

On R-trees with low query complexity

M. de Berg¹ J. Gudmundsson¹ M. Hammar² M. Overmars¹

¹ Institute of Information and Computing Sciences, Utrecht University,
PO Box 80.089, 3508 TB Utrecht, the Netherlands.
{markdb, joachim, markov}@cs.uu.nl

² Dipartimento di Informatica ed Applicazioni, Università di Salerno
Baronissi (SA) – 84081, Italy.
hammar@udsab.dia.unisa.it

Abstract

The R-tree is a well-known bounding-volume hierarchy that is suitable for storing geometric data on secondary memory. Unfortunately, no good analysis of its query time exists. We describe a new algorithm to construct an R-tree for a set of planar objects that has provably good query complexity for point location queries and range queries with ranges of small width. For certain important special cases, our bounds are optimal. We also show how to update the structure dynamically, and we generalize our results to higher-dimensional spaces.

1 Introduction

Background. Spatial data structures are important in many areas of computer science where the data is geometric—computer graphics, virtual reality, geographic information systems (GIS), and CAD/CAM are obvious examples. Such data structures store a set \mathcal{S} of geometric objects such that various queries can be answered efficiently. Widely used queries are *range queries*, which ask for all objects in \mathcal{S} intersecting a query range Q . *Point-location queries* are a special case of range queries, where the query range is a point. Another important class of queries are *nearest-neighbor queries*, where one asks for the object from \mathcal{S} closest to a query point.

Researchers in computational geometry have developed data structures for many such queries. The asymptotic worst-case behavior of these data structures is usually quite good—or at least close to the theoretical lower bounds. In practice, however, other kinds of data structures are often used. One reason is that in many applications storage is a very critical issue: $\Theta(n \log n)$ storage and even linear storage with a large constant factor is sometimes too expensive. Another reason is that the structures developed in computational geometry are usually dedicated to a very specific setting: a structure for range searching with rectangular ranges in a set of line segments will not work for searching with rectangular ranges in a set of curve segments, or for searching with circular ranges in a set of line segments. In a typical application one needs to perform several different types of queries, and it is desirable to have a data structure that supports all, or at least many, of them.

A versatile geometric data structure that is used in practice is the *bounding-volume hierarchy*—see for example van den Bergen’s thesis [6] and the references therein. This is a tree structure, whose leaves store the input objects and whose internal nodes store a bounding box¹ (or some other bounding volume) for the objects in the subtree rooted at that node. A bounding-volume hierarchy uses linear space and it can store any type of objects. It can perform range queries with any type of range; this means it can also do point location, since this is simply a range query with a point range. It can also serve to answer nearest-neighbor queries. Although bounding-volume hierarchies are used quite often in practice, there is hardly any theoretical analysis supporting

¹The bounding box of a (collection of) object(s) is the smallest axis-aligned box containing the object(s).

their efficiency. This is the goal of our paper: we want to design a bounding-volume hierarchy for which we can prove bounds on the worst-case query time. More precisely, we will do this for a special class of bounding-volume hierarchies that is used a lot in GIS, namely *R-trees*.

The R-tree, which was proposed by Guttman [12], is a bounding-volume hierarchy that is suitable for storing data on secondary storage. It can be considered a geometric version of a B-tree. More precisely, an R-tree for a set \mathcal{S} of n objects in the plane is defined as follows. Let $\mathcal{R}_{\mathcal{S}}$ denote the set of bounding boxes of the objects in \mathcal{S} . Let t , the *minimum degree* of \mathcal{T} , be a fixed parameter. Then an R-tree for \mathcal{S} of minimum degree t is a tree \mathcal{T} with the following properties:

- (i) Each leaf node of \mathcal{T} contains between t and $2t$ rectangles from $\mathcal{R}_{\mathcal{S}}$; the only exception is when a leaf is also the root, in which case it can contain between 1 and $2t$ rectangles from $\mathcal{R}_{\mathcal{S}}$. With each rectangle a pointer is stored to the corresponding object in \mathcal{S} .
- (ii) Each internal node ν of \mathcal{T} has between t and $2t$ children,² except when it is the root node, which has between 2 and $2t$ children. An internal node stores for each child the bounding box of all objects stored in the subtree rooted at that child.
- (iii) All leaves are at the same level.

The depth of an R-tree storing n objects in its leaves is $\Theta(\log n / \log t)$. The idea, like for B-trees, is to choose t as large as possible in order to minimize the depth of the tree, while making sure that each internal node still fits into one page of external memory. The R-tree is one of the most widely used geometric data structure in geographic information systems and spatial databases—see for example the survey articles by Nievergelt and Widmayer [14] and by Gaede and Günter [9].

A range query with a range Q is answered by traversing \mathcal{T} top-down: starting at the root, we recursively query the subtree of each child whose bounding box (which is stored at the root) intersects Q . When we arrive at a leaf, we check for each rectangle from $\mathcal{R}_{\mathcal{S}}$ stored at the leaf whether it intersects Q ; if so, we test the corresponding object from \mathcal{S} for intersection with Q and report it if it does. Because an R-tree resides in secondary storage, the efficiency of the query answering process is determined by the number of disk accesses. Each node fits in one page of memory, so we have exactly one disk access for each visited node. A node is visited only when the corresponding bounding box (which is stored at the parent) intersects the range Q . Hence, for a range Q we define the query complexity of \mathcal{T} as the number of bounding boxes stored in the R-tree (including those in internal nodes) intersecting Q . Our goal is to construct the R-tree in such a way that the maximum query complexity, over all possible ranges Q , is provably small.

The key to the efficiency of an R-tree is how the underlying objects are grouped together in subtrees. Intuitively, for each subtree we would like the objects in its leaves to be clustered, so that their bounding box does not have too much empty space or overlap too many other bounding boxes. The hope is that this means that only few nodes are visited unnecessarily. A number of heuristics has been proposed to achieve this [5, 7, 10, 11, 12, 15]. To our knowledge, however, no worst-case performance bounds have been proved for the query complexity in R-trees.

The only analytic result that we know of is by Faloutsos et al. [8]. Their setting is rather limited, however: they consider a 1-dimensional version of the R-tree, and assume that the input intervals have only one or two different sizes and that they are distributed uniformly. For this case they bound the number of nodes visited when answering a point-location query. They consider two heuristics to build the R-tree, and obtain $\Theta(\log n / \log t)$ bounds. A related result is by Becker et al. [4], who gives an optimal solution to a problem arising for some of the heuristics used to update an R-tree dynamically. The goal of our paper is to describe an algorithm for constructing R-trees whose worst-case query complexity is good. We show this for point-location queries and for range queries with ranges of small width.

