Frank, Andrew U., and R. Barrera. "The Fieldtree: A Data Structure for Geographic Information Systems." In *Symposium on the Design and Implementation of Large Spatial Databases*, edited by A. Buchmann, O. Günther, T.R. Smith and Y.-F. Wang, 29-44. New York, NY: Springer-Verlag, 1990.

Andrew U. Frank
Renato Barrera
National Center for Geographic Information and Analysis
and
Department of Surveying Engineering
University of Maine
Orono, ME 04469, USA
FRANK@MECAN1.bitnet
RENATO@MECAN1.bitnet

Abstract

Efficient access methods, such as indices, are indispensable for the quick answer to database queries. In spatial databases the selection of an appropriate access method is particularly critical since different types of queries pose distinct requirements and no known data structure outperforms all others for all types of queries. Thus, spatial access methods must be designed for excelling in a particular kind of inquiry while performing reasonably in the other ones. This article describes the Fieldtree, a data structure that provides one of such access methods. The Fieldtree has been designed for GIS and similar applications, where range queries are predominant and spatial nesting and overlaping of objects are common. Besides their hierarchical organization of space, Fieldtrees are characterized by three other features: (i) they subdivide space regularly, (ii) spatial objects are never fragmented, and (iii) semantic information can be used to assign the location of a certain object in the tree. Besides describing the Fieldtree this work presents analytical results on several implementations of those variants, and compares them to published results on the Rtree and the R^+tree.

1 Introduction

Spatial databases deal with the description of the geometry and other attributes of objects in space. Present technology provides a persistent storage media (hard disk) with a linear address space, partitioned into pages. We will call *original* space to the one that contains the locus of the spatial entities.

*This research was partially funded by grants from NSF under No. IST 86-09123 and Digital Equipment Corporation (Principal Investigator: Andrew U. Frank). The support from NSF for the NCGIA under grant number SES 88-10917 is gratefully acknowledged.

The purpose of spatial access methods is to provide a mapping from regions in original space to sets of pages in *disk* space. To be efficient, that mapping should have two characteristics: (i) use disk space efficiently and (ii) require the least possible amount of disk accesses.

Spatial databases are used in several fields: CAD-CAM, GIS, VLSI design, image processing, etc. Spatial access methods are also used in non-spatial databases for the implementation of multikey indices, of joins in relational databases [Kitsuregawa 1989], etc.

This paper is organized as follows: The next section presents several existing spatial access methods that provide a foundation for section 3, in which the Fieldtree is described. Section 4 presents two implementations of the Fieldtree and the following section discusses the operations upon the trees. Section 6 presents analytical results for different implementations of the Fieldtree, and finally, we conclude with some comments.

2 Access Methods for Spatial Objects

A spatial access method should provide a mapping from a (multidimensional) original space to a (unidimensional) disk space. Ideally this mapping should preserve vicinities, i.e., should map neighboring objects in original space into neighboring disk pages. This is unfortunately impossible, since any bijective mapping from the 2-d plane to a line is discontinuous.

There are mappings, such as Morton keys [Samet 1984] or z-order [Orenstein 1986], that preserve some vicinities. The selection of an adequate mapping is complicated further by the fact that pages can hold a limited amount data; if the capacity of one of them is exhausted, a mechanism (such as splitting or chaining) for passing data to other pages should be provided, thus interfering with the desire of preserving vicinity relationships. Moreover, a good spatial access method ought to take into account the dynamic characteristics of data.

This section will deal with the types of spatial queries, with a method to simplify the solution to those inquiries, and with a taxonomy of spatial access methods.

2.1 Types of Spatial Queries

General-purpose DBMS's usually provide two types of access methods: the *primary* one that fetchs a record when given a unique identifier, and the *secondary* ones, retrieving sets of records that obey a specific predicate. The methods that provide the fastest primary access are ill-suited for secondary access; secondary access methods are not efficient enough for primary access.

In a similar fashion, spatial queries can be divided into two categories: (i) *point* queries, that return objects containing that point, and (ii) *range* queries that deal with objects fulfilling a given relationship with respect to a window W in original space. That relationship can be one of: (i) intersection (all objects with points in common with W), (ii) inclusion (all objects that contain W), (iii) containment (all objects wholly inside W).

In agreement with the access methods used in general-purpose DBMS's, no known spatial access method performs optimally for all applications. Hence, we conclude that they should be designed for an outstanding performance under the most frequent conditions and a reasonably good one during the rest of the time. This implies that spatial access methods should be selected depending on the application

area and that a spatial DBMS may provide more than one.

2.2 Simplifying the Description of Shape

Geometric information can be decomposed into two constituents: (i) position and (ii) shape. It is straightforward to include position into an indexing schema, but the expenses involved in utilizing a thorough description of the shape outweigh the advantages.

Thus, the simplification of the shape of an objects is very convenient for the design of efficient algorithms. That simplification is exclusively provided for the spatial index; the exact description of the shape should continue to be stored into the database. The most common simplification is based upon the tight enclosure of the object inside a simple figure; that figure will be later used by the access method as the object's spatial surrogate. The selection of a suitable figure involves two criteria: (i) a simple description, and (ii) good fit of the spatial objects to be considered. The most common ones are rectangles [Guttman 1984], circles [Hinrichs 1985] or a convex bodies [Günther 1989].

Circumscription with a simple figure has two advantages: (i) it reduces the storage needed for the access method and (ii) it simplifies and accelerates the processing of locational queries. It might, however, include references to objects that do not satisfy the desired spatial predicate.

Enclosing the objects within simple figures leads to the concepts of *transformed* spaces of higher dimension. A rectangle, for example, is completely characterized by the position of two diagonally opposed vertices, each described by two coordinates. Hence, any rectangle can be represented by four numbers, and thus, rectangles can be considered as a *point* in a four dimensional spaces. The coordinate selection for that four-dimensional space is not unique. Figure 1 illustrates three of those selections. Fig. 1a) shows a set of line segments, and figures 1b), 1c) 1d) display three coordinate selections, that use respectively: i) The coordinates of two opposed vertices ii) The coordinates of one vertex together with the extents of the rectangle along the ordinate and the abscissa axis iii) The coordinates of the centroid, together with the half-extents of the rectangle along the ordinate and abscissa axis.

