Discrete & Computational Geometry (2024) 72:407–450
https://doi.org/10.1007/s00454-024-00632-2

Decomposing the Complement of the Union of Cubes and Boxes in Three Dimensions

Pankaj K. Agarwal[1] · **Micha Sharir**[2] · **Alex Steiger**[1]

Received: 16 June 2022 / Revised: 24 December 2023 / Accepted: 16 January 2024 /
Published online: 2 March 2024

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 $\Theta(n^2)$. We present a partition of $\mathrm{cl}(\mathbb{R}^3 \setminus \mathcal{U})$ into $O(\kappa \log^4 n)$ axis-aligned boxes with pairwise-disjoint interiors that can be computed in $O(n \log^2 n + \kappa \log^6 n)$ time if the faces of $\partial\mathcal{U}$ are pre-computed. We also show that a partition 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 if the faces of $\partial\mathcal{U}$ are pre-computed. The complexity and runtime bounds improve to $O(n \log n)$ if all cubes in $\mathcal{C}$ are congruent and the faces of $\partial\mathcal{U}$ are pre-computed. Finally, we show that if $\mathcal{C}$ is a set of arbitrary axis-aligned boxes in $\mathbb{R}^3$, then a partition of $\mathrm{cl}(\mathbb{R}^3 \setminus \mathcal{U})$ into $O(n^{3/2} + \kappa)$ boxes can be computed in time $O((n^{3/2} + \kappa) \log n)$, where κ is, as above, the number of vertices in $\mathcal{U}(\mathcal{C})$, which now can vary between $O(1)$ and $\Theta(n^3)$.

Keywords Decomposition · Free space · Fat boxes · Union complexity

Mathematics Subject Classification 52C45 · 68U05

Editor in Charge: Csaba D. Tóth

Dedicated to the memory of Eli Goodman.

Pankaj K. Agarwal
pankaj@cs.duke.edu

Micha Sharir
michas@tauex.tau.ac.il

Alex Steiger
asteiger@cs.duke.edu

[1] Department of Computer Science, Duke University, Box 90129, Durham, NC 27708-0129, USA

[2] Blavatnik School of Computer Science, Tel Aviv University, 69978 Tel Aviv, Israel

1 Introduction

Decomposing the common exterior of a set of geometric objects is an important problem in motion planning [20] and solid modeling [15, 19]. 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, which we refer to as the *free space*. 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 faces of $\partial\mathcal{U}$ is proportional to the number of its vertices. The value of κ can be anywhere between $\Omega(1)$ 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 $O(n \log^2 n)$ [3]. 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.

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, 10, 16]. In this line of work the region that needs to be decomposed is given explicitly. However, in many applications, the 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 [20]. In either case, the combinatorial complexity of the region ($\mathcal{K}$ in our case) and the complexity of its decomposition may differ significantly (see, e.g., [7]), making the decomposition task an even harder problem. A general solution to the decomposition problem was given by Schwartz and Sharir [24] who 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 decomposing a region of complex shape into simpler regions, which is more parsimonious than the cylindrical algebraic decomposition, is the "vertical decomposition;" see, e.g., [9, 25]. In our context, it will decompose $\mathcal{K}$ into axis-aligned boxes. However, the size of the vertical decomposition of $\mathcal{K}$ 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 input polyhedron [8], and the known lower bounds show that one cannot hope to do better [7]. The construction in [7] actually gives a set of n pairwise-disjoint prisms in $\mathbb{R}^3$ such that any convex decomposition of their common exterior has $\Omega(n^2)$ size. Paterson and Yao [23] 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 for that matter) 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 [2, 13, 17, 22, 23, 26], is a possible approach to decompose $\mathcal{K}$ into axis-aligned boxes, but its worst-case complexity can be $\Theta(\kappa^2)$. This can be improved, using the technique of [23], 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 [23]. 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.*[2] and Tóth [26] 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 the largest and smallest edge lengths). Unfortunately, the rectangles that arise in a decomposition of 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 [2, 26] 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 partitions $\mathcal{K}$ into $O(\kappa \operatorname{polylog}(n))$ axis-aligned boxes with pairwise-disjoint interiors. By the general-position assumption, every face of $\partial\mathcal{U}$ lies on a face of a single cube in $\mathcal{C}$, which is an important property to have for our algorithm and analysis. Concretely, our first result is the following theorem:

Theorem 1.1 *Let $\mathcal{C}$ be a set of n axis-aligned cubes in $\mathbb{R}^3$ in general position, let κ be the number of vertices on $\partial\mathcal{U}(\mathcal{C})$, and let $t(n,\kappa)$ be the time to compute the faces of $\partial\mathcal{U}(\mathcal{C})$.*

The free space of $\mathcal{C}$ can be partitioned, in time $t(n,\kappa) + O(n\log^2 n + \kappa\log^6 n)$, into $O(\kappa\log^4 n)$ axis-aligned boxes with pairwise-disjoint interiors.

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

Theorem 1.2 *Let $\mathcal{C}$ be a set of n axis-aligned cubes in $\mathbb{R}^3$ in general position, let κ be the number of vertices on $\partial\mathcal{U}(\mathcal{C})$, let $\sigma \le \min\{n,\kappa\}$ be the number of cubes in $\mathcal{C}$ that appear on $\partial\mathcal{U}$, and let $t(n,\kappa)$ be the time to compute the faces of $\partial\mathcal{U}(\mathcal{C})$. The free space $\mathcal{C}$ can be partitioned, in time $t(n,\kappa) + O(n\log^2 n + \sigma\log^8 n + \kappa\log^6 n)$, into $O(\sigma\log^4 n + \kappa\log^2 n)$ axis-aligned boxes with pairwise-disjoint interiors.*

The *depth* of $\mathcal{C}$ is the size of the largest subset of $\mathcal{C}$ with non-empty intersection. If the depth of $\mathcal{C}$ is bounded by a constant, then we obtain a slightly improved result.

Corollary 1.3 *Let $\mathcal{C}$ be a set of n axis-aligned cubes in $\mathbb{R}^3$ in general position and with bounded depth, let κ be the number of vertices on $\partial\mathcal{U}(\mathcal{C})$, let $\sigma \le \min\{n,\kappa\}$ be the number of cubes in $\mathcal{C}$ that appear on $\partial\mathcal{U}$, and let $t(n,\kappa)$ be the time to compute the*

faces of $\partial\mathcal{U}(\mathcal{C})$. *The free space of* $\mathcal{C}$ *can be partitioned, in time* $t(n,\kappa) + O(n\log^2 n + \kappa\log^4 n)$ *(resp.,* $t(n,\kappa) + O(n\log^2 n + \sigma\log^6 n + \kappa\log^5 n)$*), into* $O(\kappa\log^2 n)$ *(resp.,* $O(\sigma\log^2 n + \kappa\log n)$*) axis-aligned boxes with pairwise-disjoint interiors.*

We remark that our algorithm can be extended to degenerate configurations of cubes using symbolic perturbation (also known as simulation of simplicity [14]), but the running time will be proportional to the union-size of the perturbed configuration, which may be 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 partitioned 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 only a constant factor. Therefore we obtain the following result:

Corollary 1.4 *Let* $\mathcal{C}$ *be a set of* n *fat axis-aligned boxes in* $\mathbb{R}^3$ *in general position so that the aspect ratio of every box is bounded by a constant* α*, let* κ *be the number of vertices on* $\partial\mathcal{U}(\mathcal{C})$*, let* $\sigma \le \min\{n,\kappa\}$ *be the number of boxes in* $\mathcal{C}$ *that appear on* $\partial\mathcal{U}$*, and let* $t(n,\kappa)$ *be the time to compute the faces of* $\partial\mathcal{U}(\mathcal{C})$*. The free space of* $\mathcal{C}$ *can be partitioned, in time* $t(n,\kappa) + O(n\log^2 n + \kappa\log^6 n)$ *(resp.,* $t(n,\kappa) + O(n\log^2 n + \sigma\log^8 n + \kappa\log^6 n)$*), into* $O(\kappa\log^4 n)$ *(resp.,* $O(\sigma\log^4 n + \kappa\log^2 n)$*) axis-aligned boxes with pairwise-disjoint interiors. The constant of proportionality in the bounds depends on* α*.*

Agarwal and Steiger [1] described an output-sensitive algorithm to compute the vertices of $\partial\mathcal{U}(\mathcal{C})$ in $O(n\log^3 n + \kappa)$ time. Using standard line-sweep techniques, the edges and faces of $\partial\mathcal{U}(\mathcal{C})$ can be computed from the vertices in $O(\kappa\log\kappa)$ time. Hence, $t(n,\kappa) = O(n\log^3 n + \kappa\log\kappa)$ in the corollary above. Then the overall runtimes are $O(n\log^3 n + \kappa\log^6 n)$ and $O(n\log^3 n + \sigma\log^8 n + \kappa\log^6 n)$, respectively.

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* $\mathbb{R}^3$ *in general position, and let* $\kappa = O(n)$ *be the number of vertices on* $\partial\mathcal{U}(\mathcal{C})$*. The free space of* $\mathcal{C}$ *can be partitioned into* $O(\kappa\log\kappa)$ *axis-aligned boxes with pairwise-disjoint interiors in time* $t(n)$*, where* $t(n) = \Omega(n\log n)$ *is the time to compute the faces of* $\partial\mathcal{U}(\mathcal{C})$*.*

Analogous to Corollary 1.4, we obtain the following corollary:

Corollary 1.6 *Let* $\mathcal{C}$ *be a set of* n *axis-aligned boxes in* $\mathbb{R}^3$ *in general position so that the aspect ratio of every box is bounded by a constant* $\alpha \ge 1$ *and the ratio of the largest to the smallest size box is bounded by a constant* β*, and let* $\kappa = O(n)$ *be the number of vertices on* $\partial\mathcal{U}(\mathcal{C})$*. Then the free space of* $\mathcal{C}$ *can be partitioned into* $O(\kappa\log\kappa)$ *boxes with pairwise-disjoint interiors in time* $t(n)$*, where* $t(n) = \Omega(n\log n)$ *is the time to compute the faces of* $\partial\mathcal{U}(\mathcal{C})$*. The constant of proportionality in the bounds depends on* α *and* β*.*

When the input cubes are congruent, the $\kappa = O(n)$ vertices of $\partial\mathcal{U}(\mathcal{C})$ can be computed in $O(n\log^2 n)$ time [1]. Then the edges and faces of $\partial\mathcal{U}(\mathcal{C})$ can be computed

from the vertices in $O(n \log n)$ time, and hence $t(n) = O(n \log^2 n)$ in the corollary above. Then the overall runtime is $O(n \log^2 n)$.

At a high level, our partitioning techniques are inspired by the BSP construction schemes described in [2, 23, 26], but their implementation exploits the geometry of cubes and attains significantly improved performance bounds. The algorithms are quite simple; only the notations and analyses are somewhat involved.

Lastly, we consider partitioning the free space of a set $\mathcal{C}$ of n arbitrary axis-aligned boxes and obtain the following result:

Theorem 1.7 *Let $\mathcal{C}$ be a set of n axis-aligned boxes in $\mathbb{R}^3$ in general position, and let κ be the number of vertices on $\partial\mathcal{U}(\mathcal{C})$. Then the free space of $\mathcal{C}$ can be partitioned into $O(n^{3/2} + \kappa)$ axis-aligned boxes with pairwise-disjoint interiors in time $O((n^{3/2} + \kappa) \log n)$.*

As mentioned above, there are instances where $\kappa = O(n)$ but any partition of the free space into convex regions has size $\Omega(n^{3/2})$. Therefore the size of the partition constructed by our algorithm has optimal worst-case dependence on both n and κ. We note that here κ can range between $O(1)$ and $\Theta(n^3)$.

Roadmap of the paper. We begin by giving a brief overview of balanced-box decomposition trees [4], 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 Sect. 2, and our second algorithm (given in Theorem 1.2) in Sect. 3. We describe the more efficient algorithm for congruent cubes (given in Theorem 1.5) in Sect. 4. Finally, we describe a simple algorithm for arbitrary boxes (given in Theorem 1.7) in Sect. 5.

2 Decomposing the Free Space of Cubes of Arbitrary Sizes

Let $\mathcal{C}, \mathcal{U}, \mathcal{K}$ be as defined at the beginning of the Introduction. We assume that the faces of $\partial\mathcal{U} := \partial\mathcal{U}(\mathcal{C})$ have been pre-computed, e.g., by using the algorithm in [1] to compute the vertices of $\partial\mathcal{U}$ and then using the vertices to compute the edges and faces of $\partial\mathcal{U}$ with standard line-sweep techniques. 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 the *cut* that splits B into two boxes. For an axis-aligned plane $h : x_i = a$, we use h^- (resp., h^+) to denote the closed halfspace $x_i \le a$ (resp., $x_i \ge a$). 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 (Sect. 2.1), then describe the algorithm (Sects. 2.3 and 2.4), followed by its analysis (Sect. 2.5).

