Ladex: A New Index Mechanism in Spatial Object Database System

Xiao Weiqi Feng Yucai
Department of Computer Science, Huazhong University of Science & Technology, Wuhan, Hubei Province, People's Republic of China.
Post code: 430074 Fax: +86 27 7801738

Commission Ⅲ, Intercommission Working Group Ⅲ/Ⅳ

KEY WORDS: Spatial, Object, Database, Spatial Object, Database System, Spatial Index.

ABSTRACT

We propose a new index mechanism called Ladex(Lattice index) that supports all kinds of spatial object retrieval efficiently. Ladex organizes spatial objects with respect to their position and extension in the data space, it is an innovative spatial index structure that can be utilized in rapidly retrieving objects in a region with arbitrary shape or on certain location. In a sense, Ladex outperforms R-tree and its variants, it has a lot of extra advantages of which the other index structures lack. Ladex is a promising index structure for the future. In this paper, the structure of Ladex is described in detail. The search, insertion, deletion and modification algorithms of Ladex are presented, some implementation issues are also discussed in detail. Finally, the performance analyses and experimental results of Ladex are provided.

1 Introduction

Modern database management systems have been widely used in many applications, including: Geographic Information System (GIS) [Monehouse, 1992; Medeiros, 1994]; Cartography [White, 1981]; Computer-Aided Design (CAD) [Ousterhout, 1984]; City planning [Monehouse, 1992; Medeiros, 1994]; Real estate managing[Medeiros, 1994], etc. In the aforementioned applications, there exist a large quantity of graphic/image entities, each of them has its own geometric form and size, that is to say, every one of them has position and extension in data space. In this paper, we call these entities spatial objects. A lot of spatial queries may arise in the above applications. The best way improving spatial search/retrieval speed is to construct an efficient index mechanism suitable for spatial operations. In the past decade several spatial access approaches have been proposed. A resent survey can be found in[Lu, 1993]. Quad-tree[Aref, 1991], Grid-file[Hinrichs, 1983] and R-tree [Guttman, 1984] are typical spatial index structures. The structure of quad-tree is simple, it lacks flexibility in partition operation on itself [Aref, 1991], however it suits the needs of describing the single object's extension in space, though it does not support the search of objects in a specific area. The Grid-file structure is originally established for point query [Hinrichs, 1983]. R-tree [Guttman, 1984] and its variants (R^+-tree [Faloutsos, 1987], R^*-tree [Bechmann, 1990] and Hilbert R-tree [Ibrahim, 1994], etc.) support spatial range query, they are based on multidimensional non-linear index techniques. However, R-tree-based index mechanisms have a lot of drawbacks. First, as the structure of R-tree-based index modified dynamically, the performance would decrease significantly, and reorganization of the index structure is indispensable. Second, R-tree-based index structures do not adapt to irregular range query. Another disadvantage is that they support no control of query precision level.

The main idea in this paper is to present an innovative index method called Ladex (stands for lattice index), which supports spatial query efficiently, especially irregular range query at various precision levels. The remainder of the paper is organized as follows. Section 2 gives the structure of Ladex. Section 3 describes fundamental operations and algorithms for Ladex. Section 4 presents performance analyses and experimental results. Section 5 discusses the advantages of Ladex in detail. Section 6 gives the conclusions and directions for future research.

2 Structure of Ladex

2.1 Basic Concepts

Ahead of description of the structure of Ladex, some prerequisite concepts should be introduced.

• Spatial Index (SpIdx): This is a carefully managed data structure with the position and extension features of spatial obje ts, and with object identifiers and information used to locate the underlying objects in database.

• Spatial Search/Query/Retrieval: This is a kind of operation on spatial database, which return the objects that satisfy a given query condition QC. Usually, this operation is supported by Spatial Index (SpIdx) . A Spatial search operation can be represent as OP(DB, QC, SpIdx) , where DB is a spatial object database.

• Point Search: This type of operation is to search the objects that locate at or are near to a given point in a specific data space[Ibrahim, 1994]. This type of operation is expressed by PointQ (DB, PointC, SpIdx).

• Line Search: This type of operation is to search the objects that locate at or are near to a given line (such as straight line, polyline or even curve) in a specific data space. This can be described as LineQ (DB, LineC, SpIdx).

• Region Search: This type of operation is to search the objects that fall into or intersect with a given area in a specific data space[Ibrahim, 1994], it can be denoted as RegionQ (DB, RegionC, SpIdx).