²The original definition allows between t and s children for some given s with $s \geq 2t$, but for simplicity we assume $s = 2t$.

Our results. Define the *stabbing number* of a set of rectangles as the maximum number of rectangles containing a common point. Note that the worst-case complexity of a point-location query is equal to the stabbing number of the set of rectangles stored with the internal and leaf nodes of \mathcal{T} . Unfortunately, the stabbing number of $\mathcal{R}_{\mathcal{S}}$, the set of bounding boxes of the input objects, can already be n . This can even be the case when the input object objects are disjoint: take a set of n parallel line segments of slope 1 that are very close to each other, as in Fig. 1. Hence,

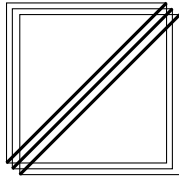


Figure 1: A set of disjoint input objects—the diagonal segments—whose set of bounding boxes has linear stabbing number.

we cannot achieve a sublinear bound on the number of visited nodes for general scenes. Therefore we will express our bounds in terms of σ , the stabbing number of $\mathcal{R}_{\mathcal{S}}$. A second parameter that we will use in our analysis is ρ , the *x-scale factor*, or *scale factor* for short, of \mathcal{S} . This is the ratio of the largest x -extent to the smallest x -extent of the objects in \mathcal{S} . (The x -extent of an object is the length of its projection onto the x -axis.) The scale factor has also been used by Zhou and Suri [17] for a different, but related problem: they analyzed how many intersections there can be among the bounding boxes of a given set of objects, as compared to the number of intersections among the original objects.

Our results can now be stated as follows.

We describe in Section 2 a new algorithm to construct an R-tree for a given set \mathcal{S} of n objects in the plane. It is similar to some known algorithms [7, 15] to construct R-trees in that it works by ordering the bounding boxes of the objects along a 1-dimensional curve. We deviate from those methods mainly by the way we deal with input objects that differ a lot in size. This is essential to be able to prove good worst-case bounds on the query complexity.

In Section 3 we analyze the query complexity of our structure. We show that a point-location query visits $O((\sigma(\log \rho + 1) \log n / \log t))$ nodes. When σ and ρ are constant, which we expect to be true in many applications, this is optimal. We can get rid of the dependency of ρ at the expense of an extra $O(\log n)$ factor, leading to an $O(\sigma \log^2 n / \log t)$ bound on the number of visited nodes. We also analyze the number of nodes visited by a range query. Here we obtain a bound of $O((\sigma(\log \rho + 1) + w + k) \log n / \log t)$, where w is the ratio of the x -extent of the query range to the smallest x -extent of any object in \mathcal{S} and k is the number of reported objects.

In Section 4 we show how to update the R-tree dynamically, using $O(\log n / \log t)$ disk accesses per operation. In Section 5 we generalize our results to higher dimensions. We conclude by mentioning some open problems in Section 6.

2 The construction of R-trees

Let \mathcal{S} be a set of n objects in the plane, and let $\mathcal{R} = \mathcal{R}_{\mathcal{S}}$ be the set of bounding boxes of these objects. Let σ be the stabbing number of \mathcal{R} , that is, the maximum number of rectangles in \mathcal{R} containing any query point. Let ρ denote the scale factor of \mathcal{R} (and, hence, of \mathcal{S}) as defined above.

Before we proceed, we need some notation. Let \mathcal{T} be an R-tree for \mathcal{S} . We denote the subset of objects stored in the subtree rooted at a node ν —more precisely, the subset of objects for which we have pointers stored in the leaves of this subtree—by $\mathcal{R}(\nu)$. We let $b(\nu)$ denote the bounding box of the objects in $\mathcal{R}(\nu)$. We say that $\mathcal{R}(\nu)$ is the *defining set* of $b(\nu)$. Notice that $b(\nu)$ is not stored at ν , but at the parent of ν .

To introduce the key idea behind our construction we start with a simple algorithm that assumes that $\rho \leq 2$. In Section 2.2 we modify the algorithm to handle arbitrary scale factors efficiently.

2.1 Sets with scale factor at most two

When the scale factor of \mathcal{R} is two or less, we proceed as follows. Assume without loss of generality that the smallest x -extent of any rectangle in \mathcal{R} is equal to one. We partition the plane into vertical strips of unit width. We associate each rectangle in \mathcal{R} with the strip containing its left edge, where strips are closed to the left and open to the right. A strip that has no rectangles associated with it is called *empty*, otherwise it is *non-empty*. Let s_1, \dots, s_k be the sequence of strips starting at the leftmost non-empty strip and ending at the rightmost non-empty strip. Notice that the sequence can contain empty strips—see Figure 2. Denote the set of rectangles associated to s_i by $\mathcal{R}(s_i)$, and let $n_i := |\mathcal{R}(s_i)|$. We order the rectangles in \mathcal{R} in a left-to-right and bottom-to-top fashion, based on the strips: the rectangles associated to the leftmost strip are called r_1, \dots, r_{n_1} in order of increasing y -coordinate of their bottom edges, the rectangles associated to the second leftmost non-empty strip are called $r_{n_1+1}, r_{n_1+2}, \dots$ in order of increasing y -coordinate of their bottom edges, and so on. We call the resulting ordering on the rectangles the *strip order*; it is illustrated in Figure 2. The following observation will be crucial; it follows trivially from the fact that the

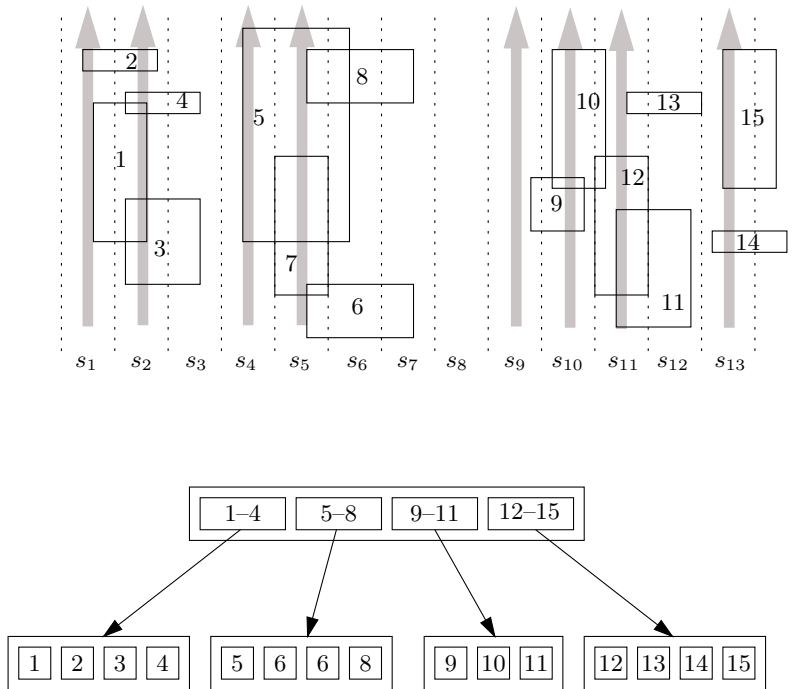


Figure 2: The strip order for a set of rectangles, and an R-tree respecting the strip order. The defining set of a bounding box is written inside the box in each node. For example, the bounding box stored in the root for its rightmost child is the bounding box of rectangles 12–15.

x -extents of the rectangles are between one and two.

