ANALYSIS OF OBJECT ORIENTED SPATIAL ACCESS METHODS

Christos Faloutsos[1]
Timos Sellis[2]
Nick Roussopoulos[1,2]

Department of Computer Science
University of Maryland
College Park

Abstract

This paper provides an analysis of R–trees and a variation (R^+–trees) that avoids overlapping rectangles in intermediate nodes of the tree The main contributions of the paper are the following We provide the first known analysis of R–trees Although formulas are given for objects in one dimension (line segments), they can be generalized for objects in higher dimensions as well We show how the the transformation of objects to higher dimensions [HINR83] can be effectively used as a tool for the analysis of R– and R^+– trees Finally, we derive formulas for R^+–trees and compare the two methods analytically The results we obtained show that R^+–trees require less than half the disk accesses required by a corresponding R–tree when searching files of real life sizes R^+–trees are clearly superior in cases where there are few long segments and a lot of small ones

1. Introduction

It has been recognized in the past that existing Database Management Systems (DBMSs) do not handle multi–dimensional data efficiently Multi–dimensional data arise in applications such as cartography, Computer–Aided Design (CAD), computer vision, robotics and computer graphics Since database management systems can be used to store one–dimensional data, like integer or real numbers and strings, considerable interest has been developed in using DBMSs to store multi– dimensional data as well In that sense the DBMS can be the single means for storing and accessing any kind of information required by applications more complex than traditional business applications However, the underlying structures, data models and query languages are not sufficient for the manipulation of more complex data The problem of extending current data models and languages has been considered by various people in the past [CHAN81,STON83,GUTT84b,ROUS85] In this paper we focus on the problem of deriving efficient data access methods for multi–dimensional objects

The main operations that have been addressed by other researchers in the past can be summarized as follows

Point Queries Given a certain point in the space, find all objects that contain it

Region Queries Given a region (user window), find all objects that intersect it

Of course these operations can be augmented with additional constraints on simple one–dimensional (scalar) data In addition, operations like insertions, deletions and modifications of objects should be supported in a dynamic environment

---

[1] Also with University of Maryland Institute for Advanced Computer Studies (UMIACS)

[2] Also with University of Maryland Systems Research Center

This research was sponsored partially by the National Science Foundation under the grant CDR–85–00108 and by UMIACS

Samet in [SAME84] provides a good survey for various access methods that can be used for the efficient storage and retrieval of complex data We review these structures here without going into details, the interested reader is referred to [SAME84] Most of the structures that have been proposed are hierarchical ones with the exception of the Grid File [NIEV84] and EXCELL [TAMM82] which are bucket methods based on extendible hashing Methods based on hierarchical decomposition are very attractive because they are recursive in nature and use the *divide and conquer* paradigm to organize multi-dimensional data That is, going down a hierarchical structure one focuses on more and more interesting subsets of the data which in turn results to efficient execution times [SAME84] In the single dimension case we have seen such a successful structure in the form of B-Trees [BAYE72] The most common case of multi-dimensional data that has been studied in the past is points k-d Trees [BENT75], Quad-Trees [FINK74,SAME84] and k-D-B-Trees [ROBI81] are hierarchical structures for storing point data Some of these structures are very inefficient for secondary storage though The k-D-B-Tree is a direct extension of B-Trees in k-dimensional spaces and is taking paged secondary memory under consideration

Considering region data (polygons), some variations of quad trees have been suggested in the past [KEDE81,SAME83] However, none of them can be easily extended to work in secondary storage based systems Another interesting approach has been suggested by Orenstein [OREN86] whereby k-dimensional objects are transformed to line segments, using the "z-transform" This transformation induces the very same ordering that a quad-tree uses to scan pixels in a 2-dimensional space These line segments are then stored in relations and are processed using the so-called "spatial join" This way minimal changes are required in database systems to support spatial operations

A very promising method is Guttman's R-Trees [GUTT84a] R-trees are a natural extension of B-trees for multi-dimensional objects that are either points or regions However, if R-Trees are built using the dynamic insertion algorithms, the structures may contain excessive space overlap and "dead-space" in the nodes that result in bad performance A packing technique proposed in [ROUS85] alleviates this problem for relatively static databases For update intensive spatial databases with highly overlapping data objects R^+-Trees [STON86] partition the space into subspaces that do not necessarily cover the spatial object This partitioning of the space avoids the overlap problem and the benefit offsets by far the small increase of the height of the tree Gunther [GUNT86] also suggests a similar scheme for general polygon data

The purpose of this paper is to analyze R-Trees and R^+-trees In the next section we describe the original R-tree scheme and the extended R^+-trees Section 3 then presents a formal analysis of the searching performance of both structures followed by section 4 where the analytical results observed for various database sizes and distributions are shown Finally, we conclude in Section 5 by summarizing our contributions and giving hints for future research in the area of multi-dimensional data indexing structures

2. R-Trees and R^+-Trees

2.1. R-trees

As mentioned above R-trees [GUTT84a] where proposed as a natural extension of B-trees [BAYE72] in higher than one dimensions They combine the good features of both the B-trees and quadtrees Like the B-trees, they remain balanced, while they maintain the flexibility of dynamically adjustable windows that deal with "dead-space" on the pictures, like the quadtrees do A second important feature of R-trees is the fact that, at the leaf level, they store full and non-atomic spatial objects This feature provides a natural and high level **object oriented search** Furthermore, because the storage organization of R-trees is similar to that of B-trees, they are good in dealing with paging [GUTT84a]

R-trees have been proposed as an advanced indexing technique for **direct spatial search** that can deal with non-atomic spatial objects They can, however, be used as a representation medium like quadtrees The full potential of R-trees has not been yet conceptualized

The decomposition used in R-trees is dynamic, driven by the spatial data objects Therefore, if a region of an n-dimensional space includes dead-space, no entry in the R-tree is made Leaf nodes of the R-tree contain entries of the form

(I, object-identifier)

where *object-identifier* is a pointer to a data object and I is an n-dimensional minimal rectangle (called

MBR) which bounds its constituent data objects The possibly non-atomic spatial objects stored at the leaf level are considered atomic, as far as the search is concerned, and, in the same R-tree, they are not further decomposed into their pictorial primitives, i e into quadrants, line segments, or pixels

