a is an inner vertex and b is an outer vertex (or vice versa). In this case, we charge a to the newly created outer vertex b. Each outer vertex is charged at most once by this case. See Figure 11(b).

The second case is when both a and b are inner vertices. We regard the face φ of Π_ρ that is split by γ as a rectangle, and let e_a (resp. e_b) be the edge of this rectangle that contains a (resp. b). Note that e_a (resp. e_b) may contain vertices of $\widetilde{\Pi}$ in its interior; see Figure 11(c). Neither e_a nor e_b lies on $\partial\rho$. Since φ is exposed, at least one of the other two edges of (the rectangle) φ lies on $\partial\rho$. Let ξ_a (resp. ξ_b) be an endpoint of e_a (resp. e_b) lying on $\partial\rho$, i.e., ξ_a, ξ_b are outer vertices in $\widetilde{\Pi}$. We charge a (resp. b) to the outer vertex ξ_a (resp. ξ_b). See Figure 11(c).

We claim that each outer vertex p is charged by $O(\log^2 n)$ inner vertices. Indeed, let δ_p be the segment connecting p to its opposite point on $\partial\rho$. (Note that at least part of δ_p, but not necessarily all of it, is covered by edges of Π_ρ.) Each inner vertex charged to p lies on δ_p and is an intersection point of δp with an orthogonal edge of Π_f (which is orthogonal to δ_p). By Lemma 3.6,[3] δ_p contains $O(\log^2 n)$ such intersection points. See Figure 11(d). Hence, p is charged by $O(\log^2 n)$ inner vertices, as claimed. This completes the proof of the lemma. ◻

An immediate corollary of the above lemma is the following:

Corollary 3.8. *A face f of $\partial\mathcal{U}$ with κ_f vertices is split into $O(\kappa_f \log^4 n)$ eternal fragments.*

Putting everything together, we conclude that the size of $\mathcal{B}_F$ is $O(\kappa \log^4 n)$, thereby proving the first part of Theorem 1.1.

Runtime analysis. We now show that the algorithm described above can be implemented in $O(n \log^2 n + \kappa \log^6 n)$ time by carefully maintaining some auxiliary information.

Recall that, at any time during the execution of the algorithm, $\mathcal{B}$ and Φ denote the current set of boxes and fragments, respectively. Let $\varphi \in \Phi$ be a fragment. For each connected component of $\partial\varphi$, we store the sequence of its vertices in cyclic order in a doubly linked list. Let L_φ be this list. For each box $B \in \mathcal{B}$, let $\Phi_B \subseteq \Phi$ be the

[3]As similarly remarked earlier, to use the lemma, we choose a mast parallel and very close to δ_p.

set of fragments that lie in the interior of B, and let Γ_B be the *multiset* of vertices of fragments in Φ_B; since at most four fragments share any vertex, each element in Γ_B has multiplicity at most four. For each box B, we maintain the set Φ_B and three lists $\mathsf{X}_B, \mathsf{Y}_B, \mathsf{Z}_B$ storing Γ_B sorted by their x-, y-, and z-coordinates, respectively. We store L_φ, X_B, Y_B, and Z_B as doubly linked lists and store cross pointers among them so that for a vertex in one of the lists, we can locate it in the other lists in $O(1)$ time. In addition, whenever we make a call GLOBALCUT(Δ, h), we ensure that we have the set $\mathcal{A}_\Delta$ of active boxes at our disposal, so that the procedure does not have to compute $\mathcal{A}_\Delta$ from scratch.

Since GLOBALCUT is the only procedure that modifies $\mathcal{B}$, we sketch how to implement GLOBALCUT(Δ, h) efficiently, omitting various tedious details:

1. Without loss of generality, assume that $h : z = z_0$ is parallel to the xy-plane. For each box $B \in \mathcal{A}_\Delta$, we scan Z and find the last vertex ξ^- with z-coordinate less than z_0. Next, we scan the set Φ_B of its fragments. For each fragment $\varphi \in \Phi_B$, by scanning the list L_φ, we test in $O(|\varphi|)$ time whether φ intersects h, where $|\varphi|$ is the number of vertices of φ. If the answer is no, we determine in $O(1)$ time whether φ lies in h^- or in h^+.

2. If a fragment intersects h or if both h^- and h^+ contain fragments, we split B into two boxes $B^- := B \cap h^-$ and $B^+ := B \cap h^+$.

3. If B is split into B^+ and B^-, then we perform the following steps:

   (a) By scanning the list L_φ, for each fragment $\varphi \in \Phi_B$, we first generate the intersection points of h with the edges of φ. Using ξ^- and cross pointers, we can store each new vertex in the lists $\mathsf{X}_B, \mathsf{Y}_B$, and Z_B in $O(1)$ time.

   (b) After having computed all new fragment vertices in B, we scan the lists X_B and Y_B and compute the new fragment edges that lie on h.

   (c) We then split the fragments intersecting h and create the lists L_φ for each newly created fragment φ. A fragment φ may be split into many fragments (see Figure 8). Each fragment now either lies in B^- or in B^+.

   (d) By scanning the lists $\Phi_B, \mathsf{X}_B, \mathsf{Y}_B$, and Z_B we construct the lists Φ_{B^-}, Φ_{B^+}, X_{B^-}, X_{B^+}, Y_{B^-}, Y_{B^+}, Z_{B^-}, and Z_{B^+}.

   (e) We identify fragments in Φ_{B^-}, Φ_{B^+} that induce free cuts. All these fragments are parallel to each other and orthogonal to h, i.e., all of them are parallel to the xz-plane or to the yz-plane. We split D by each free cut and construct the lists $\Phi_D, \mathsf{X}_D, \mathsf{Y}_D$, and Z_D for each newly created box D. The fragments that become eternal — either because they lie on h or they become free cuts — are discarded.

4. Finally, set $\Delta^+ := \Delta \cap h^+$ and $\Delta^- := \Delta \cap h^-$. GLOBALCUT ensures that $\mathcal{A}_{\Delta^+}$ (resp. $\mathcal{A}_{\Delta^-}$) is compatible with Δ^+ (resp. Δ^-). The procedure partitions the modified set $\mathcal{A}_\Delta$ into $\mathcal{A}_{\Delta^+}$ and $\mathcal{A}_{\Delta^-}$ and returns them.

Next, we analyze the total time spent by GLOBALCUT(Δ, h). Let $\nu_B := |\Gamma_B|$ denote the number of vertices of the fragments that lie in box $B \in \mathcal{A}_\Delta$, when the procedure is called. Note that Γ_B is a multiset here, and we count its elements with multiplicity. For each box $B \in \mathcal{A}_\Delta$, at most one new fragment vertex is created on any edge of a fragment in Φ_B during the execution of the procedure, namely in step (3.a). Thus, for each box $B \in \mathcal{A}_\Delta$, steps (1)–(4) are performed in $O(|\Gamma_B|)$ time. It follows that the total running time of GLOBALCUT(Δ, h) is $O(\nu_\Delta)$, where $\nu_\Delta := \sum_{B \in \mathcal{A}_\Delta} \nu_B$ is the number of vertices in the fragments that lie in a box of $\mathcal{A}_\Delta$ when the procedure was called.

Next, we note that STAIRCASE(Δ, e, X) spends $O(|X|)$ time to compute the cutting planes g_y and g_z, calls GLOBALCUT(Δ, g_y) and GLOBALCUT(Δ, g_z), each of which takes $O(\nu_\Delta)$ time, where ν_Δ is the number of vertices in Δ when GLOBALCUT is called, and then recursively calls STAIRCASE(Δ^-, e^-, X^-) and STAIRCASE(Δ^+, e^+, X^+). Using the list of active boxes returned by the two calls of the GLOBALCUT procedure, $\mathcal{A}_{\Delta^-}$ and $\mathcal{A}_{\Delta^+}$ can be computed in $O(\nu_\Delta)$ time. Each call to GLOBALCUT creates new fragment vertices, so the value of ν_Δ increases after each call. To handle this increase in the value of ν_Δ, for a region Δ, we define $\widetilde{\nu}_\Delta := |\Gamma \cap \Delta|$ to be the number of vertices of the eternal fragments that lie inside Δ at the end of the algorithm, counted with multiplicity. Then $\nu_\Delta \le \widetilde{\nu}_\Delta$ and $\widetilde{\nu}_{\Delta^-} + \widetilde{\nu}_{\Delta^+} \le \widetilde{\nu}_\Delta$. Using the fact that $|X^-|, |X^+| \le |X|/2$, a simple recurrence shows that STAIRCASE(Δ, e, X) takes $O((|X| + \widetilde{\nu}_\Delta) \log n)$ time.