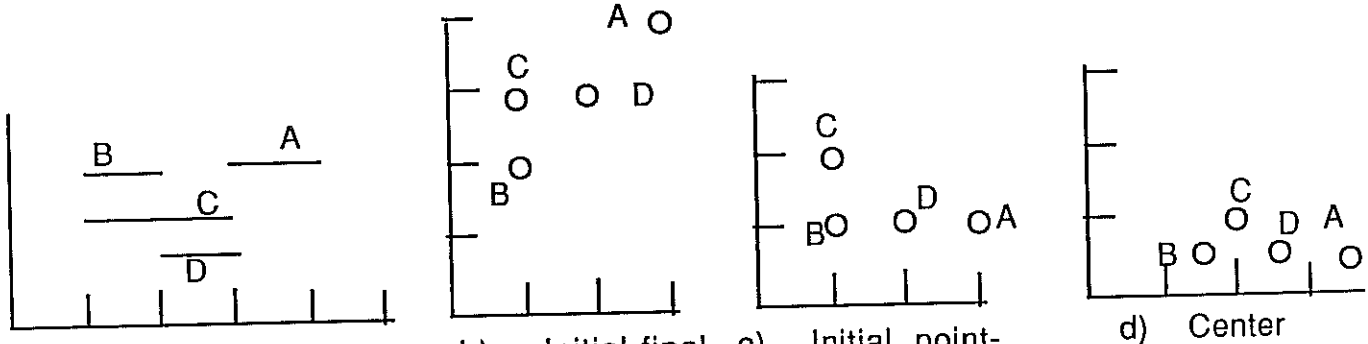


Fig. 1 A set of segments and different representations

2.3 Types of Spatial Retrieval Methods

A brief review of existing spatial retrieval methods will be given, as a preamble to the presentation of the Fieldtree with the intention of relating its characteristics to those of other methods.

Spatial retrieval methods can be classified in several categories, according to:

Type of geometric data (point vs. region)

A spatial access provides a map from a multidimensional original space to *disk* space. The access method is simpler if all spatial information refers to isolated points, since each can be assigned to a unique position in disk. Ambiguity may arise, however, while dealing with region data, because the mapping of an object from original to disk space can yield several disk addresses. That causes a need for methods specially tailored to a particular type of geometric data.

Handling of objects (non-fragmenting vs. fragmenting)

A spatial access method provides a map from the object spatial characteristics to a page in disk; if objects have non-zero dimensions, a fraction of them might be mapped into more than one page. Two alternatives follow: either (i) divide the object (at least conceptually) so each fragment is assigned to a unique page and increase memory size , or (ii) mantain the integrity of the objects and perform extra disk accesses. The R^+tree [Sellis 1987] is an example of the first alternative, while the Rtree [Guttman 1984] exemplifies the second one.

Retrieval method (direct vs. hierarchical)

This classification takes into account the implementation of the mapping between original and disk spaces and subdivides access methods into two categories: *direct* methods (such as the grid file [Nievergelt 1984]), that implement this mapping as a function, and *hierarchical* methods (such as the PR quadtree [Samet 1984]) that navigate through a data structure to obtain the desired disk address.

Space subdivison (regular vs. data determined)

Both direct and hierarchical methods consider subdivisions of space. That subdivision can be done in either of two fashions: (i) in a regular one, or (ii) according to the object's geometry and a criterion of minizing space usage and maximizing the speed of operations. Examples of data determined vs regular hierarchical partition direct access methods are kd-tree and the PR quadtree for hierarchical methods [Samet 1984] respectively

3 The Fieldtree

This section will present the characteristics of the Fieldtree and of the objects that can be placed into it.

The Fieldtree is a data structure, initially developed at ETH Zurich [Frank 1983] and used in PANDA, an object-oriented database system [Frank 1982] It has been designed for its usage in Geographic Information Systems, where the following circumstances prevail:

- Point, line and area objects are coexistent.
- Spatial nesting and partition of objects is common
- Range queries are frequent
- Spatial coverage among objects induce a lattice rather than a hierarchy.

3.1 Characterization of the Fieldtree

The Fieldtree provides a spatial access method that is:

- Region-oriented, works in the original space.
- Non-fragmenting.
- Hierarchical in nature.
- Based upon regular decomposition.

3.1.1 Original Space-Region Oriented

The Fieldtree is a hierarchical organization of (not necessarily disjoint) regions, called fields, each one associated to a disk page. If a disk page contains the data of an object, then the field associated to that page must cover the spatial extent component of the object. Organization in fields insures that many of the neigborhood relationships among objects are preserved in disk space.

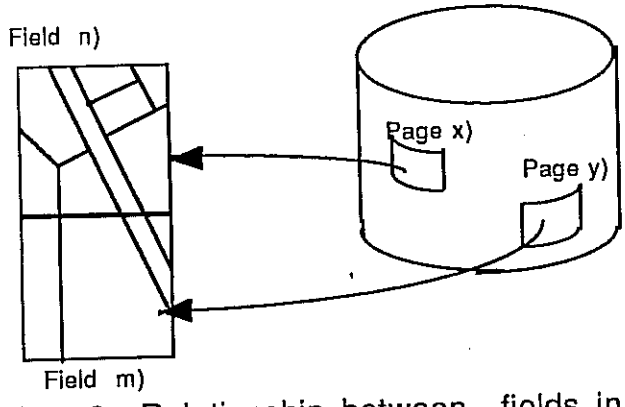


Fig. 2 Relationship between fields in original space and pages in disk space

A minimal bounding rectangle, as those introduced in Section 2.2, is associated to each object to facilitate geometric manipulation of data.