Non-leaf R-tree nodes contain entries of the form

(I, child-pointer)

where *child-pointer* is a pointer to a successor node in the next level of the R-tree, and I is a minimal rectangle which bounds all the entries in the descendent node The term *branching factor* (also called *fan-out*) can be used to specify the maximum number of entries that a node can have, each node of an R-tree with branching factor four, for example, points to a maximum of four descendents (among non-leaf nodes) or four objects (among the leaves) To illustrate the way an R-tree is defined on some space, Figure 2 1 shows a collection of rectangles and Figure 2 2 the corresponding tree built for a branching factor of 4

In considering the performance of R-tree searching, the concepts of **coverage** and **overlap** [ROUS85] are important Coverage is defined as

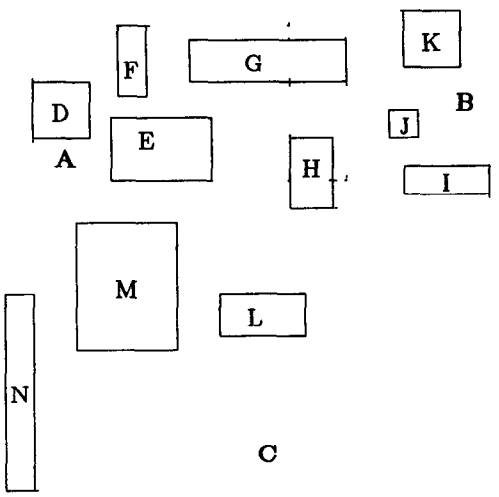


Figure 2.1 Some Rectangles organized in an R-tree

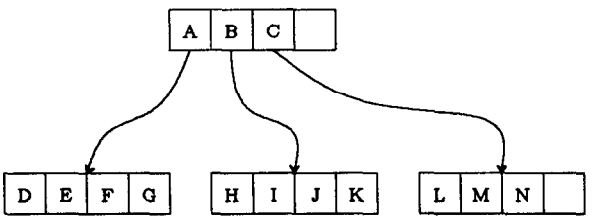


Figure 2.2 The Corresponding R-tree

the total area of all the MBR's of all leaf R-tree nodes, and overlap is defined as the total area contained within two or more leaf MBR's Obviously, efficient R-tree searching demands that both overlap and coverage be minimized, although overlap seems to be the more critical of the two issues For a search window falling in the area of N overlapping leaves, in the worst case, N paths from the root to each of the overlapping leaves have to be followed slowing down the search from h to hN, where h is the height of the tree Clearly, since it is very hard to control the overlap during the dynamic splits of R-trees, efficient search degrades and it may even degenerate the search from logarithmic to linear Minimal coverage reduces the amount of dead space covered by the leaves

It has been shown, that zero overlap and coverage is only achievable for data points that are known in advance and, that using a packing technique for R-trees, search is dramatically improved [ROUS85] In the same paper it is shown that zero overlap is not attainable for region data objects

2.2. R$^+$-Trees

In this paper we use a variation of R-trees called R$^+$-trees [STON86] which avoids overlap at the expense of space which indirectly increases the height of the tree The main difference here is that rectangles can be split into smaller sub-rectangles in order to avoid overlap among minimal bounding rectangles That is, in some cases, in which a given rectangle covering a spatial object at the leaf level overlaps with another rectangle, we decompose it into a collection of non-overlapping sub-rectangles whose union makes up the original rectangle All the pointers of the sub-rectangles point to the same object These sub-rectangles can be

judiciously chosen so that no bounding rectangle at any level need be enlarged The same sub-splitting is propagated up to the non-leaf nodes, thus overlap is forced to stay zero Figures 2 3 and 2 4 show how the boxes of Figure 2 1 will be organized in an R^+-tree, with a branching factor of 4

R^+-trees can be thought as an extension of K-D-B-trees to cover non-zero area objects (i e not only points but rectangles as well) Hence, for any object within a given search window, a single path has to be investigated despite the fact that there may be several paths leading to the same

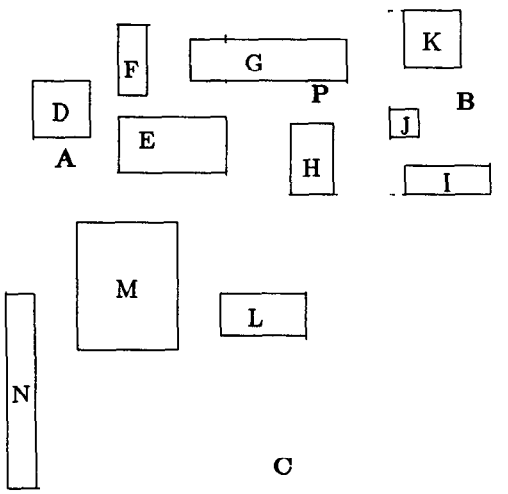


Figure 2.3 Some Rectangles, organized in an R^+ tree

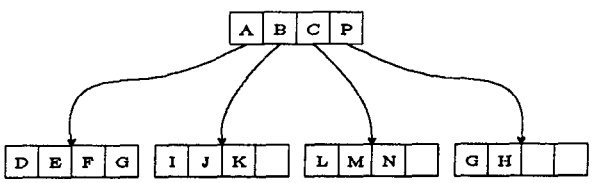


Figure 2.4 The Corresponding R^+-tree

object The additional cost incurs in terms of the increased height of the tree Intuitively, for non-pathological split cases, the increased number of sub-rectangles for two overlapping rectangles at a node may cause a split of it and this adds one more level to the height of the tree However, for each such increase to the height, the corresponding search on a regular R-tree for any object in the overlapping area may require h additional page accesses, where h is the height of the tree

Given the above brief description of the two data structures, we proceed in the next section to analyze their search performance

3. Analysis

In this section we analyze the performance of both R^+- and R- trees In the first part we introduce the methodology used through out this section and then continue in the other two subsections with the analysis of the two data structures The analysis examines the performance of both structures for point queries In section 5 we comment on the performance of region queries

3.1. Methodology – Assumptions

The proofs of the forthcoming analyses can be derived more easily if we consider boxes as points in a 4-dimensional space [HINR83] For a box aligned with the axes, four coordinates are enough to uniquely determine it (the x and y coordinates of the lower-left and upper-right corners)

