Box-Trees for Collision Checking in Industrial Installations

Herman J. Haverkort

Mark de Berg

Joachim Gudmundsson

institute of information and computing sciences, utrecht university

technical report UU-CS-2002-027

www.cs.uu.nl

Box-Trees for Collision Checking in Industrial Installations*

H. J. Haverkort M. de Berg J. Gudmundsson

Institute of Information and Computing Sciences
Utrecht University, PO Box 80.089, 3508 TB Utrecht
{herman,markdb,joachim}@cs.uu.nl

Abstract

A box-tree is a bounding-volume hierarchy that uses axis-aligned boxes as bounding volumes. We describe a new algorithm to construct a box-tree for objects in a 3D scene, and we analyze its worst-case query time for approximate range queries. If the input scene has certain characteristics that we derived from our application—collision detection in industrial installations—then the query times are polylogarithmic, not only for searching with boxes but also for range searching with other constant-complexity ranges.

Keywords: bounding-volume hierarchy, box-tree, window query, orthogonal range query, slicing number

1 Introduction

Motivation. Collision checking is an important operation in all applications where objects move around in a 3D scene—virtual reality, computer animation, and robotics are obvious examples. A popular way of doing collision checking is the following two-phase approach. In the first phase, the *filtering phase*, one finds all primitive objects in the scene whose bounding box intersects the query object (or its bounding box). In the second phase, the *refinement phase*, one tests for each of these primitives (if any) whether it actually intersects the object. To speed up the filtering phase, the set S of bounding boxes of the primitives in the scene is often stored in a bounding-volume hierarchy. This is a binary tree whose leaves store the boxes in S, and where each internal node ν stores the bounding box $b(\nu)$ of all boxes stored in the subtree rooted at ν. We call such a tree a *box-tree*; sometimes it is more precisely called an *axis-aligned-bounding-box tree*, or *AABB-tree* for short. A query with a query range Q is performed by traversing the tree in a top-down manner, only visiting nodes ν such that $b(\nu)$ intersects Q. This way we end up exactly in the leaves storing boxes that intersect Q.

The query time in a box-tree is determined by the number of nodes visited, and the goal is therefore to organize the tree in such a way that this number is kept as small as possible. Agarwal *et al.* [1] recently showed that a box-tree exists that has $O(n^{2/3} + k)$ query time for ranges that are axis-parallel boxes, where n is the total number of boxes in S and k is the number of boxes intersecting the query range. This bound is rather disappointing: if

*The work by H.H. is supported by the Netherlands' Organization for Scientific Research (NWO). The work by J.G. is supported by the Swedish Foundation for International Cooperation in Research and Higher Education.

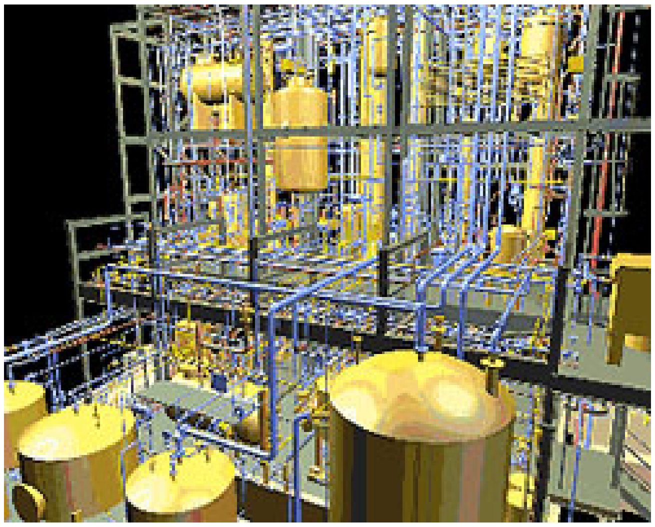

Figure 1: CAD model of a carbon black unit. Designed by OLAJTERV Process and Energy, Hungary.

the query time would really be that bad, box-trees would not be used so much in practice. Unfortunately, the bound is optimal. Agarwal *et al.* prove that there are sets of input boxes for which the worst-case query time of any box-tree is $\Omega(n^{2/3} + k)$.[1] This is the starting point for our work: we want to understand what makes box-trees perform well in practical applications even though in theory they may perform badly.

The application we have in mind comes from the MOLOG project [9]. The goal of this project is to add motion support to CAD systems used to design large industrial installations, such as depicted in Fig. 1.

Adding motion support will help the designer of an industrial installation to decide whether it will be possible to move certain parts out of the installation, for maintenance or replacement. The approach taken in the MOLOG project is based on the *probabilistic path planner* [2, 8, 12], a technique for motion planning that has proved very successful in many applications. A basic test performed many times by the probabilistic path planner is collision checking: given a query object—the object for which we are planning a motion, at a certain position and orientation—does it collide with the CAD model? We can now state the goal of this paper as follows: we want to design a provably efficient box-tree for storing scenes that are CAD models of large industrial installations.

Further background. The lower bounds of Agarwal *et al.* mentioned earlier imply that, to be able to design provably efficient box-trees for CAD models of large industrial installations, we have to make use of the properties of the bounding boxes of the primitives in such CAD models. The *realistic input models* [4] suggested in the literature do not seem applicable in our setting: the industrial installation of Fig. 1, for instance, contains many long and thin pipes that are relatively close together. But if we forget about the pipes, the scene seems to be well-behaved. Hence, the assumption we make is that the boxes in S can be partitioned into

[1]In general, the worst-case query time of a box-tree in d-dimensional space is $\Theta(n^{1-1/d} + k)$. In this paper we focus on 3-dimensional box-trees, because this is most natural in our application.

two subsets, one containing only long and thin (almost) disjoint pipes, and one forming a low-density scene [4]. Here a pipe is defined to be an axis-aligned box whose shortest dimension is at most a constant β times shorter than its middle dimension—see Section 2.3 for formal definitions of these concepts. It is important to note that our algorithm to construct the box-tree does not need this assumption; we only use it in the analysis.

Unfortunately, with the assumption just stated one still cannot prove good bounds: the $\Omega(n^{2/3} + k)$ lower bound for range queries with a box even holds if the input consists of disjoint unit squares arranged in a grid-like fashion. Therefore we analyze approximate range queries. More precisely, instead of the parameter k in the time bound, we use k_ϵ, which is the number of boxes intersecting the extended range Q_ϵ. For a given $\epsilon > 0$, the extended range Q_ϵ is the set of points lying at L_∞-distance at most ϵw from Q, where w is the length of the longest edge of Q. The expectation is that in practice k_ϵ will not be much larger than k for moderately small ϵ, at least when the query range is rather fat. Note that in our application, the query range is (the bounding box of) an object for which we are planning a motion. If the object is a forklift truck or some other car-like device, its bounding box is likely to be fat. The concept of approximate range searching was also used by Arya and Mount [3], who considered approximate range queries on a set of points. The parameter ϵ is not used by our query algorithm—the algorithm still visits only nodes whose bounding boxes are intersected by Q—but it is only used in the analysis. (So perhaps approximate range searching is a slight misnomer.)

Our results. We describe a new, simple algorithm to construct a box-tree on a set of boxes in 3D. This algorithm generalizes the 2D kd-interval tree described by Agarwal *et al.* [1] to 3D, with one additional crucial twist: We partition the input boxes into three subsets, according to the orientation of their longest edge, and construct separate box-trees for these subsets; these subtrees are then combined to form the final tree. Our main contribution is a rather involved analysis of the worst-case query time of this box-tree in the setting described above, showing it is polylogarithmic. More precisely, we prove that the number of visited nodes is $O((1/\epsilon + \lambda) \log^4 n + k_\epsilon)$, where λ is a constant depending on the scene parameters. Typically, λ will only be large if the input contains many flat 'plates' that are very close together—see section 2.2 for details. Note that the choice of ϵ determines a trade-off between the terms in the bound: choosing ϵ small will cause a large factor in the first term, but k_ϵ will be close to k. On the other hand, choosing ϵ big keeps the first term down, but k_ϵ might grow to $O(n)$. In each situation, the best bound on the query time will be the lowest bound over all possible values of ϵ; in other words: $O(\min_{0<\epsilon\leq 1}\{(1/\epsilon)(1/\epsilon + \lambda) \log^4 n + k_\epsilon\})$.

This result should be compared with the results for approximate range searching in a set of points in 3-space. Here, the best result that uses boxes as bounding volumes is by Dickerson *et al.* [5], who show that the query time in a so-called *longest-side-first kd-tree* is $O(\min_{0<\epsilon\leq 1}\{(1/\epsilon^2) \log^3 n + k_\epsilon\})$. Our result is more general than this, as we store boxes instead of points and the bounds we get are only slightly worse.

We also introduce a variant of the box-tree, where an interior node uses a different type of bounding volume: instead of a bounding box, it can use a donut-like shape, namely the difference of two boxes. This was inspired by Arya and Mount [3], who show that a similar structure for points—they call it BBD-tree—outperforms kd-trees in the worst case: the time for approximate range queries in 3D in a BBD-tree is $O(\min_{0<\epsilon\leq 1}\{\log n + (1/\epsilon)^2 + k_\epsilon\})$. (The same result can be obtained using BAR-trees [6, 7]. BAR-trees use convex, but not necessarily axis-parallel, bounding volumes whose facets have a bounded number of different orientations.) We prove that a similar improvement is possible in our case: our BBD-interval tree has a worst-case query time of $O(\min_{0<\epsilon\leq 1}\{\log^3 n + (\lambda/\epsilon)\log^2 n + (1/\epsilon^2)\log n + k_\epsilon\})$.

Finally, we extend our results to constant-complexity query ranges of arbitrary shape, showing that the time for approximate queries with such ranges is $O(\min_{0<\epsilon\leq 1}\{(\lambda/\epsilon^2)\log^4 n + k_\epsilon\})$ in a LSF-interval tree and $O((\log^3 n + \lambda \log^2 n)/\epsilon^2 + k_\epsilon)$ in a BBD-interval tree. Similar extensions were given for the case of point data by Dickerson *et al.* [5] and by Arya and Mount [3], who achieved query times of $O((\log^3 n)/\epsilon^3 + k_\epsilon)$ and $O(\log n + 1/\epsilon^3 + k_\epsilon)$, respectively. Note that the dependency on ϵ in our bounds is better by a factor of $O(1/\epsilon)$; only for convex ranges they are able to prove the dependency we get for general ranges. Our proof technique also applies to their structures, which implies an improvement of their query time by a factor of $O(1/\epsilon)$ for non-convex ranges.

2 The LSF-interval tree

In this section we first describe how to construct a kd-interval tree with longest-side-first splitting, or LSF-interval tree for short, for a set of boxes in 3-space. After that we analyse its performance for approximate range queries.

2.1 The construction

Our 3-dimensional LSF-interval tree is a generalisation of the 2-dimensional kd-interval tree with longest-side-first splitting as described by Agarwal *et al.* [1]. In fact, the 2-dimensional substructures in our 3-dimensional structure are basically their 2-dimensional structures.

Our construction algorithm takes as input a set of 3-dimensional axis-parallel boxes and their joint bounding box. The algorithm then works top-down, recursively constructing subtrees on subsets of the input. In a generic step of the construction, we have as input a set S of 3-dimensional axis-parallel boxes and a *defining region* R. The construction is started with the full input set as input and the bounding box of the entire scene as defining region. In the recursive steps, the defining regions can be axis-parallel boxes, rectangles, line segments, or points. Each input box $b \in S$ will intersect R; more precisely, the defining regions will always be such that if $\mathrm{aff}(R)$ denotes the affine hull of R, then $b \cap \mathrm{aff}(R) \subset R$. If the defining region R is d-dimensional, for some $d \in \{0, 1, 2, 3\}$, then we call the subtree storing S a *d-LSF-interval tree*, and we call its root a *d-node*.

We will now describe an algorithm to construct a d-LSF-interval tree for a set S of input boxes and a defining region R. The algorithm produces a tree whose nodes have degree at most nine; conversion to a binary tree can easily be done and does not affect the asymptotic bounds.

We proceed as follows:

1. We create a root node ν, storing the bounding box $b(\nu)$ of the boxes in S.
2. For each of the six directions $+x$, $-x$, $+y$, $-y$, $+z$, and $-z$ we take the box in S extending farthest in that direction. Each of these at most six boxes is stored in a separate leaf, called a *priority leaf*, immediately below the root node ν. Let S' denote the set of remaining boxes. Assume S' is non-empty; otherwise we are done.

3. If $d = 0$, we recursively build a 0-LSF-interval tree for S' using the point R as defining region, and we make the root of this tree a child of ν. (In fact, for $d = 0$, building a cs-priority-box-tree [1] could make a better choice, but in our analysis the better performance of a cs-priority-box-tree would be overshadowed by other terms. In the analysis presented in this paper, we only need the priority leaves, and the division of boxes among the children does not matter.)

   Otherwise, if $d > 0$, let e be a longest edge of R, where $e = R$ if R is a line segment. Let h be a plane orthogonal to e. Define h^- to be the halfspace on one side of h, and h^+ to be the halfspace on the other side of h. Define S^- to be the subset of boxes in S' lying completely in h^-, S^+ to be the subset of boxes in S' lying completely in h^+, and $S^\times$ to be the subset of boxes intersecting h. We choose h such that $|S^-| < |S'|/2$ and $|S^+| \leq |S'|/2$. We then recursively construct three subtrees whose roots become children of the root node ν:

   - The subset S^- is stored in a d-LSF-interval tree with $R \cap h^-$ as defining region.
   - The subset S^+ is stored in a d-LSF-interval tree with $R \cap h^+$ as defining region.
   - The subset $S^\times$ is stored in a $(d-1)$-LSF-interval tree with $R \cap h$ as defining region.

We could start the construction with the entire input set S and any box R completely containing S as defining region. To achieve good performance, however, we first need to apply one simple but crucial step: we divide S into three 'oriented' subsets S_x, S_y, and S_z, where S_x, S_y and S_z contain all boxes whose longest edges are parallel to the x-axis, y-axis and z-axis, respectively, with ties broken arbitrarily. We then build an LSF-interval tree for each of these three subsets separately, and combine them at the top level. For each of the subsets, we say that the *primary axis* is the axis that corresponds to the orientation of the longest edges of the boxes in the set; the other axes are called *secondary axes*.