3.1.2 Non-Fragmenting

All of the data of an object is stored in a page associated with the field that covers the spatial component of the object. Fields are not necessarily mutually disjoint. If an object can be stored in several existing fields, additional rules are provided to allocate it to the one most suited to its type and size.

3.1.3 Regular Decomposition

Space is decomposed into squares, each one of them being a field. Similarly to a quadtree, there can exist several *levels* of decomposition; the squares at a certain level are regularly spaced and have the same extent, and the extents of the fields is typically halved from one level to the next. Two facts must be stressed:

i) Fields of a certain level need not be mutually disjoint (i.e, they form a cover, athough not necessarily a partition)

ii) A given level of decomposition nees not refine previous levels, i.e., a given field need not be exactly described as a union of fields at following levels.

Figure 3 shows two regular decompositions of space: (i) a partition and (ii) a covering. Dotted lines represent the boundaries of the so-called "median subfield", that is the subfield spanned by the medians of the centroids of the fields.

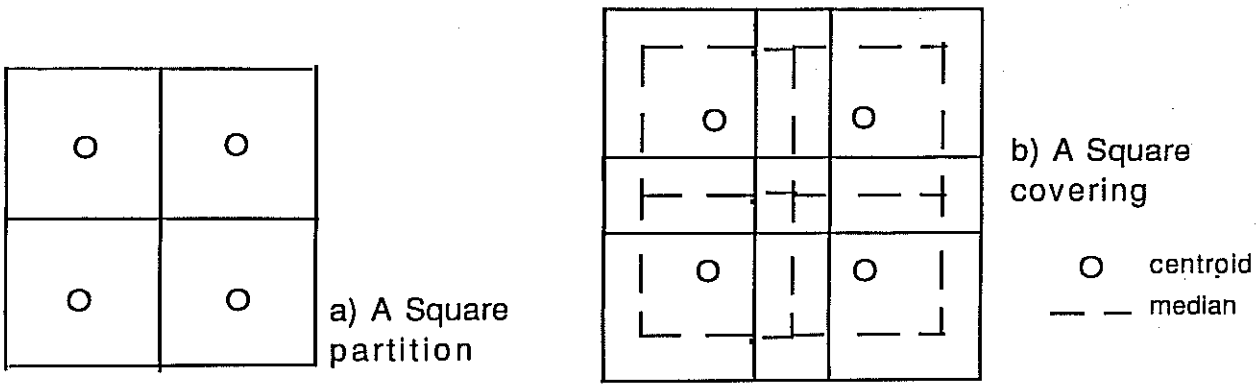


FIG 3 Two regular decompositions

3.1.4 Hierarchical organization

Fields at different levels of decomposition form a Directed Acyclic Graph (DAG) rather than a hierarchy. The descendants of a field $\mathbf{F}$ are all those at the next subdivision level that have their centroids spanned by the median subfield of $\mathbf{F}$. That hierarchy serves a dual purpose: (i) as a navigation device for access to data, (ii) and as a pathway for data migration when fields become filled.

Since successive levels of decomposition need not refine each other, we are free to specify the relative displacement among the centroids of different levels of decomposition. The criterion for that relative displacement is one that avoids the placement of objects into fields far bigger than their size. Fig. 4.b) 4.d) show two different one-dimensional equivalents of the Fieldtree; the trees' fields are shown in 4.a). 4.c); those of 4.a) form a partition and the ones of 4.b) a cover. Section 4 will present two variations of the Fieldtree: one involving partitions and the other covers.

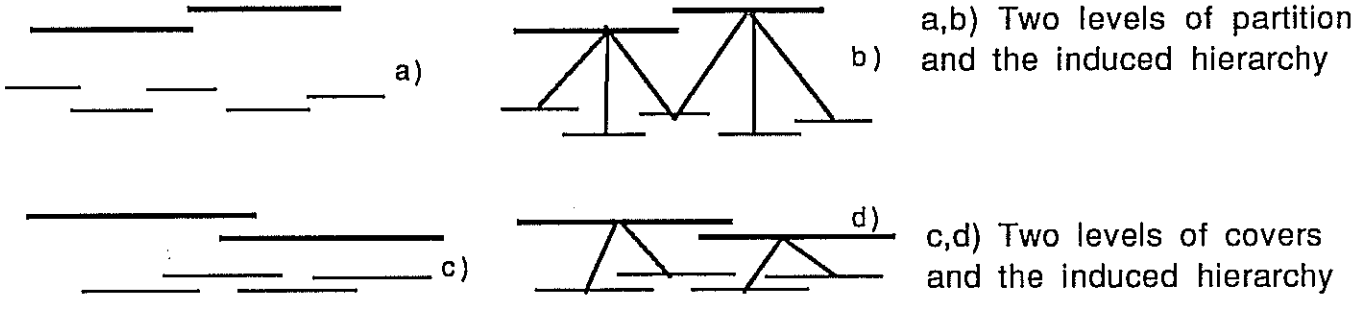


Fig. 4 Hierarchies induced by partitions and coverings (1 dimension)

3.2 Optimization of Object Allocation

Two topics will be treated here: how to use semantic information to facilitate the retrieval and creation operations of objects, and how to handle overflow after an insertion.

3.2.1 Semantic Information

Queries in a database specify *a priori* the object types to be retrieved, as shown in the following query posed in a spatial SQL dialect [Egenhofer 1988a] that asks for the name of all cities in Maine and that utilizes the object types *city* and *state*:

```
SELECT city.name
FROM   city, state
WHERE  state.name = "Maine" AND
       city.geometry INSIDE state.geometry
```

The approximate size of cities is much bigger than the one of many other entities (e.g., farms), and will be necessarily placed in the levels of the Fieldtree where fields cover larger areas. If that information is not available to the DBMS, an exhaustive and unnecessary search in the lower levels of the Fieldtree would be performed. It seems reasonable to attach to each type an 'importance' attribute that indicates the minimum size of a field where it can be placed, and thus limit the levels of the tree to be searched for objects of this type.

