An Improved Algorithm for Computing the Volume of the Union of Cubes*

Pankaj K. Agarwal
Department of Computer Science
Duke University

Abstract

Let $\mathcal{C}$ be a set of n axis-aligned cubes in $\mathbb{R}^3$, and let $\mathcal{U}(\mathcal{C})$ denote the union of $\mathcal{C}$. We present an algorithm that computes the volume of $\mathcal{U}(\mathcal{C})$ in time $O(n\,\mathrm{polylog}(n))$. The previously best known algorithm takes $O(n^{4/3}\log^2 n)$ time.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Computations on discrete structures, geometrical problems and computations*

General Terms

Algorithms, Theory.

Keywords

Geometric data structures, Union of cubes.

1. INTRODUCTION

Let $\mathcal{C}$ be a set of n axis-aligned cubes in $\mathbb{R}^3$, and let $\mathcal{U}(\mathcal{C})$ denote the union of $\mathcal{C}$. Computing the volume of $\mathcal{U}(\mathcal{C})$ efficiently is related to the well-known *Klee's measure problem*. In 1977 Victor Klee [13] had presented an $O(n\log n)$ time algorithm for computing the union of n intervals in $\mathbb{R}^1$ and had asked whether his algorithm was optimal. An $\Omega(n\log n)$ lower bound was proved by Fredmen and Weide [11]. Bentley [4] studied the two-dimensional version of Klee's measure problem. When extended to computing the volume of the union of n d-dimensional axis-aligned boxes, the running time of Bentley's algorithm is $O(n^{d-1}\log n)$. Later, van Leeuwen and Wood [12] improved the running time to $O(n^2)$ for $d = 3$. The problem lay dormant for a while until Overmars and Yap presented an algorithm with $O(n^{d/2}\log n)$ running time [15]. Recently, Chan [7] slightly improved the running time of their algorithm to $n^{d/2}2^{O(\log^* n)}$; see also [8, 9] for some other related results. The lower bounds proved in [7, 16] suggest that improving the running time to $O(n^{d/2-\varepsilon})$, for any constant $\varepsilon > 0$, might be hard. See [1] for a brief history of the problem. Boissonnat *et al.* [5] proved that the combinatorial complexity of n axis-aligned cubes in $\mathbb{R}^d$ is $\Theta(n^{\lceil d/2\rceil})$, and it is $\Theta(n^{\lfloor d/2\rfloor})$ if all the cubes have the same size. Note that this bound is considerably better than the $\Theta(n^d)$ worst-case bound on the union of n axis-aligned boxes in $\mathbb{R}^d$. This suggests that it might be easier to compute the volume of the union of n cubes. Indeed, the volume of the union of n axis-aligned unit cubes in $\mathbb{R}^3$ can be computed in $O(n\log n)$ time, by computing their union explicitly (which has linear complexity), but this will not lead to an efficient algorithm for cubes of different sizes. Agarwal *et al.* [2] presented an $O(n^{4/3}\log^2 n)$ algorithm for computing the volume of the union of n axis-aligned cubes in $\mathbb{R}^3$, by exploiting the special structure of cubes. Recently Bringmann [6] extended the result to higher dimensions and proposed an $O(n^{(d+2)/3})$ algorithm for computing the volume of the union of n hypercubes in $\mathbb{R}^d$. In this paper we significantly improve the result in [2].

THEOREM 1.1. *The volume of the union of a set of n axis-aligned cubes in $\mathbb{R}^3$ can be computed in time $O(n\log^4 n)$.*

Although there are some similarities between the new algorithm and [2], several new ideas and data structures are needed to improve the running time. The algorithm asserted in Theorem 1.1 is presented in the following four sections. Section 2 gives a high-level description of the algorithm, and describes a data structure for maintaining the area of the union of a set of squares in $\mathbb{R}^2$ under certain assumptions on the sequence of updates. Section 3 describes the update procedure for the data structure, Section 4 presents auxiliary procedures needed by our algorithm in Section 3, and Section 5 presents secondary data structures needed in Section 4.

2. THE GLOBAL STRUCTURE

We assume that the cubes of $\mathcal{C}$ are in *general position*. In particular, we assume that no plane support facets of two distinct cubes in $\mathcal{C}$. Let $z_1 < \cdots < z_{2n}$ be the (distinct) z-coordinates of the vertices of cubes in $\mathcal{C}$, sorted in increasing order. We sweep a horizontal plane Π in the $(+z)$-direction from $-\infty$ to $+\infty$, stopping at each z_i. Let $\Pi(t)$ denote the horizontal plane at $z = t$. For each $1 \le i < 2n$, the cross-section $\mathcal{U}(\mathcal{C}) \cap \Pi(z)$ is the same for all $z \in (z_i, z_{i+1})$. Let α_i denote the area of this cross-section. Then $\mathrm{Vol}\,\mathcal{U}(\mathcal{C}) = \sum_{i=1}^{2n-1} \alpha_i(z_{i+1} - z_i)$. We thus need to maintain α_i as we sweep the horizontal plane. The intersection of $\Pi(z)$ with $\mathcal{U}(\mathcal{C})$ is the union of a set $\mathbb{S}$ of squares that changes dynamically—a square is added to the intersection when Π sweeps through the bottom facet of its corresponding cube, and is removed from the intersection when Π sweeps through the top facet of its cube. Let

*Supported by NSF under grants CNS-05-40347, CCF-06 -35000, IIS-07-13498, and CCF-09-40671, by ARO grants W911NF-07-1-0376 and W911NF-08-1-0452, by an NIH grant 1P50-GM-08183-01, by a DOE grant OEG-P200A070505, and by a grant from the U.S.–Israel Binational Science Foundation.

SCG'10, June 13–16, 2010, Snowbird, Utah, USA.

S_1 and S_2 be two squares that are inserted into $\mathbb{S}$. If S_1 is bigger than S_2 and S_1 is inserted after S_2, then S_1 is deleted after S_2 has been deleted (because S_1 and S_2 are intersections of cubes with the sweep line). We capture this observation by defining the notion of a *size preserving* update sequence:

> *A sequence of insertions and deletions is called size preserving if a square S is deleted only after all squares that were smaller than S and inserted before S have been deleted.*

We describe a dynamic data structure that maintains, in $O(\log^4 n)$ amortized time (see Lemma 3.2), the area of the union of $\mathbb{S}$, denoted by $\mu(\mathbb{S})$, for a size-preserving update sequence. This implies Theorem 1.1.

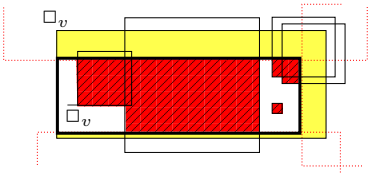

Figure 1. Long (dotted lines) and short (solid lines) squares in $\square_v$, $\bar{\square}_v$ (thick rectangle). Lightly shaded region is $\mathcal{U}(\mathbb{L}_v) \cap \square_v$, and the hatched region is $\mathcal{U}(\mathbb{G}_v) \cap \bar{\square}_v$.

Here is a rough sketch of the overall data structure for maintaining $\mu(\mathbb{S})$. $\mathbb{S}$ is stored in a partition tree $\mathbb{T}$, which is a variant of a kd-tree. Each node v of $\mathbb{T}$ is associated with a rectangle $\square_v$. The squares of $\mathbb{S}$ intersecting $\square_v$ are partitioned into two sets $\mathbb{L}_v$ and $\mathbb{G}_v$, the sets of *long* and *short* squares, respectively. A square is long if at most one of its edges intersects $\square_v$, and short if at least two edges intersects $\square_v$. Let $\bar{\square}_v = \square_v \setminus \mathcal{U}(\mathbb{L}_v)$, which is a rectangle; see Figure 1. The basic idea is to maintain $\mathbb{L}_v$ and $\bar{\square}_v$ at v, to store $\mathbb{G}_v$ recursively at the descendents of v, and to compute $\tilde{\alpha}(v) = \operatorname{Area}\mathcal{U}(\mathbb{G}_v) \cap \bar{\square}_v$ from the information stored at the children of v. There are, however, several challenges in accomplishing this: First, a square may appear short at $\Omega(\sqrt{n})$ nodes in a kd-tree, so how do we ensure that a square appears as a short square only at $\operatorname{polylog}(n)$ nodes of $\mathbb{T}$? Exploiting that $\mathbb{S}$ is a set of squares, we describe a variant of kd-tree that guarantees this property. Second, and even more challenging issue, is to maintain $\bar{\square}_v$ and $\tilde{\alpha}(v)$ efficiently. A square can be long at too many nodes of $\mathbb{T}$, so a long square is stored only at the highest nodes at which it is long (analogous to how intervals are stored in a segment tree). But then $\bar{\square}_v$ cannot be not maintained explicitly at v, making the maintenance of $\tilde{\alpha}(v)$ hard. We circumvent this problem by maintaining an approximation $\boxplus_v$ of $\bar{\square}_v$, maintaining $\alpha(v) = \operatorname{Area}\mathcal{U}(\mathbb{G}_v) \cap \boxplus_v$, pushing certain information related to $\mathbb{L}_v$ to the descendents of v in a lazy manner, and maintaining secondary structures to compute $\tilde{\alpha}(v)$ from $\alpha(v)$ quickly whenever needed. A clever charging scheme is used to bound the time spent in maintaining this information. The charging scheme exploits, in a crucial though subtle way, the special (obvious) property that the *life-time* of a square of side length h (regarding the z-direction as "time") is also h time units.

For a horizontal (resp. vertical) edge e lying on the line $y = a$ (resp. $x = a$), we refer to a as the *value* of e and denote it by $\varphi(e)$. For two sets A and B of squares and a rectangle ρ, let $\mathcal{U}(A, B, \rho)$ denote $(\mathcal{U}(A) \cap \rho) \setminus \mathcal{U}(B)$, and $\mu(A, B, \rho)$ denote $\operatorname{Area}\mathcal{U}(A, B, \rho)$. For brevity, we set $\mathcal{U}(A, \rho) := \mathcal{U}(A, \emptyset, \rho)$ and $\mathcal{U}(A, B) := \mathcal{U}(A, B, \mathbb{R}^2)$. We define $\mu(A, \rho)$ and $\mu(A, B)$ analogously. We will repeatedly use the following simple equality

$$\mu(A \cup B, \rho) = \mu(B, \rho) + \mu(A, B, \rho)\,. \tag{1}$$

Primary data structure. In our case, we know in advance the set $\mathcal{S}$ of all squares that will ever be inserted into $\mathbb{S}$. $\mathcal{S}$ is the set of the xy-projections of cubes in $\mathcal{C}$; $|\mathcal{S}| = n$. Let $\square$ be the smallest axis-parallel rectangle containing all squares in $\mathcal{S}$. We build a binary space partition tree $\mathbb{T}$ on $\mathcal{S}$, the so-called *longest-sided kd-tree* [10], that is similar to a kd-tree [3]. The structure of $\mathbb{T}$ will be static but the information stored at each node of $\mathbb{T}$ will change as the set $\mathbb{S}$ changes.

Each node v of $\mathbb{T}$ is associated with a rectangle $\square_v = [x_v^-, x_v^+] \times [y_v^-, y_v^+] \subseteq \square$ and a subset $\mathcal{S}_v \subseteq \mathcal{S}$ of squares. For the root node u of $\mathbb{T}$, $\square_u = \square$ and $\mathcal{S}_u = \mathcal{S}$. If the horizontal edge of $\square_v$ is longer than its vertical edge (i.e., $x_v^+ - x_v^- \geq y_v^+ - y_v^-$), then we call v a $\oplus$-type node, otherwise we call v a $\ominus$-type node; see Figure 3. We classify a square $S \in \mathcal{S}$ that intersects $\square_v$ into one of the following mutually disjoint categories (see Figure 2): S is called a *cover* if $\square_v \subseteq S$. S is an *anchor* if S contains exactly one edge of $\square_v$ — exactly one edge e of S intersects the interior of $\square_v$. We set $\varphi(S) := \varphi(e)$. If S contains a long edge of $\square_v$, then S is called a *long* anchor, and a *short* anchor otherwise. S is a *bar* if $\square_v$ does not contain a vertex of S and two parallel edges of S—the ones parallel to a short edge of $\square_v$—intersect the interior of $\square_v$. If v is $\oplus$-type (resp. $\ominus$-type), then only the vertical (resp. horizontal) edges of S can intersect the horizontal (resp. vertical) edges of $\square_v$. Finally, S is called a *corner* and a *floater* if $\square_v$ contains one vertex and at least two vertices of S, respectively, in its interior.