2.2 Analysis for box-intersection queries

We will analyse the query time in 3-dimensional LSF-interval trees for a box-intersection query in the subtree constructed for S_x. The analysis for S_y and S_z is similar; therefore, the asymptotic bounds we obtain hold for the entire tree as well. Recall that a query with a range Q visits all nodes ν whose bounding box $b(\nu)$ intersects Q. In the analysis, however, we work with a slightly extended range Q_ϵ, and we will charge the visiting of some of the nodes to 'approximate answers', that is, to input boxes intersecting Q_ϵ.

In the analysis we will use the following notation:

Q: the query range;

$w = w(Q)$: the length of the longest edge of the query range;

$\epsilon > 0$: the factor determining the size of the extended query range; to simplify the formulae we assume that $\epsilon \leq 1$, although the analysis can easily be adapted to values greater than 1. Our analysis holds for any $0 < \epsilon \leq 1$. Since ϵ is only used in the analysis and not by the algorithm, this implies that the actual query time is bounded by the minimum over all ϵ with $0 < \epsilon \leq 1$.

Q_ϵ: the *extended query range*, which consists of Q and all points within a distance ϵw from Q in the L_∞-metric;

k_ϵ: the number of input boxes intersecting the extended query range Q_ϵ; by $k_\epsilon(\mathcal{T})$ we will denote the number of input boxes in a subtree $\mathcal{T}$ that intersect Q_ϵ.

We also use a parameter that describes certain properties of the distribution of the input boxes over the space.

$\lambda \geq 1$: the *slicing number* of S, defined as follows. Let the slicing number λ_C of S with respect to a cube C be the maximum number of input boxes that intersect four parallel edges of C; then the overall slicing number λ is the maximum value of λ_C over all possible cubes C. Note that a box also intersects an edge if it fully contains that edge. Hence, λ is also an upper bound on the *stabbing number* σ of S, which is defined as the largest number of input boxes with a non-empty common intersection.

At the end of this section, we will show that if the input consists of a set of pipes with small stabbing number, together with a set of arbitrary boxes with low density, the complete input set will have low slicing number.

We will do the analysis bottom-up, first analysing the query time in 1-dimensional subtrees, then in 2-dimensional subtrees, then in 3-dimensional subtrees. We will denote the subtree we are analyzing by $\mathcal{T}$, and its defining region by $R(\mathcal{T})$. The subtree rooted at a node ν is denoted by $\mathcal{T}_\nu$. Sometimes we will speak of the defining region $R(\nu)$ of a node ν, which is simply the defining region $R(\mathcal{T}_\nu)$ of its subtree.

Before we proceed we state a lemma that we will need at various occasions.

Lemma 2.1 *Let $\mathcal{T}$ be a d-dimensional LSF-interval-tree and let C be a k-dimensional cube, with $1 \leq k \leq d \leq 3$. Then there are only $O(\log^{k-1} n)$ d-nodes in $\mathcal{T}$ whose defining regions are disjoint and intersect opposite facets of C.*

Proof: The d-nodes in an LSF-interval tree basically form a d-dimensional longest-side-first kd-tree. Hence, the result follows from Lemma 6.6 in Duncan's thesis [6]. This lemma is only stated for the case $k = d$, but the proof holds for $k < d$ as well. □

2.2.1 1-dimensional subtrees

In a 1-dimensional subtree $\mathcal{T}$, the defining region $R(\mathcal{T})$ is a line segment that intersects all input boxes stored in $\mathcal{T}$. The worst-case query time in $\mathcal{T}$ depends on the relation of $R(\mathcal{T})$ to the query range. In particular, we distinguish three cases, depending on how many of the two axis-parallel planes containing $R(\mathcal{T})$ intersect Q_ϵ.

Case 1: Two planes containing $R(\mathcal{T})$ intersect Q_ϵ. This case is illustrated in Fig. 2. Parts (a) and (b) of the figure correspond to part (i) in the lemma below, part (c) to part (ii).

Lemma 2.2 *Let $\mathcal{T}$ be a 1-LSF-interval tree storing n boxes. Suppose we query $\mathcal{T}$ with a box Q such that both axis-parallel planes containing $R(\mathcal{T})$ intersect Q_ϵ.*

(i) If the axis-parallel projection of Q_ϵ onto the line containing $R(\mathcal{T})$ contains at least one endpoint of $R(\mathcal{T})$, we visit $O(k_\epsilon(\mathcal{T}))$ nodes.

(ii) Otherwise, we visit $O(\log n + k_\epsilon(\mathcal{T}))$ nodes.

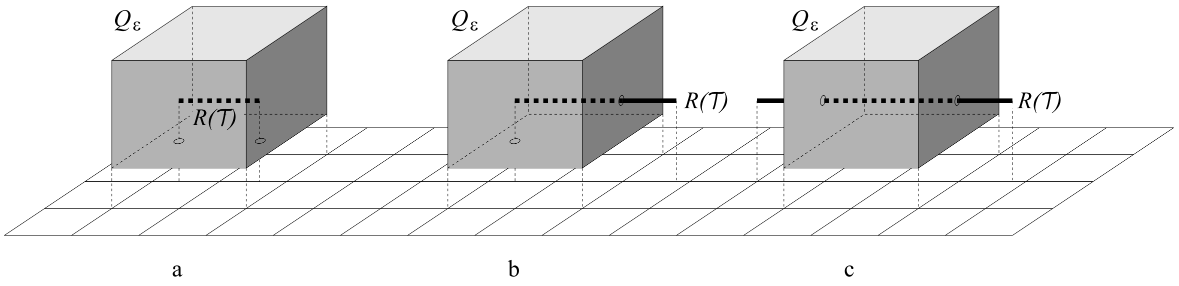

Figure 2: Two planes containing the line segment $R(\mathcal{T})$ intersect Q_ϵ.

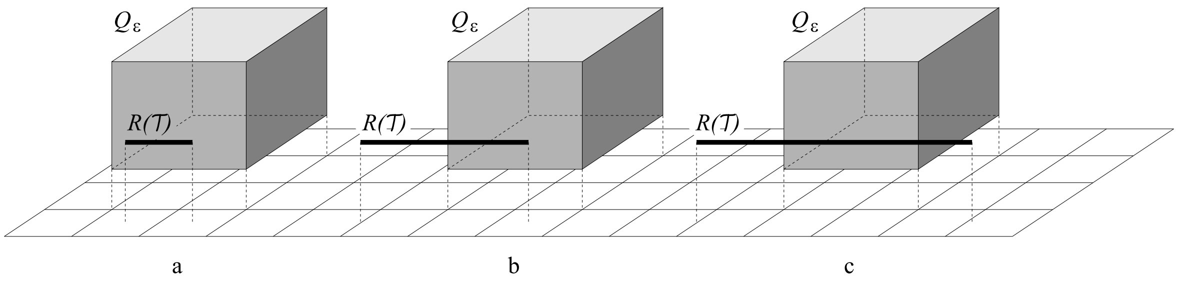

Figure 3: One plane containing the line segment $R(\mathcal{T})$ intersects Q_ϵ.

Proof: Since both axis-parallel planes containing $R(\mathcal{T})$ intersect Q_ϵ, we know that $R(\mathcal{T})$ itself must intersect Q_ϵ. Hence, an (input or bounding) box b stored in $\mathcal{T}$ intersects Q_ϵ if and only if $b \cap R(\mathcal{T})$ intersects $Q_\epsilon \cap R(T)$. We can therefore analyse the query time in this case as if the situation were completely 1-dimensional, that is, as if $\mathcal{T}$ were a 1-tree storing segments on a line, which is queried with a segment on the same line. An analysis of this case, proving the lemma, can be found in the paper by Agarwal *et al.* [1]. □

Case 2: One plane containing $R(\mathcal{T})$ intersects Q_ϵ. This case is illustrated in Fig. 3. Part (a) of the figure corresponds to part (i) in the lemma below, parts (b) and (c) to part (ii).

Lemma 2.3 *Let $\mathcal{T}$ be a 1-LSF-interval tree storing n boxes with stabbing number σ. Suppose we query $\mathcal{T}$ with a box Q such that one axis-parallel plane containing $R(\mathcal{T})$ intersects Q_ϵ.*

(i) If the axis-parallel projection of Q_ϵ onto the line containing $R(\mathcal{T})$ contains $R(\mathcal{T})$ completely, then we visit $O(k_\epsilon(\mathcal{T}))$ nodes.

(ii) Otherwise, we visit $O(\log n + \sigma + k_\epsilon(\mathcal{T}))$ nodes.

Proof: Let g be the axis-parallel plane containing $R(\mathcal{T})$ and intersecting Q_ϵ. For any (input or bounding) box b stored in $\mathcal{T}$, we know that b intersects Q_ϵ if and only if $b \cap g$ intersects $Q_\epsilon \cap g$. We can therefore analyse the query time in this case as if the situation were completely 2-dimensional, that is, as if $\mathcal{T}$ were a 1-tree storing rectangles in the plane, which is queried

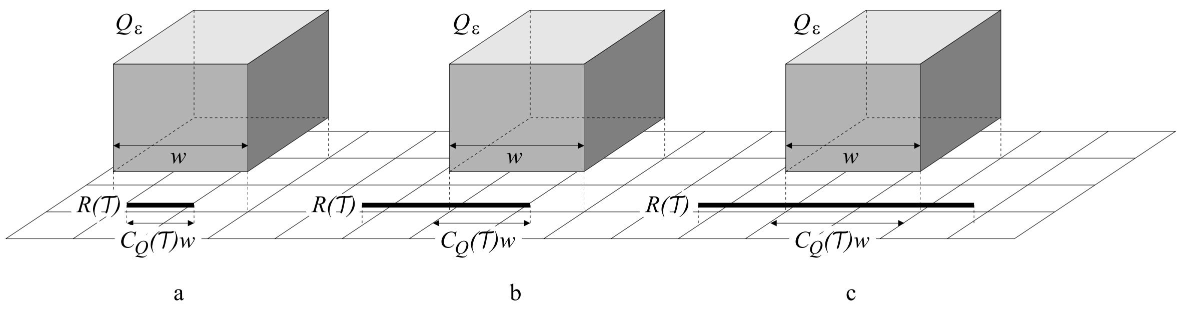

Figure 4: No plane containing the line segment $R(\mathcal{T})$ intersects Q_ϵ.

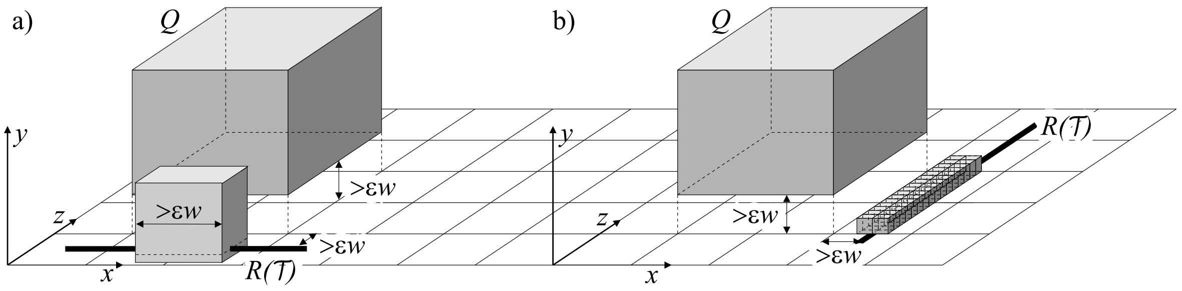

Figure 5: a. A shield on a defining region parallel to the primary axis. — b. Arrangement of cubes intersected by shields on a defining region parallel to a secondary axis.

with a rectangle in the plane. An analysis of this case, proving the lemma, can be found in the paper by Agarwal *et al.* [1]. $\square$

Case 3: No plane containing $R(\mathcal{T})$ intersects Q_ϵ. In the analysis of this case we will take into account how much of the query range is 'within reach' of the tree. More precisely, consider the intersection of $R(\mathcal{T})$ with the projection of Q_ϵ on the line containing $R(\mathcal{T})$. We denote by $C_Q(\mathcal{T})$ the length of this intersection divided by the length of the longest edge of Q—see Fig. 4. In the next subsection we will sum the bound for several different disjoint subtrees $\mathcal{T}$, and then we will use the fact that their $C_Q(\mathcal{T})$-values sum up to at most $1 + 2\epsilon$.

Figure 4 illustrates the cases that arise in the next lemma, with part (a) of the figure corresponding to part (i) of the lemma, and parts (b) and (c) corresponding to part (ii).

Lemma 2.4 *Let $\mathcal{T}$ be a 1-LSF-interval tree storing n boxes with slicing number λ. Suppose we query $\mathcal{T}$ with a box Q such that no axis-parallel plane containing $R(\mathcal{T})$ intersects Q_ϵ.*

(i) If the axis-parallel projection of Q_ϵ onto the line containing $R(\mathcal{T})$ contains $R(\mathcal{T})$ completely, then we visit $C_Q(\mathcal{T}) \cdot O(\lambda/\epsilon) + O(\lambda)$ nodes.

(ii) Otherwise, we visit $O(\log n + \lambda/\epsilon)$ nodes.

Proof: Since the maximum degree of each node is nine, the number of visited leaf nodes is at most nine times the number of visited internal nodes. Hence, we can restrict our attention to bounding the latter number. Let $\overline{Q}_\epsilon$ denote the axis-parallel projection of Q_ϵ onto the line containing $R(\mathcal{T})$, and let $\overline{R} := \overline{Q}_\epsilon \cap R(\mathcal{T})$. Let ν be a visited internal node of $\mathcal{T}$, and let $b(\nu)$ be its bounding box. We distinguish two cases: $b(\nu) \cap R(\mathcal{T}) \subset \overline{R}$, and $b(\nu) \cap R(\mathcal{T}) \not\subset \overline{R}$. We claim that the number of nodes to which the first case applies is $C_Q(\mathcal{T}) \cdot O(\lambda/\epsilon) + O(\lambda)$, and that the number of nodes to which the second case applies is $O(\sigma + \log n)$, where σ is the stabbing number of the boxes stored in the tree. Note that in part (i) of the lemma the second case cannot arise. Together with the fact that $\lambda \geq \sigma$ and $C_Q(\mathcal{T}) \leq 1 + 2\epsilon$, this means that proving the claim above will establish the lemma.

We first bound the number of nodes for which $b(\nu) \cap R(\mathcal{T}) \not\subset \overline{R}$, since this is the easier case. Let ν be such a node. Since $b(\nu) \cap R(\mathcal{T})$ cannot be disjoint from $\overline{R}$—otherwise $b(\nu)$ would not intersect Q and ν would not be visited—it follows that $b(\nu)$ must contain an endpoint $\overline{p}$ of $\overline{R}$. Now there are two possibilities.