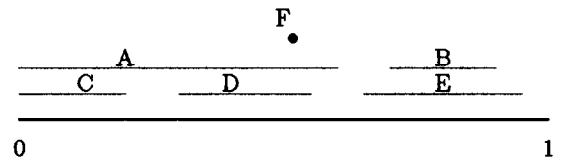
Since 4-d spaces are impossible to be illustrated, in this paper we examine line segments (1-d objects) instead of boxes (2-d objects), and we transform the segments into points in a 2-d space Each segment is uniquely determined by (x_{start}, x_{end}), the coordinates of its start and end points Obtaining formulas and results for line segments is a first step to the analysis of 2-d boxes, or even rectangles of higher dimensionality

In the case of line segments, the "screen" collapses to a line segment, which, by convention, starts at 0 and ends at 1 Figure 3 1a-b shows some line segments and their 2-d representation

Some important observations, with respect to the transformed space

(1) There are no points below the diagonal, since $x_{start} \leq x_{end}$

(2) Line segments of equal size, like B and C, are represented by points that lie on a line



(a)

Figure 3.1a Some line segments

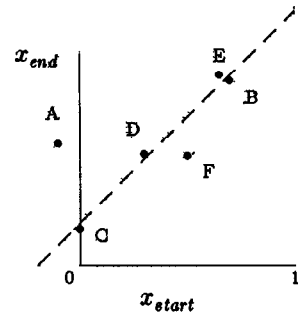


Figure 3.1b The 2–d representation of the line segments of Figure 3 1a

parallel to the diagonal (dashed line in Figure 3 1b) Zero–size segments (i e points), like F, are represented by points on the diagonal

(3) Line segments not entirely within the screen, such as A are allowed In the analysis we consider their remainder after clipping (solid part of A in Figure 3 1a)

(4) Points outside the shaded area in Figure 3 1b are of no interest, because the corresponding segments do not intersect with the screen

(5) The segments covering a given point x_0 of the screen are transformed to points in a shaded area as shown in Figure 3 2

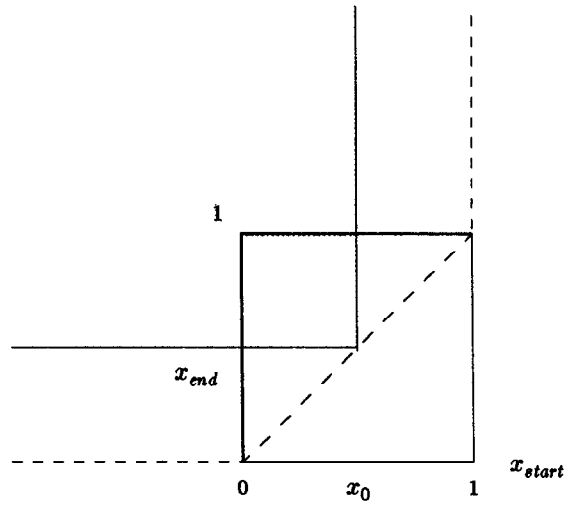


Figure 3.2 The shaded area corresponds to segments containing the point x_0

(6) The shaded area in Figure 3 3 corresponds to all the segments intersecting with the segment (x_1, x_2)

(7) The shaded area in Figure 3 4 corresponds to all the segments covered completely by the segment $S(x_1, x_2)$

To make the analysis tractable, we assume that

A1 the line segments of a given size are uniformly distributed

A2 they need not be totally within the screen The starting points of these segments divide the interval $(-\sigma, 1)$ to $N+1$ equal subintervals, where N is the number of segments and σ their size

Figure 3 5a and 3 5b illustrates a set of N=5 segments of size $\sigma = 25$, that are uniformly distributed

The reason for considering the clipped–off "left-overs" of some of the segments is to maintain a constant overlap for all the points on the screen

Definition For a given point, let Overlap O_v be the number of segments that contain it

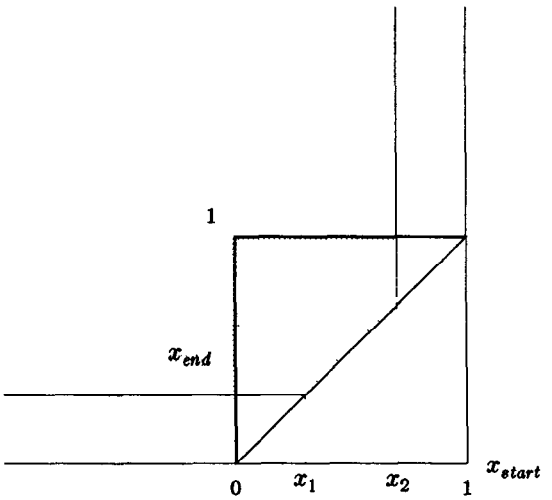


Figure 3.3 The shaded area corresponds to segments intersecting with the segment (x_1, x_2)

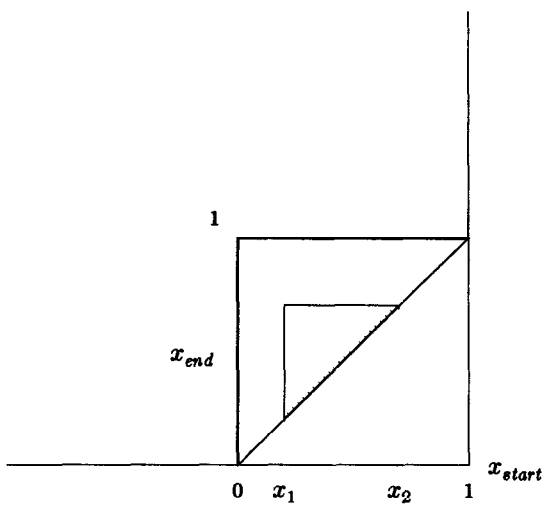


Figure 3.4 The shaded area corresponds to segments covered completely by the segment $S(x_1, x_2)$