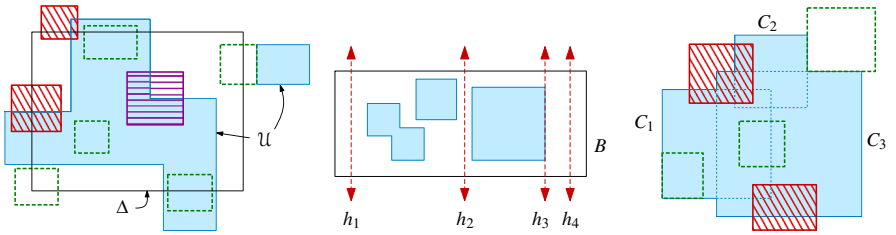

Fig. 1 In each 2D illustration, $\mathcal{U}$ is depicted as blue. (left) The dashed, green boxes are passive and compatible with Δ, the purple box with horizontal stripes is active and compatible with Δ, and the red boxes with diagonal stripes are active but not compatible with Δ. (middle) A box B that 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 dashed, green boxes are void of $\mathcal{C}$; e.g., the bottom, striped, red box is void of $\{C_1, C_2\}$, but not of $\{C_3\}$, since it intersects the boundary of C_3 on $\partial\mathcal{U}$

2.1 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, which in our case 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 $\text{int}(B) \cap \partial\mathcal{U} \subseteq \Delta$. See Fig. 1(left).

Abusing the notation a little, we say that B is *compatible* with a plane h if B is compatible with one of the open 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. We describe in Sect. 2.4 a procedure GLOBALCUT($\mathcal{Z}, h$) that refines the boxes in $\mathcal{Z}$ to make them compatible with h. See Fig. 1(middle) for an illustration of these notions in 2D.

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 boundary of cubes in X, i.e., $\partial\mathcal{U}_X := \partial\mathcal{U}(\mathcal{C}) \cap \partial\mathcal{U}(X)$. 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. We extend the definition of compatible/void to a subset $\mathcal{Z} \subseteq \mathcal{B}$ if the condition holds for all boxes in $\mathcal{Z}$. See Fig. 1(right).

2.2 Balanced-Box Decomposition (BBD) Trees

Let $P \subseteq \mathbb{R}^3$ be a set of n points. Introduced by Arya *et al.* [4], 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

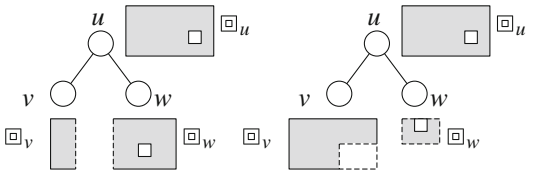

Fig. 2 Two planar renderings of BBD subtrees with identical regions $\boxdot_u$ at the root nodes. On the left, $\boxdot_u$ is partitioned into $\boxdot_v$, $\boxdot_w$ by a splitting plane. On the right, $\boxdot_u$ is partitioned by a splitting box

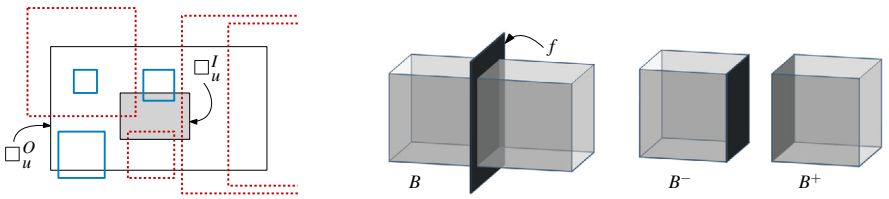

Fig. 3 (left) A 2D illustration of dashed, red, long cubes and solid, blue, short cubes (here squares) that intersect an annular region $\boxdot_u$. (right) An illustration of a box B with a free cut defined by some face f of $\partial\mathcal{U}$, where $B^- := B \cap \text{span}(f)^-$ and $B^+ := B \cap \text{span}(f)^+$

respective children, v and w. See Fig. 2. 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 [4]. See Appendix A and the original paper [4] 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 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 establish 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 where none of its vertices lie in the interior of $\boxdot_u$) contains an edge of each box that it intersects.*

2.3 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 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 Fig. 3(left).

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 $\square_u$, then $\square_u \subseteq \mathcal{U}$ and no refinement of $\square_u$ is needed. Similarly if $\mathcal{L}_u \cup \mathcal{S}_u = \varnothing$, then $\square_u \subseteq \mathcal{K}$ and there is no need to refine $\square_u$. So assume $\partial\mathcal{U}$ intersects $\square_u$. Set $\mathcal{B}_u := \mathcal{B}_{\square_u}$ and $\mathcal{A}_u := \mathcal{A}_{\square_u}$.

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 Fig. 3(right). This notion is similar to the one used in the construction of binary space partitions [22].

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 $\square_u$. That is, for any box B of $\mathcal{A}_u$, $\text{int}(B) \cap \partial\mathcal{U} \subseteq \square_u$. See Fig. 4(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 Fig. 4(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 $\mathcal{B}$, of which only those contained in $\mathcal{K}$ are of interest, 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 $\mathcal{T}$. 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 $\square_u$ 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($\square_u, h_i$) 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 $B \in \mathcal{A}_R$, $\text{int}(B) \cap \partial\mathcal{U} \subseteq R$.

(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 were short at the parent node) 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 $\square_u$ 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 $\text{int}(\square_u)$. We call GLOBALCUT($\square_u, g_i$), $i = 1, 2, 3$, to ensure that $\mathcal{A}_u$ is compatible with g_i. This step ensures that $\mathcal{A}_u$ is void of $\mathcal{S}_u$, which implies that $\mathcal{B}_u$ is void of $\mathcal{C}$.

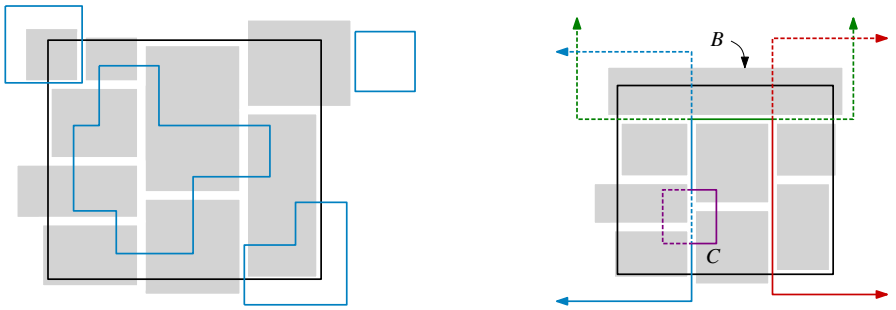

Fig. 4 In both figures, a 2D box $\boxdot_u$ (black) and boxes of $\mathcal{A}_u$ (grey) are shown; for better visualization, the boxes are slightly shrunk towards their centers. (left) An example of invariant (I1): $\mathcal{A}_u$ is compatible with $\boxdot_u$. $\partial\mathcal{U}$ is shown in blue. (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

(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). The algorithm terminates when the recursion terminates, back at the root $\square$ of $\mathcal{T}$, and we output the subcollection of those boxes of $\mathcal{B}$ that are contained in $\mathcal{K}$.

2.4 The Procedures GLOBALCUT and STAIRCASE

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 $B \in \mathcal{A}_\Delta$, $\text{int}(B) \cap \partial\mathcal{U}$ 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., $\text{int}(B) \cap \partial\mathcal{U} \cap h = \varnothing$ 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 $\text{int}(B) \cap \partial\mathcal{U} \cap h = \varnothing$ 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 Fig. 5. If either of the boxes B^-, 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 Fig. 5(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

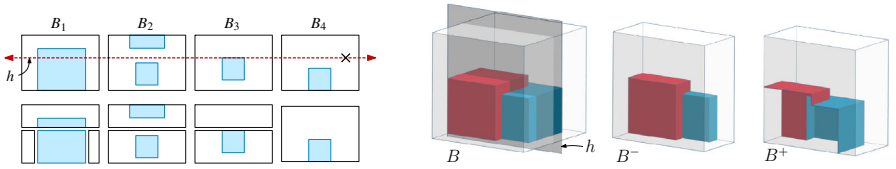

Fig. 5 (left) A 2D view of boxes $B_1, B_2, B_3, B_4 \in \mathcal{A}_\Delta$ before (top) and after (bottom) the call GLOBALCUT(Δ, h), where $\mathcal{U} \cap \Delta$ is depicted in blue. Boxes B_1, B_2, B_3 are split by h during the call, but not B_4. After the call, sub-box of B_1 admitted free cuts supporting each face of $\partial\mathcal{U}$ 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): $\text{int}(B) \cap \mathcal{U}$ 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

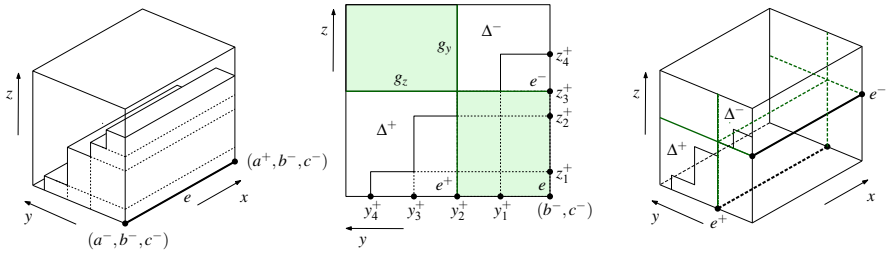

Fig. 6 An illustration of STAIRCASE(Δ, e, X)

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 $\partial\mathcal{U}(X) \cap \text{int}(\Delta)$ because otherwise we can simply ignore C in the present invocation of the procedure (removing C does not alter $\partial\mathcal{U}(X) \cap \text{int}(\Delta)$ because C is long at Δ). 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 Fig. 6(middle). 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 that $\mathcal{A}_\Delta$ 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 Fig. 6 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^+).)

2.5 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 B 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 Fig. 5. 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 B of $\mathcal{B}_i$ into two boxes B_1 and B_2, i.e., $\mathcal{B}_{i+1} = (\mathcal{B}_i \setminus \{B\}) \cup \{B_1, B_2\}$, 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 $B \in \mathsf{V}$ 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 Fig. 7 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 and performs a cut of B, either φ does not intersect the cut, or φ is divided into smaller fragments by the cut, or φ is "trapped" by the cut, meaning that the cut contains φ. Once φ is trapped by a cut, it stops being a fragment and is not divided further anymore; we refer to φ as becoming *eternal* — 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.

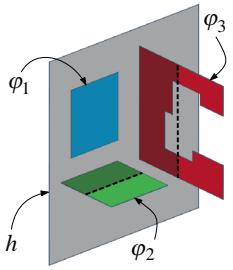

Fig. 7 Examples of fragments that are alive in a common box (not shown) that is cut by plane h. φ_1 becomes eternal, φ_2 and φ_3 die, and the maximally connected portions of φ_2, φ_3 in h^-, h^+ are newly created fragments — two for φ_2 and three for φ_3

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 (because none of the cuts intersect φ) 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 Fig. 7.

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 is not considered to 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 $\mu := |\Phi_F|$ and $\nu := |\Gamma|$. 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 prove below that $|\mathcal{B}_F| = O(\mu)$ (*cf.*Lemma 2.3) and $\nu = O(\kappa \log^4 n)$ (*cf.*Corollary 2.9). Together, these bounds imply the stated upper bound for $|\mathcal{B}_F|$.

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

Lemma 2.2 *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.

First, we prove (I1) and (I2) by induction on the depth of the node v in $\mathcal{T}$. (I1) states that upon reaching a node u of $\mathcal{T}$, $\mathcal{A}_u$ is compatible with $\boxdot_u$. It is trivially true

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 $p(v)$. 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 $\text{int}(B) \cap \partial\mathcal{U}$ 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 in $\Sigma_{p(v)}$, which would have made the cut along σ, exactly because $\text{int}(B) \cap \partial\mathcal{U}$ lies in both open halfspaces bounded by σ. This in turn 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 (I2). Namely, 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). 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, 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). □

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

Lemma 2.3 *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 across B'. We claim that h supported a live fragment φ lying in $\text{int}(B')$. Suppose h did not support a fragment inside B'. Since B' was split by h, $\partial\mathcal{U} \cap B'$ lies in both halfspaces h^- and h^+ bounded by h. Hence, either h divided a live fragment in B' into two fragments, or $\text{int}(B') \cap \partial\mathcal{U} \cap h = \varnothing$ (see Fig. 5). In either case both $\text{int}(B)$ and $\text{int}(\overline{B})$ must intersect $\partial\mathcal{U}$, or else the cut along h would not have been made, 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, $|\mathcal{B}_F| \leq 2\mu$. □

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

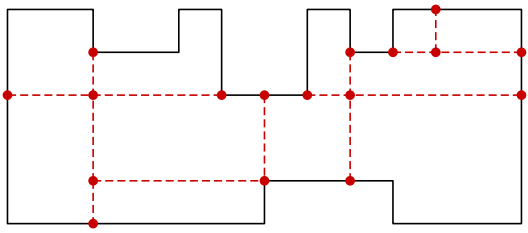

Fig. 8 A 2D view of the subdivision Π_f (red) of a face f of $\partial\mathcal{U}$

f induced by the cuts that cross f throughout the execution of the algorithm, and monitor how $\widetilde{\Pi}_f$ evolves over time. (When a portion of f is trapped by a cut, that portion becomes eternal and is not further subdivided.) In particular, whenever a box B with $f \cap \text{int}(B) \neq \varnothing$ is split by an axis-aligned plane h that crosses 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 that lie on f. See Fig. 8. By definition, every edge of Π_f is a portion of a segment γ corresponding to some cut that was applied during the algorithm and that crossed f. Many edges of Π_f 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 of Π_f with a node v of the BBD tree $\mathcal{T}$ if the call to GLOBALCUT during which it was created was executed while processing v.