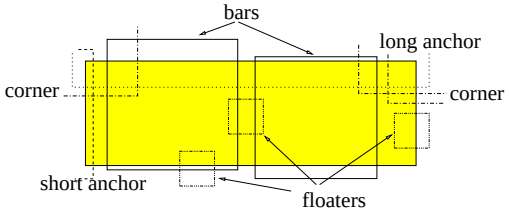

Figure 2. Partition of $\mathcal{S}$ into various categories.

We refer to bars, corners, and floaters as *short* squares (at least two edges of a short square intersect $\square_v$), and to anchors and covers as *long* squares. We call a square S *active* at v if it is not a cover or a long anchor at any proper ancestor of v. All short squares at v are active, and an active anchor at v is allowed to be an active short anchor at $p(v)$, the parent of v. Let $\mathcal{S}_v \subseteq \mathcal{S}$ be the subset of active squares at v. For a square $S \in \mathcal{S}$, let $\Xi(S)$ be the set of lowest nodes of $\mathbb{T}$ at which S is active, and let $\mathcal{N}(S)$ be the set of (proper) ancestors of nodes in $\Xi(S)$. S is active at all nodes in $\mathcal{N}(S) \cup \Xi(S)$, and it is stored at these nodes. Similarly, we call an orthogonal segment e intersecting $\square_v$ *short* at v if $\square_v$ contains an endpoint of e, and *long* otherwise. We call e *active* at v if there is no proper ancestor u of v at which e is long and parallel to a long edge of $\square_w$; this definition is similar to a square being active at a node v. Finally, we define $\Xi(e)$ and $\mathcal{N}(e)$ for an edge e as for a square.

If there are no short squares at v, then v is a leaf. Otherwise, v is an interior node, and we split $\square_v$ into two rectangles by splitting its longer edges, as follows. Suppose v is $\oplus$-type. Let $\mathcal{X}_v$ be the set of x-coordinates of the vertices of short squares at v that lie in the interval $[x_v^-, x_v^+]$, i.e., the set of x-coordinates of the vertices of: (i) $\mathcal{S}$ that lie inside $\square_v$, and (ii) of bars at v. Let χ_v be the median of $\mathcal{X}_v$. We split $\square_v$ into two rectangles by drawing the vertical

line $\ell_v : x = \chi_v$. The resulting left (resp. right) subrectangle is associated with the left (resp. right) child of v. If v is $\ominus$-type, then we split $\square_v$ by the horizontal line $\ell_v : y = \chi_v$, where χ_v is the median of the y-coordinates of the vertices of short squares at v that lie in the interval $[y_v^-, y_v^+]$; see Figure 3. We associate the bottom (resp. top) subrectangle with the left (resp. right) child of v. We call ℓ_v the *separator line* at v.

We define a subtree $\mathbb{T}(v)$ of $\mathbb{T}$ rooted at v, as follows: we recursively visit the subtree rooted at v. Suppose we are at a node w. If w is a leaf of $\mathbb{T}$ or a node of different type than v, we stop. Otherwise, we recursively visit the children of v. $\mathbb{T}(v)$ is the set of nodes visited by this procedure. Let $\mathbb{D}(v)$ denote the set of leaves of $\mathbb{T}(v)$. All interior nodes of $\mathbb{T}(v)$ are of the same type as v. See Figure 3.

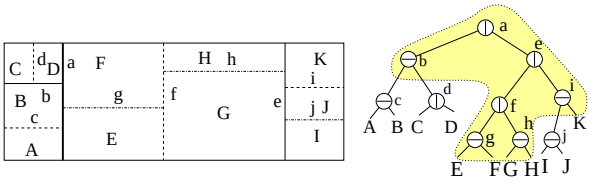

Figure 3. A recursive partition of $\square$ and the corresponding $\mathbb{T}$; lower-case letters denote the separator lines, and upper-case letters denote the cells associated with the leaves of $\mathbb{T}$; shaded region denotes $\mathbb{T}(\mathrm{rt})$.

Each square is a floater at the root of $\mathbb{T}$. Let $S \in \mathcal{S}$ be a square, and let v be a node such that S is active at v and the separating line ℓ_v intersects the interior of S. If $S \subset \operatorname{int} \square_v$, S splits into two floaters. Assume that S intersects $\partial\square_v$. If S is a floater at v, then either S splits into two corner squares, or into a bar and a floater. If S is a bar at v, it splits into two (active) anchors. (Note that since ℓ_v splits the long edges of $\square_v$, a bar is never split into two bars.) If S is a corner at v, then it splits into an (active) anchor and a corner. Finally, if S is an active short anchor at v, it splits into a cover and an anchor. See Figure 4. The following lemma will be used to bound the size of the data structure.

LEMMA 2.1. *(i) $\mathbb{T}$ has $O(n \log n)$ leaves, and the depth of $\mathbb{T}$ is $O(\log n)$. (ii) A square appears as a floater, a corner, or an active long anchor at $O(\log n)$ nodes, and as a bar, an active short anchor, or an active cover at $O(\log^2 n)$ nodes.*

Proof: Let Δ be the depth of $\mathbb{T}$, and let S be a square in $\mathcal{S}$. If S is a floater or a corner at a node v, then $\square_v$ contains at least one vertex of S. Since there are at most four nodes v of $\mathbb{T}$ at any given level for which $\square_v$ contains a vertex of S, S appears as a floater or a corner in at most 4Δ nodes. Next, if S appears as a bar at a node v, then S appears at $p(v)$ as a bar or a floater and does not appear as a bar at the sibling of v. Therefore the nodes at which S appears as a bar can be partitioned into a family $\Pi_B(S)$ of paths, each of length at most Δ, such that S appears as a floater at the parent of the root of each path. See Figure 4. Since S appears as a floater in at most 4Δ nodes, S appears as a bar in at most $4\Delta^2$ nodes. If S appears as an active short anchor at v, then there are two cases:

(i) S appears as a bar at $p(v)$, S intersects $\ell_{p(v)}$, and S appears as an anchor at the sibling of v, or

(ii) S appears as a corner or an active short anchor at $p(v)$ and does not appear as a short anchor at the sibling of v.

We can thus partition the nodes at which S appears as an active short anchor into a family Π_A of paths each of length at most Δ so that either the parent w of the root of each path is a leaf of a path in Π_B, or S appears as a corner at w. Hence, the number of paths in Π_A is $O(\Delta)$, and S appears as an active short anchor at $O(\Delta^2)$ nodes. If S appears as an active cover at a node v, then S appears as an active short anchor at $p(v)$, implying that S appears as an active cover at $O(\Delta^2)$ nodes. Finally, if S appears as a long anchor at v, then either $p(v)$ is the leaf of a path in $\Pi_A \cup \Pi_B$ or S appears as a corner at $p(v)$. Since there are $O(\Delta)$ such nodes, S appears an active long anchor at $O(\Delta)$ nodes. Next, we prove that $\Delta = O(\log n)$, which will complete the proof of (ii).

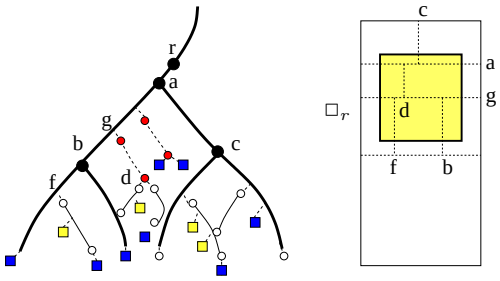

Figure 4. Trace of a square S through $\mathbb{T}$. Left: S appears as a corner or floater along thick paths; paths in $\Pi_B(S)$ (resp. $\Pi_A(S)$) are denoted by dashed (resp. solid) lines; S appears as a cover (resp. long anchor) at light (resp. dark) squares, and as a short anchor (resp. bar) at hollow (resp. filled) circles. Right: Separator lines inside $\square_r$ corresponding to the marked nodes on the left.

Let $L(m,k)$ denote the maximum number of leaves in a subtree rooted at a node v so that $\mathcal{S}_v$ contains m bars and $\square_v$ contains k vertices of $\mathcal{S}$. Let $\Delta(m,k)$ denote the maximum depth of such a subtree. If $m + k \leq 1$, then $\Delta(m,k) \leq 1$ and $L(m,k) \leq 2$. For $m + k \geq 1$, let w_1, w_2 be the two children of v. For $i = 1, 2$, let m_i, k_i denote the number of bars at w_i and the number of vertices that lie in $\square_{w_i}$. Then

$$\begin{aligned} L(m,k) &\leq L(m_1,k_1) + L(m_2,k_2), \\ \Delta(m,k) &\leq 1 + \max\{\Delta(m_1,k_1), \Delta(m_2,k_2)\}. \end{aligned}$$

A bar at v contributes at most one bar at a child of v, and a floater at v contributes at most one bar at a child of v. At least two of the vertices of a floater at v lie in $\square_v$, so the number of floaters at v is at most $k/2$, thereby implying that $m_1 + m_2 \leq m + k/2$ and $k_1 + k_2 \leq k$. Since the separator line chooses the median of $2m+k$ values, namely the (x- or y-) coordinates of the vertices of bars and of the vertices that lie in $\square_v$, $2m_i + k_i \leq (2m+k)/2$ for $i = 1, 2$. The solution to the above recurrence is

$$\begin{aligned} \Delta(m,k) &\leq A_1 \log_2(2m+k), \\ L(m,k) &\leq A_2(m + (k/2)\log_2(2m+k)), \end{aligned}$$

where $A_1, A_2 \geq 1$ are constants. Since there are no bars at the root of $\mathbb{T}$ and $\mathcal{S}$ has at most $4n$ vertices, the depth of $\mathbb{T}$ is $O(\log n)$ and it has $O(n \log n)$ leaves, as claimed. $\square$

Information stored at a node v. Let $\mathbb{S} \subseteq \mathcal{S}$ be the current set of squares. Let $\mathbb{S}_v = \mathcal{S}_v \cap \mathbb{S}$ be the subset of current squares that are active at v; $\mathbb{T}$ stores $\mathbb{S}_v$ at v. Let $\mathbb{B}_v$ (resp. $\mathbb{C}_v$, $\mathbb{F}_v$) be the subset of squares in $\mathbb{S}_v$ that are bar (resp. corner, floater) squares at v. Set $\mathbb{G}_v := \mathbb{F}_v \cup \mathbb{B}_v \cup \mathbb{C}_v \subseteq \mathbb{S}_v$ to be the set of short squares of $\mathbb{S}_v$. Let $\mathbb{X}_v$ (resp. $\mathbb{L}_{\mathbb{A}_v}$, $\mathbb{S}_{\mathbb{A}_v}$) be the subset of active covers (resp. long anchors, short anchors) at v. Set $\mathbb{L}_v := \mathbb{L}_{\mathbb{A}_v} \cup \mathbb{S}_{\mathbb{A}_v} \cup \mathbb{X}_v \subseteq \mathbb{S}_v$ to be the set of long squares of $\mathbb{S}_v$, and $\mathbb{L}_v^* = \bigcup_u \mathbb{L}_v$ where the union is taken over all ancestors of v including v itself. The following lemma is straightforward:

LEMMA 2.2. *Let w be a descendent of v. Then $\mathbb{F}_w \subseteq \mathbb{F}_v$ and $\mathbb{G}_w \subseteq \mathbb{G}_v$. Moreover, if w is a node of $\mathbb{T}(v)$ of the same type as v, then $\mathbb{B}_w \subseteq \mathbb{B}_v$, otherwise $\mathbb{B}_w \subseteq \mathbb{F}_v$.*

We call $\bar{\square} := \square_v \setminus \mathcal{U}(\mathbb{L}_v{}^*)$ an *anchored window*, which is a (possibly empty) rectangle lying inside $\square_v$. By (1),

$$\begin{aligned}\mu(\mathbb{S}, \square_v) &= \mu(\mathbb{L}_v^*, \square_v) + \mu(\mathbb{G}_v, \mathbb{L}_v^*, \square_v) \\ &= \text{Area}(\square_v) - \text{Area}(\bar{\square}_v) + \mu(\mathbb{G}_v, \bar{\square}_v)\,.\end{aligned}$$

See Figure 5 (left). Ideally, we would like to maintain $\bar{\square}_v$ and $\mu(\mathbb{G}_v, \bar{\square}_v)$ at each node v, from which $\mu(\mathbb{S}, \square_v)$ can be computed. It is, however, too expensive to maintain them explicitly at each node because the insertion or deletion of a square may change $\bar{\square}$ at several nodes. We therefore maintain another window $\boxplus_v$ called *exposed window*, an approximation of $\bar{\square}_v$ defined below, and maintain $\mu(\mathbb{G}_v, \boxplus_v)$.