One is that $R(\nu)$, the defining region of ν, is a line segment containing $\overline{p}$. Since the defining regions of 1-nodes at a fixed level of the tree are disjoint and the depth of the tree is $O(\log n)$, there are only $O(\log n)$ such nodes.

The other possibility is that $R(\nu)$ is a point. But then the priority leaf immediately below ν storing the box extending farthest into the direction of $\overline{p}$ must contain $\overline{p}$. We charge the visit of ν to this leaf. Since a leaf gets charged only from its parent, and there are at most σ input boxes containing any given point, there are at most 2σ such nodes.

Thus we find a bound of $O(\log n + \sigma) = O(\log n + \lambda)$ for the case of $b(\nu) \cap R(\mathcal{T}) \not\subset \overline{R}$.

Now consider the nodes ν such that $b(\nu) \cap R(\mathcal{T}) \subset \overline{R}$. We shall charge the visit of ν to a certain priority leaf directly below it, called a *shield*. Each shield will be charged at most once, namely from its parent. Bounding the maximum number of shields will then prove this part of the claim.

We start by defining the shields. Recall that the primary axis of S_x—the axis parallel to the longest edges of the boxes in S_x—is the x-axis. Since the two remaining (secondary) axes play equivalent roles, we can assume that the y-axis is not parallel to $R(\mathcal{T})$. Let us also assume w.l.o.g. that the y-coordinate of $R(\mathcal{T})$ is smaller than the smallest y-coordinate of Q. A *shield* is now defined as a priority leaf whose corresponding input box b extends into the positive y-direction from $R(\mathcal{T})$ over a distance of at least ϵw. That is, if $y_{\max}(b)$ is the maximum y-coordinate of b and $y(R(\mathcal{T}))$ is the y-coordinate of $R(\mathcal{T})$, then b is a shield if $y_{\max}(b) - y(R(\mathcal{T})) \geq \epsilon w$.

We now argue that each visited internal node ν for which it holds that $b(\nu) \cap R(\mathcal{T}) \subset \overline{R}$, has at least one shield as a child. Indeed, since none of the two axis-parallel planes containing $R(\mathcal{T})$ intersects Q_ϵ, the y-distance of $R(\mathcal{T})$ and Q must be at least ϵw. This means that the bounding box of ν must extend over a distance at least ϵw into the y-direction from $R(\mathcal{T})$, otherwise ν would not be visited. Hence, the input box extending farthest into the y-direction, extends that far; the priority leaf directly below ν storing this box is a shield.

It remains to bound the number of shields. We consider two subcases.

The first subcase is that $R(\mathcal{T})$ is parallel to the x-axis, as in Fig. 5a. In this case the length of any box in S_x along $R(\mathcal{T})$ is at least its length in any other direction. In particular, a shield will cover a portion of $\overline{R}$ of length at least ϵw. Since no point is contained in more than σ input boxes, there can be at most $\sigma \cdot \text{length}(\overline{R})/(\epsilon w)$ shields in this case. Because $\text{length}(\overline{R}) = C_Q(\mathcal{T}) \cdot w$ by definition, the number of shields is bounded by $\sigma \cdot C_Q(\mathcal{T})/\epsilon$.

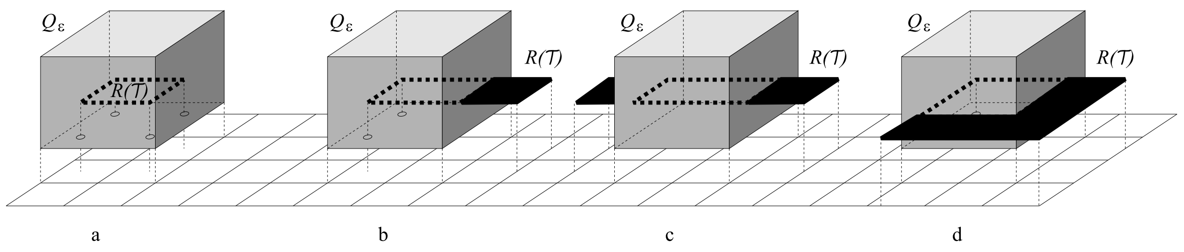

Figure 6: The plane containing the rectangle $R(\mathcal{T})$ intersects Q_ϵ.

The second subcase is that $R(\mathcal{T})$ is parallel to the z-axis—see Fig. 5b. In this case, a shield must extend over a distance of at least ϵw upwards from $R(\mathcal{T})$ and over a distance of at least $\epsilon w/2$ into either the positive of negative x-direction from $R(\mathcal{T})$. Now imagine a line-up of $\lceil 2C_Q(\mathcal{T})/\epsilon \rceil$ cubes of size $\epsilon w/2$ whose lower right edges together cover Q's projection on $R(\mathcal{T})$. Add a copy of this line-up shifted right over a distance of $\epsilon w/2$, so that in the second line-up, the lower *left* edges together cover Q's projection—see Fig. 5b. Since a shield extends away from $R(\mathcal{T})$ in both orthogonal directions over a distance greater than the size of the cubes in the line-up, it must intersect the four edges parallel to $R(\mathcal{T})$ of at least one of these cubes. Since the slicing number of the input boxes is at most λ, there can be at most $2\lambda\lceil 2C_Q(\mathcal{T})/\epsilon \rceil \leq 2\lambda + 4C_Q(\mathcal{T})\lambda/\epsilon$ shields in this case.

Using $\lambda \geq \sigma$, we conclude that the bounds for both subcases are within $O(\lambda) + C_Q(\mathcal{T}) \cdot O(\lambda/\epsilon)$, which finishes the proof of our claim. □

2.2.2 2-dimensional subtrees

Let $\mathcal{T}$ be a 2-dimensional subtree. As before, it will be useful to take into account how much of the query range's boundary is 'within reach' of the tree. More precisely, consider the edges of Q_ϵ's projection on the plane containing $R(\mathcal{T})$. Denote by $C_Q(\mathcal{T})$ the sum of the lengths of the intersections of these edges with $R(\mathcal{T})$, divided by w, the length of the longest edge of the query range.

We distinguish two cases, depending on whether or not the plane containing the 2-dimensional defining region $R(\mathcal{T})$ intersects Q_ϵ.

Case 1: The plane containing $R(\mathcal{T})$ intersects Q_ϵ. This case is illustrated in Fig. 6. Parts (a) and (b) of the figure correspond to case (i) in the lemma below, part (c) to case (ii), and part (d) to case (iii).

Lemma 2.5 *Let $\mathcal{T}$ be a 2-LSF-interval tree storing n boxes with stabbing number σ. Suppose we query $\mathcal{T}$ with a box Q such that the plane containing $R(\mathcal{T})$ intersects the extended query range Q_ϵ. Let $\overline{Q}_\epsilon$ denote the intersection of Q_ϵ with the plane containing $R(\mathcal{T})$.*

(i) If at most one edge of $\overline{Q}_\epsilon$ intersects $R(\mathcal{T})$, then we visit $O(k_\epsilon(\mathcal{T}))$ nodes.

(ii) If two opposite edges, and no other edges, of $\overline{Q}_\epsilon$ intersect $R(\mathcal{T})$, then we visit $O(\log^2 n + k_\epsilon(\mathcal{T})) + C_Q(\mathcal{T}) \cdot O((\log^2 n)/\epsilon)$ nodes.

(iii) Otherwise we visit $O((\log^2 n)/\epsilon + \sigma \log n + k_\epsilon(\mathcal{T}))$ nodes.

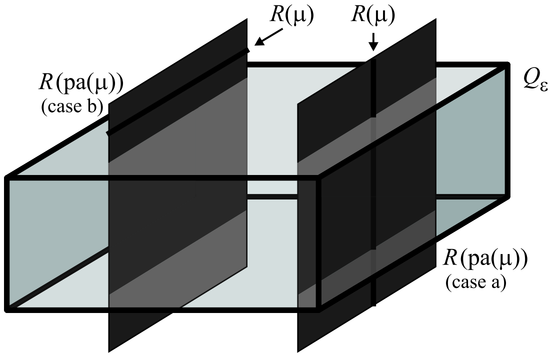

Figure 7: μ is a 1-node whose defining region cuts a 2-node intersecting opposite edges of $\overline{Q}_\epsilon$, that is: opposite facets of Q_ϵ.

Proof: First we observe that the longest edge of Q_ϵ has length $(1+2\epsilon)w$ and that its shortest edge has length at least $2\epsilon w$. Hence, the aspect ratio of Q_ϵ and the aspect ratio of $\overline{Q}_\epsilon$ are at most $1 + 1/(2\epsilon)$.

Since $R(\mathcal{T})$ intersects Q_ϵ, we know for any (input or bounding) box b stored in $\mathcal{T}$ that b intersects Q_ϵ if and only if $b \cap R(\mathcal{T})$ intersects $Q_\epsilon \cap R(T)$. We can therefore analyse the query time in this case as if the situation were completely 2-dimensional, that is, as if $\mathcal{T}$ were a 2-tree storing rectangles in the plane, which is queried with $\overline{Q}_\epsilon$. Since $\overline{Q}_\epsilon$ has aspect ratio at most $1 + 1/(2\epsilon)$, parts (i) and (iii) of the lemma now immediately follow from the results by Agarwal *et al.* [1].

For part (ii), we need a bit more refined analysis. Consider the collection N of all visited 2-nodes ν in $\mathcal{T}$ whose defining region $R(\nu)$ intersects two opposite edges of $\overline{Q}_\epsilon$, and no other edges. This collection forms a subgraph $\mathcal{G}(N)$ of $\mathcal{T}$, which is a tree rooted at the root of $\mathcal{T}$. We shall first bound the number of nodes in N, and then the number of visited descendants.

To bound the number of nodes in N, we cover $\overline{Q}_\epsilon$ with at most $\lceil\alpha\rceil$ squares with side length $(1 + 2\epsilon)w/\alpha$, where $\alpha \leq 1 + 1/(2\epsilon)$ is the aspect ratio of $\overline{Q}_\epsilon$. From a bound on the number of nodes intersecting these squares, we can derive a bound on the number of nodes in N as follows. At most $\alpha C_Q(\mathcal{T})+1$ of the squares intersect $R(\mathcal{T})$. Now consider a node $\nu \in N$. Since $R(\nu)$ intersects two opposite sides of $\overline{Q}_\epsilon$, it intersects two opposite sides of at least one of the $\alpha C_Q(\mathcal{T}) + 1$ squares used to cover $\overline{Q}_\epsilon \cap R(\mathcal{T})$. Observe that the leaves of $\mathcal{G}(N)$—that is, the nodes that have no children in N; they need not be leaves of $\mathcal{T}$—have disjoint defining regions. Lemma 2.1 implies that the number of such leaves is $O(\log n) + C_Q(\mathcal{T}) \cdot O(\alpha \log n)$. If we include their ancestors in the count, we obtain a bound of $O(\log^2 n) + C_Q(\mathcal{T}) \cdot O(\alpha \log^2 n)$ on the number of nodes in N.

It remains to bound the number of descendants of the nodes in N. These are organized into subtrees whose roots are children of nodes in N and are not in N themselves. Consider such a root node μ. Let $\mathrm{pa}(\mu) \in N$ be the parent of μ. There are three cases.

- The first case is that μ is a 2-node. In this case $R(\mu)$ intersects at most one edge of $\overline{Q}_\epsilon$, as in part (i) of the lemma; if it would intersect two opposite edges it would be in N, and the case where a vertex of $\overline{Q}_\epsilon$ lies in $R(\mu)$ cannot occur when we are handling part (ii) of the lemma. The total number of visited nodes of $\mathcal{T}_\mu$ is $O(k_\epsilon(\mathcal{T}_\mu))$ by part (i) of the lemma. Summing over all nodes μ thus gives us a total bound of $O(k_\epsilon(\mathcal{T}))$ for these

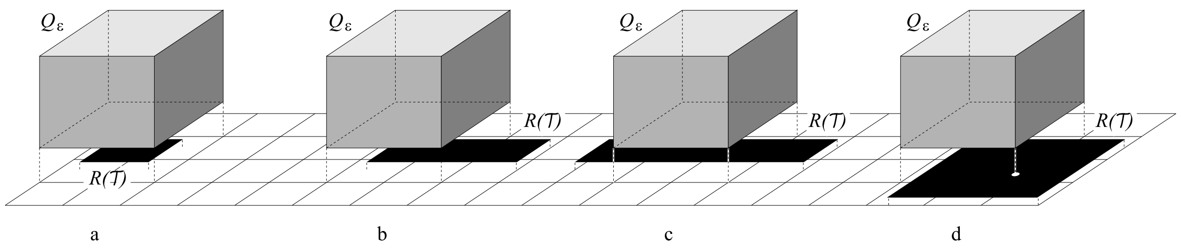

Figure 8: The plane containing the rectangle $R(\mathcal{T})$ is disjoint from Q_ϵ.

subtrees.

- The second case is that the root is a 1-node μ and $R(\mu)$ cuts $R(\text{pa}(\mu))$ such that $\text{pa}(\mu)$ has two children in N—see Fig. 7 case (a).

  The number of nodes of degree two in $\mathcal{G}(N)$ is no more than the number of leaves in $\mathcal{G}(N)$, so there can be at most $O(\log n) + C_Q(\mathcal{T}) \cdot O(\alpha \log n)$ such nodes μ. Lemma 2.2(ii) states that the query time in each such tree is $O(\log n + k_\epsilon(\mathcal{T}_\mu))$, so the total query time in these trees is $O(\log^2 n + k_\epsilon(\mathcal{T})) + C_Q(\mathcal{T}) \cdot O(\alpha \log^2 n)$.

- The third case is that the root is a 1-node μ, where $R(\mu)$ cuts $R(\text{pa}(\mu))$ such that $\text{pa}(\mu)$ has at most one child in N—see Fig. 7 case (b).

  Now $R(\mu)$ must lie completely inside the projection of Q_ϵ onto the line containing $R(\mu)$. Lemma's 2.2(i) and 2.3(i) state that the query time in each tree rooted at such a node is $O(k_\epsilon(\mathcal{T}_\mu))$. Since the number of such nodes is asymptotically bounded by the size of N, the total query time in these 1-trees is $O(\log^2 n + k_\epsilon(\mathcal{T})) + C_Q(\mathcal{T}) \cdot O(\alpha \log^2 n)$.