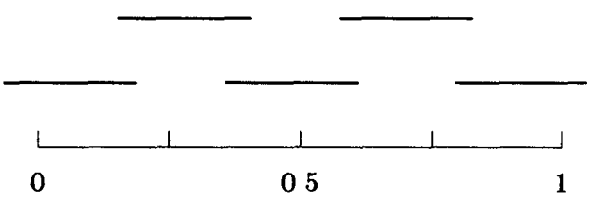


Figure 3.5a
N=5 segments of size σ = 25, uniformly distributed on the screen

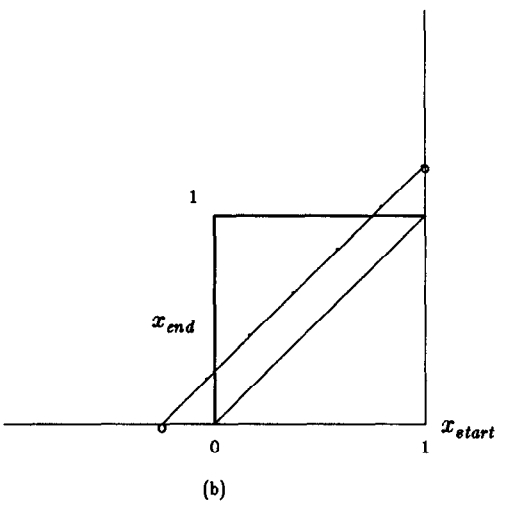


Figure 3.5b
The 2-d transformations of the segments of Figure 3 5a

Table 1 indicates the parameters common to all the forthcoming analyses

3.2. Analysis of R$^+$-trees

We examine two cases In the former, referred to by *one-size* case, we consider N segments of size σ, uniformly distributed as described before In the latter, *two-size* case, we consider two sets of segments, set 1 with N_1 segments of size σ_1 and set 2 with N_2 segments of size σ_2 The segments of each set are uniformly distributed

Parameter	Description
C	the capacity of the data pages (= data records per page)
f	the fanout of the internal nodes of the tree f sons per node
h	the height of an R–tree (the root considered at level 1)
h_+	the height of an R$^+$–tree

Table 1 Problem Parameters

An important Lemma first

Lemma Given N segments of size σ, uniformly distributed on the screen, a line segment (*query segment*) of size q intersects with $intsect(N,\sigma,q)$ segments, where

$$intsect(N,\sigma,q) = \frac{(\sigma+q)}{(1+\sigma)}(N+1) \qquad (1)$$

Proof Consider Figure 3 6 Projecting the line AB on the horizontal axis, we have the line AB', of size $1+\sigma$ The query region (shaded area) intersects the line AB on a segment CD, whose projection $C'D'$ is of size $\sigma+q$ The fraction of the intersected segments (= circles on line CD, or circles on line $C'D'$) is $length(CD)$ / $length(AB)$ = $(\sigma+q)$ / $(1+\sigma)$ Since the line AB is divided into $N+1$ equal intervals by the circles, the line CD will contain on the average

$$\frac{\sigma+q}{1+\sigma}(N+1)$$

circles, which is exactly the number of segments that the query segment q will intersect Notice that the formula does not depend on the position of the query segment on the screen, exactly because the of the way we have defined the uniform distribution of the N segments For q=0, that is, the query segment is a point, we have the formula for the overlap

Corrolary Given N segments with size σ, uniformly distributed on the screen, the overlap is constant, and given by the formula

$$O_v(N,\sigma) = \frac{\sigma}{1+\sigma}(N+1) \qquad (2)$$

From (1) and (2) we have

$$intsect(N,\sigma,q) = O_v(N,\sigma) + q(N+1-O_v(N,\sigma)) \qquad (1')$$

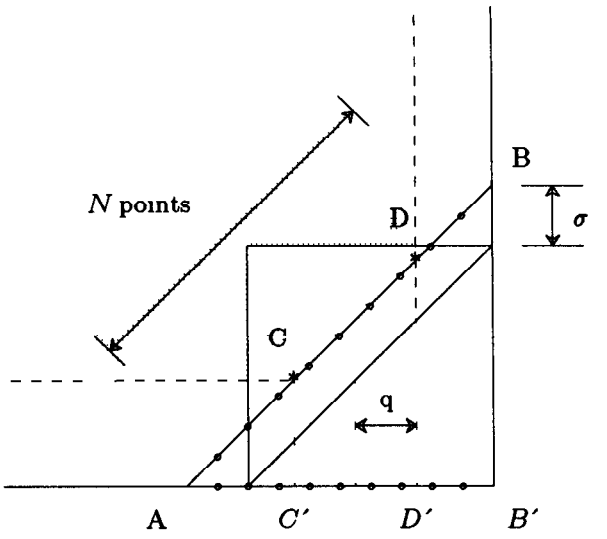


Figure 3.6 N segments of size σ (represented as circles), uniformly distributed on the screen

For example, consider the case of N=5 and σ=0 5 Then the segments are shown in Figure 3 7, for an overlap of [0 5 / (1+0 5)] * (5+1) = 2 Every point on the screen is covered by 2 segments

3.2.1. R$^+$–trees, One–Size Case

Assume that we have N segments of size σ, uniformly distributed on the screen, as explained before Let h_+ be the height of the resulting R$^+$–tree, which is assumed to be **full**, that is, every data page contains C entries, each internal node has f sons The total number of data pages will be

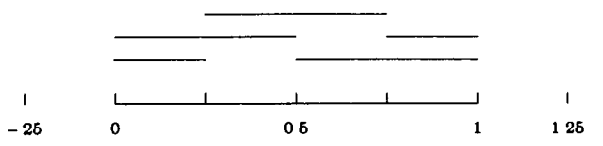


Figure 3.7 5 segments of size 0 5 each, overlap 2

f^{h_+} dividing the screen into f^{h_+} intervals Due to the uniformity assumption, each interval will be of size $1/f^{h_+}$ Each page corresponds to one such interval The segments that intersect such an interval will be entered in the corresponding page Since each page is full, we should have

$$C = intsect(N,\sigma,\frac{1}{f^{h_+}})$$

or

$$C = O + \frac{1}{f^{h_+}}(N+1-O)$$

or

$$h_+ = \log_f \frac{N+1-O}{C-O}$$

where O is the overlap $O = O_v(N,\sigma)$ of the given set of segments The total number of disk accesses r^+da for a point query is the height incremented by one, to account for the retrieval of the data page The root of the tree is assumed to be on the disk Thus

$$r^+da = 1 + \log_f \frac{N+1-O}{C-O} \qquad (3a)$$

or, since $N \gg 1$ and $N \gg O$,

$$\boxed{r^+da \approx 1 + \log_f \frac{N}{C-O}} \qquad (3b)$$

The intuitive explanation is that the capacity of every data page is "reduced" by O, because it has to accommodate the remainders of O segments from each right neighbor (while this very page forces O entries into its left neighbor)