If v is $\oplus$-type (resp. $\ominus$-type), then we define $\text{LB}_v^-, \text{LB}_v^+$ to be the bottom and top (resp. left and right) edges of $\square_v$, and $\text{SB}_v^-, \text{SB}_v^+$ to be the left and right (resp. bottom and top) edges of $\square_v$—the former two are the long edges of $\square_v$ and the latter two are the short edges of $\square_v$. Let $\text{lb}_v^-, \text{lb}_v^+, \text{sb}_v^-$, and sb_v^+ denote the values of LB_v^- (i.e., $\text{lb}_v^- = \varphi(\text{LB}_v^-)$), $\text{LB}_v^+, \text{SB}_v^-$, and SB_v^+, respectively. $\square_v$ is thus $[\text{sb}_v^-, \text{sb}_v^+] \times [\text{lb}_v^-, \text{lb}_v^+]$ if v is $\oplus$-type, and $[\text{lb}_v^-, \text{lb}_v^+] \times [\text{sb}_v^-, \text{sb}_v^+]$ otherwise. We divide active long and short anchors at v into two sets each: $\mathbb{LA}_v^-$ (resp. $\mathbb{LA}_v^+$) is the subset of $\mathbb{LA}_v$ that contains the edge LB_v^- (resp. LB_v^+), and $\mathbb{SA}_v^-$ (resp. $\mathbb{SA}_v^+$) is the subset of $\mathbb{SA}_v$ that contains the edge SB_v^- (resp. SB_v^+). To maintain $\boxplus_v$ and to compute $\bar{\square}_v$, we maintain the following information at v:

I. The sets $\mathbb{F}_v, \mathbb{C}_v, \mathbb{B}_v, \mathbb{LA}_v, \mathbb{SA}_v$, and the size $|\mathbb{X}_v|$.
II. The sequence $\mathsf{La}_v^- = \langle \varphi(S) \mid S \in \mathbb{LA}_v^- \rangle$ in decreasing order, and the sequence $\mathsf{La}_v^+ = \langle \varphi(S) \mid S \in \mathbb{LA}_v^+ \rangle$ in increasing order. For technical reasons, we add lb_v^- (resp. lb_v^+) at the end of La_v^- (resp. La_v^+).
III. The sequence $\mathsf{Sa}_v^- = \langle \varphi(S) \mid S \in \mathbb{SA}_v^- \rangle$ in decreasing order, and the sequence $\mathsf{Sa}_v^+ = \langle \varphi(S) \mid S \in \mathbb{SA}_v^+ \rangle$ in increasing order. For technical reasons, we add sb_v^- (resp. sb_v^+) at the end of Sa_v^- (resp. Sa_v^+).
IV. F_v: the edges of squares in $\mathbb{F}_v$ parallel to the long edges of $\square_v$ sorted in increasing order of their values.

Anchored window. $\bar{\square}_v$ is also defined by four edges $\text{LA}_v^-, \text{LA}_v^+$, $\text{SA}_v^-, \text{SA}_v^+$, and let $\text{la}_v^-, \text{la}_v^+, \text{sa}_v^-, \text{sa}_v^+$ be their values. If $\bar{\square} = \emptyset$, then these values are not well defined. We define them directly so that they are well defined even if $\bar{\square} = \emptyset$, and view $\bar{\square}_v$ as the intersection of four halfplanes defined by the lines supporting these edges. We describe la_v^+: Set $\tilde{\lambda}_v = \min_{\varphi \in \mathsf{La}_v^+} \varphi$, and let $\hat{\lambda}_v = \text{la}_{p(v)}^+$ if $p(v)$ and v are of the same type and $\hat{\lambda}_v = \text{sa}_{p(v)}^+$ otherwise. Then la_v^+ is defined to be the minimum of $\tilde{\lambda}_v$ and $\hat{\lambda}_v$ but its value cannot be less than lb_v^- (to ensure that $\text{la}_v^+ \in [\text{lb}_v^-, \text{lb}_v^+]$), i.e., $\text{la}_v^+ = \max\{\text{lb}_v^-, \min\{\hat{\lambda}_v, \tilde{\lambda}_v\}\}$. Values of the other edges of $\bar{\square}_v$ are described similarly. These values, and thus $\bar{\square}_v$, can be computed by traversing the path in $\mathbb{T}$ from the root to v.

Exposed window. The exposed window at v, $\boxplus_v \subseteq \square_v$, is also composed of four edges, called *virtual walls*, $\text{LE}_v^-, \text{LE}_v^+, \text{SE}_v^-, \text{SE}_v^+$ and whose values are $\text{le}_v^-, \text{le}_v^+, \text{se}_v^-, \text{se}_v^+$, respectively. See Figure 5. For each node, the algorithm maintains these edges and the quantity

$$\alpha(v) := \mu(\mathbb{G}_v, \boxplus_v) = \mu(\mathbb{F}_v \cup \mathbb{B}_v \cup \mathbb{C}_v, \boxplus_v), \tag{2}$$

i.e., the area of the union of short squares (at v) lying inside $\boxplus_v$. If $\text{se}_v^- \geq \text{se}_v^+$ or $\text{le}_v^- \geq \text{le}_v^+$, then $\boxplus_v = \emptyset$. For the root rt of $\mathbb{T}$, the algorithm ensures that $\boxplus_{\text{rt}} = \square_{\text{rt}}$, implying that $\alpha(\text{rt}) = \mu(\mathbb{S})$. For a leaf v and for a node v with $\boxplus_v = \emptyset$, $\alpha(v) = 0$. If $\boxplus_v = \bar{\square}_v$, then $\boxplus_v$ is called *tight* (see Figure 5 (i)); otherwise it is called *loose*. If $\boxplus_v$ is tight, then $\alpha(v) = \mu(\mathbb{G}_v, \bar{\square}_v)$, as desired. As mentioned above, it is too expensive to ensure that $\boxplus_v$ is tight at all nodes of $\mathbb{T}$. The algorithm lets $\boxplus_v$ be loose at many nodes, and updates them carefully in a lazy manner so that the following two invariants are maintained at every node $v \in \mathbb{T}$; see Figure 5 (ii):

(I1) *If v is $\oplus$-type (resp. $\ominus$-type), then the y-coordinate (resp. x-coordinate) of none of the vertices of a square in $\mathbb{F}_v$ lies between le_v^- and la_v^- or between le_v^+ and la_v^+.*
(I2) *If v is $\oplus$-type (resp. $\ominus$-type), then the x-coordinate (resp. y-coordinate) of none of the vertices of a square in $\mathbb{F}_v \cup \mathbb{B}_v$ lies between se_v^- and sa_v^- or between se_v^+ and sa_v^+.*

Figure 5 (iii) illustrates an example of an exposed window that violates (I1) and (I2). Intuitively, (I1) and (I2) ensure that no edge of $\mathbb{F}_v \cup \mathbb{B}_v$ that is parallel to LE_v^+ lies between LE_v^+ and LA_v^+, though an edge of $\mathbb{C}_v$ may lie there; the same holds for the other three virtual walls of $\boxplus_v$. These invariants enable us to compute $\mu(\mathbb{F}_v \cup \mathbb{B}_v, \rho)$, for any $\rho \subseteq \boxplus_v \oplus \bar{\square}_v$, by reducing it to a 1D problem. The secondary data structures described below will facilitate the computation of $\mu(\mathbb{G}_v, \rho)$ (provided I1 and I2 are maintained), which in turn is used to update $\alpha(v)$ as $\boxplus_v$ is made tight.

Secondary structures at v. For a square S, with a slight abuse of notation, we use S_v^- (resp. S_v^+) to denote the interval in $\mathbb{R}^1$ formed by the intersection of $S \cap \square_v$ and the line containing the edge LE_v^- (resp. LE_v^+) of $\boxplus_v$. (See Figure 6.) Let $\mathbb{I}_v^- = \{S_v^- \mid S \in \mathbb{B}_v \cup \mathbb{F}_v\}$, and let $\mathbb{I}_v^+ = \{S_v^+ \mid S \in \mathbb{B}_v \cup \mathbb{F}_v\}$. For an interval $I \in \mathbb{I}_v^- \cup \mathbb{I}_v^+$, let I'' be the rectangle $I \times [y_v^-, y_v^+]$ if v is $\oplus$-type and $[x_v^-, x_v^+] \times I$ if v is $\ominus$-type. Set $\mathbb{P}_v^- = \{I'' \mid I \in \mathbb{I}_v^-\}$ and $\mathbb{P}_v^+ = \{I'' \mid I \in \mathbb{I}_v^+\}$. The following two secondary data structures are maintained at v.

Wall structure. Store $\mathbb{I}_v^-$ into a dynamic segment tree that supports the following two operations in $O(\log n)$ time [3]: (i) insertion or deletion of an interval to $\mathbb{I}_v^-$, and (ii) for a query interval δ, return $\text{UL}^-(v, \delta) = |\mathcal{U}(\mathbb{I}_v^-) \cap \delta|$. Similarly, store $\mathbb{I}_v^+$ into a segment tree so that $\mathbb{I}_v^+$ can be updated and $\text{UL}^+(v, \delta) = |\mathcal{U}(\mathbb{I}_v^+) \cap \delta|$, for a query interval δ, can be computed in $O(\log n)$ time.

Corner structure. We process $\mathbb{C}_v$ and $\mathbb{I}_v^-$ into a data structure that supports the following operations: (i) insertion or deletion of a square to $\mathbb{C}_v$ and an interval to $\mathbb{I}_v^-$; (ii) for a query rectangle $\rho \subseteq \square_v$, compute $\text{CA}^-(v, \rho) = \mu(\mathbb{C}_v, \mathbb{P}_v^-, \rho)$. Similarly, maintain $\mathbb{C}_v$ and $\mathbb{I}_v^+$ into a data structure to compute $\text{CA}^+(v, \rho) = \mu(\mathbb{C}_v, \mathbb{P}_v^+, \rho)$ for a query rectangle ρ. The corner data structure is described in Section 5 and used in Section 4. By Lemma 5.1, a query is answered in $O(\log n)$ time and an update can be performed in $O(\log^2 n)$ time.

3. UPDATE PROCEDURES

This section describes the procedures for updating $\mathbb{T}$ and the secondary structures stored at its various nodes, as a square is inserted or deleted. The update procedures maintain (I1) and (I2) and use the following auxiliary procedures.

ADJUSTWALL(v, WL, ξ). Shift $\text{WL} \in \{\text{LE}_v^-, \text{LE}_v^+, \text{SE}_v^-, \text{SE}_v^+\}$ of the exposed window $\boxplus_v$ at v from its current position to ξ. The value of ξ is such that the invariants (I1) and (I2) are not violated by the shift. The procedure returns the updated value of $\alpha(v)$.