We call an axis-parallel segment contained in a face of $\partial\mathcal{U}$ a *mast* (as in [26]).

Lemma 2.4 *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 $\ell := h \cap \text{span}(f)$. If γ is parallel to h, then γ is parallel to ℓ (and possibly contained in 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 $\mathcal{E}_{\Delta,h}$. □

Lemma 2.5 *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^+

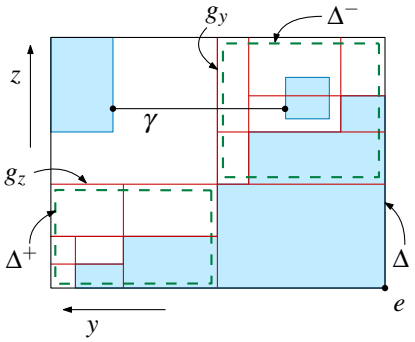

Fig. 9 A 2D view of the edges $\mathcal{E}_{\Delta,e,X}$ (red) created in subdivision Π_f on $\Delta \cap \text{span}(f)$ after calling STAIRCASE(Δ, e, X). The shaded regions (blue) form the cross section $\mathcal{U} \cap \Delta$, which includes a long cube that contains the top-left x-edge of Δ and a short cube on which an endpoint of mast γ lies. The boundaries of Δ^-, Δ^+ in the initial call are shown slightly shrunk towards their centers for better visualization

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 2.4, 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 2.4, γ crosses no edge created by GLOBALCUT(Δ, g_z) and at most one edge created by GLOBALCUT(Δ, g_y). See Fig. 9. 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 $|X^-|, |X^+| \leq \lceil |X|/2 \rceil - 1 \leq |X|/2$, a simple recursive argument shows that γ crosses $O(\log|X|)$ edges of $\mathcal{E}_{\Delta,e,X}$. □

Remark 1 We note that if the depth of $\mathcal{C}$ is bounded by a constant, then $|X| = O(1)$ and the mast γ crosses $O(1)$ edges of $\mathcal{E}_{\Delta,e,X}$.

Lemma 2.6 *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 2.4. 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 $\gamma_R := \gamma \cap R$. For each edge e of R, by Lemma 2.5, γ_R crosses $O(\log|\mathcal{C}_{R,e}|) = O(\log n)$ 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).

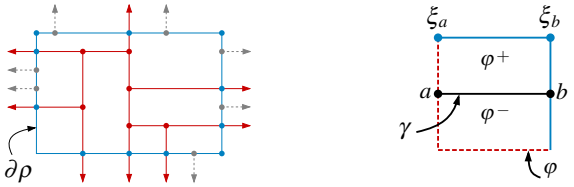

Fig. 10 (left) An illustration of Π_f clipped within ρ is shown in red, where the outer vertices, inner vertices, and vertices of Π_f that are on edges of $\partial\rho$ but not in Π_ρ are depicted as blue, red, and grey, respectively. (right) An exposed face φ of $\widetilde{\Pi}$ is split by segment γ, where the solid blue edges (resp., dashed red edges) lie on $\partial\rho$ (resp., in $\text{int}(\rho)$). b is an outer vertex, which a charges, being an inner vertex

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 2.7 *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 $i_v \leq F$ 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 in $\mathcal{B}_{i_v}$, i.e., for all $B \in \mathcal{A}_v^<$, $\text{int}(B) \cap \partial\mathcal{U} \cap \boxdot_v \neq \varnothing$. By invariant (I1), $\text{int}(B) \cap \partial\mathcal{U} \subseteq \boxdot_v$ 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 (children of at most 8 nodes at a fixed level at which C is short) 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 2.6, 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 yz-plane. From each vertex q of f, we draw a ray in the $(+z)$-direction or in the $(-z)$-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. The subdivision of $f^{||}$ is constructed only for the analysis and is not part of the algorithm. Consider any rectangle ρ of $f^{||}$. The y-edges of ρ are portions of ∂f but the z-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 Fig. 10(left). 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.the grey vertices in Fig. 10(left)). 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 [26], bounds the number of vertices of Π_ρ.

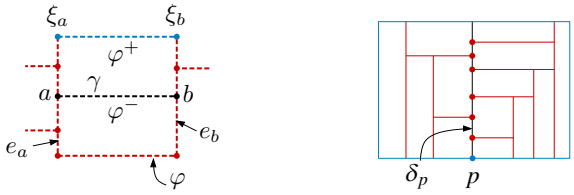

Fig. 11 (left) An exposed face φ of $\widetilde{\Pi}$ is split by segment γ, where the solid blue edges (resp., dashed red edges) lie on $\partial\rho$ (resp., in $\text{int}(\rho)$). a and b are both inner vertices and a (resp., b) charges the outer vertex ξ_a (resp., ξ_b). e_a, e_b are the left and right vertical edges of ρ. (right) An illustration of a mast δ_p, all of whose incident inner vertices (red) charge the outer vertex p

Lemma 2.8 *For each rectangle ρ of $f^{||}$, Π_ρ contains $O(\log^4 n)$ vertices.*

Proof We call a vertex of Π_ρ lying in $\text{int}(\rho)$ 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 2.7 implies[1] that ε contains $O(\log^2 n)$ vertices of Π_ρ. Hence, Π_ρ has $O(\log^2 n)$ outer vertices. See Fig. 10(left). We carefully charge each inner vertex to an outer vertex so that each outer vertex is charged by only $O(\log^2 n)$ 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 gets 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, $\widetilde{\Pi}$ either remains unchanged or 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 accounted 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 Fig. 10(right).

The second case is when both a and b are inner vertices. Let φ be the face of Π_ρ that is split by γ, which is a rectangle. 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 Fig. 11(left). 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

[1] Since vertices of Π_f that are not vertices of Π_ρ might lie on ε, we should apply Lemma 2.7 to the segment ε' that is the slight translation of $\text{int}(\varepsilon)$ into $\text{int}(\rho)$; 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.

(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 Fig. 11(left).

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.) See Fig. 11(right). Each inner vertex charged to p lies on δ_p and is an intersection point of δ_p with an edge of Π_f (which is orthogonal to δ_p). By Lemma 2.7,[2] δ_p contains $O(\log^2 n)$ such intersection points. 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 2.9 *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 size bound in Theorem 1.1. If the depth of $\mathcal{C}$ is bounded by a constant then using Remark 1, we can conclude that a mast γ lying on a face f of $\partial\mathcal{U}$ crosses $O(\log n)$ edges of Π_f, which implies that f contains $O(\kappa_f \log^2 n)$ fragment vertices. This in turn implies the first size bound in Corollary 1.3.

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 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 the points of Γ_B sorted by their x-, y-, and z-coordinates, respectively. We store $\mathsf{L}_\varphi, \mathsf{X}_B, \mathsf{Y}_B$, 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_B and find the last vertex ξ^- with z-coordinate less than z_0. Next, we scan the set Φ_B of the fragments in B. 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] As similarly remarked earlier, to use the lemma, we choose a mast parallel and very close to δ_p.

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 X_B, 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 Fig. 7). Each fragment now either lies in B^- or in B^+.
   (d) By scanning the lists Φ_B, X_B, 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. For each newly created box D, we split D by each free cut and construct the lists Φ_D, X_D, Y_D, and Z_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, and then 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. Then it 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.

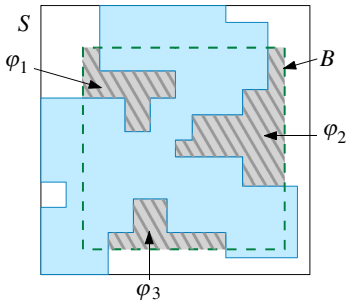

Fig. 12 A 2D view of a boundary square S of some cube that induces a free cut, by the new definition, in the dashed box B (green), but none of the fragments $\varphi_1, \varphi_2, \varphi_3$ (grey with stripes) on $S \cap B$ induce a free cut in B by the old definition. $S \cap \text{int}(\mathcal{U})$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

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 2.3 and Corollary 2.9. The analysis in Arya *et al.*[4] 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 $O(n \log^2 n + \kappa \log^6 n)$. This proves the running time bound in Theorem 1.1. If the depth of $\mathcal{C}$ is bounded by a constant then using the improved bound on the size of the decomposition we conclude that the running time is $O(n \log^2 n + \kappa \log^4 n)$, which proves the first running time bound of Corollary 1.3.

3 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 $\sigma \leq \min\{n, \kappa\}$ 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[3] 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 Fig. 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 Sect. 2.3 but use this definition of a free cut in the GlobalCut procedure.

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

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 2.2 and 2.3 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 Sect. 2, 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 $\kappa_S > 0$ 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 $\mathcal{B}_F$. (If S contains no vertices of $\partial\mathcal{U}$ then $S \subset \text{int}(\mathcal{U})$ and no fragment vertices are created on S.)

The overall structure of the proof is similar to that in Sect. 2.5 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 that 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 Fig. 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 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 Fig. 14.) The number of blue vertices is κ_S, by definition, so we need to bound the number of red and purple vertices.

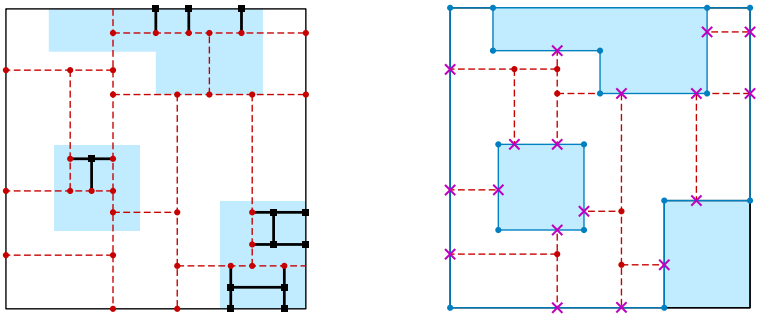

Fig. 13 A 2D view of Π_S (left) and Π_S^{∇} (right) on a square S; $S \cap \text{int}(\mathcal{U})$ is shaded blue and its interior does not belong to Π_S^{∇} (only its blue boundary edges belong to Π_S^{∇}). On the left, the red edges are dashed, and the black edges are thick. On the right, the red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

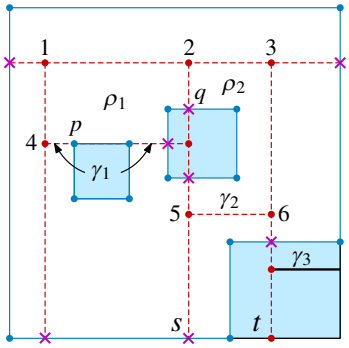

Fig. 14 A 2D view of $\widetilde{\Pi}_S$ that illustrates the various cases for new edges with segments $\gamma_1, \gamma_2, \gamma_3$. We assume that all other segments were created before them. $S \cap \text{int}(\mathcal{U})$ is shown in blue. γ_1 intersects blue edges of $\mathcal{K}_S$, γ_2 lies in $\mathcal{K}_S$, and γ_3 lies in $S \cap \text{int}(\mathcal{U})$. 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

We define a *mast* to be an axis-aligned segment contained in S. Note that unlike Sect. 2 where a mast lies on $\partial\mathcal{U}$, a mast may now intersect $\text{int}(\mathcal{U})$. The following lemma is analogous to Lemma 2.4.

Lemma 3.1 *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 an 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 2.4. □

Using Lemma 3.1 and following the same arguments as in the proofs of Lemmas 2.5–2.7, we obtain the following:

Corollary 3.2 *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 2.8.

Lemma 3.3 *Π_S^∇ has $O(\log^4 n + \kappa_S \log^2 n)$ vertices.*

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 3.2, each edge of $\partial\mathcal{K}_S$ contains $O(\log^2 n)$ purple vertices. (As in the proof of Lemma 2.8, strictly speaking, we choose a mast parallel and very close to the edge so as to use Corollary 3.2). 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 Fig. 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 $\widetilde{\Pi}_S$, which we view as a rectangle, was split into two faces by the creation of an axis-aligned segment γ with endpoints a and b, which become vertices of $\widetilde{\Pi}_S$. There are three cases. First, if $\gamma \subseteq S \setminus \mathcal{K}_S$, then a and b are not vertices of Π_S^∇. Next, assume that $\gamma \nsubseteq S \setminus \mathcal{K}_S$ and $\gamma \nsubseteq \mathcal{K}_S$. If a (resp., b) lies in the interior of $\mathcal{K}_S$, it is a red vertex of Π_S^∇ (and thus 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 Fig. 14), or 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 Fig. 14). It is easily seen that η is charged at most twice[4] in this way, so the number of such red vertices of Π_S^∇ is $O(\kappa_S \log^2 n)$.

