Direct Spatial Search on Pictorial Databases Using Packed R-trees

Nick Roussopoulos
and
Daniel Leifker

Department of Computer Science
University of Maryland
College Park, Maryland 20742

Abstract

Pictorial databases require efficient and direct spatial search based on the analog form of spatial objects and relationships instead of search based on some cumbersome alphanumeric encodings of the pictures R-trees (two-dimensional B-trees) are excellent devices for indexing spatial objects and relationships found on pictures Their most important feature is that they provide high level object oriented search rather than search based on the low level elements of spatial objects This paper presents an efficient initial packing technique for creating R-trees to index spatial objects Since pictorial databases are not update intensive but rather static, the benefits of this technique are very significant

1. Introduction

Pictorial databases have been introduced for more than a decade now Chang [Chang 1981] provides a survey of most of the attempts in this area The techniques used in the design, implementation and access languages of pictorial databases were influenced by the corresponding techniques in alphanumeric databases, but, many of the researchers discovered that these techniques had to be extended in several respects [Chang & Fu 1981], [Chang & Kunii 1981], [Lin & Chang 1980], [Tang 1980] Some other researchers felt that the capabilities of these approaches are not adequate because they are merely providing simple table look-ups of spatial facts and vector-based displays of digitized map data [McKeown 1983a,b] McKeown feels that more advanced query processing capabilities are necessary, including precomputation and utilization of spatial relationships, dynamic computation of spatial relationships from the pictures, and other specialized features which can be classified as expert routines for special purpose picture manipulation tasks

We believe that pictorial and alphanumeric databases must be integrated but the representation and processing of the two must be clearly distinguished First, pictures are not naturally representable in alphanumeric encodings and they should, therefore, be presented to the user in their analog form Furthermore, user queries searching for pictorial objects and spatial relationships among them must be **direct,** specified in terms of the analog form This allows the user to do **direct manipulations** on the pictorial database Alphanumeric data associated with pictures can be displayed on or beside the picture Second, although the manipulation language must combine powerful features for handling pictures and alphanumeric data, it should not embed pictorial access features and spatial operators into an alphanumeric data manipulation language Instead of forcing the pictorial syntax specification to fit a preexisting alphanumeric language, the user interface should naturally coordinate the query specification that addresses the pictorial part with the one that addresses the alphanumeric part But each part must have whatever syntax is natural for handling it, which implies that the pictorial syntax must be developed from the beginning with no compromises to fit existing alphanumeric query language Third, the processing of pictures which requires special purpose processors and tailored indexing techniques must be left outside the database system processor so that these special purpose processors can be replaced,

modified, or improved, according to the requirements and the sophistication of the software and hardware

The above three premises suggest a system architecture that can support from a very sophisticated, large and high resolution pictorial database to a very simple one using inexpensive graphics hardware found in today's micro-computers The alphanumeric data processor and the pictorial processor are different but they need to exchange control and data Figure 1 1 shows this architecture

From the user's point of view, the following is a list of requirements that an integrated database must satisfy

1 During the access of the database the user should be able to obtain displays which show the correspondence between the spatial objects on the picture and the data associated with it The picture contains these objects and the spatial relations among them Therefore, it would be very natural to selectively display the spatial objects satisfying the user's query along with other data associated with them

2 The database must support **direct spatial search** which locates the spatial objects in a given geographic area of the picture This accommodates queries of the form "Find all the bridges in a given area," where the area is specified on the picture by a graphics direct data entry device

3 The database must support **indirect spatial search** which locates objects based on some non-spatial attributes and use the associations between the spatial objects on the picture to place them on it This accommodates queries of the form "Display the city and its elevation if the population exceeds 2 million," where elevation is extracted from an elevation map

4 The database must support a more advanced user interface which allows for a direct graphics input specification (pointing devices such as a mouse, joystick, etc) and output display coordination between the pictorial and the alphanumeric data

Direct spatial search requires more advanced indexing techniques [Stonebraker et al 1983] because of the non-atomicity of spatial objects For two-dimensional spatial objects, R-trees [Guttman 1984] are excellent speed-up devices They can be thought of as two-dimensional B-trees [Bay & McCreight 1972], and, although they are similar in nature to Quad-trees [Finkel & Bentley 1984], they are more flexible and their dynamic nature can better deal with "dead-space" on the pictures The most important feature that distinguishes R-trees from Quad-trees is the fact that, at the leaf level, the former store full and non-atomic spatial objects whereas the latter may indiscriminately decompose the objects into lower level pictorial primitives such as quadrants, line segments, or, even pixels This feature provides a natural and high level **object oriented search** Similar search in Quad-trees requires an elaborate reconstruction process of the spatial objects from the low level primitives of the leaves Furthermore, because the storage organization of R-trees is based on B-trees, they are better in dealing with paging and disk I/O buffering [Guttman 1984]

In this paper we present a compaction technique for "packing" and reducing dead-space on R-trees We show that by carefully creating the index on the spatial objects, we obtain significant performance improvement during the search The initial construction of the index is not in a conflict with the dynamic nature of the R-trees which can then be updated in the usual way However, since most of the pictorial databases are relatively static, the benefits of this initial packing are very significant

The packed R-trees have been implemented in the context of PSQL [Roussopoulos & Leifker 1984], a query language for pictorial databases which supports direct and indirect spatial search A brief introduction of PSQL is given in Section 2 Section 3 deals with the compaction of R-trees and their use for direct spatial search Section 4 contains the conclusions

2. PSQL - A Query Language for Pictorial Databases

PSQL is a relational based language for retrieving information from a pictorial database It extends the power of SQL [Chamberlin et al 1976] for retrieving alphanumeric data by allowing direct spatial search The pictorial database maintains the associations between the spatial and alphanumeric objects This is necessary to support direct spatial and indirect alphanumeric search