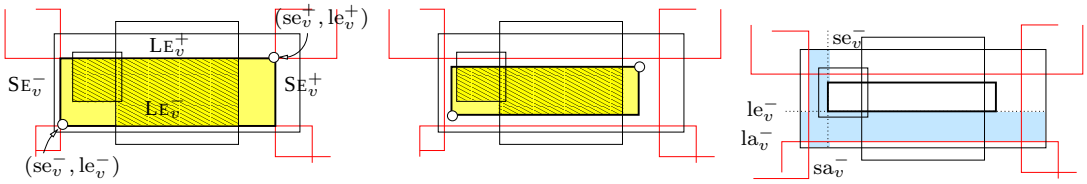

Figure 5. Exposed window $\boxplus_v$ (lightly shaded rectangles) and its virtual walls, hatched region is $\mathcal{U}(\mathbb{G}_v, \boxplus_v)$. Left: $\boxplus_v$ is tight; middle: $\boxplus_v$ satisfies invariants (I1) and (I2); right: $\boxplus_v$ violates both (I1) and (I2) — shaded strips contain vertices of floaters.

TIGHTENWINDOW(v). Tighten the exposed window $\boxplus_v$ so that it becomes the same as $\bar{\square}_v$. The procedure returns the updated value of $\alpha(v)$.

PUSHUP(v). Let w, z be the children of v. Assuming $\boxplus_v, \boxplus_w, \boxplus_z$ are tight, the procedure computes the value of $\alpha(v)$ from $\alpha(w)$ and $\alpha(z)$ and returns it.

These procedures are described in Section 4. ADJUSTWALL and TIGHTENWINDOW procedures take $O(\log^2 n)$ time, and PUSHUP takes $O(1)$ time. Assuming that we have these procedures at our disposal, we describe the procedure for inserting a square; the deletion procedure is symmetric.

Inserting a square. Let S be a square that we wish to insert into $\mathbb{S}$. We update the information stored at various nodes of $\mathbb{T}$, including $\alpha(\cdot)$. The new value of $\alpha(\mathrm{rt})$ is precisely the new value of $\mu(\mathbb{S})$.

We first visit the nodes of $\mathcal{N}(S)$ and their children in a top-down manner, starting from the root, and call the function TIGHTENWINDOW at each node. This ensures that the exposed window is tight at all these nodes before S is actually inserted. Next, for each node $v \in \mathcal{N}(S) \cup \Xi(S)$, we insert S at v, as described below, in a top-down manner. Finally, we visit the nodes $v \in \mathcal{N}(S) \cup \Xi(S)$ in a bottom-up manner to update the value of $\alpha(v)$. For $v \in \Xi(S)$, $\alpha(v)$ is updated directly as described below, and for $v \in \mathcal{N}(S)$, it is updated using the procedure PUSHUP(v); the exposed windows at v and its children will be tight before PUSHUP(v) is invoked at v; it ensures that $\alpha(v)$ is correctly computed.

The processing of S at a node $v \in \mathcal{N}(S) \cup \Xi(S)$ depends on the status of S at v. Recall that if S is a cover or a long anchor at v, then $v \in \Xi(S)$; if S is a floater, corner, or bar, then $v \in \mathcal{N}(S)$; if S is an active short anchor and v is a leaf (resp. an interior node) of $\mathbb{T}$, then $v \in \Xi(S)$ (resp. $v \in \mathcal{N}(S)$). If S is a short square or a short anchor at v, then we simply insert S at v and update the information (I)-(IV) and the secondary structures at v in a straightforward manner in $O(\log^2 n)$ time. Inserting an active long anchor or cover at v is more involved, as it may violate (I1) and (I2) at many of its descendents. Additional processing is needed at these descendents to restore (I1) and (I2). We describe these two cases in detail.

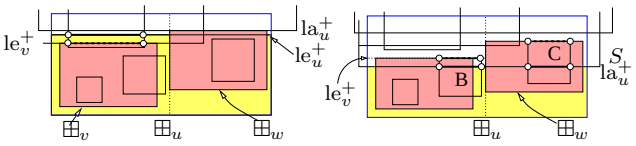

Figure 6. Insertion of a long anchor. Solid (dashed) intervals belong to $\mathbb{I}_u^+$ ($\mathbb{I}_v^+$). Left: before the insertion of S. Right: after the insertion of S; PUSHDOWN procedure lowers LE_u^+ to the bottom edge of S, LE_v^+ to the top edge of B, and LE_w^+ to the top edge of C.

Long anchor. If $S \in \mathbb{LA}_v^+$, we insert $\varphi(S)$ into La_v^+, and into La_v^- otherwise. Next, we update the exposed windows at v and some of the descendents of v to ensure that (I1) and (I2) are not violated after the insertion of S. We refer to this procedure as PUSHDOWN; it also updates the value of $\alpha(\cdot)$. For the sake of concreteness, we assume that $S \in \mathbb{LA}_v^+$ and that v is $\oplus$-type; this implies that $\mathrm{LA}_v^+, \mathrm{LE}_v^+$ are the top edges of $\bar{\square}_v, \boxplus_v$. TIGHTENWINDOW was called at v in the beginning of the insertion procedure, so $\boxplus_v$ was tight before the insertion of S. If $\varphi(S) \geq \mathrm{la}_v^+$, then $\boxplus_v$ remains tight even after the insertion of S and we do nothing. If $\varphi(S) < \mathrm{la}_v^+$ then LA_v^+ lowers to $\varphi(S)$. We *lower* LE_v^+ of $\boxplus_v$ to $\varphi(S)$, by sweeping from its current position to $\varphi(S)$ and stopping at all values in F_v, which we refer to as *events*. An event due to an edge $e \in \mathsf{F}_v$ indicates that (I1) or (I2) is violated at v (and some of its descendents) because of e. For technical reasons, we lower $\mathrm{LA}_v^+, \mathrm{LE}_v^+$ simultaneously through the sweep process. Let ξ denote the value of le_v^+ at the current event and η at the last event. Initially, we set ξ to the original value of le_v^+ (before the insertion of S) and $\eta = +\infty$. If LE_v^+ reaches $\varphi(S)$, set $\alpha(v)$:=ADJUSTWALL(v, LE_v^+, $\varphi(S)$) and $\mathrm{le}_v^+ = \mathrm{la}_v^+ = \varphi(S)$, and stop. If LE_v^+ reaches an edge $e \in \mathsf{F}_v$, where e is an edge of a floater $\sigma \in \mathbb{F}_v$, set η to ξ and ξ to $\varphi(e)$.

Let $\mathcal{Z}(v, e)$ be the subtree consisting of the descendents w of v that lie in $\mathcal{N}(e)$, at which σ is floater or bar, and for which its top virtual wall if above e. $\mathcal{Z}(v, e)$ can be computed in $O(\log^2 n)$ time by traversing $\mathbb{T}$ in a top-down manner, starting from v. Let w be a node in $\mathcal{Z}(v, e)$. The insertion of S causes the top edge of $\bar{\square}_w$ to pass below γ and thus the top virtual wall of $\boxplus_w$, denoted by WL, needs to be aligned with e to maintain (I1) and (I2); WL is SE_w^+ (resp. LE_v^+) if v is $\ominus$-type (resp. $\oplus$-type). We set $\alpha(w)$:=ADJUSTWALL(w, WL, ξ). If w is $\oplus$-type, we also update the secondary data structures at w: we insert the interval σ_v^+ into $\mathbb{I}_w^+$ if e is the top edge of σ (LE_w^+ starts intersecting σ as we lower it beyond e) and delete it from $\mathbb{I}_w^+$ if e is the bottom edge (LE_w^+ no longer intersects σ as we lower it beyond e), and we update the wall and corner data structures at w. See Figure 6.

For each node $w \in \mathcal{Z}(v, e)$, we say that S *encountered* e at node w. The procedure spends $O(\log^2 n)$ time in computing $\mathcal{Z}(v, e)$ and $O(\log^2 n)$ time at each node $w \in \mathcal{Z}(v, e)$; we charge the former to the "encounter" of e at v. For $S \in \mathbb{LA}_v^-$ and for a $\ominus$-type node, the PUSHDOWN procedure is symmetric. The total running time is $O((1 + \sum_w \kappa_w) \log^2 n)$, where κ_w is the number of edges encountered by S at w.

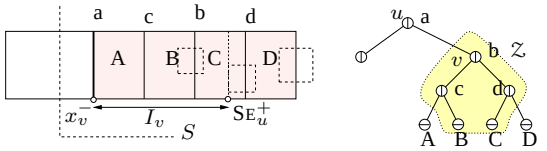

Figure 7. Insertion of a cover square and the subtree $\mathcal{Z}$. Left: shaded region is $\square_v$, dotted line is SE_v^+.

Cover. We increment the value of $|\mathbb{X}_v|$. If $|\mathbb{X}_v| = 1$, we update $\boxplus_v$ and further processing is needed at v and its descendents to

ensure (I1) and (I2) and to update the secondary structures. Let $u = p(v)$. Suppose u is $\oplus$-type and v is the right child of u, in which case $S \in \mathbb{SA}_u^+$; see Figure 7. We move the right virtual wall of $\boxplus_v$ (which is $\textsc{Le}_v^+$ if v is $\ominus$-type and $\textsc{Se}_v^+$ otherwise) from its current position, say, ξ, to x_v^-, the value of the left boundary of $\square_v$. Let $I_v = [x_v^-, \xi]$, and let $\mathcal{Z} \subseteq \mathbb{T}(u)$ be the set of nodes z such that $[x_z^-, x_z^+] \cap I_v \neq \emptyset$ and $\mathbb{F}_z \cup \mathbb{B}_z \neq \emptyset$. $\mathcal{Z}$ forms a subtree of $\mathbb{T}(u)$ rooted at v; if v is $\ominus$-type, then $\mathcal{Z} = \{v\}$. The total time spent in computing $\mathcal{Z}$ is $O((1+\beta_v)\log n)$ where β_v is the number of edges in $\mathbb{F}_v \cup \mathbb{B}_v$ whose values lie in the interval I_v. To ensure (I1) and (I2), we call the PUSHDOWN procedure at the leaves w of $\mathcal{Z}$ that are of $\oplus$-type to move $\textsc{Le}_w^+$ to the left edge of $\square_w$. At other nodes w of $\mathcal{Z}$, we set $\alpha(w) = 0$ and $\textsc{Se}_w^+$ to the left edge of $\square_w$. The insertion procedure for other cases—when v is the left child of u or when u is $\ominus$-type—is similar. The total time spent in inserting a cover at v is $O((1+\sum_w \kappa_w)\log^2 n + \beta_v \log n)$, where w is a descendent of a leaf of $\mathcal{Z}$ and κ_w is the number of edges encountered by the PUSHDOWN procedure at w.

A square can be deleted using a similar procedure. The correctness of the update procedures follows from the following lemma. The proof of the lemma, tedious but not difficult, is omitted from this extended abstract because of lack of space.

LEMMA 3.1. *Invariants (I1)–(I2) are maintained at each node of $\mathbb{T}$ after inserting a square into $\mathbb{S}$ or deleting from it.*

Since (I1) and (I2) hold at all times, the auxiliary procedures compute the value of $\alpha(\cdot)$ correctly, which in turn implies that the algorithm correctly maintains $\alpha(v)$ at each node v of $\mathbb{T}$ at all times. In particular $\alpha(\mathrm{rt}) = \mu(\mathbb{S})$ at all times. The next lemma bounds the amortized update time.

LEMMA 3.2. *We can maintain $\mathbb{S}$ into a data structure so that $\mu(\mathbb{S})$ can be updated in $O(\log^4 n)$ amortized time, as we insert or delete a square, provided that the sequence of updates is size preserving.*

Proof: The argument in the update procedure implies that the insertion/deletion time of a square S is $O(\log^4 n + \sum_v \kappa_v \log^2 n + \sum_w \beta_w \log n)$, where κ_v is the number of edges encountered at v by S during the update procedure, w is a node at which S is a cover, and β_w is as defined above. We charge $O(\log^4 n)$ time to the insertion of S. We prove that the amortized cost of the second term is also $O(\log^4 n)$; a similar argument bounds the amortized cost of the third term by $O(\log^4 n)$. We charge $\log^2 n$ units to each edge encountered by S during its insertion or deletion. We claim that an edge γ of a square $\sigma \in \mathbb{S}$ is charged at most twice at a node w of $\mathbb{T}$ by the insertion and deletion procedures each. Since $w \in \mathcal{N}(\gamma)$ and $\sigma \in \mathbb{F}_v \cup \mathbb{B}_v$, γ is charged a total of $O(\log^2 n)$ times. Hence γ is charged a total of $O(\log^4 n)$ units, which can be charged to the insertion of σ. This implies that the amortized update time of a square is $O(\log^4 n)$.