Observation 2.1 Any rectangle intersecting a given strip s_i must be assigned to s_i , to s_{i-1} , or to s_{i-2} .

We say that an R-tree *respects the strip-order* when the left-to-right order of the rectangles in the leaves corresponds to the strip order: the leftmost rectangle in the leftmost leaf is the first rectangle in the strip order, and so on—see Figure 2. Once we have an order on the rectangles in

\mathcal{R} it is easy to construct an R-tree respecting that order—but this will be explained later—but first we will modify the order to handle sets with scale factor larger than two.

2.2 The general case

So far we assumed that ρ , the scale factor of the set \mathcal{R} of rectangles, is at most two. The strip order we developed can also be used for larger ρ , but the dependency of the query complexity on ρ will be linear. The following modification of the order works efficiently in the general case.

We partition \mathcal{R} into at most $m := \lceil \log \rho \rceil + 1$ subsets,³ each with scale factor at most two. We call these subsets *classes*. Class \mathcal{R}_i contains all rectangles $r \in \mathcal{R}$ with $2^i \leq x\text{-extent}(r) < 2^{i+1}$, where $x\text{-extent}(r)$ is the length of the projection of r onto the x -axis. We call i the *index* of class \mathcal{R}_i . Note that there can be classes whose index is negative, namely if there are rectangles whose x -extent is smaller than 1. Still, if the overall scale factor is ρ , then there are at most $\lceil \log \rho \rceil + 1$ non-empty classes.

Next we define a new ordering on the rectangles, as follows: rectangles are ordered by the index number of the class \mathcal{R}_i they are in, and rectangles with the same index number are ordered using the strip order, as above. Here the strip ordering for a class \mathcal{R}_i is defined using strips of width 2^i . In other words, the sorted sequence of rectangles is obtained by sorting each class separately according to the strip order and concatenating these sorted sequences in order of the class index. We call the new order the *index-strip order*.

The index-strip order can be defined more formally as follows. Define the following functions for a rectangle r :

$$\begin{aligned} \text{index-nbr}(r) &:= \lfloor \log(x\text{-extent}(r)) \rfloor \\ \text{strip-width}(r) &:= 2^{\text{index-nbr}(r)} \\ \text{strip-nbr}(r) &:= \lfloor (x\text{-coordinate of left edge of } r) / \text{strip-width}(r) \rfloor \\ y\text{-nbr}(r) &:= y\text{-coordinate of the bottom edge of } r \end{aligned}$$

(For a negative number x , we define $\lfloor x \rfloor$ to be the largest integer smaller than or equal to x . For instance, $\lfloor -1/2 \rfloor = -1$.) We now define the following representation for r :

$$\text{rep}(r) := (\text{index-nbr}(r), \text{strip-nbr}(r), y\text{-nbr}(r)).$$

Observation 2.2 *The ordering of the rectangles in \mathcal{R} induced by a lexicographical ordering on the representations $\text{rep}(r)$ is identical to the index-strip order.*

As before, we will say that an R-tree *respects the index-strip order* when the left-to-right ordering of the rectangles in the leaves corresponds to the index-strip ordering on the rectangles.

Theorem 2.1 *For a given set of n objects in the plane, one can construct an R-tree of minimum degree t that respects the index-strip order using $O((n/t) \log(n/t) / \log(N/t))$ disk accesses, where N is the number of items fitting into internal memory.*

Proof. After scanning the input once, we can compute $\text{rep}(r)$ for a given rectangle $r \in \mathcal{R}$ in $O(1)$ time. Hence, we can sort the rectangles of \mathcal{R} according to the index-strip order using $O((n/t) \log(n/t) / \log(N/t))$ disk accesses [1]. Once we have ordered the rectangles, constructing the R-tree is basically identical to constructing a B-tree for an ordered set of items. This can be done with a simple bottom-up procedure using $O(n/t)$ disk accesses [2]. \square

³The logarithms in our paper are all with base 2.

3 Analysis of the query complexity

Next we analyze the query complexity of R-trees respecting the index-strip order. We will consider two types of queries: point-location queries and range queries. We start by introducing some more notation and stating some properties that we will need in both cases.

For a level l in \mathcal{T} , we define $\mathcal{B}(l)$ to be the collection of bounding boxes $b(\nu)$ of nodes ν at level l . (These bounding boxes are actually stored one level higher in the tree.) For a class \mathcal{R}_j , we define $\mathcal{B}_j(l) \subset \mathcal{B}(l)$ to be the subset of bounding boxes whose defining set contains only rectangles from \mathcal{R}_j .

Finally, we say that a bounding box b *straddles* a strip number i if the following two conditions hold:

- (a.) the defining set of b only has rectangles from a single class,
- (b.) the defining set of b has rectangles r_1, r_2 with $\text{strip-nbr}(r_1) < i \leq \text{strip-nbr}(r_2)$.

Lemma 3.1 *Let \mathcal{T} be an R-tree respecting the index-strip order, and let l be a fixed level in \mathcal{T} .*

- (i) *Let $b(\nu_1), b(\nu_2) \in \mathcal{B}(l)$ be two bounding boxes with ν_1 to the left of ν_2 . Then for any two rectangles $r_1 \in \mathcal{R}(\nu_1)$ and $r_2 \in \mathcal{R}(\nu_2)$ we have that r_1 comes before r_2 in the index-strip order.*
- (ii) *There are at most $m - 1$ bounding boxes in $\mathcal{B}(l)$ whose defining set contains rectangles from more than one class, where m is the number of classes.*
- (iii) *For any class \mathcal{R}_j and any strip number i , there is at most one box in $\mathcal{B}_j(l)$ that straddles i .*

Proof. Part (i) follows immediately from the fact that the ordering of the nodes at a given level is induced by the index-strip order of the input rectangles. Parts (ii) and (iii) follow directly from (i) and the definition of the index-strip order. \square

3.1 Point-location queries

We can now prove a bound on the complexity of a point-location query.