2.1. Data Definition in PSQL

PSQL supports pictorial domains whose elements are objects found on pictures Examples of these objects include geographic points on a map,

highway segments, geographic regions, etc Each domain has its own pictorial representation and form Pictorial domains have their own comparison operators for comparing their elements For example, regions have operators, such as **covers, overlaps,** etc Pictorial domains also have functions defined on them which compute some simple or aggregate attribute A simple function for a region object is **area** which computes its area An aggregate function on a set of highway segments is **northest** which finds the northest coordinates of any point in a highway

A pictorial domain in PSQL can be thought of as an abstract data type The comparison operators and functions defined on a pictorial domain hide from the user the low level implementation details which deal with the alphanumeric encodings of the domain As can be seen from the above examples, functions defined on pictorial domains are very specific to the application and that any attempt to include all useful ones in PSQL or in any other language would be pointless Instead, the language must have capabilities for user-defined (application-defined) extensions that can be invoked from the pictorial language

Relations can be defined over alphanumeric and/or pictorial domains Every tuple models relationship among those alphanumeric and pictorial objects The relation columns that correspond to alphanumeric domains are indexed the usual way The columns of pictorial domains are indexed by the R-trees, see section 3, and thus each pictorial domain element that corresponds to a tuple of the relation appears on a leaf-node of the R-tree

PSQL implements these associations between alphanumeric and pictorial domains using a backward (unique) identifier of type pointer [Powell & Linton 1983], [Zaniolo 1983] which points to the area on the picture (to the leaf-node of the R-tree) These identifiers are computed when the relations are generated or updated The identifier's value (pointer-value) is used to select the relation's tuples in the forward direct search, i e , when it retrieves using the picture Note that a pictorial relation could be associated with more than one picture In this case, one identifier is required for each picture association of this relation This increases the complexity of the updates, but provides higher data sharability

The implementation of pictorial relations and their access is similar to ordinary relations with the only difference being that each pictorial relation has an extra column named "loc" of type pointer which stores pointers to the picture

cities(city,state,population,loc)
states(state, population-density,loc)
time-zones(zone,hour-diff,loc)
lakes(lake,area,volume,loc)
highways(hwy-name,hwy-section,loc)

Although the loc column takes values of type pointer, the user can use the column name to specify spatial relationships that the tuples must satisfy The type of a pictorial object may be of type "point," as in cities, or "line segment," as in highways, or "region," as in time-zones and lakes

2.2. The Retrieve Mappings

PSQL's extended mapping is of the form

select <attribute-target-list>
from <relation-list>
on <picture-list>
at <area-specification>
where <qualification>

When specified, the **on-at**-clause selects one area on an picture and uses it in narrowing down the retrieval scope of the relations appearing in the **from**-clause

The following example is a typical simple query in PSQL

select city,state,population,loc
from cities
on us-map
at loc **covered-by** {4±4,11±9}
where population>450,000

which selects all cities in the area {4±4,11±9} (Eastern US entered by coordinates or by a mouse) having population greater than 450,000 Figure 2 1a shows the alphanumeric result of the query and 2 1b the pictorial output displayed on a graphics monitor Note that the object names are displayed on the picture to assist the user to visualize their correspondence

The <picture-list> in the mapping is just a name list, and nothing but the standard string matching for identity is performed However, a geographic area on the pictures specified by the <area-specification> is a location specification which can be either a bound variable or a location given in absolute constant coordinates or in variable coordinates The location variable may just be a name of a location predefined outside the retrieve mapping Furthermore an area in the <area-specification> may be followed by the spatial operators **covering, covered-by, overlapping, disjoined,** etc , followed by another

location specification The meaning of "loc1 **covering (covered-by, overlapping, disjoined)** loc2" is that loc1 covers (is covered by, overlaps with, is disjoined with, etc) loc2 "

The spatial operators are comparison predicates which receive two area specifications each of which is either a constant or a pointer variable whose binding during the processing points to an area of the picture The operators return true or false depending on whether or not the two argument locations satisfy the corresponding spatial relation on the picture The spatial operators are very similar to those found in Zaniolo [Zaniolo 1983] and Stonebraker [Stonebraker et al 1984] and can be implemented by extending the standard retrieve and update capabilities of current data manipulation languages A large variety of tailored operators can be implemented to further enhance PSQL It is our belief that the power of the language stems directly from the integration of SQL with these spatial operators

A very powerful operation in PSQL is the **juxtaposition** (synthesis) of dissimilar information stored in multiple but yet referring to the same geographical area pictures The following example illustrates this powerful feature by synthesizing information found on two pictures, i e , information about cities associated with us-map and time-zones associated with a time-zone-map to obtain cities together with their time-zone

select city,zone
from cities,time-zones
on us-map,time-zone-map
at cities loc **covered-by** time-zones loc

Figure 2 2a and 2 2b show the two maps and Figure 2 2c the juxtaposition of two The alphanumeric data consists of the complete relations cities and time-zones displayed next to each other if the geographic area of one spatial object (city in this case) is covered by the geographic area of the other (time-zone) Juxtaposition is performed by simultaneous search on the two (or more) spatial organizations which correspond to the same area The entries can be juxtaposed if their associated locations satisfy the **at**-clause The simultaneous use of several spatial organizations is analogous to the use of two or more secondary indexes during the query processing where the intersection of the indices speeds up the search

Juxtaposition is a very powerful operator for pictorial databases It is somehow similar to the relational join operator For the join to be meaningful, the tuples of the two relations must refer to the same entity, (see [Kent 1979] and [Roussopoulos 1984]) For the juxtaposition, it is sufficient that the two operand pictures refer to the same geographic area which in this case plays the role of the entity ("geographic join")

PSQL mappings can have several nested levels by mapping from a deeper level to the next level The query below illustrates the location binding of two nested mappings The state location passed from the interior level is used to direct the search in the exterior one to produce those lakes in the Eastern states which are within (covered by) the boundary of some state

select lake,area,lakes loc
from lakes
on lake-map
at lakes loc **covered-by**

  select state loc
  from states
  on state-map
  at states loc **covered-by** {4±4,11±9}