3.2.2 Overflow Management

The insertion of a new object into the database involves locating the proper field of the tree, using information both on the object data type and its size. Since a field has associated a certain data capacity, if that capacity has already been reached, the inclusion of an extra object causes the creation of one (or more) descendants of the field; all the objects from the original field that fit in the new field (or fields) will be transferred to their new location. If no object can migrate, the field disk storage is extended with an overflow page.

4 Variants of the Fieldtree

This section presents two versions of the Fieldtree. Some results on the behavior of those variants are presented in Section 6, and a more detailed presentation of those results can be found in [Barrera 1989].

4.1 The Partition Fieldtree

In this variant, in similarity to the PR quad-tree [Samet 1984], the fields constitute a partition of the space. In opposition to the PR tree, the centroids of the fields at the different levels of subdivision are not symmetrically placed. If objects are to be stored unfragmented, the symmetric placement of the centroids in the PR tree causes those objects that intersect edges of the Fieldtree to be stored at nodes of unnecessary large extent (Fig 5). This problem has been circumvented by some authors [Abel 1984] [Orenstein 1989] by allowing some controlled fragmentation in those cases. One of the design goals of the fieldtree was to avoid fragmentation.

The partition Fieldtree avoids this problem by shifting the positions of the centroids of one level with respect to those of the previous one. A relative displacement of one half the extent of a field in both coordinates guarantees that an object can be stored at most two levels above the lowest possible one

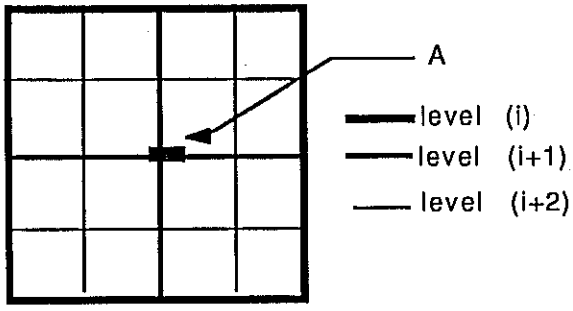


Fig 5 Small objects might be positioned at large nodes in a quadtree

(based on its size only). Even if all objects are of the same size, their x-y position will determine in which of those three levels a particular object is placed and hence, the size of the receiving field. Fig. 6 shows the relative arrangement of fields at different levels.

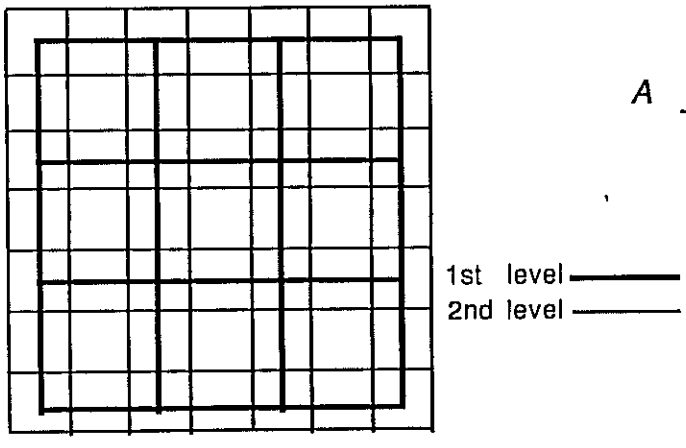


Fig. 6 Two levels of the partition tree

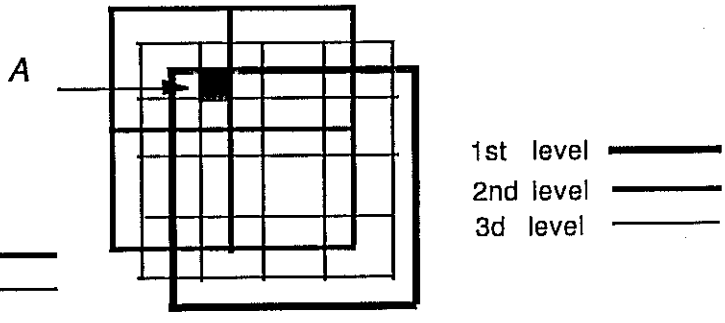
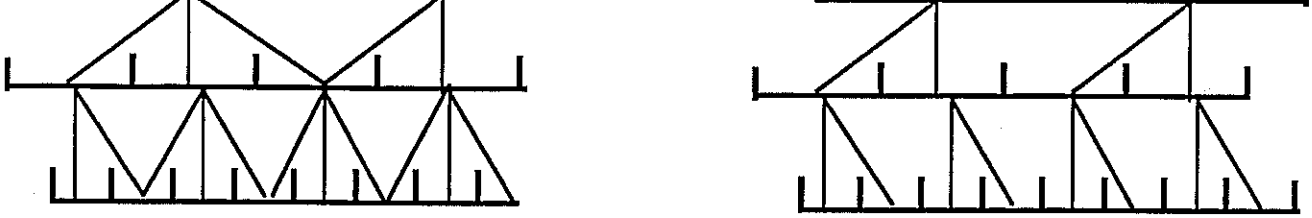


Fig. 7 Worst case situations in the partition tree

Some peculiarities of the partition Fieldtree are:

- Fields at different levels cannot have collinear edges.
- A field can have up to nine descendants. Althoug field form a DAG, then do not constitute a tree in the strict sense of the word. Depending on their position, fields can have a maximum of 1, 2 or 4 ancestors.
- Objects are intended to be placed into the field that fits them best. The worst fit involves placing an object in a field 2^3 times its size; that situation is illustrated by the black square of Fig. 7
- An object may be forced to descend two levels when a field splits (e.g. when it is placed in a position such as the one marked with an A in Fig. 7)

Even if the partition Fieldtree forms really a DAG, it can be implemented using that structure by selecting a spanning tree and treating the descendants of a field as *direct* and *indirect* children, as suggested by Banerjee et al. [Banerjee 1988]. A possible selection of the spanning tree for the one dimensional case is shown in Fig. 8. The use of a spanning tree has several advantages: it enables the adoption of a Morton-type key for each field, thus allowing a physical implementation as a linear array [Abel 1984]. It also provides a method for page clustering.