To prove the above claim, suppose γ is parallel to the long edges of $\square_w$; the other case is symmetric. First consider the insertion procedure. Suppose γ was encountered by S at w while lowering le_w^+ (actually, $\gamma \in \mathsf{F}_w$ in this case). Recall that γ is encountered at w because the PUSHDOWN procedure swept through γ at an ancestor v of w while lowering $\textsc{Le}_v^+, \textsc{La}_v^+$, where $\gamma \in \mathsf{F}_v$ and $S \in \mathbb{LA}_v \cup \mathbb{X}_v$. Since $\sigma \in \mathbb{F}_v$, one of the edges of σ lies inside $\square_v$ while S contains a long edge of $\square_v$, which implies that $|\mathsf{X}(\sigma)| < (\mathrm{le}_v^+ - \mathrm{le}_v^-) \leq |\mathsf{X}(S)|$, i.e., S is bigger than σ. (Note that this claim might not be true if S is a short anchor at $\square_v$ and thus only contains a short edge of $\square_v$.) Moreover, σ is inserted into $\mathbb{S}$ before S. We have assumed the update sequence to be size preserving (see Section 2), therefore σ will be deleted before S is deleted, implying that la_v^+, and thus la_w^+, remain less than $\varphi(\gamma)$ until σ is deleted. Since $\sigma \in \mathbb{F}_w$ and invariants (I1) and (I2) are maintained, $\mathrm{le}_w^+ \leq \varphi(\gamma)$ until γ is deleted. Hence, γ will not be encountered again at w while lowering $\textsc{Le}_w^+$. Similarly, γ will be encountered at w at most once by the insertion procedure while raising $\textsc{Le}_w^-$. Similar arguments show that γ is charged at w at most twice by the deletion procedure or when γ is parallel to the short edges of $\square_w$. $\square$

Remark. The proof of the above lemma crucially uses the observations that if the PUSHDOWN procedure at v because of the insertion of a long anchor S sweeps through the edge γ of a floater square σ then S is bigger than σ, and that the size-preserving order ensures that an edge is encountered only $O(1)$ times for a fixed node. These claims are not true if PUSHDOWN is called to move a short virtual edge. That is why short and long anchors (and edges of $\square_v$) are treated differently and auxiliary data structures are stored only with respect to long edges.

4. AUXILIARY PROCEDURES

If $\alpha(v) \neq 0$ and $\boxplus_v, \boxplus_w, \boxplus_z$ are tight, where w and z are the children of v, then

$$\alpha(v) = \alpha(w) + \alpha(z) + \mathrm{Area}(\boxplus_v \setminus (\boxplus_w \cup \boxplus_z)).$$

Using this observation, PUSHUP(v) can be implemented in $O(1)$ time. TIGHTENWINDOW can be implemented by invoking ADJUSTWALL at most four times. We thus describe ADJUSTWALL, which uses the secondary structures (wall and corner structures) stored at the nodes of $\mathbb{T}$ and relies on invariants (I1) and (I2).

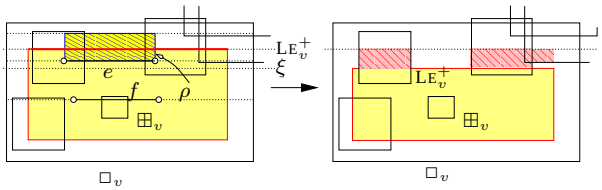

Figure 8. ADJUSTWALL(v, $\textsc{Le}_v^+$,ξ): current $\boxplus_v$ and after adjusting $\textsc{Le}_v^+$ (lightly shaded rectangles). Left: Segment e and rectangle ρ are compatible with $\textsc{Le}_v^+$; segment f is not compatible with $\textsc{Le}_v^+$. Right: area of the hatched region is the change in $\alpha(v)$.

ADJUSTWALL(v, WL, ξ). This procedure shifts the virtual wall WL of $\boxplus_v$ from its current position to ξ, assuming that the value of ξ is such that the invariants (I1) and (I2) are not violated by the shift. The procedure returns the updated value of $\alpha(v)$. See Figure 8. Let $R \subseteq \square_v$ be the rectangle obtained by sweeping WL from its current position to ξ, then $\alpha(v)$ changes by $\mu(\mathbb{G}_v, R)$.

We call an orthogonal segment $e \subseteq \square_w$ *compatible* with a virtual wall WL of $\boxplus_v$ parallel to e if the strip W_e bounded by the lines supporting e and WL does not contain a vertex of any square in $\mathbb{F}_v \cup \mathbb{B}_v$. That is, no square of $\mathbb{F}_v \cup \mathbb{B}_v$ is contained in W_e and the two boundary lines of W_e intersect the same set of squares in $\mathbb{F}_v \cup \mathbb{B}_v$. We call a rectangle $\rho \subseteq \square_v$ compatible with WL if the two edges of ρ that are parallel to WL are compatible with e. See Figure 8.

For a square $\rho \subseteq \square_v$ that is compatible with one of the virtual walls of $\boxplus_v$, we describe a procedure that computes $\mu(\mathbb{G}_v, \rho)$. By (I1) and (I2), R is compatible with WL, so $\mu(\mathbb{G}_v, R)$ can be computed using this procedure. First, consider the case when ρ is compatible with $\textsc{Le}_v^+$. In this case, ρ intersects a square $S \in \mathbb{F}_v \cup \mathbb{B}_v$ if the interval $I := S_v^+$ lies in $\mathbb{I}_v^+$. Furthermore, $S \cap \rho = I'' \cap \rho$, S

be regarded as a bar. Let $\rho = \mathsf{x}_\rho \times \mathsf{y}_\rho$. By (1),

$$\begin{aligned}\mu(\mathbb{G}_v, \rho) &= \mu(\mathbb{F}_v \cup \mathbb{B}_v, \rho) + \mu(\mathbb{C}_v, \mathbb{F}_v \cup \mathbb{B}_v, \rho) \\ &= \mu(\mathbb{P}_v^+, \rho) + \mu(\mathbb{C}_v, \mathbb{P}_v^+, \rho) \\ &= \textsc{Ul}^+(v, \mathsf{x}_\rho)|\mathsf{y}_\rho| + \textsc{Ca}^+(v, \rho).\end{aligned}$$

Similarly if ρ is compatible with $\textsc{Le}_v^-$, then

$$\mu(\mathbb{G}_v, \rho) = \textsc{Ul}^-(v, \mathsf{x}_\rho)|\mathsf{y}_\rho| + \textsc{Ca}^-(v, \rho).$$

Using the wall and corner structures stored at v, $\mu(\mathbb{G}_v, \rho)$ can be computed in $O(\log n)$ time.

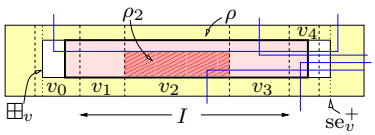

Figure 9. Computing $\mu(\mathbb{G}_v, \rho)$ for a rectangle ρ compatible with $\textsc{Se}_v^+$; thick rectangle is ρ and dashed lines are separator lines.

Next, suppose ρ is compatible with $\textsc{Se}_v^+$ and v is $\oplus$-type. Since we do not store secondary structure for short edges of $\square_v$, answering the query is more involved in this case. By (I2), no vertical edge of a square in $\mathbb{F}_v \cup \mathbb{B}_v$ intersects ρ. Hence, if a square in $\mathbb{B}_v$ intersects ρ, it contains ρ and $\mu(\mathbb{G}_v, \rho) = \mathrm{Area}(\rho)$. We can check this condition in $O(\log n)$ time using the wall data structure. So assume that ρ does not intersect any square in $\mathbb{B}_v$. Let v_0 (resp. v_k) be the node in $\mathbb{D}(v)$ (cf. Section 2) such that $\square_{v_0}$ (resp. $\square_{v_k}$) contains the left (resp. right) edge of ρ. Let I be the x-projection of $\rho \setminus (\square_{v_0} \cup \square_{v_k})$, and let $v_1, \ldots, v_{k-1}$ be the nodes in $\Xi(I)$; each v_i lies in $\mathbb{T}(v)$. See Figure 9. For $1 \le i < k$, let $\mathbb{H}_i = \mathbb{G}_v \setminus \mathbb{G}_{v_i} = \bigcup_w \mathbb{L}\mathrm{A}_w \cup \mathbb{S}\mathrm{A}_w$, where $w \neq v$ is a node lying on the path from v to v_i. Let

$$\rho_i := (\rho \cap \square_{v_i}) \setminus \mathcal{U}(\mathbb{H}_i), \tag{3}$$

which is a rectangle obtained by clipping ρ from top and bottom (e.g. hatched rectangle in Figure 9 is ρ_2). By construction, ρ_i is compatible with the right virtual wall of $\square_{v_i}$. Using (1) and (3),

$$\begin{aligned}\mu(\mathbb{G}_v, \rho) &= \sum_{i=0}^{k} \mu(\mathbb{H}_i, \rho \cap \square_{v_i}) + \mu(\mathbb{G}_{v_i}, \mathbb{H}_i, \rho \cap \square_{v_i}) \\ &= \sum_{i=0}^{k} \mu(\mathbb{H}_i, \rho \cap \square_{v_i}) + \mu(\mathbb{G}_{v_i}, \rho_i)\,.\end{aligned}$$

The first term can be computed in $O(\log n)$ time. As for the second term, if v_i is a leaf of $\mathbb{T}$, then $\mathbb{G}_{v_i} = \emptyset$ and $\mu(\mathbb{G}_{v_i}, \rho) = 0$, so assume that v_i is an interior node of $\mathbb{T}$. If $v_i \in \mathbb{D}(v)$, then v_i is $\ominus$-type and ρ_i is compatible with $\textsc{Le}_{v_i}^+$ (e.g. v_1, v_3 in Figure 9). We can therefore compute $\mu(\mathbb{G}_{v_i}, \rho_i)$ in $O(\log n)$ time, as described above. If $v_i \notin \mathbb{D}(v)$, i.e., v_i is $\oplus$-type and $1 \le i < k$ (e.g. v_2 in Figure 9), then it can be proved that $\mathbb{B}_{v_i} \cup \mathbb{F}_{v_i} = \emptyset$ and

$$\mu(\mathbb{G}_{v_i}, \rho_i) = \mu(\mathbb{C}_{v_i}, \rho_i) = \mu(\mathbb{C}_{v_i}, \mathbb{P}_{v_i}^+, \rho_i) = \textsc{Ca}^+(v, \rho_i).$$

By Lemma 5.1, $\mu(\mathbb{G}_{v_i}, \rho_i)$ can be computed in $O(\log n)$ time. Putting everything together, $\mu(\mathbb{G}_v, \rho)$ can be computed in $O(\log^2 n)$ time, and AdjustWall thus takes $O(\log^2 n)$ time.

Lemma 4.1. *AdjustWall and TightenWindow procedures take $O(\log^2 n)$ time, and PushDown takes $O(1)$ time.*

5. CORNER DATA STRUCTURE

Let $\square := [0, 0] \times [a, b]$ be a rectangle with $p_0, \ldots, p_3$ as its vertices. Without loss of generality, assume that $a \ge b$. Let $\mathbb{C}$ be a set of corner squares with respect to $\square$, let $\mathbb{B}$ be a set of rectangles, called *bars*, whose horizontal edges lie on the top and bottom edges of $\square$. Let $\mathbb{I}$ be the set of x-projections of bars in $\mathbb{B}$. We describe a data structure that supports the following operations: insertion or deletion of a corner or a bar; for a query rectangle $\rho \subseteq \square$, compute $\textsc{Ca}(\rho) = \mu(\mathbb{C}, \mathbb{B}, \rho)$. We assume that we have the wall data structure, described in Section 2, so that for any interval I, $\varepsilon(I) = |I \setminus \mathcal{U}(\mathbb{I})|$ can be computed in $O(\log n)$ time.