The binding of the top level window is dynamically done during the evaluation of the query

PSQL queries are preprocessed and translated into ordinary SQL entries The only additional requirement from SQL is the capability of executing system defined procedures from within the **where**-clause This feature is used to call the spatial operators and functions during the execution of the query

The output of PSQL queries is directed to two output devices The graphical output device displays the area of the picture containing the qualifying spatial objects and the standard terminal displays the alphanumeric data This is very useful for indirect spatial search because it allows the user to simultaneously visualize the correspondence between data about spatial objects and their picture

2.3. Database Updates

Updates of pictorial relations may involve partial reorganization of the associated pictorial index This requires that an insertion or modification of a tuple should include spatial information for updating each of the spatial index associated with the updated relation The reorganization of the spatial index is discussed in the following section

3. R-trees: The Foundation for the Direct Search of PSQL

PSQL has its foundations on SQL and a data structure known as an R-tree Originally defined by Guttman [Guttman 1984], R-trees can be loosely described as higher-dimensional generalization of B-trees [Bayer & McCreight 1972] It is appropriate that R-trees should be used in the organization of spatial databases, since the data objects of interest can be accurately represented in a form analogous to their spatial nature This section will investigate several properties of R-trees in general, and describe ways in which spatial databases may be organized by means of R-trees

As defined by Guttman, leaf-nodes of the R-tree contain entries of the form

(I, tuple-identifier)

where "tuple-identifier" is a pointer to a data object (in PSQL, data objects are pointers to tuples within relations), and I is an n-dimensional minimal rectangle 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 its 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 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" can be used to specify the number of entries per node, 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 tuples (among the leaves) For illustrative purposes in this section, we shall restrict our attention to 2-dimensional R-trees with a branching factor of four Such a small branching factor of four would possibly be undesirable in a practical application However, a factor of four greatly facilitates the presentation of the concepts that follow, and extensions to higher branching factors (that fill a logical disk block) are readily apparent

When a spatial organization on a relation is defined, an R-tree is constructed the same way a B-tree is for indexing The only difference between the two is that the spatial relationships among the objects represented by the relation tuples may not necessarily be part of the values stored in the tuple itself and thus they must be provided externally (via another file created using a cursor, mouse, or other graphic input device) [This is not necessary for a B-tree because their organization is completely specified by the values found in the tuple] The externally provided spatial organization should not be thought of as a weakness but, on the contrary, it provides further flexibility to the spatial database For example, one may choose to include in the tuple all the information needed to infer their spatial relationships As for point spatial objects, the X,Y coordinates are adequate, minimal and may be useful to the user However, for more complex objects, such as regions with no canonical shapes and sizes, the encoding of their complete spatial specification may be of no interest to the user who accesses by "pointing" to them or by their name in order to retrieve information

In a spatial database it is convenient to classify data objects as "points," "segments," or "regions" Figure 3 1 shows a partial R-tree of the spatial relation of cities in the continental United States As far as the spatial organization of this example database is concerned, the spatial objects cities are viewed as points Figure 3 2, on the other hand, shows how U S states are arranged as regions using R-trees Since the leaf nodes of an R-tree contain pointers to tuples and not the actual tuples themselves, points and regions may be freely intermixed within any R-tree

Using PASCAL-like syntax, leaf and non-leaf nodes of an R-tree with a branching factor of four can be easily defined

```
type ENTRY = record
       X1,X2,Y1,Y2   integer,
       POINTER   integer,
     end,
   NODE = record
        CLASS   (leaf, non_leaf),
        DESC    array [1  4] of ENTRY,
        VALID   integer,
     end,
```

The value of the CLASS field identifies the node as leaf or non-leaf The rectangle bounded by the lines x=X1, x=X2, y=Y1, and y=Y2 is minimal and bounds all the data objects in all descendent nodes (not only immediate successors) Pointers to those descendent objects are contained in POINTER and are interpreted as pointers to other R-tree nodes if CLASS is "non_leaf" and to database tuples if CLASS is "leaf" Since not all DESC pointers may be active for any given node, the integer VALID records the number of valid pointers By convention, elements of the DESC

array are allocated as a stack with VALID serving as the stack top The entire R-tree structure can be declared as

var RTREE **array** [1 MaxNodes] **of** NODE, where RTREE[1] is arbitrarily chosen as the root

3.1. Direct Spatial Search

R-trees offer an enormous potential for search pruning, so many types of spatial queries can be processed with unusual elegance and efficiency As an example, we present an adaptation of Guttman's recursive search algorithm for answering queries of the type "List all points and regions within target window <target-window>"

```
Procedure SEARCH(N integer),
        var K integer,
begin
if RTREE[n] CLASS = non_leaf then
 begin {recursively search feasible descendents}
  for k =1 to RTREE[n] VALID do
   if INTERSECTS(RTREE[n] DESC[k])
    then SEARCH(RTREE[n] DESC[k] POINTER)
  end
else
 begin {node N is a leaf}
  for k =1 to RTREE[n] VALID do
  if WITHIN (RTREE[n] DESC[k]) then
  DISPLAY_TUPLE(RTREE[n] DESC[k] POINTER)
 end,
end,
```

Presumably the target window is stored in a global data structure, the Boolean function INTERSECTS returns TRUE if its argument entry intersects the target window and FALSE otherwise Similarly, the function WITHIN returns TRUE if the argument entry is contained within the target window and FALSE otherwise The statement SEARCH(1) will invoke SEARCH on the root node of the R-tree and cause all data objects within the target window to be displayed

For any set of points {P1, P2, , Pn} where each Pi is a pair (xi, yi), define the "minimal bounding rectangle" or MBR, written (P1, P2, , Pn) as the rectangle bounded by the lines

$$x = \min_{1 \le i \le n} \{x_i\} \quad y = \min_{1 \le i \le n} \{y_i\}$$
$$x = \max_{1 \le i \le n} \{x_i\} \quad y = \max_{1 \le i \le n} \{y_i\}$$