a) The Partition DAG

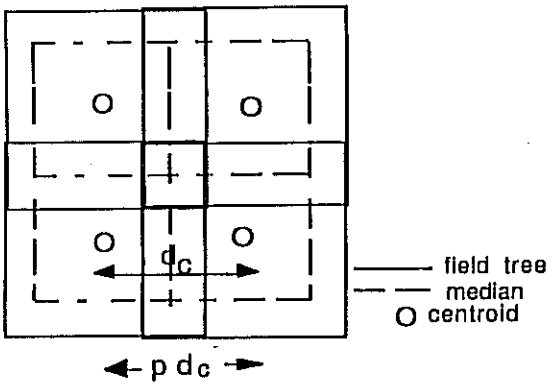
b) A spanning tree

Fig. 8 A DAG for a 1-d partition fieldtree and its spanning tree

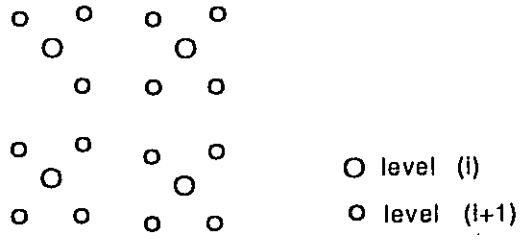
4.2 The Cover Fieldtree

Similar to the PR quad-tree [Samet 1984] this tree keeps the centroids of the different space division levels symmetrically aligned. In opposition to it, fields generate a cover, not a partition: if d_c is the distance along the coordinate axis between two consecutive centroids, adjacent fields overlap $p \times d_c$ units along that axis (see fig. 10). Hierarchical regular covers of space have been used also in computer vision [Burt 1981].

Fig. 9a) shows the arrangement of the fields at a given partition levels, Fig. 9.b) the arrangement of the centroids of two consecutive partitions.



a) Space division in fields



b) Positioning of centroids at different levels

FIG 9 FIELDS IN THE COVER FIELD TREE

When $p < 1$:

- Each field has one main subfield, four subfields that it shares with either of its four *direct* neighbors along the coordinate axis, and four subfields that it shares with two direct neigbhors and one neighbor along a diagonal.
- A field can have up to 4 descendants and only one ancestor.
- Edges of different fields at different levels never coincide.

An overlap $p > 0$ between consecutive fields enables the storage of any object of size pd_c inside a field (see Fig. 10). Unlike the partition Fieldtree, it needs not resort to several tree levels to store a set of objects of identical size; moreover, only immediate descendants need to be considered when a field is split.

A consequence for overlapping fields is that objects can be placed in more than one field at a certain level of the tree. Thus, assignment rules must be provided. Some possible rules are:

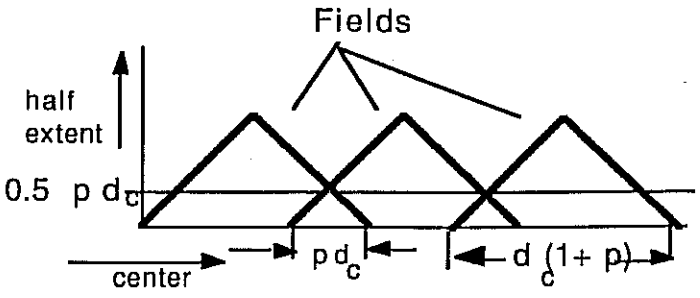


FIG 10 FIT OF OBJECTS IN A 1-D COVER FIELD TREE

Closest Centroid Assign to the field whose centroid is closest to that of the object's rectangle

Closest Corner Assign to the field whose SW (or SE, etc.) corner is closest to the corresponding corner of the object's rectangle

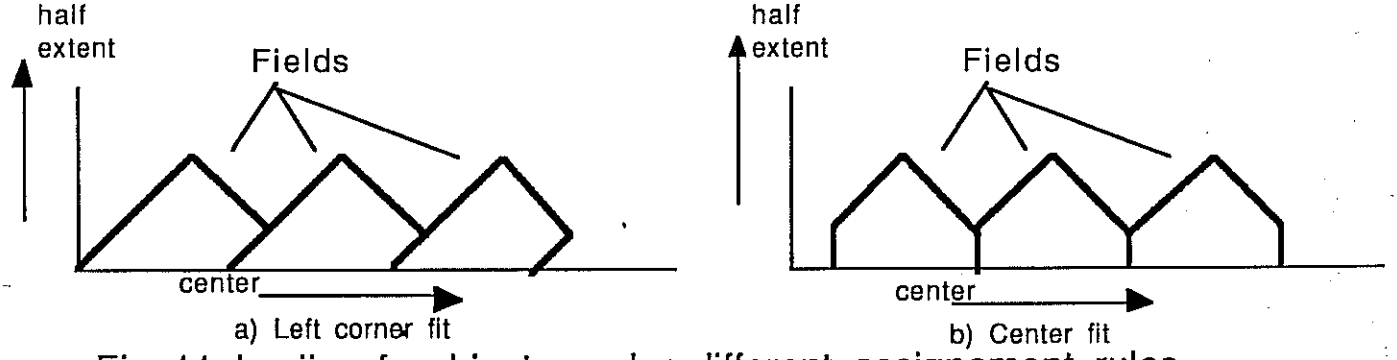


Fig 11 Locii of objects under different assignement rules

The coverage of fields under different assignment rules for the one dimensional case is shown in Fig. 11, using transformed (center, half-extent) space. As yet the authors have not found a motive for preferring among different rules.

5 Operations on Fieldtrees

Having considered the structure of the Fieldtree, its storage and retrieval algorithms will now be discussed. Afterwards, an overview of the different program modules will be presented.

5.1 Storage Algorithm

Each object is always stored into the smallest possible field, i.e., an existing field that fulfills the rules:

Fit: The object lies completely within the boundaries of the field

Importance: The field extension is large enough for the importance of the object. The importance is determined by the type of the object.