Finally, assume that $\gamma \subseteq \mathcal{K}_S$, i.e., both a and b are red vertices of Π_S^∇. 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 2.8, 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 Fig. 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.,

[4] 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.

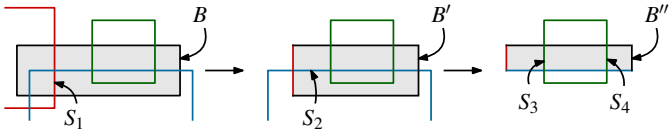

Fig. 15 A 2D view of a 3D shaded box B whose interior is intersected by three cubes (squares from this view) that are long at B (i.e., their vertices lie outside B). By the old definition of free cuts, there are no available free cuts in B since the boundary squares S_1, S_2, S_3, S_4 (segments from this view) of the long cubes intersect each other in the interior of B. However, by the new definition of free cuts, the boundary square S_1 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''

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 2.8, and using Corollary 3.2, any red vertex of Π_S on ∂S is charged $O(\log^2 n)$ times. Finally, using Theorem 3.2, 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. □

Putting everything together, the total number of fragments created by the algorithm is $O(\sigma \log^4 n + \kappa \log^2 n)$. This proves the size bound in Theorem 1.2. If the depth of $\mathcal{C}$ is bounded by a constant then the same observation as in Remark 1 implies that a mast in S crosses $O(\log n)$ red edges of Π_S, which in turn implies that Π_S^{∇} has $O(\log^2 n + \kappa_S \log n)$ vertices. This proves second size bound in Corollary 1.3.

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 Sect. 2.5.

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 Φ_B, X_B, 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 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 Fig. 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 X_D, 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 $g := D \cap \text{span}(S_v)$ lying below and above g, respectively. 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 Φ_{D^-}, 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 X_D, 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 $\nu_{D^-} := |\Gamma_{D^-}|$ and $\nu_D^g := |\Gamma_{D^-}^g|$. Because we scan the lists in lock-step manner, we can conclude that $\nu_{D^-} \leq \nu_{D^+}$, where $\nu_{D^+} := |\Gamma_{D^+}|$. Then the time spent in splitting D into D^- and D^+ is $O(\nu_{D^-} \log \nu_{D^-} + \nu_D^g)$. The time spent in the symmetric case where a free cut was found while scanning Z_D from right to left is $O(\nu_{D^+} \log \nu_{D^+} + \nu_D^g)$ with $\nu_{D^+} \leq \nu_{D^-}$. Therefore, the time spent in any case 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:

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

where $\nu_{D^-} + \nu_D^g + \nu_{D^+} \leq \nu_D$, 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 Sect. 2.5, each of the $O(1)$ calls to Staircase at u now take $O((n_u + \nu_u \log^2 \nu_u) \log n) = O((n_u + \nu_u \log^2 n) \log n)$, 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 running time bound in Theorem 1.2. If the depth of $\mathcal{C}$ is bounded, the running time is $O(n \log^2 n + \sigma \log^6 n + \kappa \log^5 n)$, which proves the second running time bound in Corollary 1.3.

4 Decomposing the Free Space of Congruent Cubes

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

4.1 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. Recall that $\kappa = O(n)$ in this setting. As before, we assume that the faces of $\partial\mathcal{U} := \partial\mathcal{U}(\mathcal{C})$ have been pre-computed in time $t(n) = \Omega(n \log n)$, e.g., by using the algorithm in [1] to compute the vertices of $\partial\mathcal{U}$ and then using the vertices to compute the edges and faces of $\partial\mathcal{U}$ with standard line-sweep techniques. Unlike the setup in Sect. 2, 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 3D 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 $\mathbb{G}^{\downarrow}$ be the 2D integer grid on the xy-plane, and let $\xi^{\downarrow}_{i,j}$ denote the unit square $[i, i+1] \times [j, j+1]$. For a pair (i, j), let $\Pi_{i,j} := \xi^{\downarrow}_{i,j} \times \mathbb{R}$ denote the unbounded prism erected on the square $\xi^{\downarrow}_{i,j}$ and extending along the z-axis, and let $\mathbb{G}_{i,j} := \{\xi_{i,j,k} \mid k \in \mathbb{Z}\}$ denote the column of grid cells stacked on $\xi^{\downarrow}_{i,j}$; $\mathbb{G}_{i,j}$ partitions $\Pi_{i,j}$ into a "stack" of 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 $\mathcal{C}$; $\mathcal{U}(\mathcal{C}) \subset \mathcal{U}(\mathcal{G}) \subset \bigcup_{(i,j)\in\mathsf{X}} \Pi_{i,j}$ and $|\mathcal{G}|, |\mathsf{X}| = O(\kappa)$.

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 $O(\kappa)$ boxes, as follows. We partition $\mathrm{cl}(\mathbb{R}^2 \setminus \bigcup_{(i,j)\in\mathsf{X}} \xi^{\downarrow}_{i,j})$ into $O(\kappa)$ axis-aligned rectangles, using, say, the standard 2D vertical decomposition. For each rectangle ρ in the decomposition, we add the unbounded prism $\rho \times \mathbb{R}$ to $\mathcal{B}_1$. See Fig. 16.

Next, for each pair $(i, j) \in \mathsf{X}$, 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 $\mathcal{B}_{i,j}$ of at most $|\mathcal{G}_{i,j}| + 1$ boxes. See Fig. 17. Set $\mathcal{B}_2 := \bigcup_{(i,j)\in\mathsf{X}} \mathcal{B}_{i,j}$. For any $(i, j) \in \mathsf{X}$, there is exactly one box in $\mathcal{B}_{i,j}$ that is unbounded in the $(-z)$-direction. Furthermore, for any grid cell $\xi \in \mathcal{G}_{i,j}$, there is a vertex of $\partial\mathcal{U}$ in the interior of ξ since the cubes of $\mathcal{C}$ are in general position. Hence, for any box $B \in \mathcal{B}_{i,j}$ bounded in the $(-z)$-direction, there exists a vertex of $\partial\mathcal{U}$ in the grid cell immediately below B. It follows that

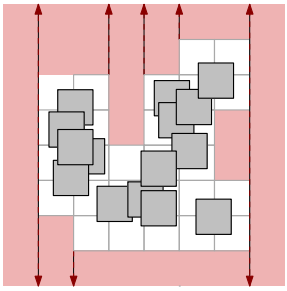

Fig. 16 A view from above of $\mathbb{G}^{\downarrow}$ with the set of cubes $\mathcal{C}$ (grey), and the partition of $\mathrm{cl}(\mathbb{R}^2 \setminus \bigcup_{(i,j)\in \mathsf{X}} \xi_{i,j}^{\downarrow})$ into axis-aligned rectangles (red)

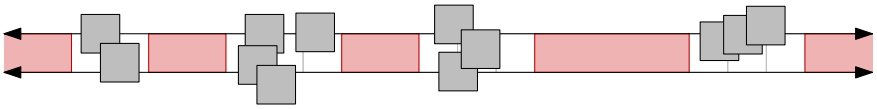

Fig. 17 A 2D view of a prism $\Pi_{i,j}$ crossed by cubes of $\mathcal{C}$ (grey), $(i,\, j) \in \mathsf{X}$. $\mathrm{cl}(\Pi_{i,j} \setminus \mathcal{U}(\mathcal{G}_{i,j}))$ is partitioned into axis-aligned boxes (red)

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

$\mathcal{B}_1 \cup \mathcal{B}_2$ partitions $\mathrm{cl}(\mathbb{R}^3 \setminus \mathcal{U}(\mathcal{G}))$ into $O(\kappa)$ 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 $\sum_{\xi\in\mathcal{G}} \kappa_\xi = O(\kappa)$. 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 4.6). Repeating this procedure for all grid cells $\xi \in \mathcal{G}$, we decompose $\mathcal{U}(\mathcal{G}) \cap \mathcal{K}$ into a total of $\sum_{\xi\in\mathcal{G}} O(\kappa_\xi \log \kappa_\xi) = O(\kappa \log \kappa)$ boxes.

Putting everything together, $\mathcal{B}_1 \cup \mathcal{B}_2 \cup \bigcup_{\xi\in\mathcal{G}} \mathcal{B}_\xi$ partitions $\mathcal{K}$ into $O(\kappa \log \kappa)$ axis-aligned boxes. Moreover, as we will show, our algorithm runs in overall $O(\kappa \log \kappa) = O(n \log n)$ time. The time $t(n)$ to compute the faces of $\partial\mathcal{U}$ is $\Omega(n \log n)$, so the overall runtime is $t(n)$. This completes the proof of Theorem 1.5.

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 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 $\kappa_\square := |\mathsf{V}_\square|$ 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.

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 particular, the above invariant 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. 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 4.1 *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.*

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 Fig. 18.

The edge e is either a *concave* edge,[5] 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 $a^\downarrow$. In the case where e is a concave edge, move q_1 slightly along γ_1 so as to make $q_1^\downarrow$ be contained in $f^\downarrow$, 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.

Corollary 4.2 *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.*

Proof Let e be a long x-edge in $\square$ and let γ be a long y-edge in $\square$. Suppose that e lies above γ in the z-direction. Then by Lemma 4.1, e lies above all long y-edges, and similarly γ lies below all long x-edges. By applying Lemma 4.1 again, we can conclude that all the long x-edges in $\square$ lie above all long y-edges in $\square$. The case where e lies below γ is handled in a fully symmetric manner. □

[5] 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 $\mathcal{U}$.

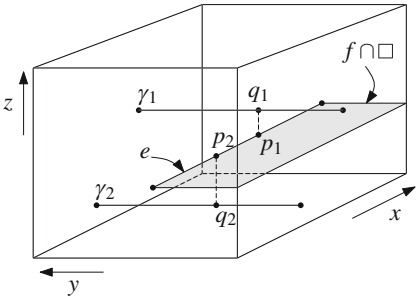

Fig. 18 An illustration of the proof of Lemma 4.1

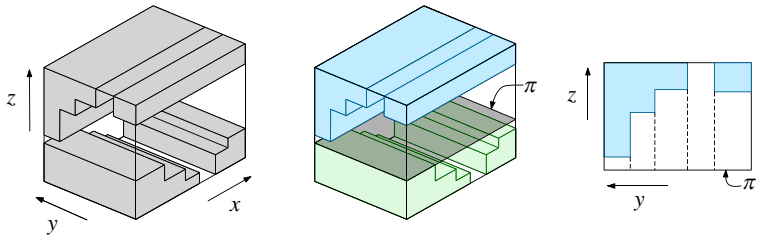

Fig. 19 An illustration of Lemma 4.3 for a box $\square$ with two families of long edges. (left) The undecomposed scenario. (middle) Separating the two families with a plane π that contains the highest long y-edge in $\square$. (right) Decomposing the portion of $\square \cap \mathcal{K}$ above π into $O(m_x)$ axis-aligned boxes, 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 4.3 *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.

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 $\mathcal{K}_\varphi := \mathcal{K} \cap \varphi$; 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 $\mathcal{K}_x = \mathcal{K}_\varphi \times [x_L, x_R]$. We partition $\mathcal{K}_\varphi$ into $O(m_x)$ axis-aligned rectangles by the standard planar vertical-decomposition method, as at the beginning of this section (see Fig. 19(right)). We extend each rectangle R in the decomposition of $\mathcal{K}_\varphi$ 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 4.4 *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 κ_i, m_i denote $\kappa_{\square_i}, m_{\square_i}$, respectively.*

(i) $\kappa_1 + \kappa_2 + \kappa_3 \leq \kappa_\square$,
(ii) $\kappa_i \leq \lceil \kappa_\square/2 \rceil$ *for every* $i = 1, 2, 3$,
(iii) $m_1 + m_2 + m_3 \leq m_\square + 2\kappa_\square$, *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 4.3, 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 $\lceil \kappa_\square/2 \rceil$ 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 $\square_3$. Otherwise, the interior of one of them, say, of $\square_x$, contains more than $\lceil \kappa_\square/2 \rceil$ vertices, and we partition $\square_x$ further into two boxes by drawing some suitable plane orthogonal to the z-axis that partitions $\square_x$ into two sub-boxes, each containing at most $\lceil \kappa_\square/2 \rceil$ 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 in $\square$, 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 $m_\square + 2\kappa_\square$, thereby proving (iii). □

Let $\psi(m_\square, \kappa_\square)$ 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 $|\mathsf{V}_\square| = \kappa_\square$ and $|\mathsf{E}_\square| = m_\square$. Lemmas 4.3 and 4.4 imply the following recurrence:

$$\psi(m_\square, \kappa_\square) \leq \begin{cases} 1 & \text{if } m_\square = \kappa_\square = 0, \\ c_1 m_\square & \text{if } m_\square > 0, \kappa_\square = 0, \\ \sum_{i=1}^{3} \psi(m_i, \kappa_i) & \text{if } m_\square \geq 0, \kappa_\square > 0, \end{cases}$$

where $\kappa_i \leq \lceil \kappa_\square/2 \rceil$, $\kappa_1 + \kappa_2 + \kappa_3 \leq \kappa_\square$, $m_1 + m_2 + m_3 \leq m_\square + 2\kappa_\square$ and $c_1 > 0$ is an absolute constant. A solution to the above recurrence is $\psi(m_\square, \kappa_\square) = O(m_\square +$

$\kappa_\square \log \kappa_\square$). We also note that the total time spent in constructing the decomposition of $\mathcal{K}_\square$ into boxes can be shown to be $O((m_\square + \kappa_\square) \log \kappa_\square)$. In conclusion, we have obtained the following result.