For a corner square S, let $q(S)$ be the vertex of S that lies inside $\square$. For $0 \le i \le 3$, let $\mathbb{C}_i \subseteq \mathbb{C}$ be the set of corners that contain the vertex p_i of $\square$, and let $\mathcal{Q}_i = \{q(S) \mid S \in \mathbb{C}_i\}$; see Figure 10 (ii). We call a (rectilinear) polygon *staircase* if it consists of a rectilinear chain that is both x- and y-monotone (in each coordinate it can be either increasing or decreasing), and the endpoints of the chain are connected together by a horizontal and a vertical edge; see Figure 10 (i). The common endpoint of the horizontal and the vertical edge is called the *apex* of the polygon. For each i, $\mathcal{U}(\mathbb{C}_i, \square)$ is a staircase polygon whose apex is p_i. We construct the following data structures:

- We preprocess $\mathbb{C}_0$ and $\mathbb{B}$ into a data structure Ψ_0 so that, for a query rectangle $\rho \subseteq \square$, $\mu(\mathbb{C}_0, \mathbb{B}, \rho)$ can be computed in $O(\log n)$ time. We construct a similar data structure Ψ_1 on $\mathbb{C}_1$ and $\mathbb{B}$ for computing $\mu(\mathbb{C}_1, \mathbb{B}, \rho)$.
- We preprocess $\mathbb{C}_0, \mathbb{C}_2, \mathbb{B}$ into a data structure Ψ_2 so that, for a rectangle $\rho \subseteq \square$, $\mu(\mathbb{C}_2, \mathbb{B} \cup \mathbb{C}_0, \rho)$ can be computed in $O(\log n)$ time. We construct a similar data structure Ψ_3 on $\mathbb{C}_1, \mathbb{C}_3, \mathbb{B}$ for computing $\mu(\mathbb{C}_3, \mathbb{B} \cup \mathbb{C}_1, \rho)$.

Before describing $\Psi_0, \ldots, \Psi_3$, we explain how they can be used to compute $\textsc{Ca}(\rho)$ in $O(\log n)$ time. For $0 \le i \le 3$, let Γ_i be the rectilinear chain of $\mathcal{U}(\mathbb{C}_i)$; Figure 10 (i). Γ_i and Γ_{i+1} intersect in at most one point, but Γ_i and Γ_{i+2} may intersect several times. For each i, we define a point r_i, as follows. If Γ_i and Γ_{i+1} (mod 4) do not intersect, then r_i is the endpoint of Γ_i that lies on the edge $p_i p_{i+1}$, otherwise r_i is the (orthogonal) projection of the intersection point of Γ_i and Γ_{i+1} on the edge $p_i p_{i+1}$. Finally, let $\square_i$ be the rectangle formed by r_{i-1} and r_i as two of its diagonally opposite vertices; see Figure 10 (iv) and (v). The interiors of $\square_i$ and $\square_{i+1}$ are disjoint but those of $\square_i$ and $\square_{i+2}$ may intersect. However, if $\square_0$ and $\square_2$ intersect, then $\square_1$ and $\square_3$ are disjoint, and vice-versa; Figure 10 (iv). Note that $\square_0, \ldots, \square_3$ may not cover $\square$; Figure 10 (v). Nevertheless, $\mathcal{U}(\mathbb{C}, \square) \subseteq \bigcup_{i=0}^3 \square_i$ and Γ_i appears on $\partial\mathcal{U}(\mathbb{C}, \square)$ only inside $\square_i$; see Figure 10. The following equality is straightforward: $\mathcal{U}(\mathbb{C}, \square) = \bigcup_{i=0}^3 \mathcal{U}(\mathbb{C}_i, \square_i)$. For a rectangle ρ, we set $\rho_i = \rho \cap \square_i$. Then

$$\mathcal{U}(\mathbb{C}, \mathbb{B}, \rho) = \bigcup_{i=0}^{3} \mathcal{U}(\mathbb{C}_i, \mathbb{B}, \rho_i).$$

Since the interiors of $\square_i$ and $\square_{i+1}$ are disjoint,

$$\begin{aligned}\mu(\mathbb{C}, \mathbb{B}, \rho) &= \mu(\mathbb{C}_0, \mathbb{B}, \rho_0) + \mu(\mathbb{C}_1, \mathbb{B}, \rho_1) + \\ &\quad \mu(\mathbb{C}_2, \mathbb{C}_0 \cup \mathbb{B}, \rho_2) + \mu(\mathbb{C}_3, \mathbb{C}_1 \cup \mathbb{B}, \rho_3)\,.\end{aligned}$$

Let $\mathcal{V}_i \subseteq \mathcal{Q}_i$ be the subset of points that appear as (convex) vertices of $\mathcal{U}(\mathbb{C}_i)$. We maintain $\mathcal{V}_i$ sorted from left to right. By performing a binary search on $\mathcal{V}_i$ and $\mathcal{V}_{i+1}$, r_i can be computed in $O(\log n)$ time. We can then compute $\rho_0, \ldots, \rho_3$ in additional $O(1)$ time. Each of the terms on the right hand side of the above equation can be computed in $O(\log n)$ time using $\Psi_0, \ldots, \Psi_3$, therefore

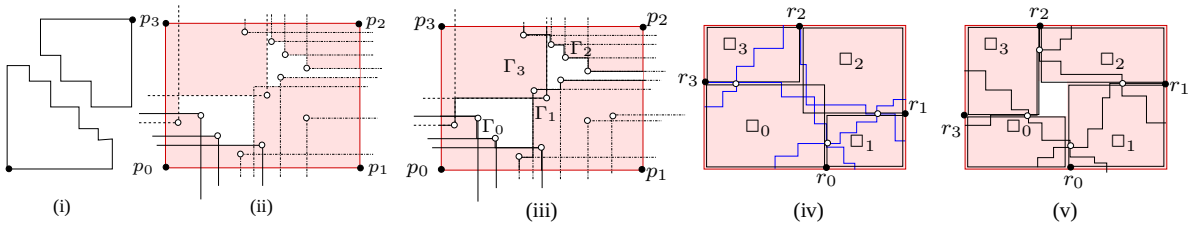

Figure 10. (i) Staircase polygons. (ii) Four different types of corner squares. (iii) rectilinear chains $\Gamma_0, \ldots, \Gamma_3$. (iv) $\square_0, \ldots, \square_3$ cover $\square$; $\square_0$ and $\square_2$ intersect. (v) $\square_0, \ldots, \square_3$ that do not cover $\square$.

$\mu(\mathbb{C}, \mathbb{B}, \rho)$ can also be computed in $O(\log n)$ time. We now describe Ψ_0 and Ψ_2; Ψ_1 and Ψ_3 are symmetric to Ψ_0 and Ψ_2, respectively. Although Ψ_0 is a special case of Ψ_2, we first describe Ψ_0 and then describe how we extend it to construct Ψ_2.

Preprocessing $\mathbb{C}_0$ and $\mathbb{B}$. For simplicity, we replace each square C in $\mathbb{C}_0$ into a quadrant with apex $q(C)$ and containing p_0; Let $\mathbb{Q}$ be the resulting set of quadrants. We describe the data structure to preprocess $\mathbb{Q}$ and $\mathbb{B}$ to compute $\mu(\mathbb{Q}, \mathbb{B}, \rho)$ for a query rectangle $\rho \subseteq \square$. $\mathcal{U}(\mathbb{Q})$ can be maintained using the data structure by Overmars and van Leeuwen [14] but the interaction of $\mathbb{Q}$ and $\mathbb{B}$ makes the maintenance of $\mathcal{U}(\mathbb{Q}, \mathbb{B})$ difficult: a bar in $\mathbb{B}$ may intersect many edges of $\partial\mathcal{U}(\mathbb{Q})$, and many bars in $\mathbb{B}$ may intersect the same edge of $\mathcal{U}(\mathbb{Q})$; see Figure 11. The insertion or deletion of a corner square or a bar may thus cause many changes in the boundary structure of $\mathcal{U}(\mathbb{Q}, \mathbb{B})$, making it expensive to update $\mathcal{U}(\mathbb{Q}, \mathbb{B})$. We circumvent this problem by maintaining the edges of $\mathcal{U}(\mathbb{Q}, \mathbb{B})$ implicitly, as described below.

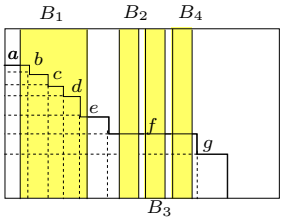

Figure 11. Interaction of $\mathbb{Q}$ and $\mathbb{B}$. $\Gamma = \langle a, \ldots, g \rangle$; b, c, d do not appear on $\Sigma(\Gamma, \mathbb{B})$; $\hat{\Sigma} = \langle \hat{a}, \hat{e}, \hat{f}.\hat{g} \rangle$.

Operations on a monotone chain. For a rectilinear xy-monotone chain $\Gamma \subset \square$, let $\mathcal{P}(\Gamma)$ be the staircase polygon formed by the set of points in $\square$ that lie below Γ. We represent Γ by its sequence of horizontal edges, and we will use Γ to denote both the chain as well as the sequence of its horizontal edges. For a subset $\mathcal{B} \subseteq \mathbb{B}$, let $\Sigma(\Gamma, \mathcal{B})$ be the sequence of horizontal edges of $\Gamma \setminus \mathcal{U}(\mathcal{B})$. An edge of Γ may get split into many edges in $\Sigma(\Gamma, \mathcal{B})$. We represent $\Sigma(\Gamma, \mathcal{B})$ implicitly: For an edge $e \in \Gamma$, define a 4-tuple $\hat{e} = \langle x^-(e), x^+(e), y(e), \varepsilon(e) \rangle$, where $x^-(e), x^+(e)$ are the x-coordinates of the left and right endpoints of e, $y(e)$ is the y-coordinate of e, and $\varepsilon(e) = |e \setminus \mathcal{U}(\mathcal{B})|$ is the *exposed* portion of e on Σ; $\hat{e}$ is the representation of e. We say that e *appears* in Σ if $\varepsilon(e) > 0$. We represent Σ as the sequence $\hat{\Sigma} = \langle \hat{e}_1, \hat{e}_2, \ldots, \rangle$, where $e_1, e_2, \ldots$ is the subsequence of the edges of Γ that appear in Σ. Note that $\mathrm{Area}(\mathcal{P}(\Gamma) \setminus \mathcal{U}(\mathcal{B})) = \sum_{\hat{e} \in \Sigma} \varepsilon(e) \cdot y(e)$. Abusing the notation slightly, we do not distinguish between Σ and its representation. For $A \subset \mathbb{R}^2$, let $\mathsf{X}(A)$ denote the x-projection of A. We perform the following operations on Σ.

(i) SPLIT(Σ, a): Given a rectilinear chain Σ and a value $a \geq 0$, let $\Sigma^+ = \langle e \in \Sigma \mid y(e) > a \rangle$ and $\Sigma^- = \langle e \in \Sigma \mid y(e) < a \rangle$. Split Σ into Σ^-, Σ^+. If Σ contains an edge e with $y(e) = a$, then the procedure returns $\hat{e}$ separately. See Figure 12.

(ii) MERGE(Σ^-, Σ^+, e): Let Σ^-, Σ^+ be two xy-monotone rectilinear and e a horizontal edge so that $\Sigma = \Sigma^- \circ \langle e \rangle \circ \Sigma^+$ is also a xy-monotone rectilinear chain. Given Σ^-, Σ^+, and $\hat{e}$, compute Σ. If the edge e is not specified or $\varepsilon(e) = 0$, the procedure computes $\Sigma_1 \circ \Sigma_2$.

(iii) QUERY(Σ, I): Given a chain Σ and an interval I, compute

$$\alpha(\Sigma, I) = \sum_{e \in \Sigma, \mathsf{X}(e) \subseteq I} \varepsilon(e) \cdot y(e).$$

Suppose $e_i, \ldots, e_j$ are the edges of Σ whose x-projections are contained in I. Then $\alpha(\Sigma, I)$ is the area of $(\mathcal{P}(\Gamma) \setminus \mathcal{U}(\mathbb{B})) \cap R_I$, where $R_I = [x^-(e_i), x^+(e_j)] \times [0, b]$.