This leads to a (simplified) version of the algorithm:

1. Descend the tree while a smaller field exists that obeys both the fit and the importance rules.
2. Store the object into that smallest field.

The previous algorithm version will never make the tree grow; to do so, rules for creating new fields must be added:

- If the tree is empty, create the first and largest field; this field is the root of the tree.
- If the page on which to store the object has reached its capacity, reorganize the corresponding field

Reorganization is a mechanism intended at distributing the contents of one field that is overfilled over other (smaller) fields; if a field becomes saturated, one or more of its descendants must be created for the reception of all the data they can hold according to the fit and importance rules. Reorganization takes care both of the creation of a minimum of descendants and of the transference of a maximum of data; if no descendant can relieve a field of its excessive burden, an overflow page would be provided. Since overflow pages preserve the logical organization of a Fieldtree, a *lazy* reorganization can be included. That procedure always uses overflow pages and marks the fields as a candidates for a thorough off-line reorganization. Two courses of action can be taken during reorganization: either i) create all the descendant pages, or ii) follow a greedy strategy and create only the one descendant field that relieves the outfilled field most.

5.2 Retrieval

The recursive algorithm for retrieval has the following structure:

1. Initialize; start with the root of the Fieldtree.
2. Test a field (recursive)

While testing a field, if the field under consideration touches (or includes, is contained, etc.) the query window and obeys the importance rules then:

- The objects of the field must be included in the answer and
- All its existing subfields must be tested

6 Performance of the Fieldtree

This section presents in brief analytical results for the partition and the cover variants of the Fieldtree. A more detailed description of those results can be found in [Barrera 1989].

For each of those variants, two methods of implementing the hierarchy of the fieldtree are considered:

i) Including pointers inside the fields. This implementation will be referred to as the *pointer* one.
ii) Providing the fields with a Morton code, and using a B^+ tree to store pairs of (*field code*, *disk address*). This implementation will be referred to as the *multiway* one.

These results are compared to those obtained for the Rtree and the R^+tree by Faloutsos et al. [1987]. Similarly to those authors:

- Objects are supposed to reside in a one dimensional space, inside the interval $[0.0, 1.0]$.
- Both point and range queries are considered.
- The analysis considers objects belonging to either one or two populations. Objects within a population have a fixed size and are uniformily distributed through space.

The same terminology of [Faloutsos 1987] will be used, namely:

- C Maximum number of data items in a data leaf.
- n Total number of data items.
- σ Size of a data item.
- f Fan-out of the tree (R^+tree, Rtree or B^+tree, depending on the context).

6.1 Comparisons, one population case

This subsection presents analytical results for point location and range queries. The performance in both cases is measured by the number of page accesses needed.

Taken from [Faloutsos 1987], a case with a population of 100.000 objects, a fan-out factor f of 50 and a field capacity C of 50 objects was considered. All results are given for different values of the paremater $\sigma n/C$, that renders the fraction of the objects that overlap the edges of an R^+tree.

6.1.1 Point location Queries

Fig. 12 compares the performance of the partition and the cover Fieldtree to that of the R^+tree and the Rtree. Only the multiway implementation of the Fieldtree appears in this figure, since for this particular case is superior to that of the pointer implementation. Two extra cases are analyzed for the partition Fieldtree: one that passes the objects that overflow a field to the field's ancestor and one that utilizes overflow buckets.

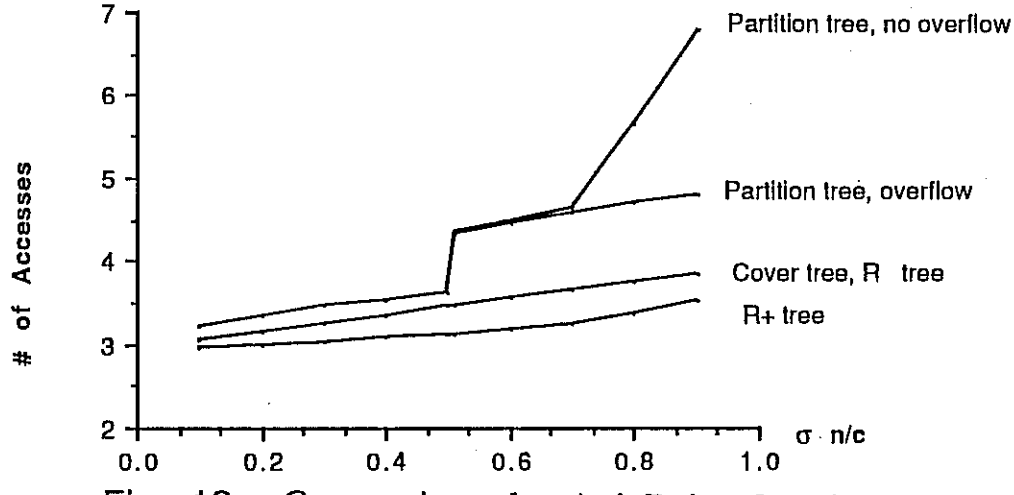


Fig 12 Comparison for 1-d Point Queries

From Fig. 12, it can be concluded that for the one-dimensional, one population case:

- The Cover Fieldtree and the Rtree perform identically.
- The Cover Fieldtree outperforms both variants of the partition Fieldtree. The variant of the partition Fieldtree that provides overflow records outperforms its alternate variant.

Further results in [Barrera 1989] show that, for point location queries, the superiority of the cover variant over the partition variant is preserved for the two-dimensional case.

6.1.2 Range Queries

Fig. 13 shows the average number of pages needed to access C' objects in a range query as a function of the size of the query window. Three sets of graphs are provided, corresponding to values of $\sigma n/C'$ of 0.1, 0.3 and 0.5. The parameter $\sigma n/C'$ is a measure of the overlap of objects: a value of 0 corresponds to no overlap; a value of 1 completely disrupts the behavior of a 1-dimensional R^+tree.