Lemma 4.5 *If $\mathcal{K}_\square$ contains at most two families of long edges, then $\mathcal{K}_\square$ can be partitioned into $O(m_\square + \kappa_\square \log \kappa_\square)$ boxes in $O((m_\square + \kappa_\square) \log \kappa_\square)$ time.*

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 4.5, $\mathcal{K}_\xi$ 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_ξ is the number of vertices of $\mathcal{U}$ that lie in the interior of ξ.

Corollary 4.6 *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.*

5 Decomposing the Free Space of Boxes

In this section we consider partitioning the free space of a set $\mathcal{C}$ of n axis-aligned boxes in general position into boxes. Let V be the set of vertices of the boxes in $\mathcal{C}$. For a set $\mathscr{A}$ of axis-aligned objects, let $\mathcal{U}(\mathscr{A})$ denote the union of the objects in $\mathscr{A}$. Set $\mathcal{U} := \mathcal{U}(\mathcal{C})$. For a box $D \subset \mathbb{R}^3$, let $\mathcal{C}_D \subseteq \mathcal{C}$ be the subset of boxes intersecting the interior of D. For a box B, let $z(B)$ be its projection onto the z-axis. A box $B \in \mathcal{C}_D$ is *long* in D if $z(B) \supseteq z(D)$, and is *short* otherwise. (Note that these definitions of long and short differ from those in the previous sections.) Let $\mathcal{L}_D$ (resp., $\mathcal{S}_D$) be the subset of boxes of $\mathcal{C}_D$ that are long (resp., short) in D. Let $B^\downarrow$ denote the xy-projection of a 3D object B. For any set $\mathscr{A}$ of 3D objects, let $\mathscr{A}^\downarrow := \{A^\downarrow \mid A \in \mathscr{A}\}$.

5.1 Algorithm

Let $\square$ be an axis-aligned box containing $\mathcal{U}$ in its interior. We partition $\square$ by horizontal planes into $r = \lceil 8\sqrt{n} \rceil$ boxes $\square_1, \ldots, \square_r$, which we refer to as *slabs*, so that each slab $\square_i$ contains at most $\sqrt{n}$ vertices of V. Since we assume that the boxes of $\mathcal{C}$ are in general position, we can ensure that the boundaries of the slabs do not contain any vertex of V. We partition $\mathbb{R}^3 \setminus \square$ into $O(1)$ boxes as before, and need to partition $\square \setminus \mathcal{U}(\mathcal{C})$.

For all $1 \le i \le r$, set $\mathcal{C}_i := \mathcal{C}_{\square_i}$, $\mathcal{U}_i := \mathcal{U}(\mathcal{C}_i) \cap \square_i$, $\mathcal{K}_i := \mathrm{cl}(\square_i \setminus \mathcal{U}_i)$, $\mathcal{L}_i := \mathcal{L}_{\square_i}$, $\mathcal{S}_i := \mathcal{S}_{\square_i}$, and $J_i := z(\square_i)$. Since no vertices of V lie on the faces of $\square_i$, we have $\mathcal{L}_1 = \mathcal{L}_r = \varnothing$ and $\mathcal{L}_i \subseteq \mathcal{C}_{i-1}, \mathcal{C}_{i+1}$ for all $1 < i < r$. Fix a slab $\square_i$. Let $\nabla_i := \mathrm{cl}(\square_i^\downarrow \setminus \mathcal{U}(\mathcal{C}_i^\downarrow))$ and $\Sigma_i := \mathrm{cl}(\mathcal{U}(\mathcal{S}_i^\downarrow) \setminus \mathcal{U}(\mathcal{L}_i^\downarrow))$. These are rectilinear regions in the xy-plane that are interior-disjoint, may be disconnected, or may have many holes. See Fig. 20a–c. Clearly $\nabla_i \subseteq \mathcal{K}_i^\downarrow$ and $\nabla_i \cup \Sigma_i \supseteq \mathcal{K}_i^\downarrow$; the latter follows because $\mathrm{cl}(\square^\downarrow \setminus (\nabla_i \cup \Sigma_i)) = \mathcal{U}(\mathcal{L}_i^\downarrow)$ and thus it is interior-disjoint from $\mathcal{K}_i^\downarrow$. Let $\nabla_i^\uparrow := \nabla_i \times J_i$, $\Sigma_i^\uparrow := \Sigma_i \times J_i$, and $\Psi_i := \mathrm{cl}(\Sigma_i^\uparrow \setminus \mathcal{U}(\mathcal{S}_i))$. We have $\mathcal{K}_i = \nabla_i^\uparrow \cup \Psi_i$.

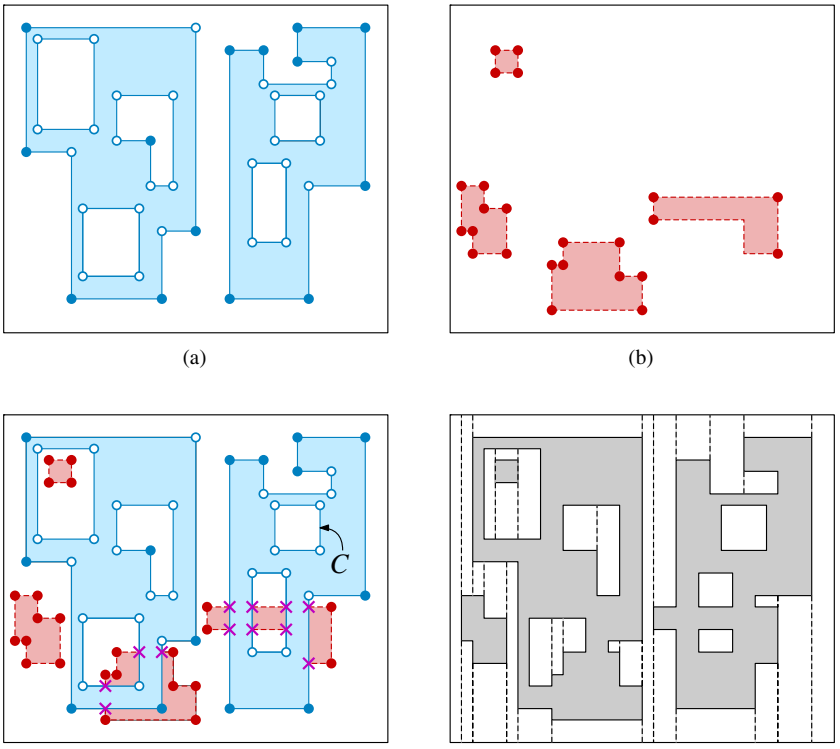

Fig. 20 In all figures, $\partial\square_i^{\downarrow}$ is shown in black. For the purpose of illustration, the boxes of $\mathcal{C}_i$ are not in general position in this example. (a) $\mathcal{U}(\mathcal{L}_i{}^{\downarrow})$ in blue. (b) $\mathcal{U}(\mathcal{S}_i{}^{\downarrow})$ in red. (c) ∇_i in white and Σ_i in red. ∇_i has a single trivial component C. The vertices of ∇_i are colored as described in the analysis for partitioning $\nabla_i^{\uparrow}$, with the primitive (resp., composite) blue vertices shown as solid (resp., hollow). (d) The white rectangles compose the 2D vertical decomposition Π_i of the non-trivial components of ∇_i

Our algorithm partitions $\bigcup_{i=1}^{r} \nabla_i^{\uparrow}$ and Ψ_i for $1 \leq i \leq r$ separately. Sects. 5.2 and 5.3 describe the decomposition of $\bigcup_{i=1}^{r} \nabla_i^{\uparrow}$ and of Ψ_i, respectively.

5.2 Partitioning $\bigcup_{i=1}^{r} \nabla_i^{\uparrow}$

We partition $\bigcup_{i=1}^{r} \nabla_i^{\uparrow}$ in two stages, as follows. We call a component C of some ∇_i *trivial* if it is a rectangle (with no holes) whose edges are the xy-projections of long boxes in $\mathcal{L}_i$, and *non-trivial* otherwise; see Fig. 20c. First, for each $i \leq r$, we do the following. We construct a partition Π_i of the non-trivial components of ∇_i into rectangles using standard 2D vertical decomposition; see Fig. 20d. We then lift every rectangle $\rho \in \Pi_i$ into the box $\rho \times J_i$, which becomes part of our final decomposition.

Next, we treat the trivial components differently and decompose them globally across the slabs. For each trivial component $C \in \nabla_i$, let j, k be the extremal indices with $1 < j \leq i \leq k < r$ such that C is also a trivial component of $\nabla_j, \ldots, \nabla_k$ but not

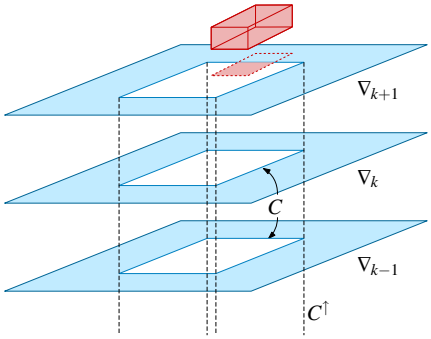

Fig. 21 Illustrations of $\nabla_{k-1}, \nabla_k, \nabla_{k+1}$, depicted in three dimensions on the bottom xy-faces of the slabs. The shaded blue (resp., red) portion of each ∇_i is contained in $\mathcal{U}(\mathcal{L}_i^{\downarrow})$ (resp., $\mathcal{U}(\mathcal{S}_i^{\downarrow}) \setminus \mathcal{U}(\mathcal{L}_i^{\downarrow})$). The white component C common to ∇_{k-1}, ∇_k is trivial and does not appear in ∇_{k+1} since the xy-projection of the red short box intersects the interior of C. The box $C^{\uparrow}$ constructed for C is depicted with its z-edges as dashed and its top xy-face lying on the bottom xy-face of slab $\square_{k+1}$. The long boxes are not shown for readability

of ∇_{j-1} or of ∇_{k+1}. We add the single box $C^{\uparrow} := C \times (\bigcup_{j \le m \le k} J_m)$ that straddles these $k - j + 1$ contiguous slabs to our decomposition. See Fig. 21. These two steps together partition $\bigcup_{i=1}^{r} \nabla_i^{\uparrow}$ into boxes.

Analysis. We now bound the size of the resulting decomposition. We begin with some notation and simple geometric observations. Fix a slab $\square_i$. We color each edge $e \in \nabla_i$ as *blue* if e lies on the projection of a vertical face of $\square_i$ or a long box in $\mathcal{L}_i$, and *red* otherwise (i.e., if e lies on the projection of a vertical face of a short box in $\mathcal{S}_i$ and is contained in an edge of Σ_i). We color a vertex $v \in \nabla_i$ as *red* if both incident edges are red, *blue* if both incident edges are blue, and *magenta* otherwise. We further classify blue vertices $v \in \nabla_i$ as *primitive* if v is the projection of a vertex of a long box in $\mathcal{L}_i$, and *composite* otherwise (i.e., v is the intersection of orthogonal edges of rectangles in $\mathcal{L}_i \cup \{\square_i^{\downarrow}\}$). See Fig. 20c for examples of the four types of vertices; the magenta vertices are depicted as crosses for readability.

For a component $C \in \nabla_i$, let pb_C, cb_C, r_C, and m_C be the number of primitive blue, composite blue, red, and magenta vertices of C. Set $\mathsf{pb}_i := \sum_{C \in \nabla_i} \mathsf{pb}_C$, and define $\mathsf{cb}_i, \mathsf{r}_i, \mathsf{m}_i$ similarly. Let κ_i be the number of vertices of $\partial\mathcal{U}$ inside $\square_i$.

Lemma 5.1 $\mathsf{pb}_i, \mathsf{r}_i = O(n)$ *and* $\mathsf{m}_i \le \kappa_i$.

Proof By definition, a primitive blue vertex is the projection of a z-edge of $\square_i$ or of a box in $\mathcal{L}_i$. Since there are $O(|\mathcal{L}_i| + 1)$ such edges, the bound on pb_i follows.

A magenta vertex is the projection of a z-edge e of $\mathcal{K}_i$ formed by the intersection of a vertical face of a box in $\mathcal{S}_i$ and an orthogonal vertical face of a box in $\mathcal{L}_i$. At least one endpoint of e lies inside $\square_i$. Hence $\mathsf{m}_i \le \kappa_i$.