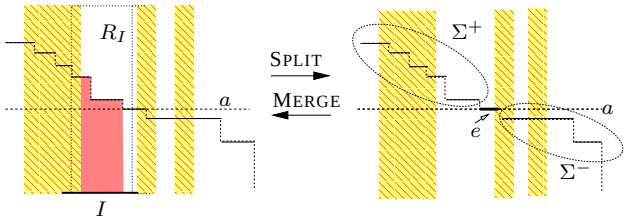

Figure 12. Operations on chains; I is a query interval, darker region is $(\mathcal{P}(\Gamma) \setminus \mathcal{U}(\mathbb{B})) \cap R_I$ (left).

By maintaining Σ in a height-balanced tree, which we denote by $\mathcal{H}(\Sigma)$, each of these operations can be performed in $O(\log n)$ time. Using the wall data structure, for an edge e of Γ, $\varepsilon(e)$ can be computed in $O(\log n)$ time.

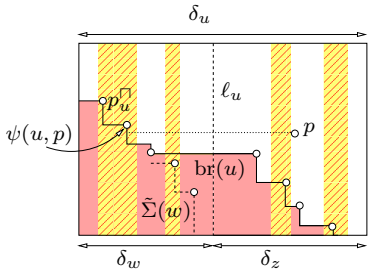

Figure 13. A node u and its children w and z. $\mathbb{B}^*(u)$ is the hatched area and $\mathcal{U}(\mathbb{Q}_u, \mathbb{B}^*_u, \square_u)$ is the darker region.

Structure of Ψ_0. Let $\mathcal{Q} = \{q(S) \mid S \in \mathbb{Q}\}$, and let $\mathcal{X} \subseteq [0, a]$ be the set of points in $\mathbb{R}$ that are x-coordinates of points in $\mathcal{Q}$ and vertices in $\mathbb{B}$. $\mathcal{X}$ partitions the interval $(0, a]$ into a set of atomic intervals; we regard each of these intervals semiclosed — the right

endpoint lies in the interval but the left endpoint does not lie in the interval. We build a height-balanced binary tree $\mathcal{T}$ on these atomic intervals. Each node $v \in \mathcal{T}$ is associated with an interval $\delta_v = (x_v^-, x_v^+]$ and a rectangle $\square_v = \delta_v \times [0, b]$. If v is the i-th leaf of $\mathcal{T}$, then δ_v is the i-th (semiclosed) atomic interval. For an interior node v, let $\delta_v = \delta_{\mathrm{L}(v)} \cup \delta_{\mathrm{R}(v)}$, $\square_v = \delta_v \times [0, b]$, and let ξ_v be the common endpoint of $\delta_{\mathrm{L}(v)}$ and $\delta_{\mathrm{R}(v)}$. Let $\ell_v : x = \xi_v$ be the separator line at v. For an interval $I \subseteq [0, a]$, let $\Xi(I)$ be the set of nodes $u \in \mathcal{T}$ for which $\mathrm{cl}\delta_u \subseteq I$ and $\mathrm{cl}\delta_{p(u)} \not\subseteq I$, and let $\mathcal{N}(I)$ be the set of ancestors of nodes in $\Xi(I)$. Similarly, we define $\Xi(B)$ and $\mathcal{N}(B)$ for a bar in $\mathbb{B}$. For a node u, let $\mathcal{Q}_u = \square_u \cap \mathcal{Q}$, $\mathbb{Q}_u = \{S \in \mathbb{Q} \mid q(S) \in \mathcal{Q}_u\}$, $\mathbb{B}_u \subseteq \mathbb{B}$ (resp. $\mathbb{B}_u^* \subseteq \mathbb{B}$) be the set of bars B such that $u \in \Xi(B)$ (resp. $u \in \Xi(B) \cup \mathcal{N}(B)$), and $p_u^{\sqcap}$ be the point of $\mathcal{Q}_u$ with the maximum y-coordinate. Let $\beta(u) = |\mathsf{X}(\mathbb{B}^*(u)) \cap \delta_v|$, which is $|\delta_u|$ if $|\mathbb{B}_u| > 0$, and $\beta(\mathrm{L}(u)) + \beta(\mathrm{R}(u))$ otherwise.

$\mathcal{U}(\mathbb{Q}_u, \square_u)$ is a staircase polygon. Let $\Gamma(u)$ be its rectilinear chain. Set $\Sigma(u) = \Sigma(\Gamma(u), \mathbb{B}_u^*)$, the portion of $\partial\mathcal{U}(\mathbb{Q}_u, \mathbb{B}_u^*, \square_u)$ that lies on $\Gamma(u)$. For an interior node u, let $\mathrm{br}(u)$ be the edge of $\Gamma(u)$ that intersects ℓ_u, and let $\tilde{\Sigma}(u) = \Sigma(u) \setminus \Sigma(p(u))$ (Figure 15 (i)). If $\varepsilon(\mathrm{br}(u)) > 0$, then $\mathrm{br}(u) \in \Sigma(u)$. For the root rt of $\mathcal{T}$, $\tilde{\Sigma}(\mathrm{rt}) = \Sigma(\mathrm{rt})$. At each node u we store: $\tilde{\Sigma}(u)$, $\mathrm{br}(u)$, $\beta(u)$, $|\mathbb{B}_u|$, and $p^{\sqcap}(u)$. For a node u and for a point p, let $\psi(u, p)$ denote the rightmost point in $\mathcal{Q}_u$ whose y-coordinate is greater than that of p; $\psi(u, p)$ can be computed in $O(\log n)$ time.

Updating Ψ_0. Using the following two procedures, each of which takes $O(\log n)$ time, a corner or a bar can be inserted/deleted in $O(\log^2 n)$ time. Let u be a node of $\mathcal{T}$, and let w and z be the left and right child of u, respectively.

ASCEND(u): It computes $\Sigma(u), \tilde{\Sigma}(w), \tilde{\Sigma}(z)$ from $\Sigma(w)$ and $\Sigma(z)$. If $\beta(u) = \delta_u$, then we set $\Sigma(u) = \emptyset$, $\tilde{\Sigma}(w) = \Sigma(w)$, and $\tilde{\Sigma}(z) = \Sigma(z)$, and we stop. So assume that $\beta(u) < \delta_u$. Let $p_R = p^{\sqcap}(z)$. We compute $p_L = \psi(w, p_R)$. If p_L does not exist, then the left endpoint of $\mathrm{br}(u)$ is $(x_u^-, y(p_R))$, otherwise it is $(x(p_L), y(p_R))$. The right endpoint of $\mathrm{br}(u)$ is p_R. Let $I = [x(p_L), x(p_R)]$ be the projection of $\mathrm{br}(u)$. We compute $\varepsilon(\mathrm{br}(u))$, in $O(\log n)$ time. If $\varepsilon(\mathrm{br}(u)) = 0$, we set $\mathrm{br}(u) = \emptyset$. Next, we call SPLIT($\Sigma(w)$, $y(p_R)$). Let Σ^-, Σ^+ be the subchains returned by the procedure. We set $\tilde{\Sigma}(w) = \Sigma^+$. If $\Sigma(z)$ contains the horizontal edge at height $y(p_R)$, we delete it, and we set $\Sigma(u)$ = MERGE(Σ^-, $\Sigma(z)$, $\mathrm{br}(u)$).

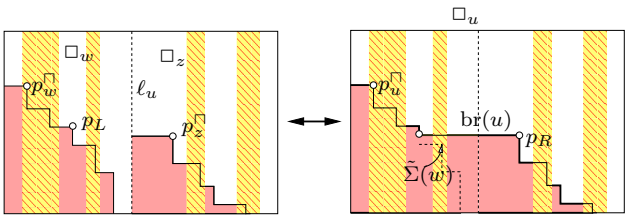

Figure 14. ASCEND(u) and DESCEND(u) procedures.

DESCEND(u): Assuming $\Sigma(u)$ is at our disposal, this procedure computes $\Sigma(w)$ and $\Sigma(z)$. If $\beta(u) = \delta_u$, then we set $\Sigma_w = \tilde{\Sigma}_w$ and $\Sigma(z) = \tilde{\Sigma}(z)$ and stop. Otherwise, let $p_R = p^{\sqcap}(z)$. We call SPLIT($\Sigma(u)$, $y(p_R)$). Let Σ^-, Σ^+ be the two chains returned by this procedure; if $\varepsilon(\mathrm{br}(u)) > 0$, then the procedure also returns $\mathrm{br}(u)$. We set $\Sigma(w)$ = MERGE(Σ^-, $\tilde{\Sigma}(w)$). Let $e = \mathrm{br}(u) \cap \square_z$. We compute $\varepsilon(e)$ and set $\Sigma(z)$ = MERGE($\emptyset$, Σ^+, e).

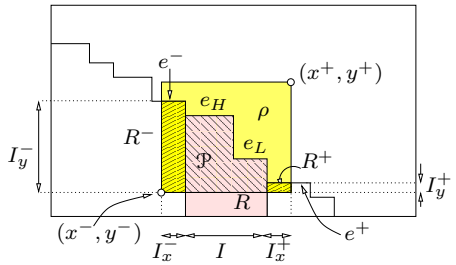

Figure 15. Answering a query on Ψ_0.

Answering a query. Let $\rho = [x^-, x^+] \times [y^-, y^+] \subseteq \square$ be a query rectangle. We wish to compute $\mu(\mathbb{Q}, \mathbb{B}, \rho)$. Let rt be the root of $\mathcal{T}$, and let $\Gamma = \Gamma(\mathrm{rt})$ and $\Sigma = \Sigma(\mathrm{rt})$. By searching with ρ in $\mathcal{H}(\Sigma)$, stored at rt, we find the highest (resp. lowest) edge e_H (resp. e_L) of Γ that appears in Σ and that lies completely inside ρ. Let $\mathcal{P}$ be the staircase polygon formed by the edges $e_H, \ldots, e_L$ of Σ. Let e^- be the predecessor of e_H and e^+ the successor of e_L in Σ. Set $I = [x^-(e_H), x^+(e_L)]$, $R = I \times [0, y^-]$, $I_x^- = \mathsf{X}(e^-) \cap [x^-, x^+]$, $I_x^+ = \mathsf{X}(e^+) \cap [x^-, x^+]$, $I_y^- = [0, y(e^-)] \cap [y^-, y^+]$, and $I_y^+ = [0, y(e^+)] \cap [y^-, y^+]$. Set $R^- = I_x^- \times I_y^-$ and $R^+ = I^+ \times I_y^+$. See Figure 15 (ii). Let A (resp. A^-, A^+) denote the area of R (resp. R^-, R^+) not lyig in $\mathcal{U}(\mathbb{B})$. Then

$$\begin{aligned}\mu(\mathbb{Q}, \mathbb{B}, \rho) &= \alpha(\Sigma, I) - \mathsf{A} + \mathsf{A}^- + \mathsf{A}^+ \\ &= \alpha(\Sigma, I) - \varepsilon(I) y_L + \varepsilon(I_x^-)|I_y^-| + \varepsilon(I^+)|I_y^+|.\end{aligned}$$

Each of these quantities can be computed in $O(\log n)$ time either querying $\mathcal{H}(\Sigma)$ or the wall data structure. Hence, $\mu(\mathbb{Q}, \mathbb{B}, \rho)$ can be computed in $O(\log n)$ time.

Preprocessing $\mathbb{C}_0, \mathbb{C}_2$, and $\mathbb{B}$. The overall structure of Ψ_2 is similar to that of Ψ_0, but it is more complex because of the interaction between $\mathbb{C}_0$ and $\mathbb{C}_2$. To keep the presentation analogous to Ψ_0, we describe the data structure for computing $\mu(\mathbb{C}_0, \mathbb{C}_2 \cup \mathbb{B}, \rho)$; $\mu(\mathbb{C}_2, \mathbb{C}_0 \cup \mathbb{B}, \rho)$ can be computed by reversing the directions of $(+x)$- and $(+y)$-axes and constructing the data structure in this new coordinate frame.