In total, we find a bound of $O(\log^2 n + k_\epsilon(\mathcal{T})) + C_Q(\mathcal{T}) \cdot O(\alpha \log^2 n)$. With $\alpha \leq 1 + 1/(2\epsilon)$, this proves part (ii) of the lemma. □

Case 2: The plane containing $R(\mathcal{T})$ does not intersect Q_ϵ. This case is illustrated in Fig. 8. Part (a) of the figure corresponds to case (i) in the lemma below, parts (b) and (c) to case (ii), and part (d) to case (iii).

Lemma 2.6 *Let $\mathcal{T}$ be a 2-LSF-interval tree storing n boxes with slicing number λ. Suppose we query $\mathcal{T}$ with a box Q such that the plane containing $R(\mathcal{T})$ does not intersect Q_ϵ. Let $\overline{Q}_\epsilon$ denote the axis-parallel projection of Q_ϵ onto the plane containing $R(\mathcal{T})$.*

(i) If $\overline{Q}_\epsilon$ contains $R(\mathcal{T})$ completely, then we visit $O(k_\epsilon(\mathcal{T}))$ nodes.

(ii) If $R(\mathcal{T})$ intersects at least one edge but no vertex of $\overline{Q}_\epsilon$, then we visit $O(\lambda \log^2 n + k_\epsilon(\mathcal{T})) + C_Q(\mathcal{T}) \cdot O(\lambda \log^2 n/\epsilon)$ nodes.

(iii) Otherwise we visit $O(\lambda \log^2 n/\epsilon + k_\epsilon(\mathcal{T}))$ nodes.

Proof: *(i)* Without loss of generality, suppose $R(\mathcal{T})$ is horizontal and lies below Q. Then every node visited in $\mathcal{T}$ must have a descendant which raises high enough to intersect Q. In particular, there is a priority leaf immediately below this node that stores an input box intersecting Q. We can charge the visit to this node to the priority leaf. Since there are at

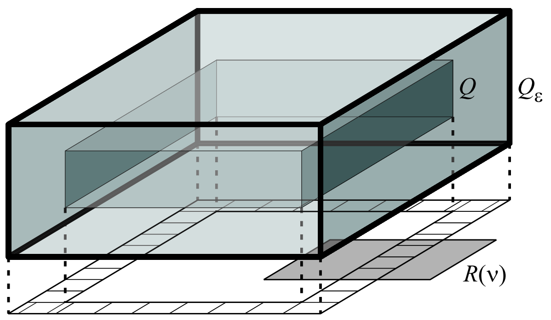

Figure 9: Covering $\overline{Q}_\epsilon \setminus \overline{Q}$ with squares.

most $k_\epsilon(\mathcal{T})$ such priority leaves and each of them is charged at most once, the bound follows. *(ii)* We can distinguish two types of visited nodes.

The first type of nodes are 2-nodes whose defining regions lie completely inside $\overline{Q}_\epsilon$ and descendants of such nodes. Here a similar argument as in the proof of part (i) applies: any such node has a priority leaf below it that intersects Q_ϵ, so there are only $O(k_\epsilon(\mathcal{T}))$ such nodes.

The second type of nodes are the remaining ones. Let N be the collection of all remaining visited 2-nodes. For any node $\nu \in N$, we know that $R(\nu)$ intersects the complement of $\overline{Q}_\epsilon$ as well as $\overline{Q}$, the projection of Q onto the plane containing $R(\mathcal{T})$.

To bound the number of nodes in N we cover $\overline{Q}_\epsilon \setminus \overline{Q}$ using at most $4(\lceil 1/\epsilon \rceil + 1)$ squares with side length ϵw, which are contained in $\overline{Q}_\epsilon \setminus \overline{Q}$—see Fig. 9. For any node $\nu \in N$ we have that $R(\nu)$ intersects two opposite edges of at least one of these squares. Since $R(\nu) \subset R(\mathcal{T})$ and $R(\mathcal{T})$ does not contain a vertex of $\overline{Q}_\epsilon$, we can restrict our attention to squares that are used to cover two opposite 'sides' of $\overline{Q}_\epsilon \setminus \overline{Q}$ and that intersect $R(\mathcal{T})$. Hence, the number of squares we have to consider is at most $2\lceil C_Q(\mathcal{T})/\epsilon \rceil$. As before, we observe that the nodes of N form a subgraph $\mathcal{G}(N)$ of $\mathcal{T}$, which is a tree whose leaves have disjoint defining regions. Hence, by Lemma 2.1 there are $O(\log n) + C_Q(\mathcal{T}) \cdot O(\log n/\epsilon)$ leaves in $\mathcal{G}(N)$. If we include their ancestors in the count, we find a bound of $O(\log^2 n) + C_Q(\mathcal{T}) \cdot O(\log^2 n/\epsilon)$ on the number of nodes in N.

It remains to bound the number of descendants of nodes in N. The descendants are organized into subtrees whose roots are children of nodes in N and are not in N themselves. Consider such a root node ν. Let $\mathrm{pa}(\mu) \in N$ be the parent of μ. There are three cases.

- The first case is that μ is a 2-node. But then μ must be of the first type—its defining region must lie completely inside $\overline{Q}_\epsilon$—so we already counted these nodes and their descendants earlier.

- The second case is that μ is a 1-node and $R(\mu)$ cuts $R(\mathrm{pa}(\mu))$ in such a way that $\mathrm{pa}(\mu)$ has two children in N.

  The analysis for this case is done the same as in the proof of Lemma 2.5(ii), now referring to Lemma 2.3 instead of Lemma 2.2.

  Since the number of nodes of degree two in $\mathcal{G}(N)$ is at most its number of leaves, there can be at most $O(\log n) + C_Q(\mathcal{T}) \cdot O(\log n/\epsilon)$ such nodes μ. Lemma 2.3(ii) states that the query time in each such tree is $O(\log n + \sigma + k_\epsilon(\mathcal{T}_\mu))$, so the total query time in these trees is

$$O(\log^2 n + \sigma \log n + k_\epsilon(\mathcal{T})) + C_Q(\mathcal{T}) \cdot O(\log^2 n/\epsilon + \sigma \log n/\epsilon).$$

(Note that the k_ϵ terms always add up to $O(k_\epsilon(\mathcal{T}))$.)

- The remaining case is that μ is a 1-node and $\mathrm{pa}(\mu)$ is cut by $R(\mu)$ such that it has at most one child in N.

  Now $R(\mu)$ lies completely inside the projection of Q_ϵ onto the line containing $R(\mu)$. Lemma 2.4(i) and Lemma 2.3(i) state that the query time in such trees is $O(\lambda) + C_Q(\mathcal{T}_\mu) \cdot O(\lambda/\epsilon)$ and $O(k_\epsilon(\mathcal{T}_\mu))$, respectively. The number of nodes to which this applies is clearly bounded by the number of nodes in N, which is $O(\log^2 n) + C_Q(\mathcal{T}) \cdot O(\log^2 n/\epsilon)$. Hence, the total query time in these 1-trees is

$$O(\lambda \log^2 n + k_\epsilon(\mathcal{T})) + C_Q(\mathcal{T}) \cdot O(\lambda \log^2 n/\epsilon) + \sum C_Q(\mathcal{T}_\mu) \cdot O(\lambda/\epsilon),$$

  where the sum is over all 1-nodes μ that are a child of a node in N and are such that $R(\mu)$ lies completely inside the projection of Q_ϵ onto the line containing $R(\mu)$. Note that each point of an edge of $\overline{Q}_\epsilon$ lies in $O(\log n)$ defining regions of 2-nodes (one per level), so $\sum_{\nu \in N} C_Q(\mathcal{T}_\nu) = O(\log n) C_Q(\mathcal{T})$. The same bound holds if we sum over the 1-nodes μ that are children of nodes in N. Hence, we find a total query time for this case of $O(\lambda \log^2 n + k_\epsilon(\mathcal{T})) + C_Q(\mathcal{T}) \cdot O(\lambda \log^2 n/\epsilon)$.

Putting the tree cases together, and using $\sigma \leq \lambda$, we find an overall bound of $O(\lambda \log^2 n + k_\epsilon(\mathcal{T})) + C_Q(\mathcal{T}) \cdot O(\lambda \log^2 n/\epsilon)$.

(iii) We can distinguish three types of visited nodes: the two types that were also considered in the proof of part (ii), and a third type, namely 2-nodes containing a corner of $\overline{Q}$ and their descendant 1-nodes and 0-nodes.

The number of nodes of the first two types can be bounded as in the proof of part (ii). Using that $C_Q(\mathcal{T}) \leq 4(1 + 2\epsilon)$, we get a bound of $O(\lambda \log^2 n/\epsilon + k_\epsilon(\mathcal{T}))$ for these types. As for the third type, we note that there are $O(\log n)$ 2-nodes containing a corner of $\overline{Q}$. If μ is a 1-node that is a child of such a node, then the query time in $\mathcal{T}_\mu$ is $O(\log n + \sigma + k_\epsilon(\mathcal{T}))$ or $O(\log n + \lambda/\epsilon)$ by Lemma 2.3 or Lemma 2.4, respectively, so we have $O(\log^2 n + \lambda \log n/\epsilon + k_\epsilon(\mathcal{T}))$ nodes of the third type. □

2.2.3 3-dimensional trees

Finally we can prove our main result.

Theorem 2.7 *Let $\mathcal{T}$ be a 3-LSF-interval tree storing n boxes with slicing number λ. Then a query in $\mathcal{T}$ with a box Q will visit $O(\min_{0<\epsilon\leq 1}\{(1/\epsilon)((1/\epsilon) + \lambda) \log^4 n + k_\epsilon\})$ nodes, where k_ϵ is the number of boxes intersecting the extended range Q_ϵ.*

Proof: Fix an arbitrary $0 < \epsilon \leq 1$. As observed before, it suffices to bound the number of visited internal nodes. These can be partitioned into four categories, namely 3-nodes ν such that $R(\nu)$ intersects:

(i) at most one facet of Q_ϵ,

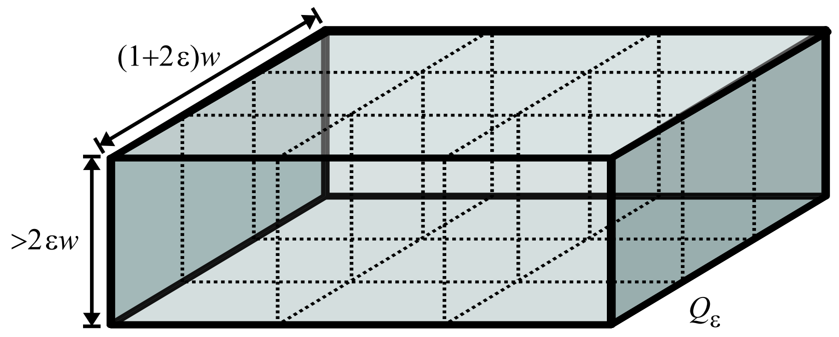

Figure 10: Covering Q_ϵ with $O(1/\epsilon^2)$ cubes.

(ii) more than one facet of Q_ϵ, but none of its edges,

(iii) at least one edge of Q_ϵ, but none of its vertices,

(iv) at least one vertex of Q_ϵ,

where each category also includes the descendant 2-nodes, 1-nodes and 0-nodes of the 3-nodes. We will now treat these cases one by one.

(i) 3-Nodes ν such that $R(\nu)$ intersects at most one facet of Q_ϵ, plus their descendant 2-nodes, 1-nodes, and 0-nodes. Any such node must have a priority leaf directly below it that stores a box intersecting Q_ϵ. Hence, the total number of nodes in this category is $O(k_\epsilon)$.

(ii) 3-Nodes ν such that $R(\nu)$ intersects more than one facet of Q_ϵ but none of its edges, plus their descendant 2-nodes, 1-nodes, and 0-nodes.

Let N be the collection of 3-nodes in this category, and let $\mathcal{G}(N)$ be the subgraph of $\mathcal{T}$ formed by these nodes. $\mathcal{G}(N)$ is a forest of trees.

To bound the number of nodes in N, we cover Q_ϵ by $O(1/\epsilon^2)$ cubes that are contained in Q_ϵ and are as big as the smallest edges of Q_ϵ — see Fig. 10. Any node in N must intersect opposite facets of at least one of these cubes. Because the leaves of $\mathcal{G}(N)$ have disjoint defining regions, their number is bounded by $O((1/\epsilon^2)\log^2 n)$ by Lemma 2.1. The total number of nodes in N is therefore bounded by $O((1/\epsilon^2)\log^3 n)$.

It remains to bound the number of descendant 2-nodes, 1-nodes, and 0-nodes of the nodes in N. These are organized in subtrees whose roots are children of nodes in N. Let μ be such a root and let $\text{pa}(\mu) \in N$ be its parent. There are two cases, as illustrated in Fig. 11.

- $R(\mu)$ cuts $R(\text{pa}(\mu))$ in such a way that $\text{pa}(\mu)$ has two children in N—see case (a) in Fig. 11.

  Since the number of nodes of degree two in $\mathcal{G}(N)$ is bounded by the number of leaves in N, there are only $O((1/\epsilon^2)\log^2 n)$ such roots. Lemma 2.5(ii) states that the query time in each subtree rooted at such a node is $O(\log^2 n + k_\epsilon(\mathcal{T}_\mu)) + C_Q(\mathcal{T}_\mu) \cdot O((\log^2 n)/\epsilon)$, so the total query time in these subtrees is

$$O((1/\epsilon^2)\log^4 n + k_\epsilon) + \sum_\mu C_Q(\mathcal{T}_\mu) \cdot O((\log^2 n)/\epsilon),$$

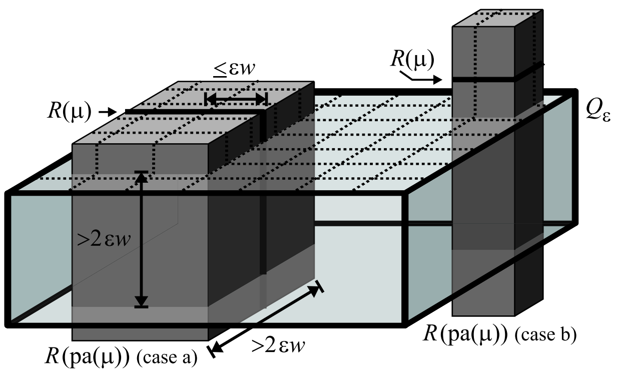

Figure 11: 3d-nodes that intersect more than one facet of Q_ϵ, but none of its edges.

where the sum is over all 2-nodes μ in the current category such that $R(\mathcal{T}_\mu)$ cuts opposite facets of Q_ϵ.