Theorem 3.1 *Let \mathcal{S} be a set of n objects in the plane such that the set of bounding boxes of \mathcal{S} has stabbing number σ and scale factor ρ . Let \mathcal{T} be an R-tree for \mathcal{S} of minimum degree t respecting the index-strip order. Then a point-location query in \mathcal{T} visits $O((\sigma(\log \rho + 1) \log n / \log t)$ nodes.*

Proof. Let l be a fixed level in the R-tree \mathcal{T} . Define $m := \lceil \log \rho \rceil + 1$. We will show that the stabbing number of $\mathcal{B}(l)$, the set of bounding boxes of the nodes at level l , is at most $3\sigma m + 8m - 1$. Multiplying this by the number of levels in \mathcal{T} then gives the desired bound.

Let q be a query point. We partition the bounding boxes in $\mathcal{B}(l)$ stabbed by q into three categories, and count the maximum number of boxes that each category can contain separately.

- *category (i): bounding boxes whose defining set has rectangles in more than one class \mathcal{R}_j .*

By Lemma 3.1(ii) there are at most $m - 1$ such bounding boxes.

- *category (ii): bounding boxes whose defining set contains rectangles from a single class that are all assigned to the same strip.*

Fix a class \mathcal{R}_j and consider a bounding box b whose defining set is a subset of \mathcal{R}_j . Let s_i be the strip defined for \mathcal{R}_j that contains the query point q . Let s_{i-1} and s_{i-2} be the two strips immediately to the left of s_i . The bounding boxes we are considering have their defining set all assigned to the same strip. By Observation 2.1 this means we only have to consider strips s_i, s_{i-1} , and s_{i-2} . Consider the bounding boxes whose defining set has only rectangles

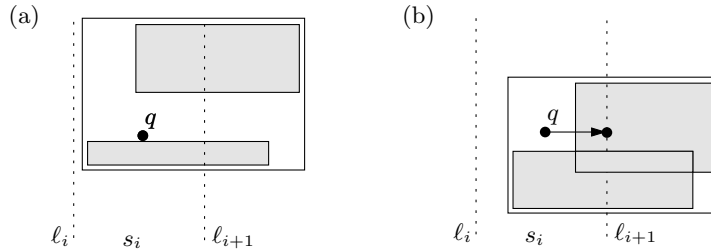


Figure 3: Illustration for the proof of Theorem 3.1.

assigned to s_i . Such bounding boxes may have a defining set containing both a rectangle with bottom edge below q and one with bottom edge above q , as shown in Figure 3(a). Because of the ordering scheme within a strip, there is at most one such bounding box. Otherwise, the defining set of the bounding box has a rectangle $[x : x'] \times [y : y']$ with $q_y \in [y : y']$, as in Figure 3(b). Because the x -extent of each rectangle is at least the width of s_i , this means that such a rectangle must be stabbed by the orthogonal projection of q onto l_{i+1} , the line bounding s_i to the right. There can be no more than σ such rectangles and, consequently, no more than σ such bounding boxes per class.

This shows that there are at most $\sigma + 1$ bounding boxes in $\mathcal{B}_j(l)$ that are stabbed by q and whose defining set has only rectangles assigned to s_i . A similar argument works for bounding boxes stabbed by q whose defining set has only rectangles assigned to s_{i-1} (or to s_{i-2}). The only difference is that we now need to consider the projection of q onto the line bounding s_{i-1} (or s_{i-2}) from the right.

Summing over all classes \mathcal{R}_j we get $3\sigma m + 3m$ boxes in total in category (ii).

- *category (iii): the remaining bounding boxes.*

These bounding boxes have a defining set with rectangles from a single class \mathcal{R}_j that are assigned to more than one strip. In other words, they straddle one (or more) strip number(s). Let s_i be the strip that contains the query point q . From Observation 2.1 it follows that such a box can only intersect s_i if it straddles a strip number i^* with $i - 2 \leq i^* \leq i + 1$. Furthermore, by Lemma 3.1(iii) there is only one such bounding box for each i^* . Multiplying by the number of classes, we get at most $4m$ boxes in category (iii).

Adding up the bounds for each of the categories, we get a total bound of $3\sigma m + 8m - 1$ per level, as claimed. \square

Remark. The only way in which the scale factor ρ plays a role in the proof of Theorem 3.1, is that it ensures that we can partition \mathcal{R} into a logarithmic number of classes with scale factor at most two. In general, our method gives a bound of $O(\sigma m \log n / \log t)$ for sets of rectangles that can be partitioned into m such classes, even when $\log \rho$ is larger than m . For instance, if \mathcal{R} contains three classes of rectangles—the large rectangles, the intermediate ones, and the small ones—each with scale factor at most two, then our method will work well even when the large rectangles are much larger than the small ones. Such a behavior may well occur for practical inputs.

3.2 Unbounded scale factors

When the scale factor gets too large, our method gives rise to many classes \mathcal{R}_i and the resulting query complexity will not be very good. We can overcome this problem with a simple trick: we replace the x -coordinate of the vertical edges of the rectangles by their rank: sort the set of x -coordinates of all the vertical edges in increasing order, and replace every coordinate by its rank

in this order. This way the x -‘coordinates’ that we are dealing with are integers between 1 and $2n$, so the scale factor is bounded by n . We then apply our algorithm to these normalized rectangles. Conversion of the resulting R-tree to an R-tree for the original rectangles is trivial: simply replace the x -‘coordinates’ of the edges of the bounding boxes by the original coordinates. The latter step does not influence the query complexity. In the analysis we can thus replace ρ by n if that gives a better result.

Corollary 3.1 *Let \mathcal{S} be a set of n objects in the plane such that the set of bounding boxes of \mathcal{S} has stabbing number σ and scale factor ρ . For a given t , we can construct an R-tree of minimum degree t for \mathcal{S} such that the number of nodes visited when answering a point-location query is $O(\sigma \log^2 n / \log t)$.*

3.3 Range-searching queries

Now suppose we want to perform a range query with an axis-parallel rectangular range Q . Let w denote the ratio of the x -extent of Q to the smallest x -extent of any object in \mathcal{S} . We call w the *width* of the range. Furthermore, let k denote the number of objects reported by the range query. We first analyze the number of nodes visited by the query procedure in terms of w , k , and the parameters introduced earlier. Then we show that in general—that is, for ranges that can be unbounded in both x - and y -direction—one cannot obtain similar (logarithmic) bounds.

Theorem 3.2 *Let \mathcal{S} be a set of n objects in the plane such that the set of bounding boxes of \mathcal{S} has stabbing number σ and scale factor ρ . Let \mathcal{T} be an R-tree for \mathcal{S} of minimum degree t respecting the index-strip order. Then a range query in \mathcal{T} with an axis-parallel rectangular range Q visits $O((\sigma(\log \rho + 1) + w + k) \log n / \log t)$ nodes, where $w = x\text{-extent}(Q) / \min_{o \in \mathcal{S}} x\text{-extent}(o)$.*

Using normalization, we can also obtain a bound of $O((\sigma \log n + w + k) \log n / \log t)$ on the number of visited nodes.