A minimal bounding rectangle for a set of regions can be defined analogously It is clear that the overall organization and density of the R-tree greatly influences the efficiency of the search Figure 3 3 shows the minimal bounding rectangles associated with the first-level entries of the root node of an R-tree Answering the query "List all cities within region W" may require substantially more searching than is tolerable, because region W intersects all the root entries and the search cannot yet be pruned If this overlap phenomenon occurs with any regularity, the advantages of R-tree organization can be greatly diminished or even lost

While considering the performance of R-tree searching, therefore, we introduce the informal concepts of "coverage" and "overlap" "Coverage" is defined as the total area of all the MBRs 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

3.2. Theoretical Issues in Organizing R-trees

We now return to Guttman's original paper [Guttman 1984] and examine the algorithms used to construct an R-tree Since Guttman wishes R-trees to be dynamic structures (such as B-trees) that need no periodic reorganization, he requires

(1) Every node except the root must be "m-filled" That is, each node must contain between m and M entries where M is the maximum number of entries that will fit into one node, and $m \le M/2$ is a parameter specifying the minimum number of entries per node

(2) New data objects to be inserted must be added to pre-existing R-tree leaves Insertion of the very first object is the only exception

Guttman's INSERT algorithm for inserting new objects is designed to create as little additional coverage as possible, but requirement (2) can sometimes cause excessive amounts of "dead space" that slows down R-tree searching To see this, consider the eight initial points in Figure 3 4a Clearly, the points could be grouped as shown in Figure 3 4b to yield two tightly-packed leaf nodes with as little coverage as possible However, if the points are inserted using INSERT, as in Figure 3 4c, requirement (2) will cause three

leaf nodes to be created with much useless space in the middle This is a simplistic example, but it does demonstrate a situation that should be avoided whenever possible

One might very reasonably ask, then, whether INSERT is appropriate for constructing the initial database Specifically, if we are dealing with spatial databases that remain relatively static over time (such as large chartographic databases), can we not design a pre-processing algorithm that will pack the initial set of data objects as tightly as possible, minimizing coverage and overlap, and thus allow for very efficient searching?

Before attempting to design an algorithm that will pack data objects with zero overlap, it would be helpful to know whether zero overlap can always be achieved for any arbitrary set of data objects We first examine the case of point objects in an R-tree with branching factor four

For any finite set of points
$S=\{P1,P2, \ ,Pn\}=\{(x1,y1),(x2,y2), \ (xn,yn)\}$
define the function of F as

$$F(S) = |\{x\ (x,y) \in S\}|$$

where the vertical bars denote cardinality That is, F(S) is the total number of distinct x-coordinates among all the points in S Now define the function $F\alpha(S)$ as

$F\alpha(S)$=F(S rotated counter clockwise through the origin by the angle α)

Clearly, $F2\pi(S)=F0(S)=F(S)$ for all S In general, however, $F\alpha(S)$ will vary as α ranges from 0 to 2π We now ask, for any S, whether an α can be found such that $F\alpha(S) = |S|$

LEMMA 3.1: For any finite set of points S in the plane, there exists an angle α such that $F\alpha(S)=|S|$

PROOF: Consider all the angles α such that $F\alpha(S)\neq S$ Each pair of points (x_i,y_i), (x_j,y_j), $i\neq j$, in S uniquely determines the line

$$y-y_i = \frac{y_j-y_i}{x_j-x_i}(x-x_i)$$

if $x_i\neq x_j$, and the line $x=x_i$ if $x_i=x_j$ If $x_i\neq x_j$, then a tedious calculation shows that there exists an angle α such that a rotation of S by α will cause the rotated points (x_i,y_i), (x_j,y_j) to have identical x-coordinates (see Figure 3 5) Since $F\alpha(S)$ is now by definition at least one less than $|S|$, $F\alpha(S)\neq|S|$ Since S is finite, there are at most $|S|/2$ angles α such that $F\alpha(S)\neq|S|$ (possibly fewer if the points in S are arranged with any degree of collinearity) However, there are an infinite number of angles α by which S may be rotated, hence it follows that there are an infinite number of angles α such that $F\alpha(S) = |S|$

The zero overlap theorem can now be stated and proved The proof is given only for an R-tree with branching factor of four, but other cases are proved similarly

THEOREM 3.2: Given any finite set of points S in the plane, there exists $\lceil |S|/4 \rceil$ minimal bounding rectangles (MBRs) such that each MBR contains not more than four points and yet all the MBRs are disjoint

PROOF: Without loss of generality we may assume that |S| is an integer multiple of four Rotate S about the origin until each point has a distinct x-coordinate By Lemma 3 1, such a rotation is guaranteed to exist Arrange the points by increasing x-coordinates {P1, P2, , Pn} and set

MBR1=(P1,P2,P3,P4),
MBR2=(P5,P6,P7,P8),
,
MBRn/4=(Pn-3,Pn-2,Pn-1,Pn)

By definition, MBR1 is bounded on the right by a point that is strictly less than any point in any MBRj for $j>1$, also, MBR1 is bounded on the left by a point that is strictly greater than any point in any MBRk for $k<1$ Hence the intersection of MBR1 and MBRj is zero for all $i\neq j$

Unfortunately, the zero overlap theorem fails when the data objects have positive area Once again, we prove this for the case of branching factor four

THEOREM 3.3: For any finite set of disjoint regions in the plane, there does not always exist a set of minimal bounding rectangles (MBRs) such that