We proceed to bound $\sum_\mu C_Q(\mathcal{T}_\mu)$. To simplify the discussion, let's assume that the defining regions $R(\mu)$ and $R(\mathrm{pa}(\mu))$ cut the top and bottom facet of Q_ϵ, as in Fig. 11, case a. Then for each node μ we have that $C_Q(\mathcal{T}_\mu)w$ is the length of $R(\mu)$ as seen from above. Note that $R(\mathrm{pa}(\mu))$ has height at least $2\epsilon w$, because the height of Q_ϵ is at least that much. Therefore, the length of the horizontal edges of $R(\mathrm{pa}(\mu))$ orthogonal to $R(\mu)$ is at least $2\epsilon w$ as well, otherwise $R(\mathrm{pa}(\mu))$ would have been cut by a horizontal plane. Cover the top facet of Q_ϵ by $O(1/\epsilon^2)$ squares of side length ϵw. Since $R(\mathrm{pa}(\mu))$ has horizontal edges of length at least $2\epsilon w$, it must intersect opposite sides of at least one such square s. If this happens for m 2-nodes μ, then there are at least m disjoint defining regions of 3-nodes that intersect opposite sides of s. Lemma 2.1 tells us that s is cut by $O(\log n)$ disjoint defining regions. Hence, the total length within s of all regions $R(\mu)$ as seen from above is $O(\epsilon w \log n)$. Summed over all squares we find that the total length of all regions $R(\mu)$ as seen from above is $O((w/\epsilon)\log n)$. This implies that $\sum_\mu C_Q(\mathcal{T}_\mu) = O((1/\epsilon)\log n)$. It follows that the total number of nodes for this case is $O((1/\epsilon^2)\log^4 n + k_\epsilon(\mathcal{T}))$.

- $R(\mathcal{T})$ cuts $R(\mathrm{pa}(\mu))$ such that $\mathrm{pa}(\mu)$ has at most one child in N—see case (b) in Fig. 11.

  In this case $R(\mu)$ lies completely inside the projection of Q_ϵ onto the plane containing $R(\mu)$. Lemma's 2.6(i) and 2.5(i) state that the number of visited nodes in each such tree is $O(k_\epsilon(\mathcal{T}_\mu))$, which adds up to $O(k_\epsilon(\mathcal{T}))$.

In total, there are $O((\log^4 n)/\epsilon^2 + k_\epsilon)$ nodes in this category.

(iii) 3-Nodes ν such that $R(\nu)$ intersects at least one edge of Q_ϵ but does not contain one of its vertices, plus their descendant 2-nodes, 1-nodes, and 0-nodes.

In this case $R(\nu)$ must intersect an edge e_ϵ of Q_ϵ *and* the corresponding edge e of Q (the edge with both endpoints lying at an L_∞-distance of ϵw from e_ϵ), otherwise ν would not be visited. For each pair e, e_ϵ of corresponding edges, we take a set of $O(1/\epsilon)$ cubes of size ϵw, such that each cube has an edge contained in e and the opposite edge contained in e_ϵ, and such that together they cover e completely — see Fig. 12. Let N be the collection of 3-nodes

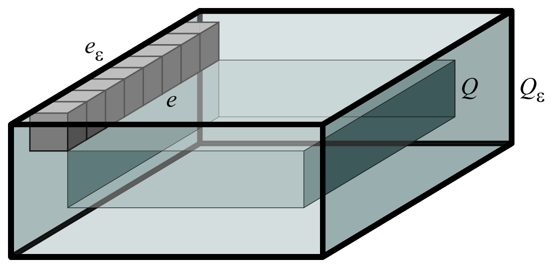

Figure 12: Covering an edge of Q with $O(1/\epsilon)$ cubes.

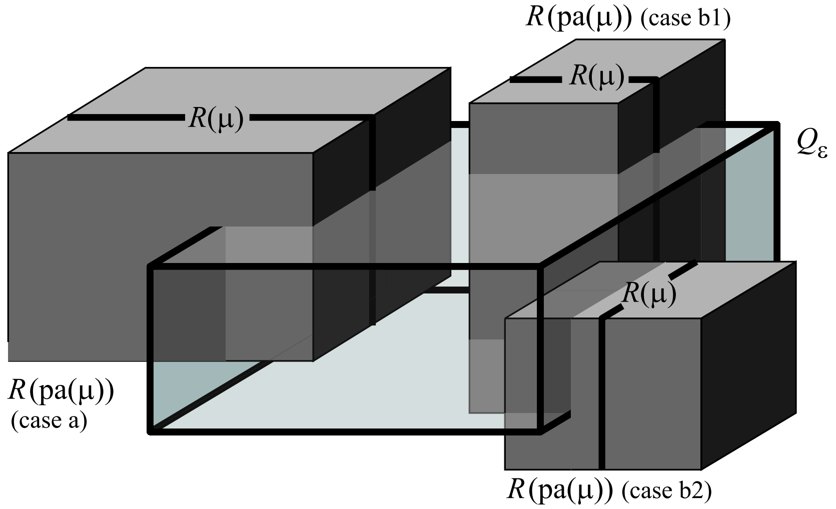
$R(\text{pa}(\mu))$ (case b1)
$R(\mu)$
$R(\mu)$
Q_ε
$R(\mu)$
$R(\text{pa}(\mu))$ (case a)
$R(\text{pa}(\mu))$ (case b2)

Figure 13: 3d-nodes that intersect an edge of Q_ϵ, but none of its vertices.

in the current category, and let $\mathcal{G}(N)$ be the subgraph of $\mathcal{T}$ formed by these nodes. $\mathcal{G}(N)$ is a forest of trees.

Any 3-node in N must intersect opposite edges of a facet of at least one of these cubes. Summing over the facets of all cubes and using Lemma 2.1 again, we find that there are only $O(\log n/\epsilon)$ leaves in $\mathcal{G}(N)$ and, hence, $O(\log^2 n/\epsilon)$ 3-nodes in N in total.

The descendant 2-nodes, 1-nodes, and 0-nodes are organized in subtrees rooted at 2-nodes μ with a node $\text{pa}(\mu)$ in N as parent. We distinguish two cases, as illustrated in Fig. 13.

- For the subtrees rooted at node μ such that $\text{pa}(\mu)$ has two children in N (case (a) in Fig. 13), we can apply Lemma 2.5(iii) to find a bound of $O(\log^2 n/\epsilon + \sigma \log n + k_\epsilon(\mathcal{T}_\mu))$ for each subtree. Since the number of such nodes is bounded by the number of leaves in $\mathcal{G}(N)$ we get a total of $O(\log^3 n/\epsilon^2 + \sigma \log^2 n/\epsilon + k_\epsilon)$ nodes.

- For the other subtrees, of which there are $O(\log^2 n/\epsilon)$, we apply Lemmas 2.5(i) and (ii) (case (b1) in Fig. 13) and Lemma 2.6(ii) (case (b2)) to find a total bound for all such subtrees of

$$O(\lambda \log^4 n/\epsilon + k_\epsilon) + \sum C_Q(\mathcal{T}_\mu) \cdot O(\lambda \log^2 n/\epsilon).$$

Because any point in 3-space lies in at most $O(\log n)$ defining regions of 3-nodes, $\sum C_Q(\mathcal{T}_\mu) = O((1+2\epsilon)\log n)$ and we get a bound of $O(\lambda \log^4 n/\epsilon + k_\epsilon)$.

In total, the number of nodes in this category is $O((1/\epsilon + \lambda)\log^4 n/\epsilon + k_\epsilon)$.

(iv) 3-Nodes ν such that $R(\nu)$ contains at least one vertex of Q_ϵ, plus their descendant 2-nodes, 1-nodes and 0-nodes.

At most $O(\log n)$ 3-nodes can contain a vertex of Q_ϵ. By Lemma 2.6(iii) each of them may have a 2-subtree $\mathcal{T}$ with query time $O(\lambda \log^2 n/\epsilon + k_\epsilon(\mathcal{T}))$, leading to a total of $O(\lambda \log^3 n/\epsilon + k_\epsilon)$ visited nodes in this category.

Since the number of visited nodes of each category is within the claimed bound, this proves the theorem. $\square$

Remark 2.8 If the query range has bounded aspect ratio, then it can be shown that the number of visited nodes reduces to $O(\min_{0<\epsilon\leq 1}\{(\lambda/\epsilon)\log^4 n + k_\epsilon\})$.

2.3 Pipes and low-density scenes

Our research is motivated by the MOLOG project [9], where we need to perform collision checking in CAD models of industrial installations such as in Fig. 1. Let S be the set of bounding boxes in the given scene. For the analysis we assume that S can be partitioned into two subsets S_P and S_D, such that S_P is a set of *pipes* and S_D forms a *low-density scene* [4, 11]. These concepts are defined as follows.

Definition 2.9 *Let b be a 3-dimensional axis-parallel box, and consider its length in x-, y-, and z-direction. The box b is called a β-*pipe *if the shortest of these three lengths is at most β times shorter than the shortest-but-one.*

Next we define the density of a scene, specialized to sets of boxes. (The original definition by van der Stappen and Overmars [11] uses balls instead of cubes, but this is equivalent up to a constant.)

Definition 2.10 *A set B of boxes in 3-space has* density δ *if the following holds: any cube C is intersected by at most δ boxes from B whose longest edge is longer than the edge length of C.*

Recall that the stabbing number of a set of boxes is defined as the maximum number of boxes with a non-empty intersection. Next we show that low-density sets and sets of pipes with low stabbing number also have low slicing number, which means that we can use the analysis of the previous subsection.

Lemma 2.11 *Let $S = S_P \cup S_D$ be a set of boxes in 3-space such that S_P is a set of β-pipes with stabbing number σ and S_D has density δ. Then the slicing number of S is at most $(\beta+2)\sigma + \delta$.*

Proof: Let C be a cube of edge length c. Since a box that slices C has edge length at least c, the set S_D has slicing number at most δ.

It remains to bound the number of pipes slicing C. A pipe slicing C has to occupy a volume of at least $c \times c \times c/\beta = c^3/\beta$ in the cube, unless it contains one of the six sides of the cube completely. In the latter case, the pipe has to contain either the top-right-back corner or the bottom-left-front corner of C, so there are at most 2σ such pipes. To bound the number of pipes in the former case, we observe that the total volume of the intersection of the pipes with C is at most σc^3. Therefore, the total number of boxes slicing the cube is at most $\delta + 2\sigma + \sigma c^3/(c^3/\beta) = \delta + (\beta + 2)\sigma$. □

By putting together Lemma 2.11 and Theorem 2.7, we get the following corollary.

Corollary 2.12 *Let $S = S_P \cup S_D$ be a set of boxes in 3-space such that S_P is a set of β-pipes with stabbing number σ and S_D has density δ. There is a box-tree for S such that the number of nodes visited by a range query with a query box Q is $O(\min_{0<\epsilon\leq 1}\{(1/\epsilon)((1/\epsilon)+\lambda)\log^4 n + k_\epsilon\})$, where $\lambda = \delta + (\beta+2)\sigma$ and k_ϵ is the number of boxes intersecting the extended range Q_ϵ.*

2.4 Analysis for other types of ranges

In the previous sections we assumed that the query range Q is an axis-parallel box. In this section we will generalize our results to constant-complexity ranges of arbitrary shape. A 3D query range is said to have constant complexity if its boundary consists of a constant number of algebraic surface patches of constant maximum degree, which are in turn bounded by a constant number of curves of constant maximum degree. In the analysis we only need the restriction that ∂Q, the boundary of Q, has a constant number of local extrema in any orthogonal cross-section, which is a condition fulfilled by the constant-complexity requirement.

We first prove a general theorem, stating that an LSF-interval tree with good query complexity for approximate range queries with boxes also has good query complexity for approximate range queries with other shapes. To this end we define a node ν to be *chargeable* with respect to a given range if all input boxes stored in $\mathcal{T}_\nu$ intersect that range, or if ν has a child with this property. Nodes for which this is not the case are *unchargeable.*

Theorem 2.13 *Let $\mathcal{T}$ be a d-dimensional box-tree on a set of n boxes, with $d \in \{2,3\}$. Suppose that, for any $0 < \epsilon \leq 1$, a query with a box B visits $O(f(n,\epsilon))$ nodes that are unchargeable with respect to the extended query box B_ϵ. Then a range query with a constant-complexity range Q visits $O(\min_{0<\epsilon\leq 1}\{(1/\epsilon)^{d-1} f(n,1) + k_\epsilon\})$ nodes of $\mathcal{T}$, where k_ϵ is the number of objects intersecting the ϵ-extended query range Q_ϵ.*

Proof: We first prove the theorem for $d = 2$.

Fix any $0 < \epsilon \leq 1$. We claim that we can cover ∂Q by $O(1/\epsilon)$ squares of edge length $\epsilon w/3$, where w is the diameter of Q (as was also shown for convex ranges by Arya and Mount [3]). To see this, consider a regular grid whose cells have size $\epsilon w/3$. Then ∂Q will intersect only $O(1/\epsilon)$ grid cells, because for any two adjacent cells intersected by a connected portion of ∂Q the following holds: either they contain a local extremum of ∂Q, or the length of the portion of ∂Q within the cells is at least $\epsilon w/3$. Since the total length of ∂Q is $O(w)$, only $O(1/\epsilon)$ grid cells can contain a portion of ∂Q of size $O(\epsilon w)$.

Now consider a query with a range Q. The number of visited nodes that are chargeable with respect to Q_ϵ is clearly $O(k_\epsilon)$. Any visited unchargeable node must have a bounding box that intersects at least one of the squares in the covering of ∂Q. To bound the number of such nodes, consider a square s in the covering. Define its extended square $s_{\epsilon'}$ as the set of points within L_∞-distance $\epsilon'\epsilon w/3$ from s. The boundary of the extended square has edge length $(1+2\epsilon')\epsilon w/3$ and intersects ∂Q, so even for ϵ' as large as 1, it is fully contained in

Q_ϵ. Hence, any node that is unchargeable with respect to Q_ϵ is unchargeable with respect to $s_{\epsilon'}$ for $\epsilon' = 1$. The number of nodes ν such that $b(\nu)$ intersects s and that are unchargeable with respect to $s_{\epsilon'}$ is $O(f(n, \epsilon'))$. Summing over all squares s and plugging in $\epsilon' = 1$, we get a bound of $O((1/\epsilon)f(n, 1))$ on the number of unchargeable nodes.