For a node u of $\mathcal{T}$, let $\nu_u := |\Gamma \cap \boxdot_u|$ and $n_u := |\mathcal{S}_u| + |\mathcal{C}_u|$. We note that $\sum_{u \in \mathcal{T}} \nu_u = O(\nu \log n) = O(\kappa \log^5 n)$ by Lemma 3.2 and Corollary 3.8. The analysis in Arya *et al.* [2] implies that $\sum_{u \in \mathcal{T}} n_u = O(n \log n)$. A straightforward analysis shows that steps (i)–(iv) of the overall algorithm at a node u can be performed in $O((n_u + \nu_u) \log n)$ time as a result of the $O(1)$ calls made to STAIRCASE and GLOBALCUT. Summing over all nodes of $\mathcal{T}$, the total running time is $O(n \log^2 n + \kappa \log^6 n)$.

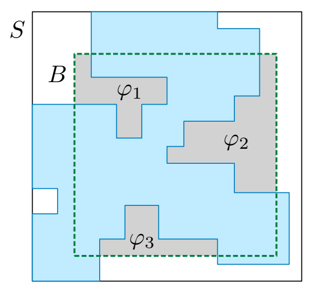

Figure 12. A 2D view of a boundary square S of some cube that induces a free cut, by the new definition, in the dashed box B (green), but none of the fragments $\varphi_1, \varphi_2, \varphi_3$ (grey) on $S \cap B$ induce a free cut in B by the old definition. $S \cap \text{int}(\mathcal{U})$ is shaded in blue.

This proves the second part of Theorem 1.1.

4 A Smaller Decomposition for Arbitrary Cubes

In this section we show that a small modification of the previous algorithm improves the size of the decomposition to $O(\sigma \log^4 n + \kappa \log^2 n)$, where $\sigma \le \min\{n, \kappa\}$ is the number of input cubes that appear on $\partial\mathcal{U}$. The only difference in the new algorithm is how we define a free cut for a box B of the current decomposition. Recall that a box B admits a free cut if there is a face f of $\partial\mathcal{U}$ that intersects $\text{int}(B)$ but $\partial f \cap \text{int}(B) = \varnothing$, i.e., $f \cap B = \text{span}(f) \cap B$. The algorithm splits B along $\text{span}(f) \cap B$ as soon as f induces a free cut in B because such a cut does not cross any fragments and $f \cap B$ no longer lies in the interior of the resulting boxes (and thus the name free cut). We observe that this property of "free" cuts holds even under a weaker condition. Namely, we say that B admits a *free cut* if B contains a fragment φ that lies on a boundary square[4] S of an input cube and $\partial S \cap \text{int}(B) = \varnothing$, i.e., $S \cap B = \text{span}(S) \cap B$. Note that, unlike the previous definition, $\partial\varphi$ may lie in the interior of B (see Figure 12), and S may cross the interior of $\mathcal{U}$. If we split B using the plane $\text{span}(S)$, φ will no longer lie in the interior of the resulting boxes and $\text{span}(S) \cap B$ will not cross any face of $\partial\mathcal{U}$ (though (a) it may meet the boundary of such a face, and (b) it may cross a portion of a boundary square that is disjoint from $\partial\mathcal{U}$). We run the algorithm described in Section 3.2 but use this definition of a free cut in the GLOBALCUT procedure.

We postpone the discussion on an efficient implementation of the modified GLOBALCUT until the runtime analysis given later in this section, and we first bound the size of the resulting decomposition $\mathcal{B}_F$. It is easily seen that Lemmas 3.1 and 3.2 still hold; the proof of the latter relies crucially on the fact that the splitting plane corresponding to a free cut contains a fragment. As in Section 3, it suffices to bound the number of eternal fragments, which we estimate by bounding the number of fragment vertices. In particular, we show that if a boundary square S of an input cube contains κ_S vertices of $\partial\mathcal{U}$, then S contains $O(\log^4 n + \kappa_S \log^2 n)$ fragment vertices, which will lead to the desired bound on the size of $\mathcal{B}_F$.

The overall structure of the proof is similar to that in Section 3.4 except that we use a more global argument. For a boundary square S, let $\mathcal{K}_S := \partial\mathcal{K} \cap S = \partial\mathcal{U} \cap S$ be the (possibly disconnected) portion of S that does not lie in $\text{int}(\mathcal{U})$. Throughout the execution of the algorithm, the splitting of boxes B with $S \cap \text{int}(B) \neq \varnothing$ by any plane h crossing S induces an evolving (rectangular) subdivision $\widetilde{\Pi}_S$ of S. Specifically, we have $\widetilde{\Pi}_S = S$ at the start (i.e., it consists of only the edges of S), and whenever such a split occurs, the axis-aligned segment $\gamma := S \cap (B \cap h)$ *creates* a new edge of $\widetilde{\Pi}_S$. The endpoints of γ lie on orthogonal edges of $\widetilde{\Pi}_S$ become new vertices of $\widetilde{\Pi}_S$ and subdivide those edges.

We color the features of $\widetilde{\Pi}_S$ as follows: Initially, we color the edges of $\widetilde{\Pi}_S = S$ as *black*. When a segment γ is created on S, we color (the interior of) γ as *red* if it intersects $\mathcal{K}_S$, and color it as black otherwise. Then we color each new vertex of $\widetilde{\Pi}_S$ induced by the endpoints of γ as *red* if it is incident to a red edge (which could be γ), and color it as black otherwise. When an edge of $\widetilde{\Pi}_S$ is subdivided, the sub-edges inherit the same color.

Let Π_S be the final (rectangular) subdivision of S when the algorithm terminates. By definition, every edge of the subdivision Π_S lies on a segment γ once created on $\widetilde{\Pi}_S$, and many edges may lie on the same γ as subsequent cuts may have subdivided γ further. We say that an edge of Π_S was *created* when its containing segment γ was created during the execution of the algorithm, and note that its color is that of γ when it was created.

Next, let Π_S^∇ be the subdivision of $\mathcal{K}_S$ obtained by overlaying Π_S with $\mathcal{K}_S$ and clipping it within $\mathcal{K}_S$. The faces of Π_S^∇ are eternal fragments. See Figure 13. We color the edges of Π_S^∇ that lie on $\partial\mathcal{K}_S$ as blue and the edges that lie in the interior of $\mathcal{K}_S$ (i.e., the clipped red edges of Π_S) as red. Note that a red edge of Π_S^∇ is a red edge of Π_S or is contained in a red edge of Π_S, and that the black edges of Π_S lie in the interior of $S \cap \text{int}(\mathcal{U})$ and do not intersect Π_S^∇. Each vertex in Π_S^∇ is one of three types: a vertex of $\partial\mathcal{K}_S$, a vertex of Π_S lying in the interior of $\mathcal{K}_S$ (all edges incident to it are red), or an intersection point of an edge of $\partial\mathcal{K}_S$ and an edge of Π_S which is not a vertex of $\partial\mathcal{K}_S$; such a vertex is incident to both red and blue edges. We color the vertex as blue, red, or purple, respectively. We note that the vertices of

[4]To distinguish from the face of the union $\mathcal{U}$, we call the boundary face of an input cube a boundary square.