Finally, since a red vertex of ∇_i is the projection of the intersection segment of a pair of vertical faces of two (not necessarily distinct) boxes of $\mathcal{S}_i$, the number of red vertices is $O(|\mathcal{S}_i|^2) = O(n)$, as claimed. □

Lemma 5.2 *For any component* C *of* ∇_i, $\mathsf{cb}_C \le \mathsf{pb}_C + \mathsf{r}_C + \mathsf{m}_C + 4$.

Proof The boundary of C consists of an outer rectilinear polygonal chain and possibly a set of inner rectilinear polygonal chains. We refer to them as *outer* and *inner* cycles, respectively. For such a cycle ω, let $|\omega|$ be the number of vertices in ω, let pb_ω be the number of primitive blue vertices on ω, and define cb_ω, r_ω, and m_ω similarly. Set $|C| = \sum_{\omega\in\partial C}|\omega|$ (where $\omega \in \partial C$ means that ω is a cycle of ∂C). A planar rectilinear cycle with $\alpha \geq 0$ reflex vertices has $\alpha + 4$ convex vertices. Composite blue vertices of C on an inner cycle are reflex vertices and they are convex on the outer cycle; see Fig. 20c again. Let ω be a component of C. If ω is an inner (resp., outer) cycle then $\mathsf{cb}_\omega \leq \mathsf{pb}_\omega + \mathsf{r}_\omega + \mathsf{m}_\omega - 4$ (resp., $\mathsf{cb}_\omega \leq \mathsf{pb}_\omega + \mathsf{r}_\omega + \mathsf{m}_\omega + 4$). Since C has only one outer cycle, summing over all cycles of ∂C,

$$\mathsf{cb}_C = \sum_{\omega\in\partial C} \mathsf{cb}_\omega \leq 4 + \sum_{\omega\in C}(\mathsf{pb}_\omega + \mathsf{r}_\omega + \mathsf{m}_\omega) = 4 + \mathsf{pb}_C + \mathsf{r}_C + \mathsf{m}_C$$

as desired. □

Lemma 5.3 *The decomposition of* $\bigcup_{i=1}^{r} \nabla_i^{\uparrow}$ *has size* $O(n^{3/2} + \kappa)$.

Proof First, any non-trivial component C of ∇_i has at least one vertex that is not composite blue, i.e., $\mathsf{pb}_C + \mathsf{r}_C + \mathsf{m}_C \geq 1$. By Lemma 5.2, $\mathsf{cb}_C \leq \mathsf{pb}_C + \mathsf{r}_C + \mathsf{m}_C + 4$, and therefore any non-trivial component has $O(\mathsf{pb}_C + \mathsf{r}_C + \mathsf{m}_C)$ vertices. Hence ∇_i has $O(\mathsf{pb}_i + \mathsf{r}_i + \mathsf{m}_i)$ vertices on its non-trivial components. Summing over all slabs and using Lemma 5.1, the number of boxes created from lifting the rectangles in the partitions Π_i is

$$\sum_{i=1}^{r} O(\mathsf{pb}_i + \mathsf{r}_i + \mathsf{m}_i) = \sum_{i=1}^{r} O(n + \kappa_i) = O(n^{3/2} + \kappa),$$

where the last inequality follows because $r = \lceil 8\sqrt{n} \rceil$.

Next, we show that the number of boxes created for the trivial components is $O(\kappa)$. Consider a box $C^{\uparrow}$ created for a trivial component C for $\nabla_j, \ldots, \nabla_k$, $1 < j \leq k < r$. Then C is not a trivial component of ∇_{j-1} or ∇_{k+1}. Let $B_1, B_2, B_3, B_4 \in \mathcal{L}_k$ be the four distinct boxes whose xy-projections support the edges of C. Since the top face of $\square_k$ does not contain any xy-face of an input box, $B_1, \ldots, B_4 \in \mathcal{C}_{k+1}$.

We claim that there exists a vertex w_C of $\mathcal{K}_{k+1}$ in $\Pi_C := C \times J_{k+1}$ (possibly on a vertical face of Π_C). To prove the claim, we sweep an xy-rectangle ρ with $\rho^{\downarrow} = C$ in the z-direction from the bottom face of Π_C to its top face. During the sweep, we maintain $\mathcal{K}_{|\rho} := \rho \cap \mathrm{int}(\mathcal{K})$. The sweep stops as soon as ρ intersects the xy-face f of a box in $\mathcal{C}_{k+1}$ (including the top faces of $B_1, \ldots, B_4$). Hence, during the sweep, $\mathcal{K}_{|\rho}^{\downarrow} = \mathrm{int}(C)$. Since C is not a trivial component of ∇_{k+1}, the sweep stops before reaching the top face of Π_C. Let $K_0 := \mathcal{K}_{|\rho}^{\downarrow}$ when the sweep stops. If f is the top face of one of $B_1, \ldots, B_4$, $K_0^{\downarrow} = \mathrm{int}(C)$ and one of the vertices of K_0 is a vertex of $\mathcal{K}$ as it is the top endpoint of the intersection segments of the vertical faces of two of these boxes. On the other hand, if f is the bottom face of a box in $\mathcal{S}_{k+1}$, then a vertex of $f \cap \Pi_C$ is a vertex of $\mathcal{K}$. Hence, when the sweep stops, the rectangle ρ contains a vertex w_C of $\mathcal{K}$.

The vertex w_C is charged by at most one box in this way, as no other trivial component C' of ∇_k can intersect C in ∇_k (including its boundary). Thus, the total charge to any vertex of $\mathcal{K}$ is at most one, which implies that trivial components lead to $O(\kappa)$ boxes in the partition.

Summing the bounds for trivial and non-trivial components, $\bigcup_{i=1}^{r} \nabla_i^{\uparrow}$ is partitioned into $O(n^{3/2} + \kappa)$ boxes. □

An efficient implementation. We now describe an efficient implementation of the above algorithm for computing the decomposition of $\bigcup_{i=1}^{r} \nabla_i^{\uparrow}$. Note that we cannot afford to compute ∇_i for each slab $\square_i$ since each ∇_i may have $\Omega(\kappa)$ trivial components and computing all of them over all slabs would take $O(\kappa\sqrt{n})$ time. Instead we process $\square_1, \ldots, \square_r$ in order. For each $\square_i$, we compute (i) the non-trivial components of ∇_i, and (ii) the trivial components of ∇_i that are not trivial components of at least one of ∇_{i-1} or ∇_{i+1}. Before processing the slabs, we compute the vertices of $\partial\mathcal{U}$ in $O(n^{3/2}\log n + \kappa)$ time by adapting the Overmars-Yap algorithm [21] in a straightforward manner. We maintain a dynamic data structure Φ that stores a set $\mathcal{A}$ of rectangles and supports the following queries. $\mathcal{A} = \mathcal{C}_i^{\downarrow}$ while processing the slab $\square_i$.

1. CONTAINED?(p): Given a point $p \in \mathbb{R}^2$, return YES if p lies in $\text{int}(\mathcal{U}(\mathcal{A}))$, and NO otherwise.
2. EXPOSED(g): Given an x- or y-segment g, return $g \setminus \text{int}(\mathcal{U}(\mathcal{A})$; g may lie on an edge of a rectangle in $\mathcal{A}$. For each connected component h of $g \setminus \text{int}(\mathcal{U}(\mathcal{A}))$, two edges of the rectangles (other than g) that contain the endpoints of h (and lie on $\partial\mathcal{U}(\mathcal{A})$) are also reported.
3. SHOOT(ρ): Given an axis-parallel ray ρ whose starting point does not lie in $\text{int}(\mathcal{U}(\mathcal{A}))$, return the first edge of a rectangle of $\mathcal{A}$ intersected by ρ if there exists one. Otherwise return NULL.
4. INSERT(R)/DELETE(R): Insert into $\mathcal{A}$ or delete from $\mathcal{A}$ a rectangle R.

The first two queries can be answered using the dynamic data structure described by Overmars and Yap [21], and the third can be answered using the orthogonal ray-shooting dynamic data structure described by Giora and Kaplan [18]. Both of these data structures support insertions and deletions to $\mathcal{A}$. CONTAINED? and SHOOT queries can be answered in $O(\log n)$ time, EXPOSED takes $O(\sqrt{n}\log n + \lambda)$ time, where λ is the output size, and INSERT/DELETE takes $O(\sqrt{n})$ time.

Suppose we have processed $\square_1, \ldots, \square_{i-1}$ and we now process $\square_i$. We color a rectangle of $\mathcal{C}_i^{\downarrow}$ red (resp., blue) if it is the projection of a short (resp., long) box of $\mathcal{C}_i$. Recall that the components of ∇_i may have holes and are represented by an outer cycle and possibly a set of inner cycles. We call a cycle (of a component) of ∇_i *trivial* if it is composed of only composite blue vertices and *non-trivial* otherwise; i.e., they have at least one primitive blue, red, or magenta vertex. A non-trivial component of ∇_i may have a trivial outer cycle but then it has at least one inner cycle, and trivial components are components with a trivial outer cycle and no inner cycles. As argued in the proof of Lemma 5.3, if a trivial component C of ∇_i is not a trivial component of ∇_{i-1} (resp., ∇_{i+1}), then C contains the projection of a vertex of $\mathcal{U} \cap \square_{i-1}$ (resp., $\mathcal{U} \cap \square_{i+1}$). Thus, our goal is to efficiently report all cycles ω of ∇_i that satisfy at least one of the following four conditions:

(i) ω contains a red edge,
(ii) ω contains a primitive blue vertex,
(iii) ω is an outer cycle and $\text{int}(\omega)$ contains a primitive blue, a red, or a magenta vertex, or
(iv) ω is an outer cycle and $\text{int}(\omega)$ contains a projection of a vertex of $\mathcal{U} \cap (\square_{i-1} \cup \square_{i+1})$.

where $\text{int}(\omega)$ is the bounded region lying inside ω. Let Ω_i denote the set of boundary cycles of ∇_i that satisfy at least one of the conditions (i)–(iv). We note that a cycle may satisfy more than one condition. In particular, non-trivial cycles satisfy at least one of the conditions (i) or (ii), and trivial cycles that we want to compute satisfy at least one of (iii) and (iv). Before computing the desired cycles of types (i)–(iv), we perform the following three steps:

I. We delete the rectangles of $\mathcal{C}_{i-1}^{\downarrow} \setminus \mathcal{C}_i^{\downarrow}$ from Φ and insert the rectangles of $\mathcal{C}_i^{\downarrow} \setminus \mathcal{C}_{i-1}^{\downarrow}$ into Φ, so that Φ now stores $\mathcal{C}_i^{\downarrow}$.
II. For each edge γ of a rectangle in $\mathcal{S}_i^{\downarrow}$, we perform the query $\textsc{Exposed}(\gamma)$, which returns the connected components of $\gamma \setminus \text{int}(\mathcal{U}(\mathcal{C}_i^{\downarrow}))$, i.e.the set of red edges of ∇_i that lie on γ. Note that for each red edge e, the procedure returns the two other edges of $\mathcal{U}(\mathcal{C}_i)$ incident on the endpoints of e. By repeating this step for all edges of rectangles in $\mathcal{S}_i^{\downarrow}$, we have the set $\mathcal{R}$ of red edges of ∇_i at our disposal.
III. For each vertex p of a rectangle in $\mathcal{L}_i^{\downarrow}$, we perform $\textsc{Contained?}(p)$ query to determine whether p is a primitive blue vertex of ∇_i. By repeating this step for all rectangles of $\mathcal{L}_i^{\downarrow}$, we compute the set P of all primitive blue vertices of ∇_i.

We construct each cycle ω of Ω_i by first computing a "seed" vertex of ω, and then by tracing it in clockwise direction. At each step, we are at a vertex u of ω and we have the segment γ of a rectangle in $\mathcal{C}_i^{\downarrow}$ that contains the edge e_u of ω next to u in clockwise direction. If γ lies on a red rectangle then we already know the other endpoint of e_u since we have computed all red edges of ∇_i. Otherwise (if γ lies on a blue rectangle) we perform a $\textsc{Shoot}(\gamma_u)$ query, where γ_u is the ray emanating from u in the desired direction along γ, and obtain the next vertex of ω in clockwise direction. We repeat this process of tracing ω until we get back to the seed vertex. To ensure that we do not trace the same cycle more than once, we store all the vertices of ∇_i that we have traced so far in a red-black tree [12]. When we get a seed vertex v, we first check in the red-black tree whether v already has been traced. The total time spent in tracing ω is $O(|\omega| \log n)$.