Hence, the total number of visited nodes is bounded by $O((1/\epsilon)f(n, 1) + k_\epsilon)$, as claimed.

The proof for $d = 3$ is similar. We start by covering ∂Q by cubes of edge length $\epsilon w/3$, where w is the diameter of Q. We claim that ∂Q intersects $O(1/\epsilon^2)$ cells of a regular grid with cells of the required size. Indeed, any intersected cell must have an intersected facet, so we can bound the number of intersected cells by summing the number of intersected facets over all $O(1/\epsilon)$ grid planes intersecting Q. Since ∂Q consists of a constant number of algebraic surface patches of constant maximum degree, which are in turn bounded by a constant number of curves of constant maximum degree, the same must hold for the intersection of ∂Q with a grid plane. Therefore, at most $O(1/\epsilon)$ facets can be intersected in each grid plane, and it follows that Q can be covered using $O(1/\epsilon^2)$ cubes of the required size. From here we can follow the proof for the case $d = 2$. $\square$

The analysis of the previous section shows that in all bounds derived there, the $O(k_\epsilon)$ term on the number of visited internal nodes is caused solely by nodes with a priority leaf as a child that stores a box intersecting the extended query range. Such nodes are chargeable, so Theorem 2.13 and Corollary 2.12 together imply the following result.

Corollary 2.14 *Let $S = S_P \cup S_D$ be a set of boxes in 3-space such that S_P is a set of β-pipes with stabbing number σ and S_D has density δ. There is a box-tree for S such that the number of nodes visited by a range query with a constant-complexity range Q is $O(\min_{0<\epsilon\leq 1}\{(\lambda/\epsilon^2)\log^4 n + k_\epsilon\})$, where $\lambda = \delta + (\beta + 2)\sigma$ and k_ϵ is the number of boxes intersecting the extended range Q_ϵ.*

Remark 2.15 The dependency on ϵ that we get is better by a factor of $O(1/\epsilon)$ than what Dickerson *et al.* [5] and Arya and Mount [3] get for queries with non-convex query ranges in point sets. Applying Theorem 2.13 to their structure, however, improves the dependency on ϵ by a factor of $O(1/\epsilon)$, leading to the same dependency as we get.

3 The BBD-interval tree

The bounding-volume hierarchy of the previous section is based on the longest-side-first kd-tree. It turns out that we can improve the results if we base the bounding-volume hierarchy on the so-called BBD-tree by Arya *et al.* [3]. The resulting hierarchy is somewhat unorthodox, however, as it uses non-convex bounding volumes.

Define a *donut* to be the set-theoretic difference of two boxes, one being contained in the other. That is, a donut is defined as $R^+ \setminus R^-$, where R^+ and R^- are boxes and $R^- \subset R^+$. The inner box R^- may be empty, in which case a donut is simply a box. The inner box may also touch the boundary of the outer box, in which case a degenerate type of donut results. It is not allowed to split the outer box, that is, $R^+ \setminus R^-$ should be connected. A *bounding donut* of a set of objects is a donut $R^+ \setminus R^-$ that contains all objects and whose outer box R^+ is the bounding box of the set. A *donut tree* for a set of objects is a bounding-volume hierarchy that uses bounding donuts.

Like a kd-tree, the BBD-tree by Arya *et al.* is a tree representing a recursive decomposition of space. Unlike in a kd-tree, however, the regions corresponding to the nodes of a BBD-tree are not boxes — they are donuts. It is possible to construct a donut tree on a set of boxes using a BBD-tree in a similar way as one can construct a box-tree from a kd-tree. The main advantage is that BBD-trees have a stronger 'packing property' than kd-trees: whereas in a longest-side-first kd-tree there can be $O(\log^{d-1} n)$ nodes whose regions are disjoint and intersect opposite facets of a cube, there can be only $O(1)$ such nodes in a BBD-tree [3]. This is the main reason that we can show the following result.

Theorem 3.1 *Let S be a set of boxes in 3-space with slicing number λ. There exists a donut-tree for S such that a query with a query box Q visits $O(\min_{0<\epsilon\leq 1}\{\log^3 n + (\lambda/\epsilon)\log^2 n + (\lambda/\epsilon^2)\log n + k_\epsilon\})$ nodes, where k_ϵ is the number of boxes intersecting the extended range Q_ϵ.*

This theorem can also be combined with Theorem 2.13 to get the following result:

Corollary 3.2 *Let S be a set of boxes in 3-space with slicing number λ. There exists a donut-tree for S such that a query with a constant-complexity range Q visits $O(\min_{0<\epsilon\leq 1}\{(1/\epsilon^2)\log^3 n + (\lambda/\epsilon^2)\log^2 n + k_\epsilon\})$ nodes, where k_ϵ is the number of boxes intersecting the extended range Q_ϵ.*

Because the details of the construction of the donut-tree and the analysis of its performance are similar to those of the LSF-interval tree, but still rather technical, we defer the details to Appendix A.

4 Concluding remarks

We have developed a new algorithm to construct box-trees, and analyzed its performance for approximate range queries when the input is a low-density scene combined with (almost) disjoint pipes. We proved that in such a setting—which was motivated by the need to perform collision checking in CAD models of industrial installations—one can achieve polylogarithmic query times. This is in sharp contrast with the $\Omega(n^{2/3} + k)$ lower bound for the query time in box-trees for arbitrary input proved by Agarwal *et al.* [1]. Our bounds almost match the best known bounds for range queries using box-trees in the much simper case of point data.

The assumptions we use in the analysis cannot be relaxed much further. In particular, we can give a lower bound construction showing that it is not possible to achieve polylogarithmic performance for box-trees when the input is uncluttered [4] instead of having low-density, even for approximate queries.

Our results can be used to perform ϵ-approximate nearest-neighbor searching, using the techniques described for instance in Duncan's thesis [6]. Thus, for input scenes satisfying the requirements above, approximate nearest-neighbor queries take time $O((\lambda/\epsilon^2)(\log^4 n)(\log\lambda + \log(1/\epsilon) + \log\log n))$ in our LSF-interval-tree, or $O(((1/\epsilon^2)\log^3 n + (\lambda/\epsilon^2)\log^2 n)(\log\lambda + \log(1/\epsilon) + \log\log n))$ in our BBD-interval-tree. (Note that for nearest-neighbor searching, ϵ is given as part of the query.)

In our future work we plan to investigate the performance of box-trees experimentally. We want to fine-tune our algorithm for constructing box-trees—in particular, we want to investigate whether the use of priority leaves, which are so convenient in the theoretical analysis, pays off in practice—and we want to compare it to existing heuristics.

References

[1] P.K. Agarwal, M. de Berg, J. Gudmundsson, M. Hammar, and H.J. Haverkort. Box-trees and R-trees with near-optimal query time. In *Proc. 17th Annu. Symp. Comput. Geom.*, pages 124-133, 2001. Accepted for publication in *Discrete Comput. Geom.*.

[2] N. Amato and Y. Wu. A randomized roadmap method for path and manipulation planning. In *Proc. IEEE Int. Conf. Robot. Autom.*, pages 113–120, 1996.

[3] A. Arya and D. Mount. Approximate range searching. *Comput. Geom. Theory Appl.*, 17(3-4):135–152, 2000.

[4] M. de Berg, M.J. Katz, A. F. van der Stappen, and J. Vleugels. Realistic input models for geometric algorithms. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 294–303, 1997. Accepted for publication in *Algorithmica*.

[5] M. Dickerson, C.A. Duncan, and M.T. Goodrich. K-D trees are better when cut on the longest side. In *Proc. 8th European Sympos. Algorithms*, volume 1879 of *LNCS*, pages 179–190, 2000.

[6] C.A. Duncan. *Balanced Aspect Ratio Trees.* PhD thesis, John Hopkins University, Baltimore, Maryland, 1999.

[7] C.A. Duncan, M.T. Goodrich, and S.G. Kobourov. Balanced aspect ratio trees: Combining the advantages of k-d trees and octrees. In *Proc. 10th Annu. ACM-SIAM Sympos. Discrete Algorithms*, pages 300–309, 1999.

[8] L. Kavraki. *Random networks in configuration space for fast path planning.* PhD thesis, Stanford University, 1995.

[9] MOLOG: Motion for Logistics. Esprit LTR Project 28226. `http://www.laas.fr/molog/`

[10] A.F. van der Stappen. *Motion Planning amidst Fat Obstacles.* PhD thesis, Utrecht University, 1994.

[11] A.F. van der Stappen, M.H. Overmars, M. de Berg, and J. Vleugels. Motion planning in environments with low obstacle density. *Discrete Comput. Geom.* 20:561–587 (1998).

[12] P. Švestka. *Robot motion planning using probabilistic roadmaps.* PhD thesis, Utrecht University, 1997.

A BBD-Interval-Trees

A.1 The construction of BBD-interval-trees

The concept of BBD-trees can serve as a basis for BBD-interval-trees, just like kd-trees with longest-side-first splitting can be used to construct LSF-interval trees. The construction algorithm is very similar to that for LSF-interval-trees. It will produce trees whose nodes have degree more than nine—but still $O(1)$. Conversion to a binary tree can easily be done and does not affect the bounds.

First, we divide the set of input boxes S into three oriented subsets S_x, S_y and S_z, as with LSF-interval-trees. We build a 3-BBD-interval-tree, that is, a BBD-interval tree whose defining region is 3-dimensional, for each of the oriented subsets separately, and combine them only at the top level.

A 3-BBD-interval-tree is now constructed as follows. We first build a 3-dimensional BBD-tree on the vertices of the input boxes. Then we convert the BBD-tree into a BBD-interval-tree in a top-down manner, as described below. Note that each node ν in the BBD-tree represents a donut-shaped cell, which is the set-theoretic difference of an outer box $R^+(\nu)$ and a possibly empty inner box $R^-(\nu)$.

Starting with the root and the set S_x (or S_y, or S_z), we handle each node ν as follows. From now on, let S be the set of boxes to be stored in the subtree rooted at ν.

1. If S is empty, the subtree rooted at ν can be ignored: it has no corresponding nodes in the BBD-interval-tree. Otherwise, we make a corresponding node ν' in the BBD-interval-tree, in which we store a bounding donut $b(S) \setminus R^-(\nu)$, where $b(S)$ is the bounding box of S. Note that $b(S) \subset R^+(\nu)$. In the analysis, the BBD-tree cell associated with ν will be referred to as the defining region $R(\nu')$ of ν'.

2. For each of the six directions $+x$, $-x$, $+y$, $-y$, $+z$, and $-z$ we take the box in S extending farthest in that direction. Each of these at most six boxes is stored in a separate leaf, called a *priority leaf*, immediately below the subtree's root node ν'. Let S' denote the set of remaining boxes in S.

3. In the BBD-tree, the region $R^+(\nu) \setminus R^-(\nu)$ corresponding to node ν is split into two subregions with a *splitting surface*. This is either a rectangle or a box; in the latter case the box always contains the inner box $R^-(\nu)$. Let R_1 and R_2 be the two resulting subregions; they are the regions corresponding to the children ν_1 and ν_2 in the BBD-tree. Define S_1 to be the subset of boxes in S that lie completely inside R_1, define S_2 to be the subset of boxes that lie completely inside R_2, and define $S^\times$ to be the subset of boxes that intersect the splitting surface.

   - We construct one subtree for ν' by recursively calling the procedure with child ν_1 and subset S_1.
   - We construct another subtree for ν' by recursively calling the procedure with child ν_2 and subset S_2.
   - The set $S^\times$ is handled as follows. In the LSF-interval tree, we could construct a 2-dimensional LSF-interval tree for the complete set $S^\times$, but now we need to be more careful.

     First of all, we need to ensure that any bounding donut that may be constructed when handling a subset of $S^\times$ is contained fully inside the region $R^+(\nu) \setminus R^-(\nu)$. To this end, we first partition $S^\times$ into six subsets such that the bounding box of each subset is disjoint from $R^-(\nu)$. This is possible because each box in $S^\times$ can be separated from $R^-(\nu)$ by a plane through one of the six facets of $R^-(\nu)$.

     Consider one subset, let's call it $\widetilde{S}$, of the resulting six subsets.

     If the splitting surface is a rectangle r, we simply store $\widetilde{S}$ in a 2-dimensional BBD-interval tree with r as defining region.

If the splitting surface is a box b, we proceed as follows. We cannot only construct a 2-BBD-interval tree for each facet f of b, because the intersection of a box in $\widetilde{S}$ with aff(f) need not be fully contained in f. Hence, we construct twenty-six subtrees: one for each vertex of b, one for each edge of b, and one for each facet of b. If a box in $\widetilde{S}$ contains one or more vertices of b, it is stored in the subtree constructed for one of these vertices (it doesn't matter which one); if it does not contain a vertex but intersects one or more edges, it is stored in the subtree constructed for one of these edges; otherwise it is stored in the subtree onstructed for one of the facets. Note that this means that a 3-node in a BBD-interval tree can have not only 3-nodes and 2-nodes, but also 1-nodes and 0-nodes as direct children.

It remains to describe the construction of d-BBD-interval-trees for $d < 3$.

2-BBD-interval-trees are built like 3-BBD-interval-trees, except that the underlying BBD-tree is a 2-dimensional tree subdividing a rectangle r instead of a 3-dimensional bounding box. Note that the bounding donuts stored in a 2-BBD-interval-tree are still 3-dimensional: the 2-dimensional defining regions are extended in the third dimension just enough to fit the 3-dimensional boxes stored in the corresponding subtree. Priority leaves are still created for both directions in all three dimensions.

1-BBD-interval-trees and 0-BBD-interval-trees are constructed just like LSF-interval-trees: there is nothing to be gained from using donuts here.

A.2 Analysis for range searching with boxes

The following lemma results from the packing constraints proven for the original BBD-tree by Arya *et al.* [3]:

Lemma A.1 *Let $\mathcal{T}$ be a d-dimensional BBD-interval-tree and let C be a k-dimensional cube, with $1 \leq k \leq d \leq 3$. Then there are only $O(1)$ d-nodes in $\mathcal{T}$ whose defining regions are disjoint and intersect opposite facets of C.*

The analysis for searching with boxes in BBD-interval trees now resembles the analysis for LSF-interval trees very much. The essential differences we have to take into account are the following.