3.2.2. R$^+$–trees, Two–Size Case

Here we assume two sets of segments, set 1 with N_1 segments of size σ_1 and set 2 with N_2 segments of size σ_2 The segments of each set are uniformly distributed on the screen Using the same arguments as before, we have that

$$C = intsect(N_1,\sigma_1,1/f^{h_+}) + intsect(N_2,\sigma_2,1/f^{h_+})$$

or

$$C - O_{v,1} - O_{v,2} = \frac{1}{f^{h_+}}(N_1 + 1 - O_{v,1} + N_2 + 1 - O_{v,2})$$

or

$$h_+ = \log_f \frac{N_1 + N_2 + 2 - O_{v,1} - O_{v,2}}{C - O_{v,1} - O_{v,2}}$$

or

$$h_+ \approx \log_f \frac{N}{C - O_v}$$

where

$N_1 + N_2 = N$

$O_{v,1} = O_v(N_1,\sigma_1)$ the overlap due to set 1 segments

$O_{v,2} = O_v(N_2,\sigma_2)$ the overlap due to set 2 segments

$O_v = O_{v,1} + O_{v,2}$ the total overlap

Thus, finally we have

$$\boxed{r^+da \approx 1 + \log_f \frac{N}{C - O_v}} \qquad (4)$$

Important observations

(1) Eq (4) holds for **any number of sets** of segments (not just two)

(2) The result depends only on the total number of segments and the total overlap, not on the characteristics of each set of segments

3.3. Analysis of R–trees

In the same way as above we do the analysis of R–trees Both one–size and two–size cases are analyzed

3.3.1. R–trees, One–Size Case

First we consider the case of one–size segments The tree is assumed to be full Thus, the segments are grouped in N/C pages, in groups of size C Each page is characterized by the "minimum enclosing segment", that covers all the segments in this page This segment corresponds to a point in the transformed space, for example, in Figure 3 8, "A" is the minimum enclosing segment for the segments A_1, A_2, A_3 and A_4

These segments will be referred to by "fathers of level 1" It can be proven [SELL86] that they are of equal size $\sigma_{father,1}$

$$\sigma_{father,1} = \frac{(1+\sigma)}{(N+1)}(C-1) + \sigma \qquad (5)$$

and that they are uniformly distributed

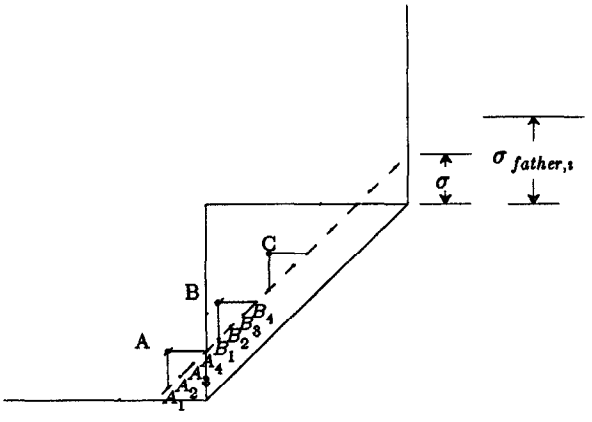


Figure 3.8 Illustration of fathers of level 1 ($C=4$)

Data pages are grouped in $N/(Cf)$ groups, each group containing f data pages (and therefore Cf segments), to form the lowest level of internal nodes of the R-tree ("fathers of level 2") In general, "fathers of level i" have size

$$\sigma_{father,i} = \frac{1+\sigma}{N+1}\left(Cf^{i-1}-1\right)+\sigma \qquad (6)$$

and are uniformly distributed, in the sense described before The root of the R-tree corresponds to a father of level $h+1$

The overlap $O_{v,father,i}$ for fathers of level i is the number of fathers of level i containing a point on the screen Since the number $N_{father,i}$ of i-level fathers is

$$N_{father,i} = \frac{N}{Cf^{i-1}} \qquad (7)$$

their overlap $O_{v,father,i}$ is, from the corollary (Eq (2))

$$O_{v,father,i} = \frac{\sigma_{father,i}}{1+\sigma_{father,i}}(N_{father,i}+1) \qquad (8)$$

which becomes

$$O_{v,father,i} = 1+\frac{O_v-1}{Cf^{i-1}} \qquad (9)$$

To find the number of disk accesses rda in answering a point query, we have to add all the fathers of any level, that cover the query point That is

$$rda = \sum_{i=1}^{h+1} O_{v,father,i} \qquad (10)$$

where h is the height of the (full) R-tree

$$h = \log_f \frac{N}{C} \qquad (11)$$

Then, (11) and (9) give

$$\boxed{rda = h+1+\frac{O_v-1}{C}\frac{f}{f-1}\left(1-\frac{1}{f^{h+1}}\right)} \qquad (12)$$

Eq (12) assumes that h is an integer If not, we can either use linear interpolation, or replace the sum of Eq (10) with an integral from $i=0.5$ to $i=h+1.5$ The difference in the two approximations is small (less than 2%)

3.3.2. R-trees, Two-Size Case

As with the R$^+$-trees, we consider N_1 segments of size σ_1 and N_2 segments of size σ_2 Each set is uniformly distributed on the screen A tight (if not the tightest) packing of segments in data pages is by storing $C_1 = N_1/N$ and $C_2 = N_2/N$ consecutive segments of set 1 and set 2 respectively in each data page, starting from left to right ($N = N_1 + N_2$) C_1 and C_2 will be referred to by *effective capacities* for set 1 and set 2 respectively Figure 3.9 illustrates the grouping of segments into pages, for $C_1 = 3$, $C_2 = 2$, $N_1 = 6$, $N_2 = 4$

Notice that each set has N/C fathers of level 1, with sizes

$$\sigma_{1,father,1} = \frac{1+\sigma_1}{N_1+1}(C_1-1)+\sigma_1$$

$$\sigma_{2,father,1} = \frac{1+\sigma_2}{N_2+1}(C_2-1)+\sigma_2$$

according to (5) It can be shown [SELL86] that every father of larger size covers completely the corresponding father of smaller size In Figure 3.9, B_1 covers A_1 and B_2 covers A_2 The set whose level 1 fathers are larger is called the **dominating** set Notice that it is possible to have set 1 as dominating set, even though $\sigma_1 < \sigma_2$