(1) Each region is contained wholly within exactly one MBR, and

(2) Each MBR bounds more than one region, but not more than four regions, and

(3) The intersection of all the MBRs has zero area

PROOF: By counterexample Assume that conditions (1), (2), and (3) of Theorem 3 3 hold for all sets of disjoint regions, in particular they then must hold for the set of skewed rectangular regions shown in Figure 3 6 By condition (1), region R0 must be contained within one MBR, call it MBR0 By condition (2), MBR0 must also bound at least one other region but no more than three other regions An exhaustive enumeration

shows that any other regions we select for inclusion in MBR0 will necessarily include parts of other unwanted regions This contradicts condition (3) and proves Theorem 3 3

Although it has been shown that zero overlap can be achieved for any set of points, there are several legitimate objections that can be raised with these results as applied to the practical construction of R-trees

(1) Attaining zero overlap may require rotating the orientation of the entire database, and this may not always be convenient or even possible

(2) Theorem 3 2 assumed ideal conditions on a continuous plane and may not always hold in a digitized database

(3) Zero overlap may be attained only at the leaf level of the R-tree The next level contains MBR's of the leaf MBR's, and therefore represents an organization of region data objects Even though these leaf MBR's may have zero overlap, Theorem 3 3 shows that zero overlap for regions (i e , the next higher level of the R-tree) is sometimes impossible

(4) Some region data objects have inherent overlap, such as counties within states

(5) Although overlap may be minimized, the issue of coverage remains Figure 3 7a shows a set of leaf R-tree nodes Although there is zero overlap, the coverage is unacceptably high A more reasonable grouping is shown in Figure 3 7b The simultaneous minimization of both coverage and overlap is a complex task

3.3. The Packing Algorithm

These problems are addressed by means of algorithm PACK, which attempts to minimize both coverage and overlap PACK takes as input a set of data objects and produces as output a near-optimally packed R-tree, but requires no special rotation or orientation of the database frame of reference

Since we assume that the spatial database will remain relatively static (maps of geographical regions, for example, do not require frequent insertions or updates), we strengthen Guttman's requirement that R-tree nodes be at least m-filled by stipulating that all nodes are to be packed as fully as possible For convenience, we again assume that the node branching factor is four, and also that the total data objects at any R-tree level is an integral multiple of four This would be highly unlikely in any real application, but the "multiple of four" assumption allows us to dispense with the trivial special cases of one partially-filled node for leftover entries per level

Algorithm PACK can be written as a recursive function, its sole argument is DLIST, a list of data objects to be packed NN is a nearest neighbor function which takes two arguments NN(DLIST,I) return the item in the list DLIST which is spatially closest to item I and has the additional effect of deleting that item from DLIST A very high level description of the algorithm follows

```
Recursive Function PACK (DLIST) ↑node,

  {Returns a pointer to the root node
   of a fully-packed R-tree containing
     all the data items in DLIST }

begin
 if DLIST contains four data objects
  then begin
    Allocate a pointer to a new R-tree node, N0,
    Cause pointers of N0↑ to point to items of DLIST,
    RETURN (N0),
    end
  else
   begin
    Order objects of DLIST by some spatial crit-
        erion, {e g ascending x-coordinate}
    NLIST =(), {initialize as the empty list}
    while DLIST is not empty do
    begin
      I1 =first object from DLIST,
      DLIST =tail(DLIST), {delete 1st object}
      I2 =NN(DLIST,I1),
      I3 =NN(DLIST,I1),
      I4 =NN(DLIST,I1),
      Allocate a new R-tree node, N1,
      Cause pointers of N1 to point to I1,I2,
                              I3 and I4,
      LIST =append(NLIST,N1),{add new node}
     end,
    RETURN(PACK(NLIST)),
   end
end
```

A simple example should illustrate the principle by which PACK operates Figure 3 8a shows a set of points representing cities on a map of the United States In the first call of PACK, DLIST is the entire list of cities stored as coordinate pairs (perhaps latitude and longitude) Since there are more than four such pairs, the ELSE-clause is executed, and the cities are grouped by nearest

neighbor (Figure 3 8b) PACK is then called recursively using the list of leaf MBR's as data objects (Figure 3 8c), and this process continues, working ever backwards, until the root is finally reached and created As defined here, PACK refuses to make any distinctions between leaf and non-leaf nodes, although such a distinction is critical and must be made in any practical implementation It should be noted that it may be preferable to select the 4 items simultaneously from DLIST such that the area of the resulting associated MBR is minimized, but this could be combinatorially explosive

It is beyond the scope of this paper to prove any abstract formal properties of PACK, but empirical results have repeatedly demonstrated that the algorithm constructs very tightly-packed R-trees which readily lend themselves to efficient searching

3.4. The Update Problem

The basic assumption of PACK is that databases that are created for the first time must be efficiently organized Another assumption is that the database will remain relatively static However, the database need not be absolutely static the INSERT and DELETE algorithms given by Guttman can still be used to insert and delete data objects Indeed, it is intuitively appealing to suppose that INSERT and DELETE will perform well on a PACKed R-tree INSERT, for example, inserts a new data object into the leaf that requires the least enlargement, and, if that leaf is already filled, propagates node splits upward toward the root Such splits, of course, would be inevitable with the first few insertions, since the packed nodes are already filled However, R-trees created by PACK are presumed to exist in at least a tentatively final state INSERT may thus select from a large number of leaves so that the "least enlargement" is minimized Hence, INSERT (and analogously DELETE) and PACK can complement each other, and such an combination can be used effectively in the creation and maintenance of dynamic R-trees

3.5. Empirical Results

We ran some experiments comparing Guttman's INSERT and PACK The experiments were performed in a straightforward fashion The parameter J, specifying the number of data objects, was allowed to range over selected values from 10 to 900 Data objects were points having coordinates (x,y), ($0 \leq x \leq 1000$, $0 \leq y \leq 1000$), and were randomly generated with a uniform distribution in the plane Each algorithm used the same set of points for equal values of J to construct an R-tree For each algorithm and each value of J, we measured and recorded the coverage (C) and overlap (O) of the constructed R-tree, the total nodes within the R-tree (N), the depth (D) of the R-tree, and the average number (A) of nodes visited during 1000 random search queries The search queries were of the simple form