It thus suffices to describe how we compute seeds of cycles in Ω_i. We use the endpoints of red edges in $\mathcal{R}$ as the seeds of type (i) cycles and the set P for type (ii) cycles. We compute the seeds of type (iii) cycles, as follows. First, recall that the faces of $\square_i$ do not intersect any boxes of $\mathcal{C}$, so $\partial\square_i^{\downarrow}$ is an outer cycle of a non-trivial component of ∇_i. Hence, $\partial\square_i^{\downarrow}$ is a type (iii) cycle and we trace it. Let V be the set of vertices on type (i) and (ii) cycles. Let ω be a type (i) or (ii) inner cycle, which we already have computed, let v be a vertex of ω with the maximum y-coordinate, and let η_v be the ray emanating from v in the $(+y)$-direction. We perform $\textsc{Shoot}(\eta_v)$. If the query returns $\textsc{Null}$, then the outer cycle of the same non-trivial component as ω is

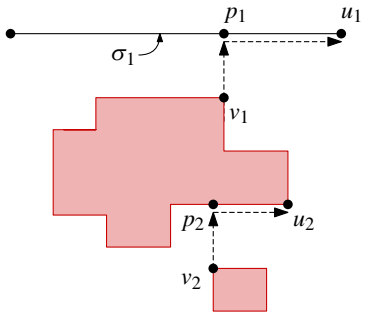

Fig. 22 Searching for type (iii) cycles with SHOOT queries. v_1 and v_2 lie on type (i) inner cycles, u_1 lies on a type (iii) cycle, and u_2 lies on the same cycle as v_1

$\partial\square_i^{\downarrow}$, which we already traced, and we are done with ω. Otherwise, the query returns a segment σ of a rectangle in $\mathcal{C}_i^{\downarrow}$ hit by η_v. The intersection point $p := \eta_v \cap \sigma$ lies on the boundary of the same connected component of ∇_i that contains ω. By performing another SHOOT query along the segment σ, we find an endpoint u of the edge of ∇_i that contains $\eta_v \cap \sigma$. See Fig. 22. Next, we check whether $u \in V$. If $u \notin V$ then u lies on the outer cycle of the same component of ∇_i as ω and hence is a seed of a type (iii) cycle. For at least one of the inner cycles of a non-trivial component (for which $\partial\square_i^{\downarrow}$ is not its outer cycle), this ray-shooting procedure will reach a vertex on its outer cycle. Hence, repeating this for all type (i) and (ii) inner cycles computes seeds of all type (iii) cycles.

Finally, we compute the seeds of type (iv) cycles. Let X_i be the set of the xy-projections of vertices of $\mathcal{U}$ that lie in $\square_{i-1}$ or $\square_{i+1}$. For each point $p \in X_i$, we do the following. First by performing a CONTAINED?(p) query, we check whether $p \in \text{int}(\mathcal{U}(\mathcal{C}_i^{\downarrow}))$. If the answer is NO, by performing two SHOOT queries as above, we compute a vertex u of the component of ∇_i that contains p. Next, we check whether u has been traced. If not then u is a seed of a type (iv) cycle that has not been computed so far. From the proof of Lemma 5.3, repeating this step for all points in X_i computes seeds of all type (iv) cycles.

Let χ_i be the total number of vertices in type (i)–(iv) cycles. Then computing the seeds and tracing the cycles takes $O((\chi_i + |X_i|)\log n)$ time. We spend $O(|\mathcal{S}_i|\sqrt{n}\log n + |\mathcal{C}_i|\log n) = O(n\log n)$ time in steps I–III. Hence, the total time spent in processing $\square_i$ is $O((n + \chi_i + |X_i|)\log n)$. We have $\sum_{i=1}^{r}|X_i| = 2\kappa$, and by Lemma 5.3, $\sum_{i=1}^{r}|\chi_i| = O(n^{3/2} + \kappa)$. Therefore the total time spent over all slabs is $O((n^{3/2} + \kappa)\log n)$.

5.3 Partitioning Ψ_i

For each $i \leq r$, we partition Ψ_i independently. Roughly speaking, we compute a (3D) vertical composition of Ψ_i. However, because many edges of Ψ_i may be coplanar and we do not wish to perform a symbolic perturbation to bring them into general

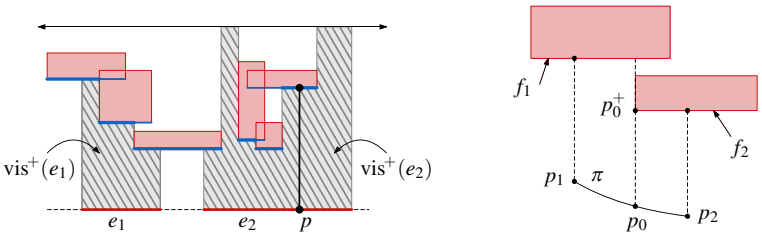

Fig. 23 (left) A 2D view of curtains $\mathsf{vis}^+(e_1), \mathsf{vis}^+(e_2)$ (hatched) erected upwards from convex edges e_1, e_2 which lie on a common edge e' of a box in $\mathcal{C}$. The xy-face of the slab $\square_i$ is shown at the top, which is partially visible from e_2. The boxes of $\mathcal{S}_i$ above e' are red. The segment $\mathsf{vis}^+(p)$ for a point p on e_2 is shown. (right) A 2D view of the proof of Lemma 5.4. The dashed vertical segments are $\mathsf{vis}^+(p_j)$ for $j \in \{0, 1, 2\}$

position, we cannot use a generic vertical decomposition construction (e.g., see [9, 25]). Therefore we need to be more careful. The construction consists of two stages.

For any point $p \in \mathcal{K}_i$, let $\mathsf{vis}^-(p)$ (resp., $\mathsf{vis}^+(p)$) be the z-segment composed of the points of $\mathcal{K}_i$ that are visible from p within $\mathcal{K}_i$ in the $(-z)$-direction (resp., $(+z)$-direction). For any x-segment (resp., y-segment) s in $\mathcal{K}_i$, $\mathsf{vis}^-(s) := \bigcup_{p \in s} \mathsf{vis}^-(p)$ and $\mathsf{vis}^+(s) := \bigcup_{p \in s} \mathsf{vis}^+(p)$ are x-monotone (resp., y-monotone) rectilinear polygons in a plane parallel to the xz-plane (resp., yz-plane).

Let $\gamma_1, \ldots, \gamma_t$ be the horizontal (x- or y-)edges of boxes of $\mathcal{S}_i$ that lie in $\square_i$. We process them in an arbitrary order. Processing an edge γ_j is done as follows. Suppose γ_j lies on the top (resp., bottom) face of a box. For each connected component e of $\gamma_j \cap \partial\mathcal{K}$, which appears as a reflex edge of Ψ_i, we draw the rectilinear polygon $\mathsf{vis}^-(e)$ (resp., $\mathsf{vis}^+(e)$), which we refer to as a *curtain* erected from e. See Fig. 23(left). We describe below how this step is implemented efficiently. After we have processed all γ_j's, we obtain a decomposition of Ψ_i, which we refer to as a *curtain decomposition* $\Psi_i^{||}$ of Ψ_i.

We prove below in Lemma 5.4 that each cell Δ of $\Psi_i^{||}$ is a prism of the form $\Delta = G \times \delta$, where $G \subseteq \mathbb{R}^2$ is a rectilinear polygon (possibly with holes), $\delta \subseteq J_i$ is its z-extent; see Fig. 24. In the second stage, for each prism $\Delta \in \Psi_i^{||}$ of the form $\Delta = G \times \delta$, we partition G into rectangles $\rho_1, \ldots, \rho_q$ using the 2D vertical decomposition, and then lift each rectangle ρ_j into the box $\rho_j \times \delta$. These boxes partition Δ. By repeating this step for all prisms of $\Psi_i^{||}$, we obtain a partition of Ψ_i into boxes. We repeat this construction for all slabs. This decomposition of the Φ_i's along with the decomposition of $\bigcup_{i=1}^r \nabla_i^{\uparrow}$ gives the desired partition of $\mathcal{K}$ into boxes.

Lemma 5.4 *Each cell Δ of the curtain decomposition $\Psi_i^{||}$ of Ψ_i is a prism of the form $\Delta = G \times \delta$, where $G \subseteq \mathbb{R}^2$ is a rectilinear polygon (possibly with holes), $\delta \subseteq J_i$ is its z-extent, at least one of its top or bottom faces lies on a xy-face of a box of $\mathcal{S}_i$ and the other on a xy-face of a box of $\mathcal{S}_i \cup \{\square_i\}$, and a vertical wall of Δ is composed of at most one face of Ψ_i and at most two curtains.*

Proof Let Δ be a cell of $\Psi_i^{||}$. We claim that the top (resp., bottom) endpoint of $\mathsf{vis}^+(p)$ (resp., $\mathsf{vis}^-(p)$) for all points $p \in \mathrm{int}(\Delta)$ lies on the same xy-face of a box in $\mathcal{S}_i \cup \{\square_i\}$.

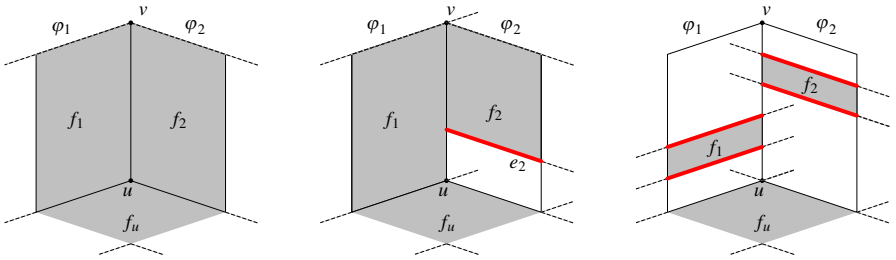

Fig. 24 Examples of edge $e = uv$ of a prism R of $\Psi_i^{||}$ incident to vertical faces φ_1, φ_2, where u lies on the top xy-face f_u of a short box. Shaded (resp., transparent) portions of φ_1, φ_2 lie on $\partial\mathcal{U}$ (resp., on curtains). (left) φ_1, φ_2 lie on faces f_1, f_2 of $\partial\mathcal{U}$, respectively. (middle) φ_1 is contained in face f_1 of $\partial\mathcal{U}$ and φ_2 intersects the curtain $\mathsf{vis}^-(e_2)$ erected downwards from edge e_2 of face f_2 of $\partial\mathcal{U}$. (right) φ_1 and φ_2 intersect both curtains erected from the horizontal edges of faces f_1 and f_2 of $\partial\mathcal{U}$, respectively

Suppose to the contrary there are two points $p_1, p_2 \in \text{int}(\Delta)$ such that the top endpoint of $\mathsf{vis}^+(p_j)$ lies on an xy-face f_j, $j = 1, 2$, and $f_1 \neq f_2$. Among all such pairs, we choose p_1, p_2 for which there is a path $\pi \subset \text{int}(\Delta)$ from p_1 to p_2 such that the top endpoint of $\mathsf{vis}^+(p)$ for all $p \in \pi$ initially lies on f_1 then switches to f_2 and remains on f_2. Without loss of generality, assume that $z(f_1) > z(f_2)$. Let $p_0 \in \pi$ be the first point such that the vertical segment $\mathsf{vis}^+(p_0)$ intersects f_2 (and f_1). Let p_0^+ be the point on $\mathsf{vis}^+(p_0) \cap f_2$. See Fig. 23(right). Then p_0^+ lies on an edge of f_2. Since $z(f_1) > z(f_2)$, this edge is a reflex edge of Ψ_i, which is an edge of the bottom face of a box in $\mathcal{S}_i$. But the algorithm erected the curtain $\mathsf{vis}^-(e)$ on e and p_0 lies on $\mathsf{vis}^-(e)$, contradicting the assumption that $\pi \subset \text{int}(\Delta)$.

Hence, the claim is true and Δ is a prism whose top and bottom faces are rectilinear polygons. Since $\Delta^\downarrow \subset \mathcal{U}(\mathcal{S}_i^\downarrow)$, both the top and bottom faces of Δ cannot lie on $\partial\square_i$ and at least one of them lies on a xy-face of a box of $\mathcal{S}_i$. The general-position assumption of $\mathcal{C}$ implies that no two vertical faces of boxes in $\mathcal{C}$ are coplanar. Therefore each vertical wall of Δ consists of at most one vertical face of a box in $\mathcal{S}_i \cup \{\square_i\}$ and at most two curtains — one erected from the top edge of a vertical face of a box and another from the bottom edge of that face; see Fig. 24. □

The next two lemmas bound the size of the decompositions of the Ψ_i's over all slabs.