As before, fathers of level 1 are uniformly distributed In fact, if the subscript *dom* stands for the dominating set, the R-tree will behave exactly the same as if

- we had N_{dom} segments of size σ_{dom}
- the fanout was f as before

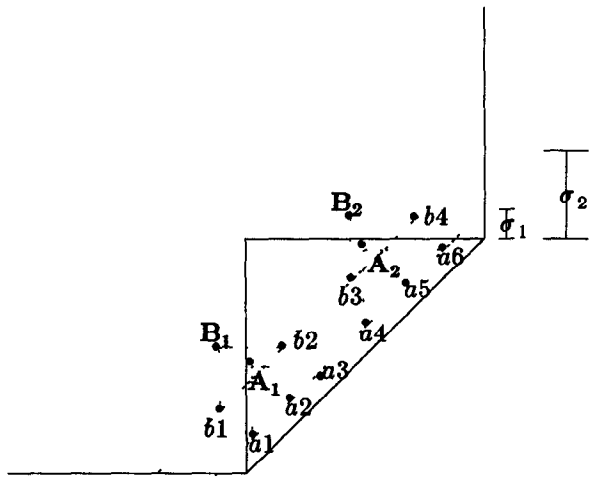


Figure 3.9 Illustration of the grouping for R-trees
$C_1 = 3$, $C_2 = 2$, $N_1 = 6$, $N_2 = 4$

- the data page capacity was C_{dom}

That is, the only effect of the dominated set is that it occupies a fraction $1-N_{dom}/N$ of the data pages reducing their effective capacity to C_{dom} for elements of the dominating set Thus, the final formula for disk accesses is derived from Eq (12)

$$rda = h+1+\frac{O_{v,dom}-1}{C_{dom}}\frac{f}{f-1}\left(1-\frac{1}{f^{h+1}}\right) \quad (13)$$

where

$$h = \log_f\frac{N}{C} = \log_f\frac{N_1}{C_1} = \log_f\frac{N_2}{C_2}$$

and

$$dom = \begin{cases} 1 & \text{if } \sigma_{1,father,1} > \sigma_{2,father,1} \\ 2 & \text{otherwise} \end{cases}$$

Notice that Eq (13) holds for **more than two sets** of segments, the only thing that matters is which set will be the dominating set

4. Analytical Results

Based on the formulas extracted in the previous section, we obtained some results that indicate how R–trees compare to R$^+$–trees The following values were assumed

P = The size of a page = 1,024 bytes
S = The size of a rectangle = 16 bytes
p = The size of a pointer = 4 bytes

Given these numbers we can derive the fan-out f and capacity C of non–leaf and leaf pages respectively We assume that the first 24 bytes of a page are taken for the header Clearly

$$f = C = \left\lfloor \frac{P}{(16+4)} \right\rfloor = 50$$

Finally, the number of data tuples N was assumed to take values between 100 and 1,000,000

The analytical results shown in this section fall under two categories First, we show the number of disk accesses required to search an R–tree or R$^+$–tree in case of a point query Since in general this figure depends on more than one parameters, we choose to fix all but one and display in a chart the number of disk accesses versus the only left parameter Second, we show in a combined chart the performance gain as it changes with the various parameters We define performance gain as

$$Perf\ Gain = \frac{rda-r^+da}{rda}100$$

where rda is the number of disk accesses for the R–tree and r^+da is the number of disk accesses for the R$^+$–tree

4.1. The One–Size Case

Figures 4 1a and 4 1b show the number of disk accesses required to process a point query in an R–tree and the corresponding R$^+$–tree for a file of N segments with O overlap 4 1a gives the disk accesses as a function of O in the case where N=100,000 4 1b gives the same values as a function of N in the case where O=40

As expected the R–tree has always worse performance than the R$^+$–tree because of the excessive searching required In both cases the number of disk accesses increases with the number of segments since the height of the tree increases Finally, increasing overlap also accounts for increased number of pages searched, in R–trees this

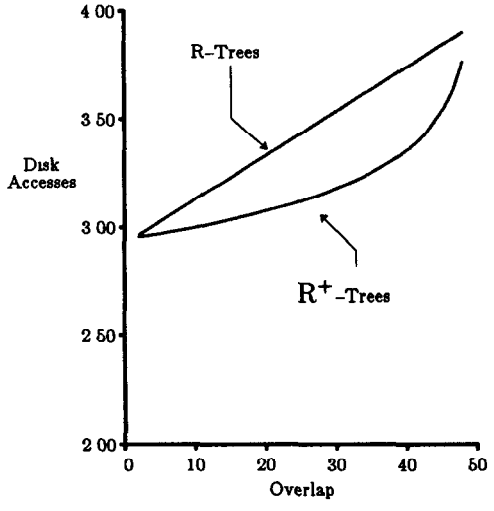


Figure 4.1a Disk Accesses for One-Size Segments, as a function of O, N=100,000

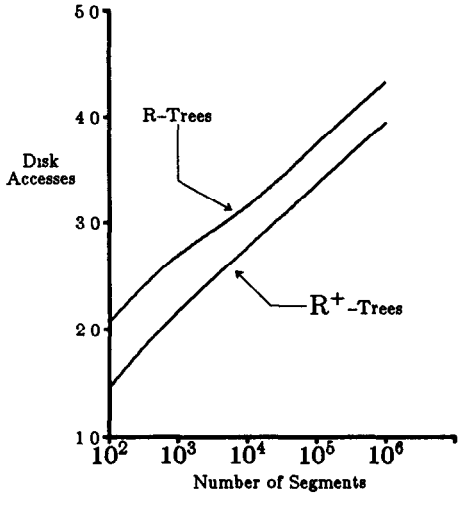


Figure 4.1b Disk Accesses for One-Size Segments, as a function of N, O=40

is due to multiple paths that have to be followed till the leaves have been reached, while in the R+-tree it is due to increasing number of sub-segments that have to be created because of splits Notice that the difference in disk accesses is not very significant This is true because of the assumption that all segments have same size In this case the R-tree does not incur a lot of overlapping segments that would cause searching multiple paths in the tree The two-size case results that follow later show the effect that lengthy segments have on the performance

The next set of figures, Figures 4 2a-b, illustrate how the performance gain varies with the overlap and number of segments for different values for the total number of segments and overlap respectively Again as expected, the gain in performance decreases as the total number of segments increases since the R+-tree will create a lot of sub-segments trying to keep all intermediate node segments non-overlapping However, the figures show that improvements up to 30% can be achieved