"Is point (x1,y1) contained in the database?"

and again were identical for both algorithms for equal values of J The results of these experiments are shown in Table 1

As the results show, packing the data objects can result in significant savings in space and search time

4. Conclusions

This paper presented a packing technique for R-trees which significantly improves direct spatial search on pictorial databases This is achieved by minimizing coverage and overlap of the leaf-nodes of the R-trees

The formal the theoretical properties of the PACK and the search algorithms will be reported in a forthcoming report The search of the spatial operators and functions employed by PSQL are currently being implemented We are currently investigating the possibility of dynamic invocation of the PACK algorithm during insertions and deletions to efficiently perform a "local" reorganization This will achieve the search performance obtained by the PACK algorithm for dynamically reorganized R-trees

5. References

[Bayer & McCreight 1972]
Bayer, R , and McCreight, E M , "Organization and Maintenance of Large Ordered Indices," *Acta Informatica*, Vol 1, No 3, 1972, pp 173-189

[Chamberlin et al 1976]
Chamberlin, D D , "SEQUEL 2 A Unified Approach to Data Definition, Manipulation, and Control," *IBM J Research and Development*, Vol 20, No 6, 1976, pp 560-575

[Chang 1981]
Chang, S K , "Pictorial Information Systems Guest Editor's Introduction," *IEEE Computer*, Vol 14, No 11, November 1981

[Chang & Fu 1981]
Chang, N S , and Fu, K S , "Picture Query Languages for Pictorial Data-Base Systems," *IEEE Computer*, Vol 14, No 11, November 1981

[Chang & Kunii 1981]
Chang, S K , and Kunii, L K , "Pictorial Database Systems," *IEEE Computer*, Vol 14, No 11, November 1981

[Finkel & Bentley 1984]
Finkel, R A , Bentley, J L , "Quad-Trees - A Data Structure for Retrieval on Composite Keys," *Acta Informatica*, Vol 4, 1984, pp 1-9

[Guttman 1984]
Guttman, A , "R-TREES A Dynamic Index Structure for Spatial Searching," *Proc of ACM SIGMOD Conference on Management of Data*, Boston, June, 1984

[Kent 1979]
Kent, W , "Limitations of Record-Based Information Models," *ACM Transactions on Database Systems*, Vol 4, No 1, 1979, pp 107-131

[Lin & Chang 1980]
Lin, B S , and Chang, S K , "GRAIN - A Pictorial Database Interface," *IEEE Proc of the Workshop on Picture Data Description and Management*, Asilomar, California, August 1980

[McKeown 1983a]
McKeown, D M , Jr , "Concept Maps," Dept of Computer Science, Carnegie-Mellon University, Technical Report CMU-CS-83-117, April 27, 1983

[McKeown 1983b]
McKeown, D M , Jr , "MAPS The Organization of a Spatial Database System Using Imagery, Terrain, and Map Data," Dept of Computer Science, Carnegie-Mellon University, Technical Report CMU-CS-83-136, July 17, 1983

[Powell & Linton]
Powell, M and Linton, M , "Database Support for Programming Environments," *Proc Engineering Design Applications of ACM SIGMOD Database Week*, San Jose, May 23-26, 1983

[Roussopoulos 1984]
Roussopoulos, N , "Intensional Semantics of the Relational Model," Dept of Computer Science, University of Maryland, January 1984

[Roussopoulos & Leifker 1984]
Roussopoulos, N , Leifker, D , "An Introduction to PSQL A Pictorial Structured Query Language," *IEEE Workshop on Visual Languages*, Hiroshima, Japan, December 6-8, 1984

[Stonebraker et al 1983]
Stonebraker, M , Rubenstein, J , Guttman, A , "Application of Abstract Data Types and Abstract Indices," *Engineering Design & Applications, Database Week, ACM SIGMOD*, San Jose, May 23-26, 1983

[Stonebraker et al 1984]
Stonebraker, M , Anderson, E , Hanson, E , "QUEL as a Data Type," *Proc of ACM SIGMOD Conference on Management of Data*, Boston, June, 1984

[Tang 1980]
Tang, G Y , "A Logical Data Organization for the I database of Pictures an Alphanumerical Data," *IEEE Proc of the Workshop on Picture Data Description and Management*, Asilomar, California, August, 1980

[Zaniolo 1983]
Zaniolo, C , "The Database Language GEM," *Proc ACM-SIGMOD Conference on Management of Data*, San Jose, May 23-26, 1983

	GUTTMAN'S INSERT					PACK ALGORITHM				
J	C	O	D	N	A	C	O	D	N	A
10	68483	43731	1	4	2 217	39590	0	1	3	1 424
25	74577	124311	2	12	4 800	31230	144	2	9	2 249
50	70718	177809	3	28	7 775	37421	1295	2	16	2 282
75	74561	229949	3	39	9 379	36152	1329	3	26	3 431
100	75234	235079	4	60	12 955	38271	994	3	35	3 645
125	77578	246084	4	73	14 024	36476	1318	3	42	3 658
150	77342	255692	4	86	14 894	40145	2729	3	51	3 784
175	79869	255523	4	103	16 277	36432	2532	3	58	3 820
200	80034	295091	4	117	17 870	33959	1394	3	68	3 873
250	79117	293730	4	142	18 585	40069	1946	3	83	3 897
300	78891	376731	4	167	20 838	38438	1527	4	102	5 397
400	82116	553650	5	233	28 935	37558	965	4	135	5 418
500	85290	698248	5	302	36 132	39820	1688	4	168	5 466
600	85253	749874	5	368	40 799	39542	2106	4	202	5 276
700	86225	852205	5	438	45 924	37016	1252	4	234	5 604
800	87418	1002339	6	507	55 462	38614	1522	4	268	5 730
900	87640	1164809	6	573	63 595	38808	1512	4	302	6 071