Proof. Define $m := \lceil \log \rho \rceil + 1$. Let l be a fixed level in the R-tree \mathcal{T} , and let $\mathcal{B}(l)$ be the set of bounding boxes of the nodes at level l . We will show that the number of bounding boxes in $\mathcal{B}(l)$ intersecting the query range Q is $O(\sigma m + w + k)$.

There are three categories of bounding boxes in $\mathcal{B}(l)$ intersecting Q . Below we count the maximum number of bounding boxes each category can contain.

- *category (i): bounding boxes whose defining set has rectangles in more than one class \mathcal{R}_j .*

By Lemma 3.1(ii) there are at most $m - 1$ such bounding boxes.

- *category (ii): bounding boxes not in category (i) containing a corner of Q .*

From the proof of Theorem 3.1—in particular categories (i) and (ii) in that proof—it follows that there are at most $3\sigma m + 7m$ such bounding boxes per corner.

- *category (iii): the remaining boxes*

Fix a class \mathcal{R}_j . Let $s_i, \dots, s_{i'}$ be the strips defined for \mathcal{R}_j intersected by Q , as illustrated in Figure 4. Define w_j to be the ratio of the x -extent of Q and the x -extent of the strips defined for \mathcal{R}_j . Note that $i' - i \leq w_j + 1$. Define j_0 to be the smallest index of any (non-empty) class. In other words, \mathcal{R}_{j_0} contains the input object of smallest x -extent. We have

$$w_j = \frac{x\text{-extent}(Q)}{2^j} = \frac{x\text{-extent}(Q)}{2^{j_0}} \cdot \frac{2^{j_0}}{2^j} \leq \frac{2w}{2^{j-j_0}},$$

which implies that $\sum_{j \geq j_0} w_j = O(w)$. We distinguish three subcategories for the boxes in category (iii).

- The first subcategory contains the boxes straddling a strip number i^* . From Observation 2.1 it follows that we can restrict our attention to strip numbers i^* with $i - 2 \leq i^* \leq i' + 1$. By Lemma 3.1(iii) each strip number is straddled at most once, so there are at most $(i' + 1) - (i - 2) + 1 \leq w_j + 5$ such bounding boxes for \mathcal{R}_j .

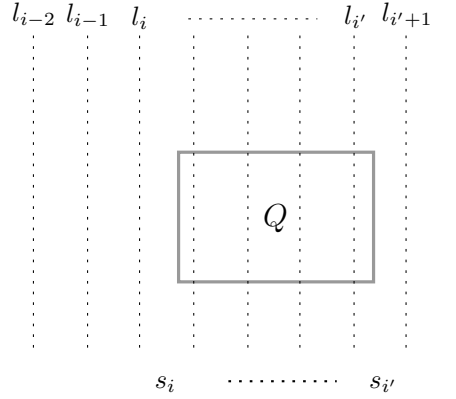


Figure 4: The strips $s_i, \dots, s_{i'}$ in \mathcal{R}_j intersected by Q .

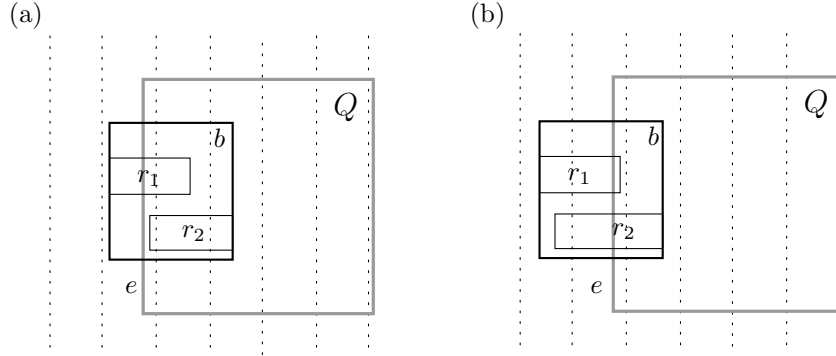


Figure 5: Illustration for the proof of Theorem 3.2.

- The second subcategory contains the boxes not in the first subcategory—hence, their defining sets have only rectangles assigned to a single strip—and not intersecting the top or bottom edge of Q . Such a box b is either fully contained in Q or it intersects the left or right edge of Q . We will argue that in both cases there is an object in the defining set of b that intersects Q . Thus we can charge the box b to this answer.

If b is fully contained in Q this is trivial: any object in b 's defining set intersects Q .

If b intersects the left or right edge of Q we can argue as follows. Consider for example a bounding box b intersecting the left edge, e , of Q . Its defining set must have a rectangle r_1 whose left edge is contained in the left edge of b , and a rectangle r_2 whose right edge is contained in the right edge of b . There are two cases. One is when e lies in the same strip that r_1 and r_2 are assigned to. In this case r_1 must intersect e and we are done—see Figure 5(a). In the other case e must lie to the right of the strip that r_1 is assigned to, which implies that r_2 intersects e —see Figure 5(b)—and we are done again. In the case when b only intersects the right edge e' of Q we have that r_1 must intersect the right edge of Q .

It follows that the total number of bounding boxes in this subcategory is $O(k)$.

- The last subcategory contains the remaining boxes. These must intersect the top or bottom edge of Q . There are two cases.

One is where the defining set of such a bounding box b has one rectangle whose top edge is below the bottom edge of Q and one rectangle whose bottom edge is above the top edge of Q . For each \mathcal{R}_j , this can happen only once for each of the strips $s_{i-2}, \dots, s_{i'}$.

Hence, there are at most $w_j + 4$ such bounding boxes for \mathcal{R}_j .

In the other case the defining set of b must contain a rectangle whose top or bottom edge is contained fully in Q . This means that the object contained in this rectangle intersects Q .

The total number of bounding boxes in this subcategory is therefore at most $w_j + 4 + k$.

Summing over all subcategories, and over all classes \mathcal{R}_j we can bound the number of boxes in category (iii) by $\sum_{j \geq j_0} \{(w_j + 4) + (w_j + 3)\} + O(k) = O(w + m + k)$.

Adding up the bounds for each of the cases, we get a total bound of $O(\sigma m + w + k)$ for the number of nodes visited on a fixed level l . Over all levels we thus get a bound of $O((\sigma m + w + k) \log n / \log t)$.