First, wherever the analysis of a LSF-interval trees refers to Lemma 2.1 for the fact that only $O(\log n)$ or $O(\log^2 n)$ disjoint cells can intersect opposite sides of a square or cube, respectively, an analysis for a BBD-interval tree would refer to Lemma A.1 for the fact that only $O(1)$ disjoint cells can intersect opposite sides of a square or cube.

Second, the donut-shaped nodes introduce additional cases in the analysis.

We will now show how to derive the bounds for the BBD-interval-trees. The reader will be assumed to be familiar with the analysis of LSF-interval-trees, so we will not explain all arguments and notation in full detail anymore. Since 1-BBD-interval trees are the same as 1-LSF-interval trees, we start with the analysis of 2-BBD-interval-trees.

A.2.1 2-dimensional subtrees

We distinguish two cases, depending on whether or not the plane containing the 2-dimensional defining region $R(\mathcal{T})$ intersects Q_ϵ.

Case 1: The plane containing $R(\mathcal{T})$ intersects Q_ϵ.

Lemma A.2 *Let $\mathcal{T}$ be a 2-BBD-interval tree storing n boxes with stabbing number σ. Suppose we query $\mathcal{T}$ with a range Q such that the plane containing $R(\mathcal{T})$ intersects Q_ϵ. Let $\overline{Q}_\epsilon$ denote the intersection of Q_ϵ with the plane containing $R(\mathcal{T})$.*

(i) If no edge of $\overline{Q}_\epsilon$ intersects $R(\mathcal{T})$, then we visit $O(k_\epsilon(\mathcal{T}))$ nodes.

(ii) If no vertex of $\overline{Q}_\epsilon$ lies in $R(\mathcal{T})$, then we visit $O(\log n + k_\epsilon(\mathcal{T})) + C_Q(\mathcal{T}) \cdot O((\log n)/\epsilon)$ nodes.

(iii) In all cases—that is, there might be a vertex of $\overline{Q}_\epsilon$ lying inside $R(\mathcal{T})$—we visit $O(\log^2 n + (1/\epsilon + \sigma)\log n + k_\epsilon(\mathcal{T}))$ nodes.

Proof: As observed before, it suffices to bound the number of visited internal nodes.

(i) In this case, $R(\mathcal{T})$ must lie completely inside Q_ϵ, otherwise $\mathcal{T}$ would not be visited at all. The bound follows trivially.

(ii) We can distinguish two kinds of visited nodes: those in 2-subtrees whose defining regions lie completely inside Q_ϵ, and those in 2-subtrees whose defining regions intersect the boundary of Q_ϵ.

For the first type, the same argument as in part (i) of the Lemma applies; their number is bounded by $O(k_\epsilon(\mathcal{T}))$.

Let N be the collection of all remaining visited 2-nodes, and $\mathcal{G}(N)$ be the subgraph of $\mathcal{T}$ formed by those nodes. We bound the number of leaves in $\mathcal{G}(N)$ by the same argument as in the proof of Lemma 2.6, now referring to Lemma A.1 instead of Lemma 2.1. The number of leaves in $\mathcal{G}(N)$ is therefore bounded by $O(1) + C_Q(\mathcal{T}) \cdot O(1/\epsilon)$, and the total number of nodes in N is bounded by $O(\log n) + C_Q(\mathcal{T}) \cdot O((\log n)/\epsilon)$.

It remains to bound the number of descendants of the nodes in N. These are organized in subtrees whose roots are children of nodes in N and are not in N themselves. Consider such a root node μ, and let $\text{pa}(\mu)$ be its parent. There are four cases.

case 1: μ is a 2-node. Then μ must either lie completely inside $\overline{Q}_\epsilon$, in which case we already counted it, or completely outside $\overline{Q}_\epsilon$, in which case it is not visited. So we can ignore this case.

case 2: μ is a 1-node and $\text{aff}(R(\mu))$ cuts $\overline{Q}_\epsilon$. If $R(\mu)$ itself does not cut any edge of $\overline{Q}_\epsilon$, it lies either inside or outside $\overline{Q}_\epsilon$ and the number of nodes visited is trivially bounded by $O(k_\epsilon(\mathcal{T}_\mu))$. Otherwise, if one or two edges of $\overline{Q}_\epsilon$ are cut, we know by Lemma 2.2 that the number of nodes visited in $\mathcal{T}_\mu$ is bounded by $O(\log n + k_\epsilon(\mathcal{T}_\mu))$. Note that in this case $R(\mu)$ cuts $R(\text{pa}(\mu))$ in such a way that $\text{pa}(\mu)$ has two children in N, so the number of such nodes μ is bounded by the number of leaves in $\mathcal{G}(N)$, which is $O(1) + C_Q(\mathcal{T}) \cdot O(1/\epsilon)$. Hence, the total number of nodes visited in subtrees rooted at such nodes μ must be $O(\log n + k_\epsilon(\mathcal{T})) + C_Q(\mathcal{T}) \cdot O((\log n)/\epsilon)$.

case 3: μ is a 1-node and $\text{aff}(R(\mu))$ does *not* cut $\overline{Q}_\epsilon$. This implies that $R(\mu)$ lies 'next to Q_ϵ', on a line parallel to the closest edge of $\overline{Q}_\epsilon$. In principle, $R(\mu)$ could contain the projection of a vertex of $\overline{Q}_\epsilon$ on $\text{aff}(R(\mu))$. But in that case, the vertex should have to lie inside the inner box of $R(\text{pa}(\mu))$, since we are in case (ii) of the Lemma. By the construction of the 1-subtrees, there must then be a plane through one of the facets of the inner box which separates the bounding box of all boxes in $\mathcal{T}_\mu$ from the inner box, and thus, also from Q_ϵ. Hence, the subtree rooted at μ will be visited only if $R(\mu)$ is completely contained in the projection of Q_ϵ on $\text{aff}(R(\mu))$. Therefore, we can apply Lemma 2.3(i) and find that the number of nodes visited in $\mathcal{T}_\mu$ is $O(k_\epsilon(\mathcal{T}_\mu))$. Hence, the total number of nodes visited in subtrees rooted at such nodes μ is $O(k_\epsilon(\mathcal{T}))$.

case 4: μ is a 0-node. Similar to case 3, either there must be a plane that separates the bounding box of $\mathcal{T}_\mu$ from $\overline{Q}_\epsilon$ (so that $\mathcal{T}_\mu$ is not visited at all), or there must be an axis-parallel line through $R(\mu)$ that intersects $\overline{Q}_\epsilon$. Then, any node visited in $\mathcal{T}_\mu$ must have a priority leaf directly below it that stores a box intersecting $\overline{Q}_\epsilon$. Hence, the total number of nodes visited in such subtrees is $O(k_\epsilon(\mathcal{T}))$.

In total, we find the claimed bound of $O(\log n + k_\epsilon(\mathcal{T})) + C_Q(\mathcal{T}) \cdot O((\log n)/\epsilon)$.

(iii) We can distinguish three types of visited nodes: the two types that were also considered in the proof of part (i) and (ii), and a third type, namely 2-nodes containing a corner of $\overline{Q}_\epsilon$ and their descendant 1-nodes and 0-nodes.

The number of nodes of the first two types can be bounded as in the proof of part (ii). As for the third type, we note that there are $O(\log n)$ 2-nodes containing a corner of $\overline{Q}_\epsilon$. If μ is a 0-node or 1-node that is a child of such a node, then the query time in $\mathcal{T}_\mu$ is bounded to $O(\log n + \sigma + k_\epsilon(\mathcal{T}_\mu))$ by Lemmas 2.2 and 2.3). This leads to a total of $O(\log^2 n + \sigma \log n + k_\epsilon(\mathcal{T}))$. Using $C_Q(\mathcal{T}) \leq 4 + 8\epsilon$, we get a final bound of $O(\log^2 n + (1/\epsilon + \sigma)\log n + k_\epsilon(\mathcal{T}))$.

□

Case 2: The plane containing $R(\mathcal{T})$ does not necessarily intersect Q_ϵ.

Lemma A.3 *Let $\mathcal{T}$ be a 2-BBD-interval tree storing n boxes with slicing number λ. Let $\overline{Q}_\epsilon$ denote the projection of Q_ϵ onto the plane containing $R(\mathcal{T})$.*

(i) If no edge of $\overline{Q}_\epsilon$ intersects $R(\mathcal{T})$, then we visit $O(k_\epsilon(\mathcal{T}))$ nodes.

(ii) If no vertex of $\overline{Q}_\epsilon$ intersects $R(\mathcal{T})$, then we visit $O(\lambda \log n + k_\epsilon(\mathcal{T})) + C_Q(\mathcal{T}) \cdot O((\lambda \log n)/\epsilon)$ nodes.

(iii) In all cases — that is, there might be a vertex of $\overline{Q}_\epsilon$ lying inside $R(\mathcal{T})$ — we visit $O(\log^2 n + (\lambda \log n)/\epsilon + k_\epsilon(\mathcal{T}))$ nodes.

Proof:

(i) In this case, $R(\mathcal{T})$ must lie completely inside $\overline{Q}_\epsilon$, otherwise $\mathcal{T}$ would not be visited at all. Using the priority leaves as with LSF-interval-trees, we find a bound of $O(k_\epsilon(\mathcal{T}))$.

(ii) This case is handled basically as in the proof of Lemma A.2(ii), now referring to Lemma 2.4 instead of Lemma 2.3 and to Lemma 2.3 instead of Lemma 2.2.

Again, we will worry only about 2-nodes visited that are not addressed by part (i) of the lemma, and their descendants. Let N be the collection of such 2-nodes, and $\mathcal{G}(N)$ be the subgraph of $\mathcal{T}$ formed by those nodes. The number of leaves in $\mathcal{G}(N)$ is $O(1) + C_Q(\mathcal{T}) \cdot O(1/\epsilon)$, and the total number of nodes in N is $O(\log n) + C_Q(\mathcal{T}) \cdot O((\log n)/\epsilon)$.

The descendants of the nodes in N are organized in subtrees whose roots are children of nodes in N and are not in N themselves. Consider such a root node μ, and let $\text{pa}(\mu)$ be its parent.

case 1: μ is a 2-node. As before, this case can be ignored.

case 2: μ is a 1-node and $\text{aff}(R(\mu))$ cuts $\overline{Q}_\epsilon$. If $R(\mu)$ itself does not cut any edge of $\overline{Q}_\epsilon$, it lies either inside or outside Q_ϵ and the number of nodes visited is bounded by $O(k_\epsilon(\mathcal{T}_\mu))$. Otherwise, if one or two edges of $\overline{Q}_\epsilon$ are cut, we know by Lemma 2.3(ii) that the number of nodes visited in $\mathcal{T}_\mu$ is bounded by $O(\log n + \sigma + k_\epsilon(\mathcal{T}_\mu))$. Note that in this case $R(\mu)$ cuts $R(\text{pa}(\mu))$ in such a way that $\text{pa}(\mu)$ has two children in N, so the number of such nodes μ is bounded by the number of leaves in $\mathcal{G}(N)$, which is $O(1) + C_Q(\mathcal{T}) \cdot O(1/\epsilon)$. Hence, the total number of nodes visited in subtrees rooted at such nodes μ must be $O(\log n + \sigma + k_\epsilon(\mathcal{T})) + C_Q(\mathcal{T}) \cdot O((\log n)/\epsilon + \sigma/\epsilon)$.

case 3: μ is a 1-node and $\text{aff}(R(\mu))$ does *not* cut $\overline{Q}_\epsilon$. This implies that $R(\mu)$ lies 'next to $\overline{Q}_\epsilon$', on a line parallel to the closest edge of $\overline{Q}_\epsilon$. As explained in the proof of Lemma A.2, we may assume that $R(\mu)$ does not contain the projection of a vertex of $\overline{Q}_\epsilon$ on $\text{aff}(R(\mu))$. Therefore, we can apply Lemma 2.4(i) and find that the number of nodes visited in $\mathcal{T}_\mu$ is $O(\lambda) + C_Q(\mathcal{T}_\mu) \cdot O(\lambda/\epsilon)$. As in the proof of Lemma 2.6, we can use the fact that the number of such nodes μ is bounded by $|N|$ and that $\sum_{\{\mu|\text{pa}(\mu)\in N\}} C_Q(\mathcal{T}_\mu) = O(\log n) C_Q(\mathcal{T})$. Hence, we find a total query time for this case of $O(\lambda \log n) + C_Q(\mathcal{T}) \cdot O(\lambda(\log n)/\epsilon)$.

case 4: μ is a 0-node. Clearly, the total asymptotic query time in the subtrees rooted at such nodes cannot be worse than the number of boxes stored in such trees, which is $O(|N| \cdot \sigma) = O(\sigma \log n) + C_Q(\mathcal{T}) \cdot O(\sigma(\log n)/\epsilon)$.

Using $\sigma \leq \lambda$, we find an overall bound of $O(\lambda \log n + k_\epsilon(\mathcal{T})) + C_Q(\mathcal{T}) \cdot O(\lambda(\log n)/\epsilon)$.

(iii) Again, we can distinguish between the types of nodes that were also considered in the proof of part (ii), and another type, namely the 2-nodes containing a corner of $\overline{Q}_\epsilon$ and their descendant 1-nodes and 0-nodes.

The number of nodes of the first types can be bounded as in the proof of part (ii), using that $C_Q(\mathcal{T}) \leq 4 + 8\epsilon$. As for the other type, we note that there are $O(\log n)$ 2-nodes containing a corner of $\overline{Q}_\epsilon$. If μ is a 0-node or 1-node that is a child of such a node, then the query time in $\mathcal{T}_\mu$ is bounded to $O(\log n + \lambda/\epsilon + k_\epsilon(\mathcal{T}_\mu))$ by Lemmas 2.3 and 2.4. This leads to a total of $O(\log^2 n + \lambda(\log n)/\epsilon + k_\epsilon(\mathcal{T}))$.

□

A.2.2 3-dimensional trees

Theorem A.4 *Let $\mathcal{T}$ be a 3-BBD-interval tree storing n boxes with slicing number λ and let ϵ be any positive real number. Then a query in $\mathcal{T}$ with a range Q will visit $O(\min_{0<\epsilon\leq 1}\{\log^3 n + (\lambda/\epsilon)\log^2 n + (\lambda/\epsilon^2)\log n + k_\epsilon\})$ nodes, where k_ϵ is the number of boxes intersecting the extended range Q_ϵ.*

Proof: As observed before, it suffices to bound the number of visited internal nodes.