Code J = Number of Spatial Data Objects
C = Coverage of R-tree
O = Overlap of R-tree
D = Depth of R-tree
N = Total number of nodes in R-tree
A = Average number of nodes visited

Table 1 Experimental results over 100 random search queries

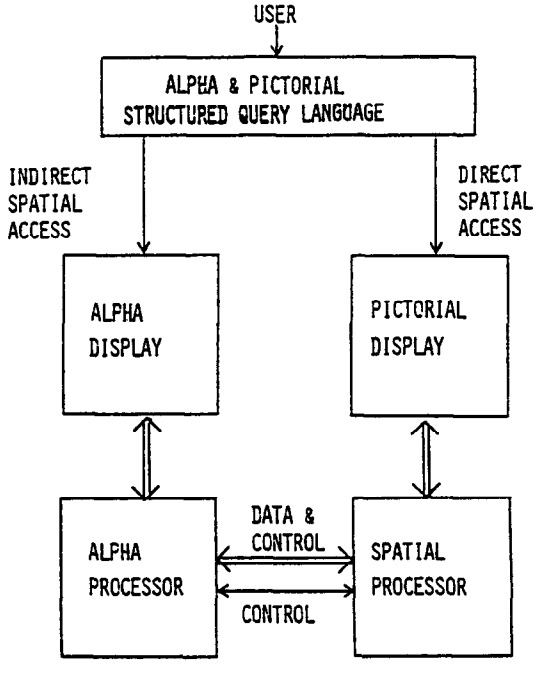

FIGURE 1.1.

char	char	real
city	state	population
ct	st	pop
JACKSONVILLE	FLORIDA	540,920
NEW ORLEANS	LOUISIANA	557,515
ST LOUIS	MISOURI	453,085
MEMPHIS	TENNESSEE	646,356
NASHVILLE	TENNESSEE	455,651