4.2. The Two-Size Case

Figure 4 3a-b shows the disk accesses required for searching an R-tree and a corresponding R+-tree used to index N_1+N_2=100,000 segments with total overlap of O_1+O_2=40 The first figure (4 3a) shows disk accesses required as a function of the small segment overlap O_1 in the case where the large segments account for 10% of the total number of segments (N_2=10,000, N_1= 90,000)

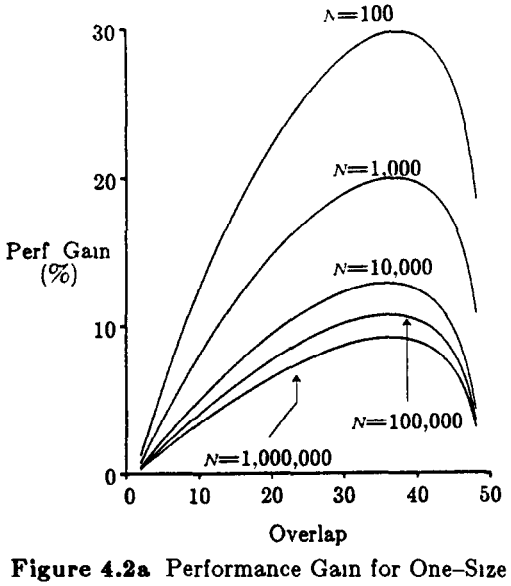


Figure 4.2a Performance Gain for One-Size Segments, as a function of O, N=100-1,000,000

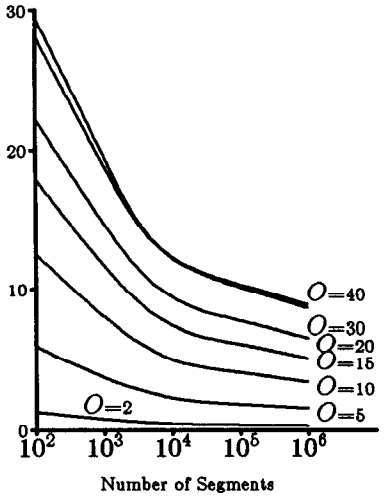


Figure 4.2b Performance Gain for One-Size Segments, as a function of N, O=2-45

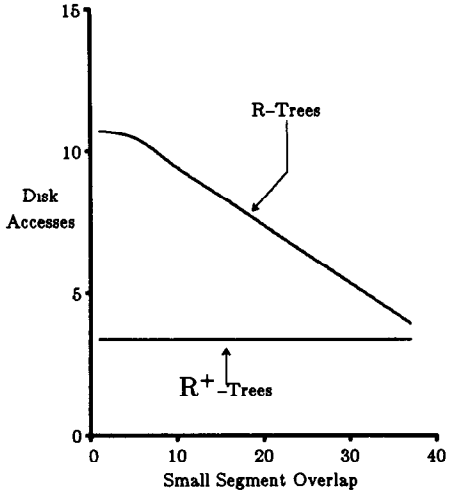


Figure 4 3a Disk Accesses for Two-Size Segments, as a function of O_1, N_2=10,000

Figure 4 3b shows the number of disk accesses as a function of the number of large segments N_2, when the small segment overlap O_1=5 These figures show clearly the problem that the R-tree has in handling few, lengthy segments In such situations, an R-tree may require more than twice the page accesses required by an R+-tree Notice also that the performance of the R+-tree is "immune" to changes in the distribution of the segment sizes The number of disk accesses depends *only* on the

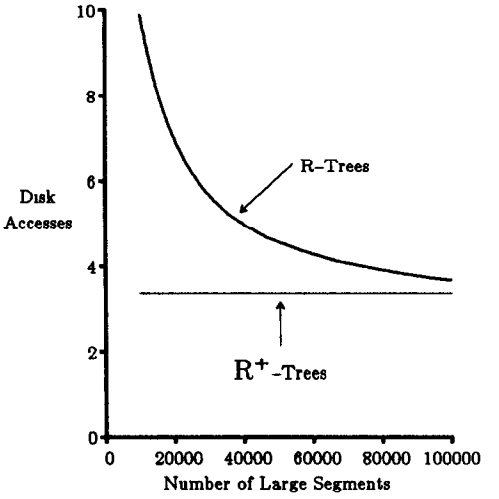


Figure 4.3b Disk Accesses for Two-Size Segments, as a function of N_2, O_1=5

total number of segments and the total overlap observed

In the final set of figures, Figures 4 4a-b, we illustrate again the performance gain as a function of the overlap observed among the small segments (O_1) and as a function of the number of large segments (N_2), respectively The advantage of R+-

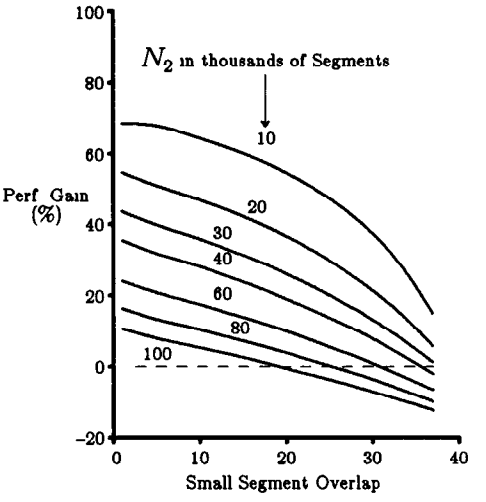


Figure 4.4a Performance Gain for Two-Size Segments, as a function of O_1, N_2=10,000-100,000, N=100,000

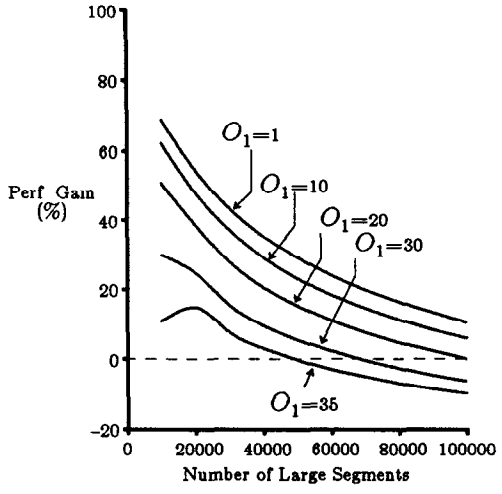


Figure 4.4b Performance Gain for Two-Size Segments as a function of N_2, O_1=5-35, N=100,000

trees of R-trees is shown clearly in these figures Improvements up to 70% can be achieved That means that the R-tree will require in some cases more than twice the page accesses required using an R+-tree Of course, when the number of large segments approaches the total number of segments, R+-trees will lose since many lengthy segments cause a lot of splits to sub-segments However, typical distributions do not have this characteristic On the contrary, lengthy segments are few compared with small ones (e g in a VLSI design)