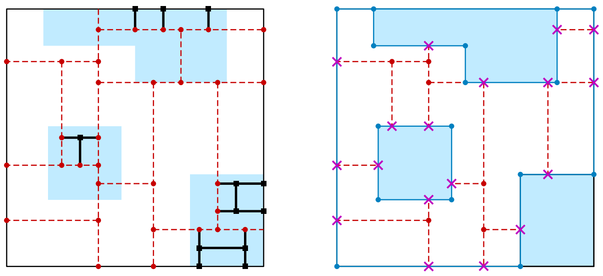

Figure 13. A 2D view of Π_S (left) and Π_S^{∇} (right) on a square S. On the left, the red edges are dashed, and the black edges are thick. On the right, the red edges are dashed, the blue edges are solid, and the purple vertices are shown as crosses. $S \cap \mathrm{int}(\mathcal{U})$ is shaded in blue, whose interior does not belong Π_S^{∇} (only its blue boundary edges belong to Π_S^{∇}).

Π_S lying in the interior of $\mathcal{K}_S$, which are also vertices of Π_S^{∇}, were colored red. (Π_S may have red vertices lying outside $\mathcal{K}_S$, namely the endpoints of black edges incident on red edges; see the red vertex incident on edge γ_3 in Figure 14.) The number of blue vertices is κ_S, by definition, so we need to bound the number of red and purple vertices.

As before, we define a *mast* to be an axis-aligned segment contained in S. The following lemma is analogous to Lemma 3.3.

LEMMA 4.1. *Let γ be a mast in S. Let $\mathcal{E}_{\Delta,h}$ be the set of red edges of Π_S created by a single call to* GLOBALCUT(Δ, h). *If γ is parallel to h it does not cross any edge of $\mathcal{E}_{\Delta,h}$. If γ is orthogonal to h then it crosses at most one edge of $\mathcal{E}_{\Delta,h}$.*

Proof. During GLOBALCUT(Δ, h), for each box $B \in \mathcal{A}_\Delta$, B is possibly split by the plane h, and if so, the resulting sub-boxes of B are split by free cuts until none admit a free cut. Recall that a free cut cannot cross $\mathcal{K}_S$. Therefore, while a free cut may cross S and generate edges of Π_S, these edges do not lie on a edge that intersects $\mathcal{K}_S$, and hence are black. Thus, any red edges created during the call lie on $h \cap S$. The proof now follows from the same argument as in Lemma 3.3. ∎

Using Lemma 4.1 and following the same arguments as in the proofs of Lemmas 3.4–3.6, we obtain the following:

COROLLARY 4.2. *A mast in S crosses $O(\log^2 n)$ red edges of Π_S.*

We are now ready to prove the main lemma, which is analogous to Lemma 3.7.

LEMMA 4.3. *Π_S^{∇} has $O(\log^4 n + \kappa_S \log^2 n)$ vertices.*

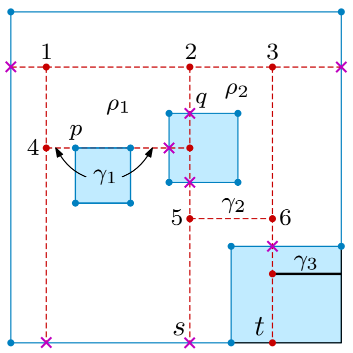

Figure 14. A 2D view of $\widetilde{\Pi}_S$ that illustrates the various cases for new edges with segments $\gamma_1, \gamma_2, \gamma_3$. We assume that all other segments were created before them. $S \cap \mathrm{int}(\mathcal{U})$ is shown in blue. γ_1 intersects blue edges of $\mathcal{K}_S$, γ_2 lies in $\mathcal{K}_S$, and γ_3 lies in $S \cap \mathrm{int}(\mathcal{U})$. Immediately before the edges are created on Π_S, all faces of Π_S are exposed; afterwards, the resulting faces ρ_1 and ρ_2 are the only shielded faces of Π_S. Vertices 2, 4, 5, and 6 are charged to q, p, s, and t, respectively.

Proof. It suffices to estimate the number of red and purple vertices. Each edge of $\partial\mathcal{K}_S$ is a mast, so by Corollary 4.2, each edge of $\partial\mathcal{K}_S$ contains $O(\log^2 n)$ purple vertices. (As in the proof of Lemma 3.7, strictly speaking, we choose a mast parallel and very close to the edge so as to use Corollary 4.2). Hence, the total number of purple vertices is $O(\kappa_S \log^2 n)$.

Next, we bound the number of red vertices of Π_S^{∇}. We note that each such vertex is also a vertex of Π_S.

We charge each red vertex of Π_S^{∇} to a purple vertex of Π_S^{∇} or to a red vertex of Π_S lying on ∂S. To describe the charging scheme, it will be more convenient to work with the dynamic subdivision $\widetilde{\Pi}_S$ of S that was refined as the algorithm progressed and pay attention to the creation of the red vertices of Π_S^{∇}. Recall that $\widetilde{\Pi}_S = S$ initially and $\widetilde{\Pi}_S = \Pi_S$ at the end. We call a face of $\widetilde{\Pi}_S$ *exposed* if one of its edges lies on an edge of S and *shielded* otherwise. (For example, in Figure 14, ρ_1 and ρ_2 are shielded faces of Π_S, and the rest are exposed.) If a shielded face ρ intersects $\mathcal{K}_S$, then by the new definition of free cut, B admits a free cut (along S). The algorithm splits boxes by free cuts as soon as they become available. Thus, ρ is not further refined, ρ becomes a face of Π_S, and all fragments on ρ become eternal. Therefore no red vertex of Π_S^{∇} lies inside ρ. If ρ does not intersect $\mathcal{K}_S$, then ρ does not contain any vertex of Π_S^{∇}, so it suffices to focus on how a vertex of Π_S^{∇} is created inside an exposed face of $\widetilde{\Pi}_S$.

Suppose an exposed face ρ of $\widetilde{\Pi}_S$ was split into two faces by the creation of an axis-aligned segment γ with endpoints a and b, which become vertices of $\widetilde{\Pi}_S$ (and thus of Π_S). If γ lies in the interior of $S \cap \mathrm{int}(\mathcal{U})$, then a and b are not vertices of Π_S^{∇}. If a (resp. b) lies in the interior of $\mathcal{K}_S$, it is a red vertex of Π_S^{∇}. We charge a (resp. b) as follows: We walk from a (resp. b) on γ until we reach a point η on $\partial\mathcal{K}_S$ and charge a (resp. b) to η, which is a blue vertex of Π_S^{∇} (if it is a vertex of $\mathcal{K}_S$, e.g. vertex 4 is charged to p in Figure 14), or it is a purple vertex of Π_S^{∇} (if η lies in the relative interior of an edge of $\mathcal{K}_S$, e.g. vertex 2 is charged to q in Figure 14). It is easily seen that η is charged at most twice[5] in this way, so the number of such red vertices of Π_S^{∇} is $O(\kappa_S \log^2 n)$.

Next, assume that γ lies in $\mathcal{K}_S$, i.e., both a and b are red vertices of Π_S^{∇}. Let ρ be the exposed face of $\widetilde{\Pi}_S$, which we view as a rectangle, that is being split by γ. If at least one of a and b lies on ∂S, say a for concreteness, we charge both a and b to a. At most two vertices are charged to a in this way. Next, we assume that both a and b lie in the interior of S. As in the proof of Lemma 3.7, let e_a (resp. e_b) be the edge of ρ that contains the endpoint a (resp. b); e.g., segments $2s$ and $3t$ for γ_2 in Figure 14. Neither e_a nor e_b lies on ∂S. Since ρ is an exposed face of S, at least one of the other two edges of the rectangle ρ lies on ∂S, and hence at least one endpoint ξ_a (resp. ξ_b) of e_a (resp. e_b) lies on ∂S. Note that the edge e_a (resp. e_b) of $\widetilde{\Pi}_S$ may be later subdivided by subsequent cuts, but since e_a (resp. e_b) intersects $\mathcal{K}_S$, all edges of the final subdivision Π_S lying on it will be colored red, so ξ_a (resp. ξ_b) is a red vertex of Π_S. We charge a (resp. b) to ξ_a (resp. ξ_b). Following the same argument as in the proof of Lemma 3.7 and using Corollary 4.2, any red vertex of Π_S on ∂S is charged $O(\log^2 n)$ times. Finally, using Corollary 4.2, only $O(\log^2 n)$ red edges of Π_S have any endpoint incident on an edge ω of S, which implies that ω contains $O(\log^2 n)$ red vertices of Π_S. Hence, the total charge to the red vertices on an edge of S is $O(\log^4 n)$. This completes the proof of the lemma. $\square$

Putting everything together, the first part of Theorem 1.2 is proved.