Lemma 5.5 *The rectilinear prisms in the curtain decomposition $\Psi_i^{||}$ of Ψ_i have a total of $O(n + \kappa_i)$ vertices.*

Proof We call a vertex of a prism of $\Psi_i^{||}$ *pure* if it is a vertex of Ψ_i, otherwise we call it *mixed*. By Lemma 5.1, there are $O(n + \kappa_i)$ pure vertices, so it suffices to bound the number of mixed vertices. Each mixed vertex lies on a curtain. We bound the number of vertices lying on the curtains erected from an edge γ_j of a box of $\mathcal{S}_i$. Without loss of generality, assume that γ_j lies on the top face of the box; the other case is symmetric. Let $T_j^{||}$ be the semi-unbounded strip obtained by drawing a vertical ray in the $(+z)$-direction from each point of γ_j. That is, $T_j^{||} = \gamma_j^\downarrow \times [z(\gamma_j), \infty)$. For each bottom

xy-face f_t of a box in $\mathcal{S}_i \cup \{\square_{i+1}\}$ that lies in $\square_i$ and intersects $T_j^{||}$, let $g_t := f_t \cap T_j^{||}$; g_t is a segment parallel to γ_j. Let G_j be the resulting set of horizontal segments (blue segments in Fig. 23(left)), and let Λ_j be the lower envelope of G_j (thick blue segments in Fig. 23(left)), i.e., the portion of segments of G_j that are visible from γ_j. Since the segments in G_j are parallel, Λ_j has $O(|\mathcal{S}_i|) = O(\sqrt{n})$ breakpoints. Now consider the curtain erected from an edge $e = pq$ of Ψ_i that lies on γ_j. Each vertex of $\mathsf{vis}^+(e)$ is either a breakpoint of Λ_j or an endpoint of $\mathsf{vis}^+(p)$, $\mathsf{vis}^+(q)$. Note that p, q are vertices of Ψ_i. Let χ_j be the number of vertices of Ψ_i that lie on γ_j. Then the curtains erected from the edges of Ψ_i lying on γ_j have a total of $O(\sqrt{n} + \chi_j)$ vertices. Since $\sum_j \chi_j = O(\kappa_i)$, summing over all γ_j's, the total number of mixed vertices of prisms in $\Psi_i^{||}$ is $O(|\mathcal{S}_i|\sqrt{n} + \kappa_i) = O(n + \kappa_i)$. □

Lemma 5.6 *The decomposition of* $\bigcup_{i=1}^r \Phi_i$ *has size* $O(n^{3/2} + \kappa)$.

Proof Fix a slab $\square_i$. The second stage partitions each rectilinear prism R in the curtain decomposition of $\Psi_i^{||}$ into $O(|R|)$ boxes, where $|R|$ is the number of vertices of R. By Lemma 5.5, the total number of vertices of the prisms, and hence the number of resulting boxes, is $O(n + \kappa_i)$. Summing over all slabs, the overall size of the decomposition is $\sum_{i=1}^r O(n + \kappa_i) = O(n^{3/2} + \kappa)$, using the fact that $r = \lceil 8\sqrt{n} \rceil$. □

An efficient implementation. We now describe how we construct the vertical decomposition of Ψ_i efficiently. We only consider the construction of the curtain decomposition $\Psi_i^{||}$ of Ψ_i, as the second stage is straightforward. By Lemma 5.4, the top and bottom faces of the prisms in $\Psi_i^{||}$ are rectilinear polygons (possibly with holes) that lie on the xy-faces of the boxes in $\mathcal{S}_i \cup \{\square_i\}$. Let Π_t be the subdivision of $f_t \cap \Psi_i$ induced by the prisms of $\Psi_i^{||}$ for each xy-face f_t of the boxes in $\mathcal{S}_i \cup \{\square_i\}$. It suffices to first compute the polygons of the Π_t's and then identify which pairs of these polygons are the top and bottom faces of the same prisms, as follows.

Π_t is induced by a set E_t of curtain edges that lie on f_t, so Π_t is the arrangement of E_t. We compute E_t as follows. Let $\gamma_1, \ldots, \gamma_s$ be the horizontal edges of boxes of $\mathcal{S}_i$ that lie in $\square_i$, and fix one of these edges γ_j. Without loss of generality, assume γ_j is on the top face of a box in $\mathcal{S}_i$. As described in the proof of Lemma 5.5, we compute the associated lower envelope Λ_j in $O((|\mathcal{S}_i| \log |\mathcal{S}_i|) = O(\sqrt{n} \log n)$ time [25]. By merging the breakpoints of Λ_j with the vertices of Ψ_i on γ_j, we can compute in $O((\sqrt{n} + \chi_j) \log n)$ time the curtains $\mathsf{vis}^+(e)$ drawn from the edges e of Ψ_i lying on γ_j. If a top edge g of $\mathsf{vis}^+(e)$ lies on a face f_t, we add g to E_t. After repeating this for all γ_j's, we have the set E_t for all faces f_t lying in $\square_i$. Next, we compute Π_t, the arrangement of E_t, in $O(|\Pi_t| \log n)$ time by a sweep-line algorithm. Summing over all faces, the total time in computing the Π_t's is $O((n + \kappa_i) \log n)$.

Finally, we report the prisms. Let $\mathcal{G}^+$ (resp., $\mathcal{G}^-$) be the set of polygons that lie on bottom (resp., top) faces of boxes in $\mathcal{S}_i \cup \{\square_{i+1}\}$ (resp., $\mathcal{S}_i \cup \{\square_{i-1}\}$), and let $\mathcal{V}^-$ be the multiset of vertices of polygons in $\mathcal{G}^-$, sorted in lexicographic order. Fix a polygon $G_t \in \mathcal{G}^+$. The goal is to find the polygon $G_s \in \mathcal{G}^-$ that is the bottom face of the prism whose top face is G_t. Note that $G_s^{\downarrow} = G_t^{\downarrow}$ and $z(G_s) < z(G_t)$. We first pick a vertex v_t of G_t and compute the set $\mathcal{V}_t \subset \mathcal{V}^-$ of vertices that are below v_t and

whose projections are $v_t^{\downarrow}$. Let $\mathcal{G}_t \subset \mathcal{G}^-$ be the set of polygons with vertices in $\mathcal{V}_t$. Then G_s is the highest polygon in $\mathcal{G}_t$ such that the edges incident to $v_t^{\downarrow}$ in $G_s^{\downarrow}$ and $G_t^{\downarrow}$ are identical and $\text{int}(G_s^{\downarrow})$ and $\text{int}(G_t^{\downarrow})$ lie on the same sides of those edges. Hence, we can identify G_s in $O(|\mathcal{V}_t|)$ time, and then report $G_t \times [z(G_s), z(G_t)]$ as a prism. The boxes of $\mathcal{C}$ are in general position, so $|\mathcal{V}_t| = O(1)$ and hence $\mathcal{V}_t$ and G_s takes $O(\log n)$ time. We repeat this process for all polygons in $\mathcal{G}^+$, reporting all prisms of $\Psi_i^{||}$. By Corollary 5.6, $|\Psi_i^{||}| = O(n + \kappa_i)$. Therefore the entire algorithm to compute $\Psi_i^{||}$ takes $O((n + \kappa_i) \log n)$ time.

Summing this bound over all slabs and adding the time spent in decomposing $\bigcup_{i=1}^{r} \nabla_i^{\uparrow}$, we construct a partition of $\mathcal{K}$ into $O(n^{3/2} + \kappa)$ axis-aligned boxes in $O((n^{3/2} + \kappa) \log n)$ time, thereby proving Theorem 1.7.

6 Conclusion

We have described algorithms to compute a partition of the complement of the union of axis-aligned 3D cubes (or fat boxes) into boxes where the runtimes are near-linear in the input and output size. In particular, let n be the number of input cubes, let $\mathcal{U}$ be the union of the cubes, and let κ be the number of vertices on $\partial\mathcal{U}$. If the input cubes have different sizes then a decomposition of size $O(\kappa \log^4 n)$ (resp., $O(\sigma \log^4 n + \kappa \log^2 n)$), where $\sigma \leq \min\{n, \kappa\}$ is the number of input cubes that appear on $\partial\mathcal{U}$, can be computed in $t(n, \kappa) + O(n \log^2 n + \kappa \log^6 n)$ (resp., $t(n, \kappa) + O(n \log^2 n + \sigma \log^8 n + \kappa \log^6 n)$) time, where $t(n, \kappa)$ is the time to compute the faces of $\partial\mathcal{U}$. If all cubes have the same size, then a decompositon of size $O(\kappa \log \kappa)$ can be computed in $t(n)$ time, where $t(n) = \Omega(n \log n)$ is the time to compute the faces of $\partial\mathcal{U}$. Currently, the best runtimes are $t(n, \kappa) = O(n \log^3 n + \kappa \log n)$ for arbitrary cubes and $t(n) = O(n \log^2 n)$ for congruent cubes [1]. We also described an algorithm to compute a partition of the complement of the union of n arbitrary axis-aligned boxes in $\mathbb{R}^3$ into $O(n^{3/2} + \kappa)$ boxes, where κ is the number of vertices on their union, which has worst-case optimal dependence on n and κ.

We conclude by mentioning two natural 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.

Appendix A: Balanced-Box Decomposition (BBD) Trees

In this appendix we prove Lemma 2.1, but we first 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 [4].

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 [4], $\square^I$ 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 $\boxdot_u = \text{cl}(\square_u^O \setminus \square_u^I)$. 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 $\text{int}(\square_u^O)$, 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. □

References

1. Agarwal, P.K., Steiger, A.: An output-sensitive algorithm for computing the union of cubes and fat boxes in 3d. 48th Inter. Colloq. on Auto., Lang., and Prog. 10, 1–20 (2021)
2. Agarwal, P.K., Grove, E.F., Murali, T.M., Vitter, J.S.: Binary space partitions for fat rectangles. SIAM J. Comput. **29**, 1422–1448 (2000)
3. Agarwal, P.K., Kaplan, H., Sharir, M.: Union of hypercubes and 3d Minkowski sums with random sizes. Discrete Comput. Geom. **65**, 1136–1165 (2021)
4. Arya, S., Mount, D.M., Netanyahu, N.S., Silverman, R., Wu, A.Y.: An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. J. ACM **45**, 891–923 (1998)
5. Bern, M., Eppstein, D.: Mesh generation and optimal triangulation. In: Computing in Euclidean Geometry, pp. 23–90. World Scientific, Singapore (1992)
6. Boissonnat, J.D., Sharir, M., Tagansky, B., Yvinec, M.: Voronoi diagrams in higher dimensions under certain polyhedral distance functions. Discrete Comput. Geom. **19**, 485–519 (1998)
7. Chazelle, B.: Convex partitions of polyhedra: a lower bound and worst-case optimal algorithm. SIAM J. Comput. **13**, 488–508 (1984)
8. Chazelle, B., Palios, L.: Triangulating a nonconvex polytope. Discrete Comput. Geom. **5**, 505–526 (1990)
9. Chazelle, B., Edelsbrunner, H., Guibas, L.J., Sharir, M.: A singly-exponential stratification scheme for real semi-algebraic varieties and its applications. Theor. Comput. Sci. **84**, 77–105 (1991)
10. Cheng, S.-W., Dey, T.K., Shewchuk, J.: Delaunay Mesh Generation. Chapman & Hall/CRC, New York (2012)
11. Collins, G.: Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In: Second GI Conference on Automata Theory and Formal Languages 33, pp. 134–183 (1975)
12. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to Algorithms, 4th edn., pp. 331–332. MIT Press, Cambridge (2022)
13. Dumitrescu, A., Mitchell, J.S.B., Sharir, M.: Binary space partitions for axis-parallel segments, rectangles, and hyperrectangles. Discrete Comput. Geom. **31**, 207–227 (2004)
14. Edelsbrunner, H., Mücke, E.P.: Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. ACM Trans. Graph. **9**(9), 66–104 (1990)
15. Foley, J.D., Dam, A., Feiner, S., Hughes, J.F.: Computer Graphics—Principles and Practice, 2nd edn. Addison-Wesley, Boston (1992)
16. Frey, P.J., George, P.-L.: Mesh Generation: Application to Finite Elements. Wiley, Hoboken (2007)

17. Fuchs, H., Kedem, Z., Naylor, B.: On visible surface generation by a priori tree structures. Proc. 7th Annu. Conf. Comp. Graph. Interact. Tech., 124–133 (1980)
18. Giora, Y., Kaplan, H.: Optimal dynamic vertical ray shooting in rectilinear planar subdivisions. ACM Trans. Algorithm (5) **3**(28), 1–28 (2009)
19. Laidla, D.H., Trumbore, W.B., Hughes, J.F.: Constructive solid geometry for polyhedral objects. Proc. 13th Annu. Conf. Comp. Graph. Interact. Tech., 161–170 (1986)
20. LaValle, S.M.: Planning Algorithms. Cambridge University Press, Cambridge (2006)
21. Overmars, M.H., Yap, C.-K.: New upper bounds in Klee's measure problem. SIAM J. Comput. **20**(6), 1035–1045 (1991)
22. Paterson, M.S., Yao, F.F.: Efficient binary space partitions for hidden-surface removal and solid modeling. Discrete Comput. Geom. **5**, 485–503 (1990)
23. Paterson, M., Yao, F.F.: Optimal binary space partitions for orthogonal objects. J. Algorithms **13**, 99–113 (1992)
24. Schwartz, J.T., Sharir, M.: On the "piano movers" problem ii: general techniques for computing topological properties of real algebraic manifolds. Adv. Appl. Math. **4**, 298–351 (1983)
25. Sharir, M., Agarwal, P.K.: Davenport-Schinzel Sequences and Their Geometric Applications. Cambridge University Press, Cambridge (1995)
26. Tóth, C.D.: Binary space partitions for axis-aligned fat rectangles. SIAM J. Comput. **38**, 429–447 (2008)