Let $\mathbb{Q}^+$ (resp. $\mathbb{Q}^-$) be the set of quadrants corresponding to the squares in $\mathbb{C}_0$ and $\mathbb{C}_2$. For a query rectangle $\rho \subseteq \square$, we wish to compute $\mu(\mathbb{Q}^+, \mathbb{Q}^- \cup \mathbb{B}, \rho)$. We begin by describing the representation of $\mathcal{U}(\mathbb{Q}^+, \mathbb{Q}^- \cup \mathbb{B}, \square)$ and the operations performed on it.

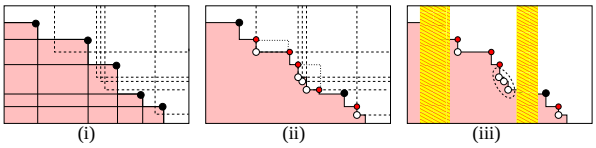

Figure 16. Representation of $\Sigma(\mathbb{Q}^+, \mathbb{Q}^-, \mathbb{B})$. (i) $\mathcal{U}(\mathbb{Q}^+)$. (ii) $\mathcal{U}(\mathbb{Q}^+, \mathbb{Q}^-)$, black (resp. white) points are vertices of squares in $\mathbb{Q}^+$ (resp. $\mathbb{Q}^-$), and gray vertices are mixed vertices. (iii) $\mathcal{U}(\mathbb{Q}^+, \mathbb{Q}^- \cup \mathbb{B})$; edges inside the ellipse form a superedge.

Representation of $\mathcal{U}(\mathbb{Q}^+, \mathbb{Q}^- \cup \mathcal{B}, \square)$. $\mathcal{U}(\mathbb{Q}^+, \mathbb{Q}^-, \square)$ is a staircase polygon, with four types of vertices: vertices of $\mathbb{Q}^+$, vertices of $\mathbb{Q}^-$, intersection points of two edges of $\mathbb{Q}^+$ (or $\mathbb{Q}^-$), and intersection points of an edge of $\partial\mathcal{U}(\mathbb{Q}^+)$ with an edge of $\partial\mathcal{U}(\mathbb{Q}^-)$. The first and the last type are convex vertices, the second type are reflex vertices, and the third type can be both convex and reflex. We refer to the last ones as *mixed* vertices. An edge of $\mathcal{U}(\mathbb{Q}^+, \mathbb{Q}^-)$ is called *purple* if one of its endpoints is a mixed vertex, *blue* if it is an edge of $\mathcal{U}(\mathbb{Q}^+)$, and *red* if it is an edge of $\mathcal{U}(\mathbb{Q}^-)$. Let

$\Gamma(\mathbb{Q}^+, \mathbb{Q}^-)$ be the rectilinear chain of $\mathcal{U}(\mathbb{Q}^+, \mathbb{Q}^-, \square)$. As in the previous case, Γ is represented as the sequence of its horizontal edges. Let $\Sigma := \Sigma(\mathbb{Q}^+, \mathbb{Q}^-, \mathbb{B})$ be the sequence of edges of $\Gamma(\mathbb{Q}^+, \mathbb{Q}^-) \setminus \mathcal{U}(\mathcal{B})$, and let $\Sigma(\mathbb{Q}^-, \mathbb{B})$ be the sequence of edges in $\Gamma(\mathbb{Q}^-) \setminus \mathcal{U}(\mathcal{B})$. Σ is represented implicitly using $\Sigma(\mathbb{Q}^-, \mathbb{B})$: Let $E = \langle e_1, e_2, \ldots, e_k \rangle$ be the sequence of edges of $\Gamma(\mathbb{Q}^+, \mathbb{Q}^-)$ that appear in Σ. Every red edge in E also appears in $\Sigma(\mathbb{Q}^-, \mathbb{B})$. We call a contiguous subsequence $e_i, e_{i+1}, \ldots, e_j$ of red edges in E as a *super-edge* σ_{ij} (see Figure 16 (iii)), and represent it as a five-tuple:

$$\hat{\sigma}_{ij} = \langle x^-(\sigma_{ij}), x^+(\sigma_{ij}), y^-(\sigma_{ij}), y^+(\sigma_{ij}), \varepsilon(\sigma_{ij}) \rangle,$$

where $x^-(\sigma_{ij}) = x^-(e_i)$, $x^+(\sigma_{ij}) = x^+(e_j)$, $y^-(\sigma_{ij}) = y(e_j)$, $y^+(\sigma_{ij}) = y(e_i)$, and $\varepsilon(\sigma_{ij}) = \sum_{l=i}^{j} \varepsilon(e_l) y(e_l)$. We represent $\Sigma = \hat{\Sigma} = \langle \gamma_1, \gamma_2, \ldots, \gamma_s \rangle$ as a sequence of blue edges, purple edges, and super-edges; either $\gamma_i = \hat{e}_j$ for a blue or purple edge in E or $\gamma_i = \hat{\sigma}_{jl}$ for a super-edge σ_{jl}. By definition, a super-edge may consist of a single red edge. We assume that each super-edge in $\hat{\Sigma}$ is maximal in the sense that if γ_i is a super-edge then γ_{i+1} is not a super-edge. By storing both $\Sigma(\mathbb{Q}^+, \mathbb{Q}^-, \mathbb{B})$ and $\Sigma(\mathbb{Q}^-, \mathbb{B})$ in height balanced binary trees, each of the following operations can be performed in $O(\log n)$ time.

(i) SPLIT(Σ, a): Given a rectilinear chain Σ and a value $a \geq 0$, let $\Sigma^+ = \langle e \in \Sigma \mid y(e) > a \rangle$ and $\Sigma^- = \langle e \in \Sigma \mid y(e) < a \rangle$. Split Σ into Σ^-, Σ^+. If Σ contains an edge e with $y(e) = a$, then the procedure returns $\hat{e}$ separately. If there is a super-edge σ_{jl} in $\hat{\Sigma}$ with $a \in [y^-(\sigma_{jl}), y^+(\sigma_{jl})]$, then the procedure splits the super-edge into two: one of which lies above a and appears at the end of Σ^-, and the other lies below a and appears at the beginning of Σ^+.

(ii) MERGE(Σ^-, Σ^+, e): Let Σ^-, Σ^+ be two xy-monotone rectilinear chains and e a horizontal edge so that $\Sigma = \Sigma^- \circ \langle e \rangle \circ \Sigma^+$ is also a xy-monotone rectilinear chain. Given Σ^-, Σ^+, and $\hat{e}$, compute Σ. If the edge e is not specified or $\varepsilon(e) = 0$, the procedure computes $\Sigma^- \circ \Sigma^+$. Finally, if there are two consecutive super-edges in $\hat{\Sigma}$, then we merge them.

(iii) QUERY(Σ, I): Given a chain Σ and an interval I, compute

$$\alpha(\Sigma, I) = \sum_{e \in \Sigma, \mathsf{X}(e) \subseteq I} \varepsilon(e) \cdot y(e).$$

Since some of the edges of Σ are implicitly represented as super-edges, computing $\alpha(\Sigma, I)$ now is more involved than for Ψ_0 but nevertheless can be computed in $O(\log n)$ time.

Structure of Ψ_2. Let $\mathcal{Q}^- = \{q(S) \mid S \in \mathbb{Q}^-\}$ and $\mathcal{Q}^+ = \{q(S) \mid S \in \mathbb{Q}^+\}$. Let $\mathcal{X} \subseteq [0, a]$ be the set of x-coordinates of points in $\mathcal{Q}^+ \cup \mathcal{Q}^-$ and edges in $\mathbb{B}$. We construct a binary tree $\mathcal{T}$ as for Ψ_0. For each node u, we define $\mathbb{Q}_u^+, \mathbb{Q}_u^-, \mathbb{B}_u, \mathbb{B}_u^*$ in a manner similar to Ψ_0. Let $\Gamma^+(u) = \Gamma(\mathbb{Q}_u^+, \mathbb{Q}_u^-)$, $\Gamma^-(u) = \Gamma(\mathbb{Q}_u^-)$, $\Sigma^+(u) = \Sigma(\mathbb{Q}_u^+, \mathbb{Q}_u^-, \mathbb{B}_u^*)$, and $\Sigma^-(u) = \Sigma(\mathbb{Q}_u^-, \mathbb{B}_u^*)$. In addition, let $\tilde{\Sigma}^+(u) = \Sigma^+(u) \setminus \Sigma^+(p(u))$ and $\tilde{\Sigma}^-(u) = \Sigma^-(u) \setminus \Sigma^-(p(u))$. We define $\mathrm{br}^+(u)$ (resp. $\mathrm{br}^-(u)$) be the edge of $\Gamma^+(u)$ (resp. $\Gamma^-(u)$) that intersects ℓ_u. At each node u, we store compact representations of $\tilde{\Sigma}^+(u)$ and $\tilde{\Sigma}^-(u)$, as described above, and we also store $\mathrm{br}^+(u)$ and $\mathrm{br}^-(u)$.

As for Ψ_0, we again describe ASCEND, DESCEND procedures to compute $\Sigma^+(u)$ from $\Sigma^+(w)$ and $\Sigma^+(z)$, and vice-versa, where w and z are the children of v. Each of them takes $O(\log n)$ time, and they are used to insert and delete corners and bars. Putting everything together, we conclude the following:

LEMMA 5.1. *A set $\mathbb{C}$ of corner squares and a set $\mathbb{B}$ of bars can be maintained in a dynamic data structure under insertion/deletion so that for a query rectangle ρ, $\mu(\mathbb{C}, \mathbb{B}, \rho)$ can be computed in $O(\log n)$ time. The insertion and deletion operations take $O(\log^2 n)$ time.*

References

[A1] http://en.wikipedia.org/wiki/Klee's_measure_problem

[A2] P. K. Agarwal, H. Kaplan, and M. Sharir, Computing the volume of the union of cubes, *Proc. 23rd Annu. Sympos. Comput. Geom.*, 2007, 294–301.

[A3] M. de Berg, M. van Kreveld, M. Overmars and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*, 2nd edition, Springer Verlag, Heidelberg, 2000.

[A4] J. L. Bentley, Algorithms for Klee's rectangle problems. Unpublished notes, Computer Science Department, Carnegie Mellon University, 1977.

[A5] J.D. Boissonnat, M. Sharir, B. Tagansky and M. Yvinec, Voronoi diagrams in higher dimensions under certain polyhedral distance functions, *Discrete Comput. Geom.* 19 (1998), 485–519.

[A6] K. Bringmann, Klee's measure problem on fat boxes, *Proc. 26th Annu. Sympos. Comput. Geom.*, 2010, to appear.

[A7] T. M. Chan, A (slightly) faster algorithm for Klee's measure problem, *Comput. Geom.: Theory Appls.* 43 (2010), 243-250.

[A8] E. Chen and T. M. Chan, Space-efficient algorithms for Klee's measure problem, *Proc. 17th Canadian Conf. Comput. Geom.*, 2005.

[A9] B. S. Chlebus, On the Klee's measure problem in small dimensions, *Proc. 25th Conf. Current Trends in Theory and Practice of Informatics*, 1998, 304-311.

[A10] M. Dickerson, C. A. Duncan, and M. T. Goodrich, K-d trees are better when cut on the longest side, *Proc. 8th European Sympos. Algorithms*, 2000, 179190.

[A11] M. L. Fredman and B. Weide, The complexity of computing the measure of $\bigcup[a_i, b_i]$, *Commun. ACM* 21 (1978), 540–544.

[A12] J. van Leeuwen and D. Wood, The measure problem for rectangular ranges in d-space, *J. Algorithms* 2 (1981), 282–300.

[A13] V. Klee, Can the measure of $\bigcup[a_i, b_i]$ be computed in less than $O(n \log n)$ steps? *Amer. Math. Monthly* 84 (1977), 284–285.

[A14] M. Overmars and J. Leeuwen, Maintenance of configurations in the plane, *J. Comput. Syst. Sci.* 23 (1981), 166–204.

[A15] M. Overmars and C.K. Yap, New upper bounds in Klee's measure problem, *SIAM J. Comput.* 20 (1991), 1034–1045.

[A16] S. Suzuki and T. Ibaki, An average running time analysis of a backtracking algorithm to calculate the measure of the union of hyperrectangles in d dimensions, *Proc. 16th Canadian Conf. Comput. Geom.*, 2004, 196–199.