There is a one-to-one correspondence between facets of Q and facets of Q_ϵ, between edges of Q and edges of Q_ϵ, and between vertices of Q and vertices of Q_ϵ. Suppose that the defining region of a node ν intersects a face f_ϵ of Q_ϵ but not the corresponding face f of Q. Then f_ϵ and f must be separated by a plane through a boundary facet of $R(\nu)$. If this is a plane through a facet of $R^+(\nu)$, the node must lie completely outside Q and will not be visited. If it is a plane through a facet of $R^-(\nu)$, then for each child of ν that is a 0-node, 1-node or 2-node intersecting f_ϵ, the plane that separates it from $R^-(\nu)$ will also separate it from Q, so it will not be visited. Therefore, when discussing bounds on the query time in 0-nodes, 1-nodes or 2-nodes which *are* visited, we may assume that whenever a facet f_ϵ of Q_ϵ is intersected, the corresponding facet f of Q is intersected as well.

The internal nodes visited can now be partitioned into four categories.

- 3-nodes ν such that $R(\nu)$ intersects no facet of Q_ϵ, and their descendant 2-nodes, 1-nodes and 0-nodes.

  In this case $R(\nu)$ must lie completely inside Q_ϵ, otherwise ν would not be visited at all. A bound of $O(k_\epsilon)$ on the number of such nodes and their descendants follows trivially.

- 3-nodes ν such that $R(\nu)$ intersects at least one facet of Q_ϵ, but none of its edges, and their descendant 2-nodes, 1-nodes and 0-nodes.

  Let N be the collection of all 3-nodes in this category, and $\mathcal{G}(N)$ be the subgraph of $\mathcal{T}$ formed by those nodes. We bound the number of leaves in $\mathcal{G}(N)$ by a similar argument as in the proof of Theorem 2.7, part (iii), now covering $Q_\epsilon \setminus Q$ with $O(1/\epsilon^2)$ cubes and referring to Lemma A.1 instead of Lemma 2.1. The number of leaves in $\mathcal{G}(N)$ is therefore bounded by $O(1/\epsilon^2)$, and the total number of nodes in N is bounded by $O((\log n)/\epsilon^2)$.

  It remains to bound the number of visited nodes in the 2-subtrees, 1-subtrees and 0-subtrees rooted at the children of the nodes in N. Let μ be such a root child and let $\mathrm{pa}(\mu) \in N$ be its parent. There are six cases.

  case 1: μ is a 2-node and $\mathrm{aff}(R(\mu))$ intersects Q_ϵ. If $R(\mu)$ does not intersect any facet of Q_ϵ, it lies either inside Q_ϵ or outside Q and the number of nodes visited is trivially bounded by $O(k_\epsilon(\mathcal{T}_\mu))$. Otherwise, if one or two facets of Q_ϵ are intersected, we know by Lemma A.2(ii) that the number of nodes visited in $\mathcal{T}_\mu$ is bounded by $O(\log n + k_\epsilon(\mathcal{T}_\mu)) + C_Q(\mathcal{T}_\mu) \cdot O((\log n)/\epsilon)$. Note that in this case $R(\mu)$ cuts $R(\mathrm{pa}(\mu))$ in such a way that $\mathrm{pa}(\mu)$ has two children in N, so the number of such nodes μ is bounded by the number of leaves in $\mathcal{G}(N)$, which is $O(1/\epsilon^2)$. Hence, the total number of nodes visited in subtrees rooted at such nodes μ is $O((\log n)/\epsilon^2 + k_\epsilon) + \sum_{\{\mu|\mathrm{pa}(\mu)\in N\}} C_Q(\mathcal{T}_\mu) O((\log n)/\epsilon)$.

  We proceed to bound $\sum_\mu C_Q(\mathcal{T}_\mu)$. To simplify the discussion, let's assume that the defining regions $R(\mu)$ and $R(\mathrm{pa}(\mu))$ intersect the top facet of Q_ϵ. As explained above, we may assume that they intersect the top facet of Q as well. We can bound $\sum_\mu C_Q(\mathcal{T}_\mu)$ as follows. For each node μ we have that $C_Q(\mathcal{T}_\mu)w$ is the length of $R(\mu)$ as seen from above. The defining regions $R(\mu)$ cut the top facet of Q_ϵ into a number of cells. Since these cuts intersect the top facets of both Q_ϵ and Q, they must have height at least ϵw. We now use the property that BBD-trees have fat donuts. In particular, if the inner box does not touch a given facet of the outer box of a donut, then the distance between the inner box and the facet is not smaller than the size of the inner box. Hence, parallel cuts of height at least ϵw cannot be arbitrarily close together — in fact, the total length of the cuts through any $\Theta(\epsilon w)$ size square of the top facet of Q_ϵ must be $O(\epsilon w)$. The top facet of Q_ϵ can be covered by $O(1/\epsilon^2)$ squares of size $\Theta(\epsilon w)$, so the total length $\sum_\mu C_Q(\mathcal{T}_\mu)w$ is $O(1/\epsilon w)$, and $\sum_\mu C_Q(\mathcal{T}_\mu)$ is $O(1/\epsilon)$.

With this, we find a total bound of $O((\log n)/\epsilon^2 + k_\epsilon)$ for this case.

case 2: μ is a 2-node and $\text{aff}(R(\mu))$ does *not* intersect Q_ϵ. This implies that $R(\mu)$ lies 'next to Q_ϵ', on a plane parallel to the closest facet of Q_ϵ. Following a similar argument as in the analysis of case 3 in Lemma A.2(ii), we conclude that $R(\mu)$ does not intersect the projection of any edge of Q_ϵ on $\text{aff}(R(\mu))$. Therefore, we can apply Lemma A.3(i) and find that the number of nodes visited in $\mathcal{T}_\mu$ is $O(k_\epsilon(\mathcal{T}_\mu))$. Hence, the total number of nodes visited is $O(k_\epsilon)$ for this case.

case 3: μ is a 1-node and $\text{aff}(R(\mu))$ intersects Q_ϵ. If $R(\mu)$ does not intersect any facet of Q_ϵ, it lies either inside Q_ϵ or outside Q and the number of nodes visited is bounded by $O(k_\epsilon(\mathcal{T}_\mu))$. If one or two facets of Q_ϵ are intersected, we know by Lemma 2.2(ii) that the number of nodes visited in $\mathcal{T}_\mu$ is bounded by $O(\log n + k_\epsilon(\mathcal{T}_\mu))$. Note that in this case $R(\mu)$ must be (part of) an edge of a box-shaped cut that cuts $R(\text{pa}(\mu))$ in such a way that $\text{pa}(\mu)$ has two children in N, so the number of such nodes μ is bounded by the number of leaves in $\mathcal{G}(N)$, which is $O(1/\epsilon^2)$. Hence, the total number of nodes visited in subtrees rooted at such nodes μ is $O((\log n)/\epsilon^2 + k_\epsilon)$.

case 4: μ is a 1-node and only one axis-parallel plane containing $R(\mu)$ intersects Q_ϵ. Consider the projection of Q_ϵ on the other axis-parallel plane containing $R(\mu)$. Following a similar argument as in the analysis of case 3 in Lemma A.2(ii), we conclude that $R(\mu)$ does not intersect any edge of this projection, and therefore, $R(\mu)$ does not contain any vertex of the projection of Q_ϵ on $\text{aff}(R(\mu))$. Therefore, we can apply Lemma 2.3(i) and find that the number of nodes visited in $\mathcal{T}_\mu$ is $O(k_\epsilon(\mathcal{T}_\mu))$. Hence, the total number of nodes visited is $O(k_\epsilon)$ for this case.

case 5: μ is a 1-node and both axis-parallel planes containing $R(\mu))$ are disjoint from Q_ϵ. Consider the edge e of Q_ϵ which is parallel to $R(\mu)$ and closest to $\text{aff}(R(\mu))$. Since $R(\text{pa}(\mu))$ does not intersect any edge of Q_ϵ by definition, e must intersect its inner box $R^-(\text{pa}(\mu))$. By the construction of the 1-subtrees, there must be a plane through one of the facets of the inner box which separates the bounding box of all boxes in $\mathcal{T}_\mu$ from the inner box, and thus, also from Q_ϵ. Therefore, such subtrees $\mathcal{T}_\mu$ are not visited at all.

case 6: μ is a 0-node. Clearly, the total asymptotic query time in the subtrees rooted at such nodes cannot be worse than the number of boxes stored in such trees, which is $O(|N| \cdot \sigma) = O(\sigma(\log n)/\epsilon^2)$.

- 3-nodes ν such that $R(\nu)$ intersects at least one edge of Q_ϵ, but none of its vertices, and their descendant 2-nodes, 1-nodes and 0-nodes.

  Let N be the collection of all 3-nodes in this category, and $\mathcal{G}(N)$ be the subgraph of $\mathcal{T}$ formed by those nodes. We bound the number of leaves in $\mathcal{G}(N)$ by the same argument as in the proof of Theorem 2.7, part (iii), now referring to Lemma A.1 instead of Lemma 2.1. The number of leaves in $\mathcal{G}(N)$ is therefore bounded by $O(1/\epsilon)$, and the total number of nodes in N is bounded by $O((\log n)/\epsilon)$.

  It remains to bound the number of visited nodes in the 2-subtrees, 1-subtrees and 0-subtrees rooted at the children of the nodes in N. Let μ be such a root child and let $\text{pa}(\mu) \in N$ be its parent. There are four cases.

case 1: μ is a 2-node and $\text{aff}(R(\mu))$ intersects Q_ϵ. If $R(\mu)$ does not intersect any facet of Q_ϵ, it lies either inside Q_ϵ or outside Q and the number of nodes visited is trivially bounded by $O(k_\epsilon(\mathcal{T}_\mu))$.

Otherwise, if $R(\mu)$ is (part of) a cut that divides $\text{pa}(\mu)$ such that it has two children in N, we know by Lemma A.2(iii) that the number of nodes visited in $\mathcal{T}_\mu$ is bounded by $O(\log^2 n + (1/\epsilon + \sigma)\log n + k_\epsilon(\mathcal{T}_\mu))$. Since the number of such cuts is bounded by the number of leaves in $\mathcal{G}(N)$, the total number of nodes visited in subtrees rooted at such nodes μ is $O((\log^2 n)/\epsilon + (\log n)/\epsilon^2 + (\sigma \log n)/\epsilon + k_\epsilon)$.

The remaining subcase is that $R(\mu)$ is part of a cut such that $\text{pa}(\mu)$ has one child in $\mathcal{G}(N)$ and one child piercing a facet (that is: one child in N as defined in the category handled above). By Lemma A.2(ii) we know that the number of nodes visited in each such tree $\mathcal{T}_\mu$ is $O(\log n + k_\epsilon(\mathcal{T}_\mu)) + C_Q(\mathcal{T}_\mu) \cdot O((\log n)/\epsilon)$. From the analysis of case 1 in the previous category we know that $\sum_\mu C_Q(\mathcal{T}_\mu)$ is $O(1/\epsilon)$, and we find that the total number of nodes visited for this subcase can be bounded by $|N|O(\log n) + \sum_\mu k_\epsilon(\mathcal{T}_\mu) + \sum_\mu C_Q(\mathcal{T}_\mu)(\log n)/\epsilon) = O((\log^2 n)/\epsilon + (\log n)/\epsilon^2 + k_\epsilon)$.

In total, we obtain a bound of $O((\log^2 n)/\epsilon + (\log n)/\epsilon^2 + (\sigma \log n)/\epsilon + k_\epsilon)$ for this case.

case 2: μ is a 2-node and $\text{aff}(R(\mu))$ does *not* intersect Q_ϵ. This implies that $R(\mu)$ lies 'next to Q_ϵ', on a plane which is parallel to the closest facet of Q_ϵ. Following a similar argument as in the analysis of case 3 in Lemma A.2(ii), we conclude that $R(\mu)$ does not intersect the projection of any vertex of Q_ϵ on $\text{aff}(R(\mu))$. Therefore, we can apply Lemma A.3(ii) and find that the number of nodes visited in $\mathcal{T}_\mu$ is $O(\lambda \log n + k_\epsilon(\mathcal{T}_\mu)) + C_Q(\mathcal{T}_\mu) \cdot O(\lambda(\log n)/\epsilon)$. Because any point on an edge of Q_ϵ lies in at most $O(\log n)$ defining regions of 3-nodes, it holds that $\sum_\mu C_Q(\mathcal{T}_\mu) = O((1 + 2\epsilon)\log n)$ and we get a total bound of $|N|O(\lambda \log n) + O(\sum_\mu k_\epsilon(\mathcal{T}_\mu)) + \sum_\mu C_Q(\mathcal{T}_\mu) \cdot O(\lambda(\log n)/\epsilon) = O((\lambda \log^2 n)/\epsilon + k_\epsilon)$ for this case.

case 3: μ is a 1-node. In the worst-case, the query time in each subtree $\mathcal{T}_\mu$ is $O(\log n + \lambda/\epsilon)$ (Lemma 2.4). Therefore, the total query time for this case is at most $|N| \cdot O(\log n + \lambda/\epsilon) + \sum O(k_\epsilon(\mathcal{T}_\mu)) = O((\log^2 n)/\epsilon + (\lambda \log n)/\epsilon^2 + k_\epsilon)$.

case 4: μ is a 0-node. Clearly, the total asymptotic query time in the subtrees rooted at such nodes cannot be worse than the number of boxes stored in such trees, which is $O(|N| \cdot \sigma) = O(\sigma(\log n)/\epsilon)$.

- 3-nodes ν such that $R(\nu)$ intersects at least one vertex of Q_ϵ, and their descendant 2-nodes, 1-nodes and 0-nodes not included in the previous categories.

  At most $O(\log n)$ 3-nodes can contain a vertex of Q_ϵ. Each of them may have $O(1)$ 2-subtrees with query time $O(\log^2 n + (\lambda \log n)/\epsilon + k_\epsilon(\mathcal{T}_\nu))$ each (by Lemma A.3(iii)), $O(1)$ 1-subtrees with query time $O(\log n + \lambda/\epsilon)$ each in the worst case (Lemma 2.4(ii)) and $O(1)$ 0-subtrees whose query times are trivially bounded by $O(\sigma)$ each. This leads to a total of $O(\log^3 n + (\lambda \log^2 n)/\epsilon + k_\epsilon)$ visited nodes in this category.

Since the number of visited nodes of each category is within the bound claimed, this proves the theorem. $\square$