Runtime analysis. Since the modification lies strictly in GlobalCut, it suffices to describe how to modify GlobalCut to identify and split by the new free cuts. We then bound the resulting runtime by adapting the previous analysis at the end of Section 3.4.

In the GlobalCut procedure, we replace only step (3.e) of the original procedure; all other steps are performed as stated there. We also maintain the same auxiliary information as before, including the lists $\Phi_B, \mathsf{X}_B, \mathsf{Y}_B$, and Z_B for each box $B \in \mathcal{A}_\Delta$. Recall that Φ_B denotes the list of fragments that lie in box B, each

[5]A purple vertex of Π_S^{∇} lying on an edge of S is a red vertex of Π_S and this red vertex may be charged $O(\log^2 n)$ times by a later stage.

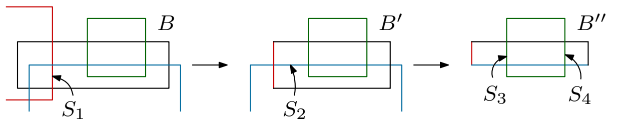

Figure 15. A 2D view of a box B intersected by three cubes (squares here) long at B (i.e., their vertices lie outside B). By the new definition of free cuts, only the boundary square S_1 (a segment here) induces a free cut in B. By splitting B by this cut, S_2 induces a newly available free cut in resulting sub-box B'. By splitting B' by this cut, boundary squares S_3 and S_4 induce newly available free cuts in the resulting sub-box B''. By the old definition of free cuts, there are no available free cuts in B.

represented by a list of its vertices in cyclic order, and that Γ_B is the multiset of the vertices of these fragments, represented as a list. In the original implementation of step (3.e), all free cuts in B are available in the beginning of this step, and no free cuts become newly available after being split by a free cut. In contrast, with the new definition of free cuts, splitting B by a free cut may create new free cuts in the resulting sub-boxes B^-, B^+ that did not exist in B (see Figure 15). We therefore carefully find free cuts, one at a time, in a recursive manner. We sketch the process, as follows.

Consider a newly created box D with set of fragments Φ_D; initially, D is either B^- or B^+. Then we iterate from each end of the sorted lists $\mathsf{X}_D, \mathsf{Y}_D$, and Z_D in a lock-step manner; each full iteration consists of six steps (two per list). We do the following at each step: For concreteness, assume we are at a vertex v while scanning the list Z_D from left to right. If the fragment φ_v containing v lies in a xy-plane (i.e., span(φ_v) is orthogonal to the z-axis), we test whether the boundary square S_v supporting φ_v induces a free cut in D. If the answer is yes, we pause the scan at v. We split D into D^- and D^+ by the free cut $g := D \cap \mathrm{span}(S_v)$. Next, we split the lists $\Phi_D, \mathsf{X}_D, \mathsf{Y}_D$, and Z_D to create the lists for D^- and D^+, as follows. Let $\Phi_D^g \subseteq \Phi_D$ denote the set of (xy-)fragments that lie on g, and let Γ_D^g be the list of vertices of these fragments. Then $\Phi_D = \Phi_{D^-} \cup \Phi_D^g \cup \Phi_{D^+}$ and $\Gamma_D = \Gamma_{D^-} \cup \Gamma_D^g \cup \Gamma_{D^+}$. By breaking ties in the lists X_D, Y_D, and Z_D carefully, we can ensure that all vertices in Γ_{D^-} (resp. Γ_{D^+}) appear before (resp. after) the vertices in Γ_D^g in Z_D. We resume the scan of Z_D from the vertex v to the right until a vertex v^+ of a fragment in Φ_{D^+} (or the end of Z_D) is reached. We remove the vertices of Z_{D^-} and Z_D^g from Z_D. The remaining list is Z_D^+. We reconstruct the list Z_{D^-}. Next, we delete the corresponding fragments from Φ_D and fragment vertices from X_D and Y_D, and we reconstruct the lists $\Phi_{D^-}, \mathsf{X}_{D^-}$, and Y_{D^-}; the last two lists require sorting the vertices of Γ_{D^-} in the x- and y-order.

We recursively call the procedure to find free cuts in D^- and D^+. On the other hand, if no free cut was found in D while scanning $\mathsf{X}_D, \mathsf{Y}_D$, and Z_D, we are done with box D.

Next, we analyze the total time spent in splitting D by free cuts with this recursive procedure. Recall that splitting by free cuts do not create any new fragment vertices. Let $\nu_D := |\Gamma_D|$ immediately before step (3.e), where $D = B^-$ or $D = B^+$. If no free cut was found in D, then we spend $O(\nu_D)$ time at D. Assuming that a free cut was found while scanning Z_D from left to right, then the procedure spends $O(\nu_{D^-} \log \nu_{D^-} + \nu_D^g)$ time in splitting D and constructing the lists for D^- and D^+, where $\nu_{D^-} := |\Gamma_{D^-}|$, $\nu_D^g := |\Gamma_{D^-}^g|$, and $\nu_{D^+} := |\Gamma_{D^+}|$. Because we scan the lists in lock-step manner, we can conclude that $\nu_{D^-} \le \nu_{D^+}$. Therefore, the time spent in splitting D into D^- and D^+ is always $O(\widehat{\nu} \log \widehat{\nu} + \nu_D^g)$, where $\widehat{\nu} := \min\{\nu_{D^-}, \nu_{D^+}\}$. Let $\tau(\nu_D)$ be the total time spent in splitting by free cuts in D, including the time taken by the recursive calls. Then we obtain the following recurrence:

$$\tau(\nu_D) \le \tau(\nu_{D^-}) + \tau(\nu_{D^+}) + O(\widehat{\nu} \log \widehat{\nu} + \nu_D^g),$$

where $\nu_{D^-} + \nu_D^g + \nu_{D^+} \le \nu_D$, and $\tau(\nu_D) = O(\nu_D)$ if no free cut was found. By induction on ν_D, we can prove that the solution to above recurrence is $\tau(\nu_D) = O(\nu_D \log^2 \nu_D)$. Summing this quantity over all sub-boxes B^-, B^+ for each $B \in \mathcal{A}_\Delta$, the total running time of GlobalCut(Δ, h) is $O(\nu_\Delta \log^2 \nu_\Delta)$, where $\nu_\Delta := \sum_{B \in \mathcal{A}_\Delta} \nu_B$ is the number of vertices of the fragments that lie in a box of $\mathcal{A}_\Delta$ when the procedure was called, counted with multiplicity.

For a node u of $\mathcal{T}$, let $\nu_u := |\Gamma \cap \boxdot_u|$ and $n_u := |\mathcal{S}_u| + |\mathcal{C}_u|$. Following the analysis in Section 3.4, each of the $O(1)$ calls to Staircase at u now take $O((|X| + \nu_u \log^2 \nu_u) \log n) = O((|X| + \nu_u \log^2 n) \log n)$, using the fact that $\log \nu = O(\log \kappa) = O(\log n)$. It follows that processing any node u of $\mathcal{T}$ during the overall algorithm takes $O((n_u + \nu_u \log^2 n) \log n)$ time. Therefore, by summing over all nodes of $\mathcal{T}$ and using the fact that $\sum_{u \in \mathcal{T}} n_u = O(n \log n)$ and $\sum_{u \in \mathcal{T}} \nu_u = O(\nu \log n) = O(\sigma \log^5 n + \kappa \log^3 n)$, the total running time is $O(n \log^2 n + \sigma \log^8 n + \kappa \log^6 n)$. This proves the second part of Theorem 1.2.

5 Algorithm for Congruent Cubes

In this section, we describe an improved decomposition scheme for a set of axis-aligned congruent cubes in $\mathbb{R}^3$.