If ρ is large, normalization can ensure $m = \log n$ —see the previous subsection. \square

Can we improve on this result? In particular, one would hope that it is possible to get rid of the dependence on w . Unfortunately, the next theorem shows that in this case one cannot get bounds close to the ones we just obtained. (This theorem is well known in the spatial-databases community, but we add a proof for completeness.)

Theorem 3.3 *For any n , there is a set \mathcal{S} of n disjoint unit squares such that for any R-tree with minimum degree t on \mathcal{S} , there is a rectangular query range for which the query procedure will visit $\Omega(\sqrt{n}/\sqrt{t})$ nodes even though the range does not intersect any of the squares.*

Proof. Assume for simplicity that n is a perfect square. Consider a configuration of $\sqrt{n} \times \sqrt{n}$ disjoint squares arranged in a regular grid. The shaded squares in Figure 6 show the construction for $n = 16$. Let \mathcal{T} be an R-tree for this collection of squares. Consider the collection of $\Theta(\sqrt{n})$

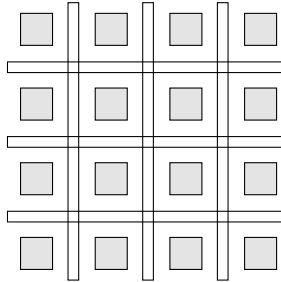


Figure 6: The lower-bound configuration for range searching.

long thin ranges separating either two consecutive columns or two consecutive rows of the set of squares. There are $\Theta(n/t)$ leaves in \mathcal{T} . It is easy to see that for any leaf, the bounding box of the squares stored at that leaf is intersected by $\Omega(\sqrt{t})$ ranges. Hence, the total number of range-box intersections is $\Omega(n/\sqrt{t})$. By the averaging principle, there must be a range intersecting $\Omega(\sqrt{n}/\sqrt{t})$ bounding boxes. The number of leaves visited by the query procedure for this range is $\Omega(\sqrt{n}/\sqrt{t})$. \square

4 Dynamization

Observation 2.2 implies that if we augment the R-tree with some additional information, we can use, with a small modification, standard leaf-oriented B-tree algorithms for insertions and deletions, as described in Chapter III.5.2 of Mehlhorns book [13]. The extra information is needed to be able to walk down the tree in order to locate the position of a new rectangle in the leaf-level of the R-tree: when we insert a rectangle into the subtree rooted at a node ν , we need to know which subtree rooted at a child of ν should contain r . Recall that at ν we store a bounding box

$b(\mu)$ for each child μ of ν . This is not enough to locate a new rectangle r . What we need is information about the index-strip order. More precisely, the extra information we need to store with $b(\mu)$ is the representation $rep_\mu := rep(r_{\mu(min)})$, where $r_{\mu(min)}$ is the minimum (according to the index-strip order) rectangle stored in the subtree of μ . This allows us to search with (the representation of) a new rectangle r , so that we can determine into which child of ν we should insert r : this is the rightmost child μ such that $r_{\mu(min)} \leq rep(r)$ or, if no such child exists, the leftmost child. Now we can use B-tree-like update algorithms, which require $O(\frac{\log n}{\log t})$ node accesses per update.

We obtain the following theorem.

Theorem 4.1 *An R-tree respecting the strip-order for a set \mathcal{S} of n objects in the plane can be updated using $O(\frac{\log n}{\log t})$ node visits.*

Remark. The extra information will force us to choose the minimum degree smaller, roughly by a factor of two, otherwise the information for a node would no longer fit into one page of external memory. This implies that the depth will increase by a factor of roughly $(1 + \log t)/\log t$.

It should also be noted that the dynamization described above cannot be used together with the normalization trick described in Section 3.2. The normalization scales the input such that the x -coordinates of the corners of the rectangles are between 1 and n . Inserting new elements into the R-tree affects the normalization. Some rectangles might change size, forcing updates in the R-tree. As a result, good worst-case update times cannot be expected.

5 Higher-dimensional R-trees

The approach for the planar case extends easily into higher dimensions. For instance, suppose we have a set \mathcal{S} of n objects in 3-dimensional space with axes x_1 , x_2 , and x_3 . As before, we let σ denote the stabbing number of the set \mathcal{R} of bounding boxes of the objects in \mathcal{S} , that is, the maximum number of boxes containing any query point. We let ρ_1 and ρ_2 denote the x_1 -scale factor and the x_2 -scale factor of \mathcal{R} , respectively.

First assume that $\rho_1 \leq 2$ and $\rho_2 \leq 2$. We partition space into three-dimensional columns by planes orthogonal to the x_1 -axis and planes orthogonal to the x_2 -axis. The spacing of the planes equals the minimum x_1 -extent and x_2 -extent, respectively, of the objects. We number the columns in increasing order primarily with respect to their x_1 -coordinates and secondarily on their x_2 -coordinates. We assign each box in \mathcal{R} to the column containing its front left edge (that is, the vertical edge with smallest x_1 - and x_2 -coordinate)—see Figure 7. We then number the boxes in

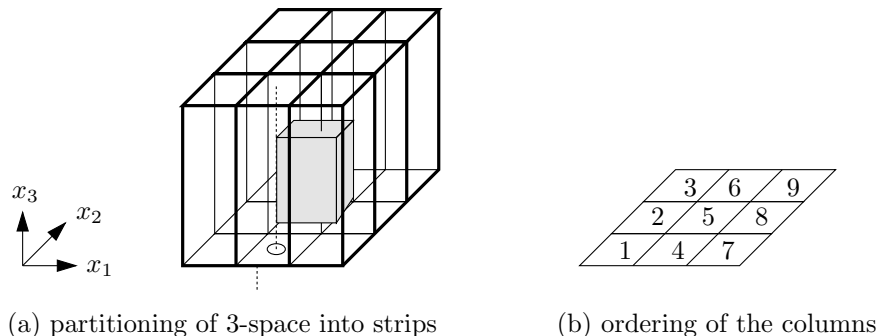


Figure 7: The strip-order in 3-dimensional space. The shaded box is assigned to column 4.

\mathcal{R} according to the ordering of the columns, where within each column we order the boxes based on the x_3 -coordinate of their bottom facet. The latter ordering is done in increasing order. The definition of the order for sets where the scale factors are more than two is similar: partition \mathcal{R} into $(\lceil \log \rho_1 \rceil + 1) \cdot (\lceil \log \rho_2 \rceil + 1)$ classes with scale factors at most two, compute an order for

each class and concatenate the orders. A similar approach works in dimensions higher than three. To describe this more precisely, we need to generalize the definition of the representation, given in the previous section. After that we will analyze the d -dimensional structures by showing the generalized results of Observation 2.1, Lemma 3.1 and Theorem 3.1.