That figure shows that:

- For the one population Fieldtree case, implementation by multiway trees outperforms that of pointers interior to the fields.
- Rtrees and Cover Fieldtrees that use multiway indexes again behave identically. They outperform all other variants.
- R^+trees are outperformed by partition Fieldtrees for high values of $\sigma n/C'$

6.2 Results for two populations: Rtree vs. Cover Fieldtree

Faloutsos [1987] considers the case of a 1 dimensional Rtree with two populations of objects. The number and the extension of the objects in each population are called (σ_1, n_1), (σ_2, n_2). That reference also proves that in the case of two populations of objects one of them dominates, i.e., all formulas are the same as if they were from a single population with the dominant characteristics.

In the cover Fieldtree two situations can happen:

1. The importance of both types of objects is the same and, thus, their items are placed in the same level of the Fieldtree, and, as in [Faloutsos 1987], one of the populations dominates.
2. The importance of both types of objects is different and they are placed in the different levels of the Fieldtree.

Situation (2) is particularly attractive for range queries since then both populations become independent; no coupling at all occurs for the case of a pointer implementation of the hierarchy, and very little in the case of a an implementation by multiway trees. In this later case, the coupling can be greatly diminished if the positional keys of the objects are ordered "by levels", i.e., keys of fields a given level of the Fieldtree preceding all those corresponding to descendant levels.

As an example of that let us consider the following case: suppose that C' and f are the same as above, and the existance of two populations such that:

$$\begin{aligned} n_2 = 16 n_1 \quad &. \quad n_1 + n_2 = 10^5 \\ \sigma_1 = 16 * \sigma_2 \quad &. \quad \sigma_1 n_1 / C' = 0.2 \\ importance_1 \quad &> \quad importance_2 \end{aligned}$$

The results of that example are illustrated in Fig. 14. In it, a case where the cover Fieldtree shows its superiority for range queries is demonstrated; it also shows that the superiority of a multiway implementation over a pointer one becomes less marked for mixed populations. Examples can be found, specially in the two dimensional case, where that superiority disappears.

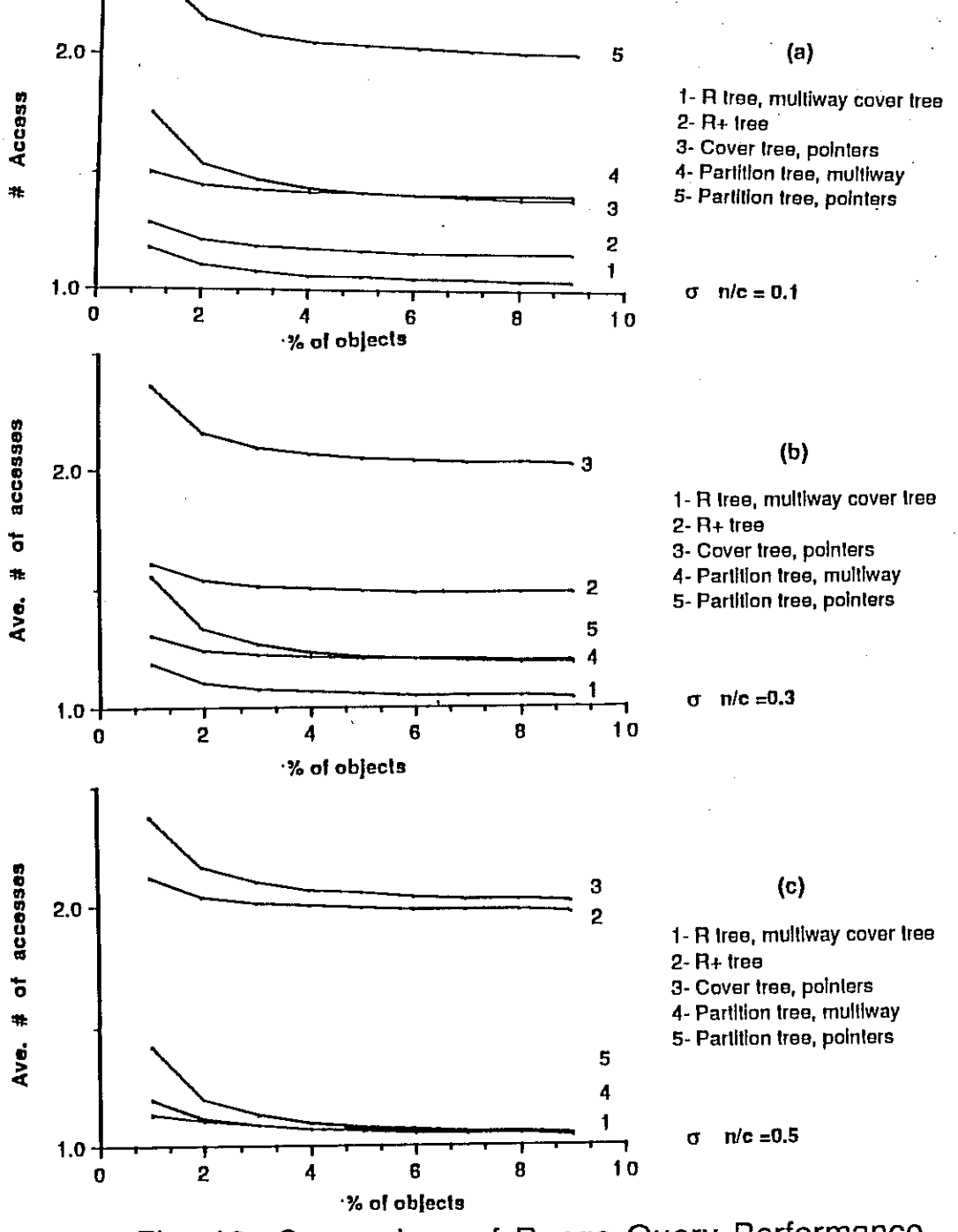


Fig 13 Comparison of Range Query Performance

7 Conclusions

The spatial access method presented here has the capability of clustering spatial objects according to their type; this feature is specially helpful in making the solution time of a query independant of the contents of the database, while still using a single spatial access mechanism. In opposition to the R^+tree its performance does not degrade indefinitely as the object overlap increases. It differs from the Rtree in the possibility of elliminating the *population dominance* effect. It is thus especially well suited for cases where data sets with many different types of spatial objects are stored, and queries refer to objects from a particular data collection.