5.1 Overall algorithm Let $\mathcal{C} := \{C_1, \ldots, C_n\}$ be a set of n axis-aligned congruent cubes, say, unit cubes, in $\mathbb{R}^3$ in general position. Unlike the setup in Section 3, where we have enclosed $\mathcal{U}$ in some sufficiently large box $\square$ and focused on constructing the decomposition of $\mathcal{K}$ within $\square$, here it is more convenient to treat the unbounded version of $\mathcal{K}$. Let $\mathbb{G}$ be the 3-dimensional integer grid, which partitions $\mathbb{R}^3$ into unit cubes. For $i, j, k \in \mathbb{Z}$, let $\xi_{i,j,k}$ denote the grid cell $[i, i+1] \times [j, j+1] \times [k, k+1]$. Let $\mathbb{G}^*$ be the 2-dimensional integer grid on the xy-plane, and let $\xi_{i,j}^*$ denote the unit square $[i, i+1] \times [j, j+1]$. For a pair (i, j), let $\Pi_{i,j} := \xi_{i,j}^* \times \mathbb{R}$ denote the unbounded vertical prism erected on the square $\xi_{i,j}^*$, and let $\mathbb{G}_{i,j} := \{\xi_{i,j,k} \mid k \in \mathbb{Z}\}$ denote the column of grid cells stacked on $\xi_{i,j}^*$; $\mathbb{G}_{ij}$ partitions $\Pi_{i,j}$ into a "stack" unit cubes. Let $\mathcal{G} \subset \mathbb{G}$ denote the set of non-empty grid cells, i.e., the ones that intersect a cube of $\mathcal{C}$, and let X be the set of pairs (i, j) such that $\Pi_{i,j}$ intersects a cube of $\mathcal{C}$; $\mathcal{U}(\mathcal{C}) \subset \bigcup \mathcal{G} \subseteq \bigcup_{(i,j) \in \mathsf{X}} \Pi_{i,j}$ and $|\mathcal{G}|, |\mathsf{X}| = O(n)$.

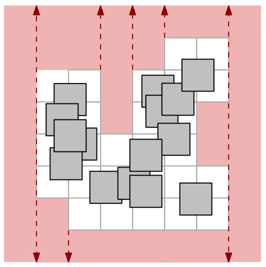

Figure 16. A view from above of $\mathbb{G}^*$ with the set of cubes $\mathcal{C}$ (grey), and the partition of $\mathrm{cl}(\mathbb{R}^2 \setminus \bigcup_{\chi \in \mathsf{X}} \xi_\chi^*)$ into axis-aligned rectangles (red).

We partition $\mathcal{K}$ into boxes in three stages. First, we decompose $\mathrm{cl}(\mathbb{R}^3 \setminus \bigcup_{(i,j) \in \mathsf{X}} \Pi_{i,j})$ into a family $\mathcal{B}_1$ of $O(n)$ boxes, as follows. We partition $\mathrm{cl}(\mathbb{R}^2 \setminus \bigcup_{(i,j) \in \mathsf{X}} \xi_{i,j}^*)$ into $O(n)$ axis-aligned rectangles, using, say, the standard 2-dimensional vertical decomposition. For each rectangle ρ in the decomposition, we add the unbounded prism $\rho \times \mathbb{R}$ to $\mathcal{B}_1$. See Figure 16.

Next, for each pair $\chi \in \mathsf{X}$, let $\mathcal{G}_{i,j} := \mathcal{G} \cap \mathbb{G}_{i,j}$ denote the set of non-empty grid cells in column (i, j). We partition the union of empty grid cells in column (i, j), i.e., $\mathrm{cl}(\Pi_{i,j} \setminus \mathcal{U}(\mathcal{G}_{i,j}))$, in a straightforward manner, into a family $\mathcal{B}_\chi$ of at most $|\mathcal{G}_\chi| + 1$ boxes. See Figure 17. Set $\mathcal{B}_2 := \bigcup_{(i,j) \in \mathsf{X}} \mathcal{B}_{i,j}$. Note that

$$\sum_{(i,j) \in \mathsf{X}} |\mathcal{B}_{i,j}| \le \sum_{(i,j) \in \mathsf{X}} |\mathcal{G}_{i,j}| + 1 \le |\mathcal{G}| + |\mathsf{X}| = O(n).$$

$\mathcal{B}_1 \cup \mathcal{B}_2$ partitions $\mathrm{cl}(\mathbb{R}^3 \setminus \bigcup \mathcal{G})$ into $O(n)$ axis-aligned boxes.

Finally, we partition $\mathcal{K} \cap \xi$, for all non-empty grid cells $\xi \in \mathcal{G}$, into boxes. Fix a cell $\xi \in \mathcal{G}$. Let $\mathcal{K}_\xi := \mathcal{K} \cap \xi$, and let κ_ξ be the number of vertices of $\mathcal{K}$ that lie in the interior of ξ; we have $\sum_{\xi \in \mathcal{G}} \kappa_\xi = O(n)$. Below we describe the main part of our procedure, a recursive algorithm that partitions $\mathcal{K}_\xi$ into a collection $\mathcal{B}_\xi$ of

$O(\kappa_\xi \log \kappa_\xi)$ axis-aligned boxes, in $O(\kappa_\xi \log \kappa_\xi)$ time (*cf.* Corollary 5.6). Repeating this procedure for all grid cells $\xi \in \mathcal{G}$, we decompose $(\bigcup \mathcal{G}) \cap \mathcal{K}$ into a total of $\sum_{\xi \in \mathcal{G}} O(\kappa_\xi \log \kappa_\xi) = O(n \log n)$ boxes.

Putting everything together, $\mathcal{B}_1 \cup \mathcal{B}_2 \cup \bigcup_{\xi \in \mathcal{G}} \mathcal{B}_\xi$ partitions $\mathcal{K}$ into $O(n \log n)$ axis-aligned boxes. Moreover, as we will show, our algorithm runs in overall $O(n \log n)$ time. This completes the proof of Theorem 1.5.

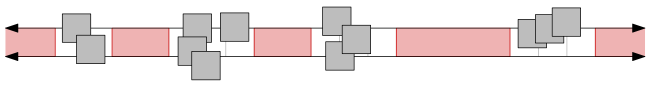

Figure 17. A 2D view of a prism Π_χ crossed by some cubes of $\mathcal{C}$ (grey). $\mathrm{cl}(\Pi_\chi \setminus \bigcup \mathcal{G}_\chi)$ is decomposed into axis-aligned boxes (red).

Decomposition within a single unit grid cell. Let $\square := [x_L, x_R] \times [y_L, y_R] \times [z_L, z_R]$ be an axis-aligned box in $\mathbb{R}^3$, each of whose side-lengths is at most 1, that intersects $\mathcal{K}$. We describe a recursive algorithm for partitioning $\mathcal{K}_\square := \mathcal{K} \cap \square$ into axis-aligned boxes. Let $\mathsf{E}_\square$ be the set of edges of $\mathcal{K}_\square$ that lie in $\mathrm{int}(\square)$ (these are the edges of $\mathcal{K}$ that intersect the interior of $\square$, clipped within $\square$), and let $\mathsf{V}_\square$ be the set of vertices of $\mathcal{K}$ that lie in $\mathrm{int}(\square)$. If $\mathsf{E}_\square = \emptyset$, then $\mathcal{K}_\square$ is a single box, bounded by portions of $\partial\square$ and of up to two parallel faces of $\partial\mathcal{U}$ (the fact that there is only one such box follows from the fact that all the side lengths of $\square$ are at most 1). We output $\{\mathcal{K}_\square\}$ and stop. So assume that $\mathsf{E}_\square \neq \emptyset$.

We call an edge of $\mathsf{E}_\square$ *short* if one of its endpoints lies in the interior of $\square$, and *long* otherwise. Let $n_\square := |\mathsf{V}_\square|$ and $m_\square$ be the number of long edges in $\mathsf{E}_\square$. We further classify the edges of $\mathsf{E}_\square$ into three families: an edge is an *x-edge* (resp., *y-edge, z-edge*) if it is parallel to the x-axis (resp., y-axis, z-axis).

We assume that $\square$ satisfies the following invariant, and we will enforce the maintenance of this invariant throughout the recursive execution of the algorithm. In particular, it will hold initially, when $\square$ is a unit cell of $\mathcal{G}$, because, by the general position assumptions, such a cell does not contain any long edge.