In general our results are very encouraging We are currently working on their experimental verification [SELL86]

5. Contributions

The contributions of this work can be summarized as follows

(1) A variation to R-trees, R+-trees that was originally proposed in [STON86] is compared to the original proposal by Guttman Such a comparison is very useful since it gives us a good understanding of how R-trees work and why they suffer under various object distributions

(2) The introduction and effective use of the transformation to higher dimension as a tool for the analysis of R- and R+- trees This transformation is especially useful for line-segments, ie , 1-dimensional boxes on a 1-dimensional screen, because the transformed space is the plane and it can be easily drawn Thus, simple proofs of the performance results can be derived The formulas for the line-segments case will be the stepping stones for the analysis of boxes of higher dimensionality

(3) In this paper we provide the first known analysis of R-trees Our formulas for the two-size case can be easily generalized to handle any number of sizes one of the sizes will "dominate" the rest and the formulas can be readily applied The uniformity assumption that the model is based on is admittedly strong This does not reduce its theoretical and practical value It corresponds to the best-case results we should expect from an R-tree Moreover, if the distribution in a given situation is uniform except for minor perturbations, the derived formulas will give good estimates for the expected performance

(4) Finally, using exactly the same assumptions, we derived formulas for the proposed R+-trees and compared the two methods analytically The results of the comparison agree completely with the intuition The R-trees suffer in the case of few, long segments, which force a lot of "forking" during the search The R+-trees handle these cases easily, because they split these long segments into smaller ones

Although our analysis has concentrated on point queries, similar results have been obtained for region queries Due to space limitations they are not presented in this paper (they will appear in [SELL86])

Our current and future work in the area focuses in two major tasks First, we are currently working in extending the analysis to the case of arbitrary dimensions This will allow us to examine objects in two dimensional spaces which are found in many applications Moreover, we try to model the problem under different assumptions than the uniform size one used in this study Our results will appear in a companion paper [SELL86] Second, we have initiated an experimentation effort that will allow us to verify our analytical results Guttman's code for R-trees is available, we are in the process of modifying it to come up with code for R+-trees and we expect to obtain experimental results soon

6. References

[BAYE72] Bayer, R and McCreight, E , *Organization and Maintenance of Large Ordered Indexes*, Acta Informatica, Vol 1, No 3, 1972

[BENT75] Bentley, J L , *Multidimensional Binary Search Trees Used for Associative Searching*, CACM, Vol 18, No 9, 1975

[CHAN81] Chang, J M and Fu, K S , *Picture Query Languages for Pictorial Database Systems*, IEEE Computer, Vol 14, No 11, November 1981

[FINK74] Finkel, R A and Bentley, J L , *Quadtrees A Data Structure for Retrieval on Composite Keys*, Acta Informatica, Vol 4, No 1, 1974

[GUNT86] Gunther, O , *The Cell Tree An Index for Geometric Data*, Memorandum No UCB/ERL M86/89, University of California, Berkeley, December 1986

[GUTT84a] Guttman, A , *R-Trees A Dynamic Index Structure for Spatial Searching*, Proceedings of the 1984 ACM–SIGMOD International Conference on Management of Data, Boston, MA, June 1984

[GUTT84b] Guttman, A , *New Features for Relational Database Systems to Support CAD Applications*, PhD Thesis, University of California, Berkeley, June 1984

[HINR83] Hinrichs, K and Nievergelt, J , *The Grid File A Data Structure to Support Proximity Queries on Spatial Objects*, Tech Report 54, Institut fur Informatik, ETH, Zurich, July 1983

[KEDE81] Kedem, G , *The Quad-CIF tree A Data Structure for Hierarchical online Algorithms*, Tech Report 91, Computer Science Dept , University of Rochester, Rochester, NY, September 1981

[NIEV84] Nievergelt, J , Hinterberger, H and Sevcik, K C , *The Grid File An Adaptable, Symmetric Multikey File Structure*, ACM Transactions on Database Systems, Vol 9, No 1, March 1984

[OREN86] Orenstein, J A , Proceedings of the 1986 ACM–SIGMOD International Conference on Management of Data, Washington, DC, May 1986 *Spatial Query Processing in an Object–Oriented Database System*,

[ROBI81] Robinson, J T , *The K-D-B tree A Search Structure for Large Multidimensional Dynamic Indexes*, Proceedings of the 1981 ACM–SIGMOD International Conference on Management of Data, April 1981

[ROUS85] Roussopoulos, N and Leifker, D , *Direct Spatial Search on Pictorial Databases Using Packed R-Trees*, Proceedings of the 1985 ACM–SIGMOD International Conference on Management of Data, Austin, TX, May 1985

[SAME83] Samet, H and Webber, R E , *Using Quad-trees to Represent Polygonal Maps*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Washington, DC, June 1983

[SAME84] Samet, H , *The Quadtree and Related Hierarchical Data Structures*, ACM Computer Surveys, Vol 16, No 2, June 1984

[SELL86] Sellis, T , Faloutsos, C and Roussopoulos, N , *Object Oriented Access Methods for Spatial Objects Algorithms and Analysis*, in preparation

[STON83] Stonebraker, M , Rubenstein, B and Guttman, A, *Application of Abstract Data Types and Abstract Indices to CAD Data Bases*, Tech Report UCB/ERL M83/3, Electronics Research Laboratory, University of California, Berkeley, January 1983

[STON86] Stonebraker, M , Hanson, E and Sellis, T , *Rule Indexing Implementations in Database Systems*, Proceedings of the First International Conference on Expert Database Systems, Charleston, SC, October 1986

[TAMM82] Tamminen, M , *The EXCELL Method for Efficient Geometric Access to Data*, Acta Polytech Scand Mathematics and Computer Science Series, No 34, Helsinki