Suppose we have a set \mathcal{S} of n input objects in \mathbb{R}^d . Let $\mathcal{R} = \mathcal{R}_{\mathcal{S}}$ be the set of bounding boxes of the objects in \mathcal{S} . For a box r we use $x_f\text{-extent}(r)$ to denote the length of the orthogonal projection of r onto the x_f -axis. We define the following functions for a d -dimensional box r , where $1 \leq f < d$:

$$\begin{aligned} \text{index-}n\text{br}_f(r) &:= \lfloor \log(x_f\text{-extent}(r)) \rfloor \\ \text{strip-width}_f(r) &:= 2^{\text{index-}n\text{br}_f(r)} \\ \text{strip-}n\text{br}_f(r) &:= \lfloor (\text{the smallest } x_f\text{-coordinate of } r) / \text{strip-width}_f(r) \rfloor \\ x_d\text{-}n\text{br}(r) &:= \text{the smallest } x_d\text{-coordinate of } r \end{aligned}$$

We now define the following representation for r as

$$\text{rep}(r) := (\text{index-}n\text{br}_1(r), \dots, \text{index-}n\text{br}_{d-1}(r), \text{strip-}n\text{br}_1(r), \dots, \text{strip-}n\text{br}_{d-1}(r), x_d\text{-}n\text{br}(r)),$$

and we define the index-strip order as the lexicographical order on the representations. Finally, we define $\mathcal{R}(j_1, \dots, j_d)$ to be the class of boxes r with $\text{index-}n\text{br}_f(r) = j_f$ for all $1 \leq f < d$. Thus we have at most $m := \prod_{1 \leq f < d} (\lceil \log \rho_f \rceil + 1)$ classes, where ρ_f is the x_f -scale factor of the objects in \mathcal{S} , and in each class the x_f -scale factor of the boxes is at most two for any $1 \leq f < d$. Note that a class defines a collection of ‘columns’ parallel to the x_d -axis. Each box in $\mathcal{R}(j_1, \dots, j_d)$ is assigned to a column according to its strip numbers. In other words, if of all the edges of a box b that are parallel to the x_d -axis, e is the one with minimum x_f -coordinate for all $f < d$, then b is assigned to the column containing e .

Consider an R-tree \mathcal{T} respecting the lexicographic ordering of the boxes defined by $\text{rep}(r)$. The analysis of the number of nodes visited when answering a point-location query, which we present next, is very similar to the planar case. We start by generalizing Observation 2.1. Fix a class $\mathcal{R}(j_1, \dots, j_d)$. For a point q , we define its f -th strip number with respect to the class $\mathcal{R}(j_1, \dots, j_d)$ as

$$\text{strip-}n\text{br}_f(q) := \lfloor (x_f\text{-coordinate of } q) / \text{strip-width}_f \rfloor,$$

where $\text{strip-width}_f = 2^{j_f}$ is the width in x_f -direction of the columns defined for $\mathcal{R}(j_1, \dots, j_d)$.

Observation 5.1 *For every input box $r \in \mathcal{R}(j_1, \dots, j_d)$ that is stabbed by a point q , we have: $\text{strip-}n\text{br}_f(q) - 2 \leq \text{strip-}n\text{br}_f(r) \leq \text{strip-}n\text{br}_f(q)$, for every $f < d$.*

The observation follows from the fact that the x_f -extent of any box $r \in \mathcal{R}(j_1, \dots, j_d)$ is at most twice the width (in x_f -direction) of the columns.

We continue with the generalization of Lemma 3.1. Part (i) and (ii) hold verbatim; we only need to reformulate part (iii). Let f be an integer with $1 \leq f < d$. A bounding box b is said to f -straddle a strip number i if the following three conditions hold:

- (a.) the defining set of b only has rectangles from a single class,
- (b.) for any two rectangles r_1, r_2 in the defining set of b and any f' with $1 \leq f' < f$, we have $\text{strip-}n\text{br}_{f'}(r_1) = \text{strip-}n\text{br}_{f'}(r_2)$,
- (c.) the defining set of b has rectangles r_1, r_2 with $\text{strip-}n\text{br}_f(r_1) < i \leq \text{strip-}n\text{br}_f(r_2)$.

Let $\mathcal{B}(j_1, \dots, j_d, l)$ be the collection of bounding boxes of all nodes at a given level l in \mathcal{T} with the property that the defining set of the bounding box has only boxes from $\mathcal{R}(j_1, \dots, j_d)$.

Lemma 5.1 *Let \mathcal{T} be an R-tree respecting the index-strip order, and let l be a fixed level in \mathcal{T} . Then for any class $\mathcal{R}(j_1, \dots, j_d)$ and any f and i , there is at most one box in $\mathcal{B}_j(j_1, \dots, j_d, l)$ that f -straddles i .*

We now come to the main result.

Theorem 5.1 *Let \mathcal{S} be a set of n objects in d -space such that the set of bounding boxes of \mathcal{S} has stabbing number σ . Let ρ_f denote the x_f -scale factor of \mathcal{R} . Let \mathcal{T} be an R-tree for \mathcal{S} of minimum degree t respecting the index-strip order. Then a point-location query in \mathcal{T} visits $O((\sigma 3^{d-1} + 4^{d-1})m \log n / \log t)$ nodes, where $m = \prod_{1 \leq f < d} (\log \rho_{x_f} + 1)$.*

Using normalization, we can also obtain a bound where $m = \prod_{1 \leq f < d} \min(\log \rho_{x_f} + 1, \log n)$.

Proof. Let l be a fixed level in the R-tree \mathcal{T} . Define $m := \prod_{1 \leq f < d} (\lceil \log \rho_{x_f} \rceil + 1)$. We will show that the stabbing number of $\mathcal{B}(l)$, the set of bounding boxes of the nodes at level l , is at most $(3^{d-1}(\sigma + 1) + 4^{d-1} + 1)m - 1$.

Let q be a query point. We consider three categories of bounding boxes in $\mathcal{B}(l)$ stabbed by q .

- *category (i): bounding boxes whose defining subset has boxes in more than one of the classes $\mathcal{R}(j_1, \dots, j_d)$.*

By Lemma 3.1(ii), which still holds, there are at most $m - 1$ such bounding boxes.

- *category (ii): bounding boxes whose defining set contains boxes from a single class that are all assigned to the same column.*

Fix a class $\mathcal{R}(j_1, \dots, j_d)$. According to Observation 5.1 we only have to consider 3^{d-1} of the columns defined for $\mathcal{R}(j_1, \dots, j_d)$. For simplicity we only consider the column containing the query point q . The analysis for the other columns is similar—see also the proof for the 2-dimensional case. So now consider a bounding box b whose defining set only contains boxes assigned to the column containing q .