2-Family Invariant: *$\mathsf{E}_\square$ contains at most two families of long edges, i.e., there is at least one axis among the x-, y-, and z-axes such that $\mathsf{E}_\square$ has no long edge parallel to that axis.*

In view of the above invariant, let us assume, without loss of generality, that $\mathsf{E}_\square$ has no long z-edges. The next two lemmas lie at the heart of our decomposition procedure.

LEMMA 5.1. *Let e be a long x-edge (resp., y-edge) of $\mathsf{E}_\square$, and let γ_1, γ_2 be two long y-edges (resp., x-edges) of $\mathsf{E}_\square$. Then either both γ_1, γ_2 lie above e (in the z-direction) or both of them lie below e.*

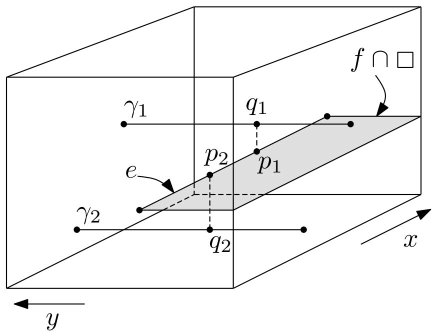

Figure 18. An illustration of the proof of Lemma 5.1.

Proof. Suppose to the contrary that, say, γ_1 passes above e and γ_2 passes below e. Denote by p_1 and q_1 the respective points on e and γ_1 that lie vertically above each other (with p_1 lying below q_1). Similarly, denote by p_2 and q_2 the respective points on e and γ_2 that lie vertically above each other (with p_2 lying above q_2). See Figure 18.

The edge e is either a *concave* edge,[6] namely a portion of an original edge of some cube $C \in \mathcal{C}$, or a *convex* edge, which is a portion of an edge formed by the intersection of two non-parallel faces of two distinct cubes $C, C' \in \mathcal{C}$. In the former case, e is adjacent to an xy-parallel face and to an xz-parallel face of C. In the latter case, we take C to be the cube for which e lies on one of its (top or bottom) xy-parallel faces. In either case, let f be the xy-parallel face of C that contains e, and assume, without loss of generality, that f is the bottom face of C.

Denote the xy-projection of an object a as a^*. In the case where e is a concave edge, move q_1 slightly along γ_1 so as to make q_1^* be contained in f^*, and move p_1 along f to make it co-vertical with q_1. In the case of a convex edge, p_1 and q_1 remain unchanged. Now the fact that $\square$ is a box of side-lengths at most 1 implies that the vertical segment p_1q_1 is fully contained in the interior of C. In particular, q_1 lies inside C, contradicting the fact that it lies on an edge of the union. The case where f is the top face of C is handled symmetrically, using γ_2 instead of γ_1. ∎

The proof of the following corollary is now straightforward and omitted here.

COROLLARY 5.2. *Either all long x-edges of $\mathsf{E}_\square$ lie above all the long y-edges of $\mathsf{E}_\square$, or all of them lie below all the long y-edges.*

[6]The terminology comes from treating the edges as edges of $\mathcal{K}$; it would be reversed if we were to regard them as edges of $\mathcal{U}$.

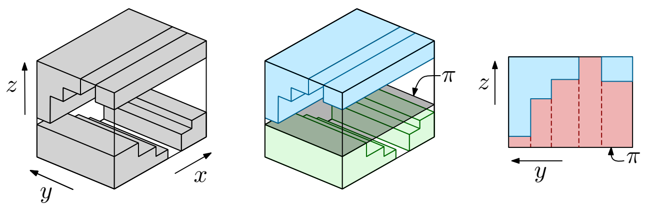

Figure 19. An illustration of Lemma 5.3 for a box $\square$ with two families of long edges. (left) The undecomposed scenario. (middle) Separating the two families with a plane π that contains the highest long y-edge in $\square$. (right) Decomposing the portion of $\square \cap \mathcal{K}$ above π into $O(m_x)$ axis-aligned boxes (red), where m_x is the number of x-edges in $\square$. The figure shows a yz-cross section of the decomposition.

Similar claims hold for the other possible combinations of long edges.

LEMMA 5.3. *If $\mathsf{V}_\square = \emptyset$, i.e., $\mathsf{E}_\square$ does not have any short edges, then $\mathcal{K}_\square$ can be partitioned into $O(m_\square)$ axis-aligned boxes.*

Proof. Suppose, without loss of generality, that all long x- edges of $\mathsf{E}_\square$ lie above all the long y-edges. Let z_0 be the maximum z-coordinate of a long y-edge in $\square$. We first partition $\square$ into two boxes $\square_x, \square_y$ by drawing the plane $\pi : z = z_0$, with $\square_x$ (resp., $\square_y$) lying above (resp., below) π. If there are no x-edges (resp. y-edges) then we set $\square_y := \square$ (resp., $\square_x := \square$) and $\square_x := \emptyset$ (resp., $\square_y := \emptyset$). Since there are no z-edges inside B, π does not cross any edge of $\mathsf{E}_\square$, and all the x-edges (resp., y-edges) lie inside $\square_x$ (resp., $\square_y$). Let m_x (resp. m_y) denote the number of x-edges (resp. y-edges) in $\square$. We describe how to partition $\mathcal{K}_x := \mathcal{K}_{\square_x}$ into $O(m_x)$ boxes. See Figure 19 (right).

Let ϕ be one of the two faces of $\square_x$ parallel to the yz-plane (it is a portion of a face of $\square$), and let $\mathcal{K}_\phi := \mathcal{K} \cap \phi$; it is a rectilinear polygonal region. Since all the edges of $\mathsf{E}_\square$ that lie in $\square_x$ are x-edges, it easily follows that $\mathcal{K}_x = \mathcal{K}_\phi \times [x_L, x_R]$. We partition $\mathcal{K}_\phi$ into $O(m_x)$ axis-aligned rectangles by the standard planar vertical-decomposition method, as at the beginning of this section (see Figure 16). We extend each rectangle R in the decomposition of $\mathcal{K}_\phi$ to a prism (within $\square$) in the x-direction, i.e., we generate the box $R^\uparrow := R \times [x_L, x_R]$, resulting in the desired partition of $\mathcal{K}_x$ into $O(m_x)$ boxes.

In a fully symmetric manner, $\mathcal{K} \cap \square_y$ can be partitioned into $O(m_y)$ axis-aligned boxes. Hence, $\mathcal{K}_\square$ can be partitioned into $O(m_\square)$ boxes, as claimed. ∎

The next lemma suggests a recursive procedure for decomposing $\mathcal{K}_\square$ into boxes when $\mathsf{V}_\square \neq \emptyset$.

LEMMA 5.4. *The box $\square$ can be partitioned into at most three cubes $\square_1, \square_2, \square_3$ such that each $\square_i$ satisfies the following properties. For $i = 1, 2, 3$, let n_i, m_i denote $n_{\square_i}, m_{\square_i}$, respectively.*

(i) $n_1 + n_2 + n_3 \le n_\square$,

(ii) $n_i \le \lceil n_\square/2 \rceil$ for every $i = 1, 2, 3$,

(iii) $m_1 + m_2 + m_3 \le m_\square + 2n_\square$, and

(iv) each $\square_i$ satisfies the 2-family invariant.

Proof. If $\square$ contains both long x-edges and long y-edges, then, similar to the analysis in the proof of Lemma 5.3, we partition $\square$ into two boxes $\square_x$ and $\square_y$, such that the long x-edges (y-edges) of $\mathsf{E}_\square$ lie in $\square_x$ (resp., in $\square_y$), by drawing the horizontal plane $\pi_1 : z = z_\square$, where $z_\square$ is the maximum z-coordinate of a long y-edge in $\square$ (assuming, as above and without loss of generality, that the long y-edges lie below the long x-edge); if $\square$ contains only long x-edges (resp., long y-edges), we set $\square_x$ (resp., $\square_y$) to $\square$, and $\square_y$ (resp., $\square_x$) is then $\emptyset$.