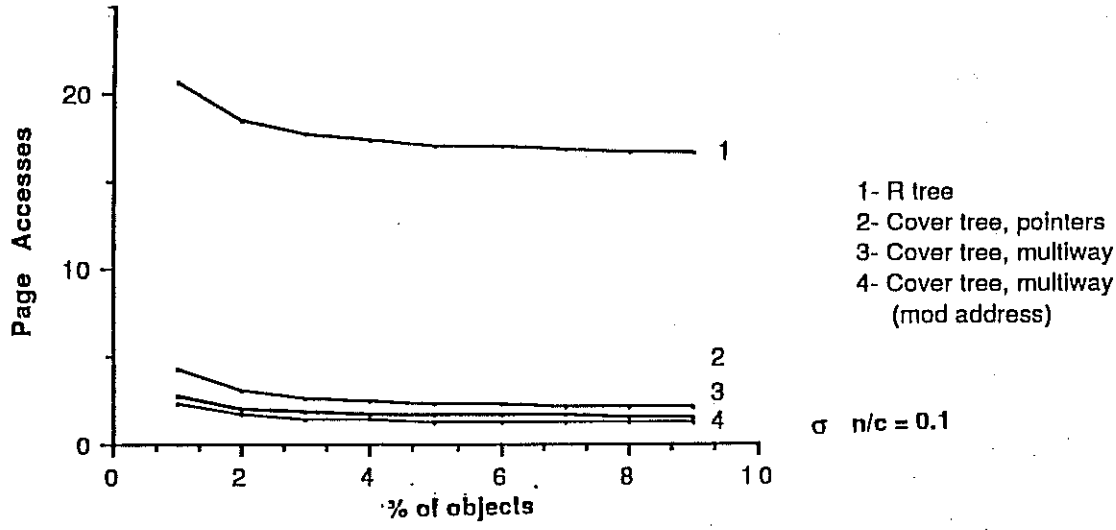


Fig 14 Average Number of Page Accesses, "C" objects

8 Acknowledgements

The authors wish to thank Max Egenhofer and Douglas Hudson for their discussions and their suggestions to improve this paper, and Jeff Jackson for his help in the preparation of the graphs.

References

[Abel 1984] D. Abel and J. Smith. A Data Structure and Query Algorithm Based on a Linear Key for a Database of Areal Entities. The Australian Computer Journal, 16(4), November 1984.

[Banerjee 1988] J. Banerjee et al. Clustering a DAG for CAD Databases. IEEE Transactions on Software Engineering, 14(11), November 1988.

[Barrera 1989] R. Barrera and A. Frank. Analysis and Comparison of the Performance of the Fieldtree. Technical Report, Department of Surveying Engineering, University of Maine, Orono, ME. March 1989.

[Burt 1981] P. Burt et al. Segmentation and Estimation of Image Region Properties through Cooperative Hierarchical Computation. IEEE Transactions on Systems, Man, and Cybernetics. 11(12). December 1981.

[Egenhofer 1988a] M. Egenhofer. A Spatial SQL Dialect. Technical Report, Department of Surveying Engineering, University of Maine, Orono, ME, September 1988. submitted for publication.

[Egenhofer 1988b] M. Egenhofer and A. Frank. Towards a Spatial Query Language: User Interface Considerations. In: D. DeWitt and F. Bancilhon, editors, 14th International Conference on Very Large Data Bases, Los Angeles, CA, August 1988.

[Faloutsos 1987] Faloutsos et al. Analysis of Object-Oriented Spatial Access Methods. In: Proceedings of SIGMOD Conference, May 1987.

[Frank 1982] A. Frank. PANDA—A Pascal Network Database System. In: G.W. Gorsline, editor, Proceedings of the Fifth Symposium on Small Systems, Colorado Springs, CO, 1982.

[Frank 1983] A. Frank. Problems of Realizing LIS: Storage Methods for Space Related Data: The Field Tree. Technical Report 71, Institut for Geodesy and Photogrammetry, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, 1983.

[Günther 1989] O. Günther. The Cell Tree: An Object-Oriented Index Structure for Geometric Databases. In: Proceedings IEEE Fifth International Conference on Data Engineering, Los Angeles, CA, February 1989.

[Guttman 1984] A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In: Proceedings of the Annual Meeting ACM SIGMOD, Boston, MA, 1984.

[Hinrichs 1985] K. Hinrichs. The GRid File System: Implementation and Case Studies of Applications (in German). PhD thesis, Swiss Federal Institute of Technology, Zurich, Switzerland, 1985.

[Kitsuregawa 1989] M. Kitsuregawa et al. Join Strategies on KD-tree Indexed Relations. In: Proceedings Fifth International Conference on Data Engineering, Los Angeles, CA, February 1989.

[Nievergelt 1984] J. Nievergelt et al. The GRID FILE: An Adaptable, Symmetric Multi-Key File Structure. ACM Transactions on Database Systems, 9(1), March 1984.

[Orenstein 1986] J. Orenstein. Spatial Query Processing in an Object-Oriented Database System. ACM-SIGMOD, International Conference on Management of Data, 15(2), 1986.

[Orenstein 1989] J. Orenstein. Redundancy in Spatial Databases. ACM-SIGMOD, International Conference on Management of Data, 1989.

[Samet 1984] H. Samet. The Quadtree and Related Hierarchical Data Structures. ACM Computing Surveys, 16(2), June 1984.

[Sellis 1987] T. Sellis et al. The R^+-Tree: A Dynamic Index for Multi-Dimensional Objects. In: P. Stocker and W. Kent, editors, 13th VLDB conference, Brighton, England, September 1987.