The first case is when b 's defining set has both an input box with minimum x_d -coordinate smaller than the x_d -coordinate of q and one with minimum x_d -coordinate greater than the x_d -coordinate of q —see Figure 3(a) for an illustration in two dimensions. Because of the bottom-to-top ordering within a column, there is at most one such bounding box.

In the second case, b 's defining set has a box with x_d -coordinates in the range $[a : a']$ with $q_{x_d} \in [a : a']$, as in Figure 3(b). We know that the x_f -extent of each box is at least the width of the column in x_f -direction for each $1 \leq f < d$. Hence, such a box must be stabbed by the orthogonal projection of q onto an edge of the column, namely the edge with maximum x_f -coordinate for each $1 \leq f < d$. There can be no more than σ such boxes and, consequently, no more than σ such bounding boxes in total.

Multiplying by the number of columns we have to consider, we see that there are at most $3^{d-1}(\sigma + 1)$ bounding boxes in category (ii) for a fixed class $\mathcal{R}(j_1, \dots, j_d)$. This leads to $3^{d-1}(\sigma + 1)m$ boxes in total in this category.

- *category (iii): the remaining bounding boxes.*

These bounding boxes have a defining set with boxes from a single class $\mathcal{R}(j_1, \dots, j_d)$ that are assigned to more than one column. In other words, they f -straddle some strip number i for some $1 \leq f < d$. By Observation 5.1 and Lemma 5.1 we have to consider at most 4^{d-1} strip numbers, each of which is straddled at most once. Multiplying by the number of classes we get at most $4^{d-1}m$ boxes in total.

Adding up the bounds for each of the cases, we get a total bound of $(3^{d-1}(\sigma + 1) + 4^{d-1} + 1)m - 1$. Multiplying by the number of levels gives the desired bound.

When one or more of the ρ_f 's are very large, we can again use normalization (in the relevant dimensions) to improve the bounds. \square

We cannot obtain bounds for range searching that are similar to the planar case. The reason is that even when $\sigma = 1$ it can happen that a range with small width intersects many of the boxes in $\mathcal{R}_{\mathcal{S}}$ without intersecting any of the corresponding objects in \mathcal{S} . Consider for example a range

Q whose x_i -extent is 1 for all $i < d$, but that is very long in x_d -direction. Such a range has small width, since the width criterion does not take the x_d -extent into account. (One might be able to prove good bounds by also restricting the x_d -extent of the range.) Then we can put many disjoint unit cubes above each other, each of which just intersects the front-left vertical of Q . This can be done in such a way that we can add an object not intersecting Q inside each cube whose bounding box is exactly that cube.

6 Concluding remarks

We have given an algorithm to construct R-trees for sets of n objects in the plane and in higher dimensional spaces. We analyzed the number of nodes visited when answering a point-location query in terms of n , and σ (the stabbing number of the initial bounding boxes), and ρ (the scale factor). When σ and ρ are constant, our results are optimal.

Our results might be improved in several ways. First of all, it would be interesting to reduce the dependency on σ in our bounds. Ideally, we would like to replace σ by $\lceil \sigma/t \rceil$. Another question is whether it is possible to improve the $O(\log^2 n / \log t)$ bound that we get for constant σ to $O(\log n / \log t)$.

It would also be nice to find another way to deal with scale factors larger than two. Our method of partitioning the set into classes with scale factor two or less works fine in theory, but it is questionable whether it works well in practice.

Acknowledgement

We thank Frank van der Stappen for stimulating discussions during the early stages of this research. We also thank an anonymous referee for providing many helpful comments.

References

- [1] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Commun. ACM*, 31:1116–1127, 1988.
- [2] L. Arge. External memory data structures. In J. Abello and M. Pardalos and M. G. C. Resende, editors, *Handbook of Massive Data Sets*. Kluwer Academic Publishers, 2001.
- [3] G. Barequet, B. Chazelle, L. Guibas, J. Mitchell, and A. Tal. BOXTREE: A hierarchical representation for surfaces in 3D. *Comput. Graph. Forum*, 15(3):C387–C396, C484, September 1996. Proc. Eurographics’96.
- [4] B. Becker, P. Franciosa, S. Gschwind, T. Ohler, G. Thiemt, and P. Widmayer. Enclosing many boxes by an optimal pair of boxes. In *Proc. 9th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, LNCS 577, pages 475–486, 1992.
- [5] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In *Proc. ACM-SIGMOD International Conference on Management of Data*, pages 322–331, 1990.
- [6] G. van den Bergen. *Collision Detection in Interactive 3D Computer Animation*. Ph.D. thesis, Technical University Eindhoven, 1999.
- [7] C. Faloutsos and I. Kamel. *Packed R-trees using fractals*. Report CS-TR-3009, University of Maryland, College Park, 1992.
- [8] C. Faloutsos, T. Sellis, and N. Roussopoulos. Analysis of object oriented spatial access methods. In *Proc. ACM-SIGMOD International Conference on Management of Data*, pages 426–439, 1987.

- [9] V. Gaede and O. Günther. Multidimensional access methods. *ACM Comput. Surv.*, 30:170–231, 1998.
- [10] D.M. Gavrilă. R-tree index optimization. In *Proc. 6th International Symposium on Spatial Data Handling*, pages 771–791, 1994.
- [11] D. Greene. An implementation and performance analysis of spatial data access methods. In *Proc. 5th International Conference on Data Engineering*, pages 606–615, 1989.
- [12] A. Guttman. R-trees: a dynamic indexing structure for spatial searching. In *Proc. ACM-SIGMOD International Conference on Management of Data*, pages 47–57, 1984.
- [13] K. Mehlhorn. Data Structures and Algorithms 1: Sorting and Searching. *EATCS Monographs on Theoretical Computer Science*, vol. 1, Springer-Verlag, Heidelberg, Germany, 1984.
- [14] J. Nievergelt and P. Widmayer. Spatial data structures: concepts and design choices. In M. van Kreveld, J. Nievergelt, T. Roos, and P. Widmayer (eds.), *Algorithmic Foundations of Geographic Information Systems*, LNCS 1340, pages 153–198, 1997.
- [15] N. Roussopoulos and D. Leifer. Direct spatial search on pictorial databases with packed R-trees. In *Proc. ACM-SIGMOD International Conference on Management of Data*, 1985.
- [16] H.-W. Six and P. Widmayer. Spatial access structures for geometric databases. In B. Monien and T. Ottmann (eds.), *Data Structures and Efficient Algorithms*, LNCS 594, pages 214–232, 1992.
- [17] Y. Zhou and S. Suri. Analysis of a bounding box heuristic for object intersection. In *Proc. 10th Annual Symposium on Discrete Algorithms (SODA)*, 1999. To appear in *Journal of the ACM*.