If the interior of each of $\square_x, \square_y$ contains at most $\lceil n_\square/2 \rceil$ vertices of $\mathcal{K}_\square$, then we have obtained a partition of $\square$ into two boxes $\square_1 := \square_x$ and $\square_2 := \square_y$, and there is no need for the third box $\square_3$. If the interior of one of them, say, of $\square_x$, contains more than $\lceil n_\square/2 \rceil$ vertices, we partition $\square_x$ further into two boxes by drawing some suitable horizontal plane that partitions $\square_x$ into two sub-boxes, each containing at most $\lceil n_\square/2 \rceil$ vertices. In either case, we obtain a partition of $\square$ into at most three boxes $\square_1, \square_2, \square_3$.

We now prove that $\square_1, \square_2, \square_3$ satisfy the properties (i)–(iv). Clearly, (i) and (ii) follow from the construction. Concerning (iv), each $\square_i$ contains either long x-edges or long y-edges of $\mathsf{L}_\square$, but not both. Since the partition is only by horizontal planes, no new long x- or y-edge can be produced. The only new long edges, in any $\square_i$ are portions of original short z-edges in $\mathsf{E}_\square$. This implies (iv).

Finally, each long (x- or y-)edge of $\mathsf{E}_\square$ lies in the interior of at most one box $\square_i$. Furthermore, each short z-edge of $\mathsf{E}_\square$ is split into at most two long z-edges (and possibly a third short z-edge), so the total number of long edges in the three boxes $\square_i$, $i = 1, 2, 3$, is at most $m_\square + 2n_\square$, thereby proving (iii). ∎

Let $\psi(m_\square, n_\square)$ be the maximum number of boxes into which $\mathcal{K}_\square$ is partitioned, where the maximum is taken over all the sets of unit cubes such that $|\mathsf{V}_\square| = n_\square$ and $|\mathsf{E}_\square| = m_\square$. Lemmas 5.3 and 5.4 imply the following recurrence:

$$\psi(m_\square, n_\square) \le \begin{cases} 1 & \text{if } m_\square = n_\square = 0, \\ c_1 m_\square & \text{if } m_\square > 0, n_\square = 0, \\ \displaystyle\sum_{i=1}^{3} \psi(m_i, n_i) & \text{if } m_\square \ge 0, n_\square > 0, \end{cases}$$

where $n_i \leq \lceil n_\square/2 \rceil$, $n_1 + n_2 + n_3 \leq n_\square$, and $m_1 + m_2 + m_3 \leq m_\square + 2n_\square$. A solution to the above recurrence is $\psi(m_\square, n_\square) = O(m_\square + n_\square \log n_\square)$. We also note that the total time spent in constructing the decomposition of $\mathcal{K}_\square$ into boxes can be shown to be $O((m_\square + n_\square) \log n_\square)$. In conclusion, we have obtained the following result.

LEMMA 5.5. *If $\mathcal{K}_\square$ contains at most two families of long edges, then $\mathcal{K}_\square$ can be partitioned, in $O((m_\square + n_\square) \log n_\square)$ time, into $O(m_\square + n_\square \log n_\square)$ boxes.*

Returning to the overall algorithm, let ξ be a cell in $\mathcal{G}$. Since ξ is a unit cube with integer vertex coordinates, our assumption of vertices of $\mathcal{C}$ not having integer coordinates implies that no face of $\mathcal{K}$ lies on $\partial\xi$, which in turn implies that $\mathcal{K}_\xi$ does not have any long edge, and thus trivially satisfies the 2-family invariant. Hence, by Lemma 5.5, $\mathcal{K}_\xi$ can be partitioned into a family $\mathcal{B}_\xi$ of $O(\kappa_\xi \log \kappa_\xi)$ axis-aligned boxes in $O(\kappa_\xi \log \kappa_\xi)$ time, where κ_τ is the number of vertices of $\mathcal{U}$ that lie in the interior of ξ.

COROLLARY 5.6. *For any cell $\xi \in \mathcal{G}$, $\mathcal{K}_\xi$ can be partitioned into $O(\kappa_\xi \log \kappa_\xi)$ boxes in $O(\kappa_\xi \log \kappa_\xi)$ time.*

6 Conclusion

We have described algorithms to compute a decomposition of the complement of the union of axis-aligned 3D cubes (or fat boxes) into a number of boxes near-linear in the complexity κ of (the boundary of) the union, and their runtimes are near-linear in the input and output size. In particular, if the input cubes have different sizes then a decomposition of size $O(\sigma \log^4 n + \kappa \log^2 n)$, where $\sigma \leq \min\{n, \kappa\}$ is the number of input cubes that appear on $\partial\mathcal{U}$, can be computed in $O(n \log^2 n + \sigma \log^8 n + \kappa \log^6 n)$ time. If all cubes have the same size, then a decompositon of size $O(\kappa \log n) = O(n \log n)$ can be computed in $O(n \log n)$ time.

We conclude by mentioning two open problems: Can the complement of the union of a set of axis-aligned cubes in $\mathbb{R}^3$ be decomposed into $O(\kappa)$ boxes? Can our results be extended to higher dimensions?

Acknowledgements. We thank Boris Aronov and Mark de Berg for helpful discussions, and the reviewers for their useful comments.

References

[1] P. K. Agarwal, H. Kaplan, and M. Sharir, Union of hypercubes and 3D Minkowski sums with random sizes, In *Proc. 45th Intl. Colloq. Auto. Lang. Program.*, (2018), 10:1–10:15.

[2] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu, An optimal algorithm for approximate nearest neighbor searching in fixed dimensions, *J. ACM* 45 (1998), 891–923.

[3] 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.

[4] P. K. Agarwal, S. Har-Peled, H. Kaplan and M. Sharir, Union of random Minkowski sums and network vulnerability analysis, *Discrete Comput. Geom.* 52 (2014), 551–582.

[5] M. Bern and D. Eppstein, Mesh generation and optimal triangulation, *Computing in Euclidean Geometry*, (D. Du and F. Hwang, eds.), 1992, World Scientific.

[6] J.D. Boissonnat, M. Sharir, B. Tagansky and M. Yvinec, Voronoi diagrams in higher dimensions under certain polyhedral distance functions, *Discrete Comput. Geom.* 19 (1998), 485–519.

[7] S.-W. Cheng, T. K. Dey, and J. Shewchuk, *Delaunay Mesh Generation*, Chapman & Hall/CRC, 2012.

[8] B. Chazelle, H. Edelsbrunner, L. J. Guibas, and M. Sharir. A singly-exponential stratification scheme for real semi-algebraic varieties and its applications. *Theoret. Comput. Sci.* 84 (1991), 77–105.

[9] B. Chazelle, Convex partitions of polyhedra: A lower bound and worst-case optimal algorithm. *SIAM J. Comput.* 13 (1984), 488–508.

[10] B. Chazelle and L. Palios, Triangulating a nonconvex polytope., *Discrete Comput. Geom.* 5 (1990), 505–526.

[11] G. Collins, Quantifier elimination for real closed fields by cylindrical algebraic decomposition, *Second GI Conference on Automata Theory and Formal Languages* 33, (1975), 134—183.

[12] H. Edelsbrunner and E. P. Mücke, Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms, *ACM Trans. Graph.* 9 (1990), 9:66–104.

[13] J. D. Foley, A. van Dam, S. Feiner, and J. F. Hughes, *Computer Graphics — Principles and Practice*, 2nd ed., Addison-Wesley, 1992.

[14] P. J. Frey and P.-L. George, *Mesh Generation: Application to Finite Elements*, Wiley, 2007.

[15] H. Fuchs, Z. Kedem, and B. Naylor, On visible surface generation by a priori tree structures, *Proc. 7th Annu. Conf. Comp. Graph. Interact. Tech.*, 1980, 124–133.

[16] S. Har-Peled, *Geometric Approximation Algorithms*, American Mathematical Soc., 2011.

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

[18] D. H. Laidla, W. B. Trumbore, and J. F. Hughes, Constructive solid geometry for polyhedral objects, *Proc. 13th Annu. Conf. Comp. Graph. Interact. Tech.*, 1986, 161–170.

[19] S. M. LaValle, *Planning Algorithms*, Cambridge University Press, 2006.