Figure 2 1a

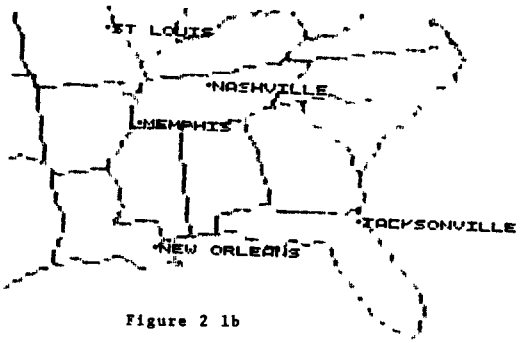

Figure 2 1b

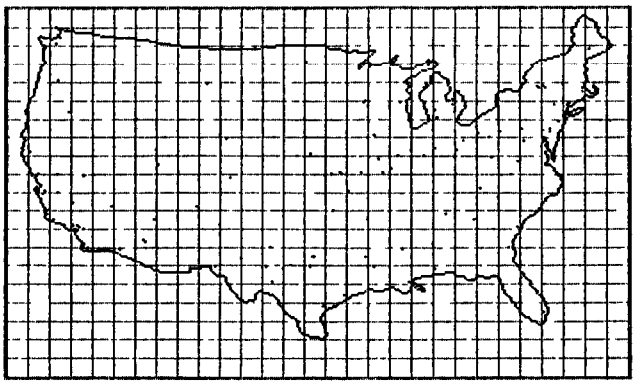

FIGURE 2.2A

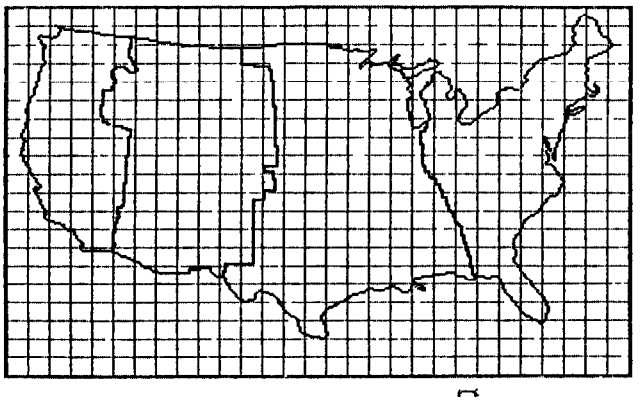

FIGURE 2.2B

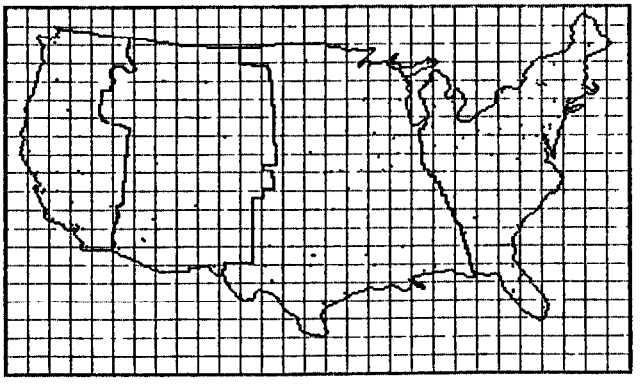

FIGURE 2.2C

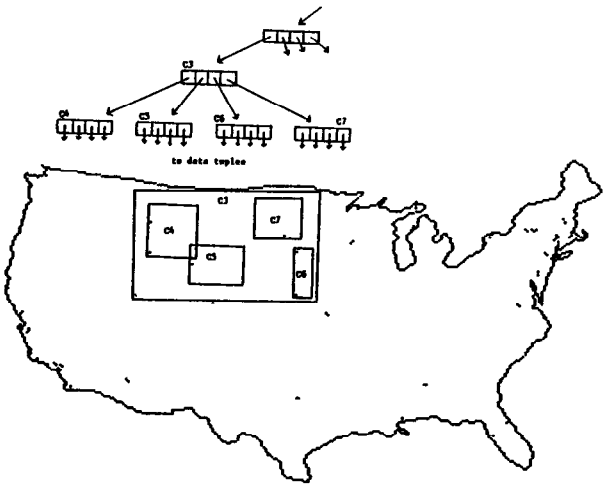

FIGURE 3.1

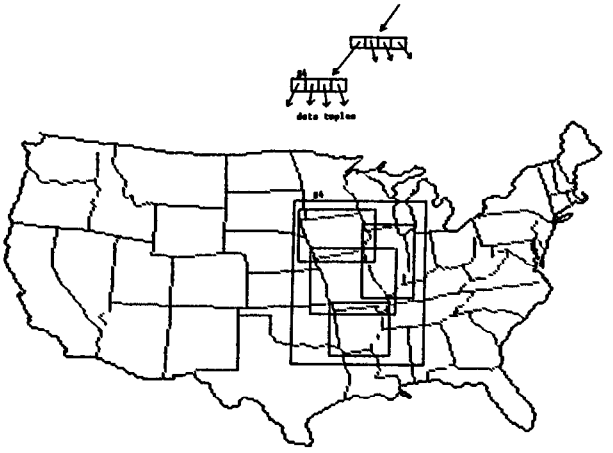

FIGURE 3.2

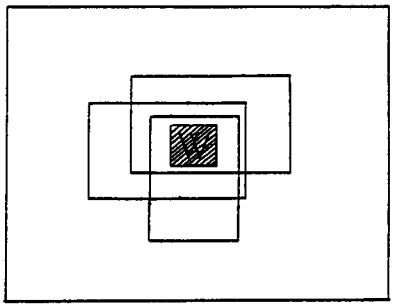

FIGURE 3 3

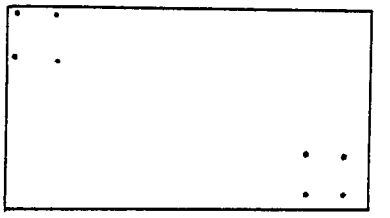

FIGURE 3 4a

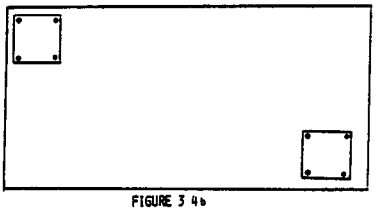

FIGURE 3 4b

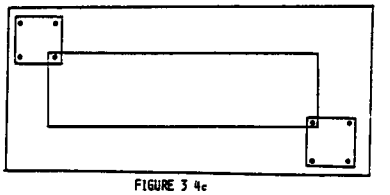

FIGURE 3 4c

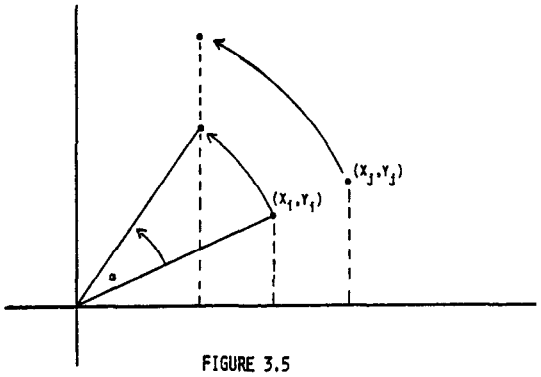

FIGURE 3.5

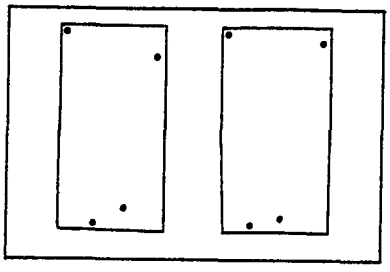

FIGURE 3.7A

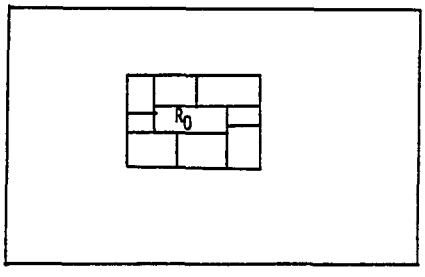

FIGURE 3.6

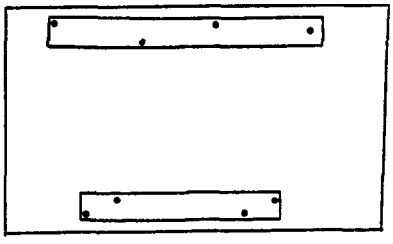

FIGURE 3 7B

FIGURE 3.8A

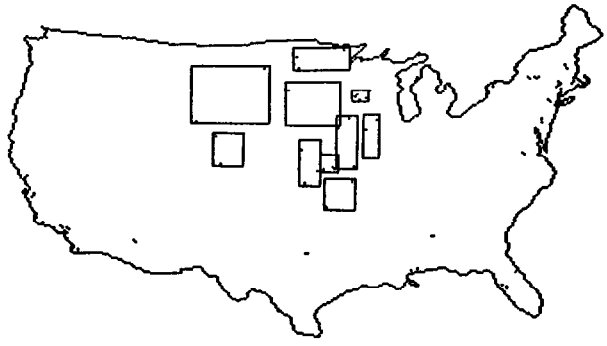

FIGURE 3.8B

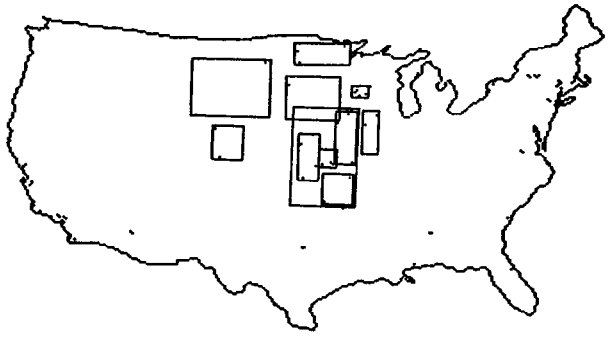

FIGURE 3.8C