[20] 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.

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

[22] M. Sharir and P. K. Agarwal, *Davenport-Schinzel Sequences and Their Geometric Applications*, Cambridge University Press, 1995.

[23] J. T. Schwartz and M. Sharir, On the "piano movers" problem II: General techniques for computing topological properties of real algebraic manifolds, *Adv. Appl. Math.*, 4 (1983), 298–351.

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

A Appendix: Balanced-Box Decomposition (BBD) Trees

Before we prove Lemma 2.1, we review additional properties of BBD trees that were not required to describe our algorithms but are needed for the proof here. For the full details of BBD trees and their construction, we refer the reader to [2].

For a box B, let $x(B)$, $y(B)$, $z(B)$ denote its projection on the x-, y-, and z-axis, respectively, and we refer to them as its *x-span*, *y-span*, and *z-span*, respectively.

Consider two nested boxes $\square^O$ and $\square^I$ such that $\square^I \subseteq \square^O$. For each axis $q \in \{x, y, z\}$, let $[q_I^-, q_I^+]$ (resp. $[q_O^-, q_O^+]$) be the q-span of $\square^I$ (resp. $\square^O$). Using the terminology from [2], $\square^I$ is said to be *q-sticky* for $\square^O$ if each of $q_I^- - q_O^-$ and $q_O^+ - q_I^+$ is either 0 or at least $q_I^+ - q_I^-$, and $\square^I$ is said to be *sticky* for $\square^O$ if $\square^I$ is q-sticky for all axes $q \in \{x, y, z\}$.

Let $P \subseteq \mathbb{R}^3$ be a set of n points, and let $\mathcal{T}$ be a BBD tree constructed on P. The following additional properties hold for each node u of $\mathcal{T}$: (i) $\square_u^I$ is sticky for $\square_u^O$ (if $\square_u^I$ exists), and (ii) $\square_u^O$ and $\square_u^I$ have aspect ratio at most three, i.e., the length of the longest span (edge length) of $\square_u^O$ (resp. $\square_u^I$) is at most three times the length of the shortest span of $\square_u^O$ (resp. $\square_u^I$).

Using these properties, we establish Lemma 2.1:

Lemma 2.1. *Let u be a node of a BBD tree $\mathcal{T}$ for a point set $P \subseteq \mathbb{R}^3$. There is a set H_u of at most 24 planes that induces a subdivision of $\boxdot_u$ into $O(1)$ axis-aligned boxes such that any axis-aligned cube C that intersects $\boxdot_u$ but none its vertices lie in the interior of $\boxdot_u$ contains an edge of each box that it intersects.*

Proof. Let C be an axis-aligned cube that intersects the interior of $\boxdot_u$ and has all vertices outside $\boxdot_u = \mathrm{cl}(\square_u^O \setminus \square_u^I)$. The proof is trivial if $\square_u^O \subseteq C$, so assume otherwise. For concreteness, we also assume $\square_u^I \neq \varnothing$; the proof for the other case is similar.

For each axis $q \in \{x, y, z\}$, let $[q_I^-, q_I^+] := q(\square_u^I)$ be the q-span of $\square_u^I$, let $[q_O^-, q_O^+] := q(\square_u^O)$ be the q-span of $\square_u^O$, and let $[q_C^-, q_C^+] := q(C)$ be the q-span of C. Let $q_I^1 := (q_I^+ - q_I^-)/3$ and $q_I^2 := 2(q_I^+ - q_I^-)/3$ be the points that trisect $q(\square_u^I)$, and let $q_O^1 := (q_O^+ - q_O^-)/3$ and $q_O^2 := 2(q_O^+ - q_O^-)/3$ be the points that trisect $q(\square_u^O)$. Set $T_{I,q} := \{q_I^-, q_I^1, q_I^2, q_I^+\}$ and $T_{O,q} := \{q_O^-, q_O^1, q_O^2, q_O^+\}$.

Let H_u be the set of planes of the form $q = t$ for each $t \in T_{I,q} \cup T_{O,q}$ and $q \in \{x, y, z\}$. Clearly $|H_u| \leq 24$. Let Ξ_u be the set of boxes in the subdivision of $\boxdot_u$ induced by H_u, and let $B \in \Xi_u$ be a box whose interior intersects C. We prove that C contains an edge of B.

First observe that if $q(C) \not\supseteq q(B)$ for all $q \in \{x, y, z\}$ then a vertex of C lies inside B, which contradicts the assumption that no vertex of C lies in $\boxdot_v$. Hence, assume that $x(C) \supseteq x(B)$. To prove that an x-edge of B lies inside C, we will prove that for each $q \in \{y, z\}$, at least one of the endpoints of the q-span $q(B) = [q_B^-, q_B^+]$ lies in $q(C)$, as this will imply that both endpoints of an x-edge of B lie inside C. Note that $q_B^-, q_B^+ \in T_{I,q} \cup T_{O,q}$, by construction.

We claim that for each $q \in \{y, z\}$, $q(C)$ contains at least one element of $T_{I,q} \cup T_{O,q}$. Assuming that the claim is true, let $q_i \in (T_{I,q} \cup T_{O,q}) \cap q(C)$. If q_i is q_B^- or q_B^+, we are done so assume that $q_i \neq q_B^-, q_B^+$. On the other hand, by construction, $q_i \notin q(B)$, so we conclude that $q(C) \not\subseteq q(B)$. But $q(B) \cap q(C) \neq \varnothing$. Hence, at least one of the endpoints of $q(B)$ lies in $q(C)$, as desired. What now remains is to prove the above claim.

The proof of the claim consists of two parts. We first consider the case where no vertex of C lies in $\square_u^I$. Then all of the vertices lie outside $\square_u^O$. If no span $q(C)$ contains $q(\square_u^O)$, a vertex of C lies in $\mathrm{int}(\square_u^O)$, which is a contradiction. Without loss of generality, assume that $x(\square_u^O) \subseteq x(C)$. By property (ii) of $\mathcal{T}$, we have that $3|x(\square_u^O)| \geq |y(\square_u^O)|, |z(\square_u^O)|$. Since C is a cube, $|x(C)| = |y(C)| = |z(C)|$, so $|y(C)| \geq |y(\square_u^O)|/3$ and $|z(C)| \geq |z(\square_u^O)|/3$. It follows that at least one point $y_i \in T_{O,y}$ (resp. $z_i \in T_{O,z}$) lies in $y(C)$ (resp. $z(C)$), thereby proving the claim in this case.

Next, suppose at least one vertex of C lies in $\square_u^I$. For each axis $q \in \{x, y, z\}$, if $q(C) \subseteq q(\square_u^I)$ we say $q(C)$ is *enclosed*, and *crossing* if $q(C) \supseteq [q_O^-, q_I^-]$ or $q(C) \supseteq [q_I^+, q_O^+]$. If all spans of C are enclosed, C is contained in $\square_u^I$, a contradiction. Hence, there is a crossing span of C, say, $x(C)$. Without loss of generality, assume each crossing span $q(C)$ contains $[q_O^-, q_I^-]$. In particular, $|x(C)| \geq x_I^- - x_O^-$ and $x(C)$ contains $x_I^- \in T_{I,x}$ and $x_O^- \in T_{O,x}$. By property (i) of $\mathcal{T}$, we have $x_I^- - x_O^- \geq |x(\square_u^I)|$, and by property (ii) of $\mathcal{T}$, we have $3|x(\square_u^I)| \geq |y(\square_u^I)|, |z(\square_u^I)|$. Since C is a cube, $|x(C)| = |y(C)| = |z(C)|$, so $|y(C)| \geq |y(\square_u^I)|/3$ and $|z(C)| \geq |z(\square_u^I)|/3$. Hence, if $y(C)$ is enclosed, then it contains either y_I^1 or y_I^2; otherwise, $y(C)$ is crossing and contains y_O^- and y_I^-. In either case, at least one point $y_i \in T_{I,y} \cup T_{O,y}$ lies in $y(C)$. By a symmetric argument, some point $z_i \in T_{I,z} \cup T_{O,z}$ lies in $z(C)$. This completes the proof of the claim and of the lemma. $\square$