2. 2 Description of Ladex

The Ladex index mechanism we present here can support point search, line search and region search. The main idea for Ladex is to partition a map into a lot of small blocks with equivalent size, each small block can be used as a bucket, and every object on a certain small block is stored in the corresponding bucket. In order to reach higher precision, each small block could be subdivided further, this procedure could proceed until the small block can not be partitioned any more. Different partition levels correspond to different precision levels. First level small block corresponds to first level precision, and second to second, and so on. Figure 1 is a map of MxN first level small blocks. Suppose the origin of the coordinates is at the bottom-left corner, a first level small block could be represented as Block[k, j], $0 \leqslant k < N$, $0 \leqslant j < M$. A lattice of MxN first level blocks on a map has MxN buckets, the i-th bucket is represented as BK [i], the relationship between a bucket and a block is described as:

$$BK[i] \leftrightarrow Block[i \text{ div } M, i \text{ mod } M]$$

where $0 \leqslant i < MxN$, '$\leftrightarrow$' is a symbol of one to one relationship. Figure 2 demonstrates the case that a point-shaped object, a line-shaped object and a region-shaped object take spaces on a lattice structure. Suppose the set of geographic objects in bucket BK[i] is S_BK[i]($0 \leqslant i < MxN$). Ladex has following characteristics:

(1) For any point-shaped object P_Obj_x, if it is in bucket BK[i], then we have $P\_Obj_x \in S\_BK[i]$;

(2) For any line-shaped object L_Obj_x, suppose the set of buckets occupied by L_Obj_x is $\{k_1, k_2, \ldots, k_m\}$, if $\forall\ i \in \{k_1, k_2, \ldots, k_m\}$, then we have $L\_Obj_x \in S\_BK[i]$;

(3) For any region-shaped object R_Obj_x, suppose the set of buckets overlapped by R_Obj_x is $\{r_1, r_2, \ldots, r_n\}$, if $\forall\ i \in \{r_1, r_2, \ldots, r_n\}$, then we have $R\_Obj_x \in S\_BK[i]$.

The data structure of Ladex is composed of an array of buckets and their corresponding linked nodes, this has been shown in figure 3.

In general, blocks can be subdivided at different levels, in our inplementation, Ladex supports three levels of subdivision which corresponds to three levels of precision. At the second level, any first level block Block[i, j] ($0 \leqslant i < M, 0 \leqslant j < N$)can be subdivided into M_2xN_2 sub-blocks: $Block_2[i_2, j_2]$($0 \leqslant i_2 < M_2, 0 \leqslant j_2 < N_2$), where '2' denotes second level subdivision. At the third level, every second level block

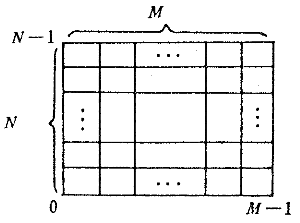


Figure 1 :The lattice on a map

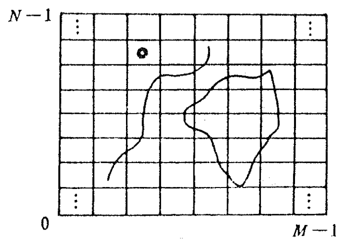


Figure 2: The position and extension of objects on a lattice

$Block_2[i_2, j_2]$($0 \leqslant i_2 < M_2, 0 \leqslant j_2 < N_2$), can be subdivided into M_3xN_3 points, further partition is forbidden. In consideration of efficiency, second and third level blocks should not be organized as bucket structures. The spatial information of an object in a second or third level block can be expressed by a feature information field in a node of Ladex structure. Following is the discussion of representation of feature information field.

A point-shaped object at the second level must be on a second level small block which has a relative block number within its first level block, so the relative block number can be used as feature information; and at the third level, that is the same case. Therefore, the feature information of a point-shaped object can be represented as "Second level block number+Third level block number".

As for line-shaped objects, the feature information is more complicated than that of point-shaped objects. Suppose a line-shaped object pass through m second level blocks(in a first level block of number i), the blocks are in the set $\{L_1, L_2, \ldots, L_m\}$ represented by Set_i^2. For $\forall\ j \in Set_i^2$, Set_j^3 represents the set of third level blocks that the object overlaps in the j-th second level block. Hence, in the i-th bucket, the feature information of a line-shaped object Obj_x can be expressed by $\{L_1.Set_{L1}^3, L_2.Set_{L2}^3, \cdots, Lm.Set_{Lm}^3\}$, the above set is represent as $Obj_x\_INFO_i$, including feature information of Obj_x in the i-th bucket.

As for region-shaped objects, the blocks they overlap fall into two classes. The first class of blocks are completely overlapped by the objects, we call this class of blocks inner blocks. Another class of blocks are partly occupied by the objects, we call them edge blocks. In figure 4, the shaded blocks are inner blocks. Obviously, there is no need to subdivide inner blocks further. No feature information are needed for inner objects. However, we have to subdivide edge blocks further. The partition method of edge blocks and the extraction approach of feature information of the objects are the same as that of line-shaped objects.

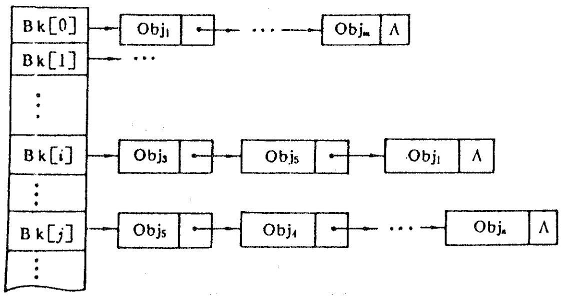


Figure 3: The basic structure of Ladex

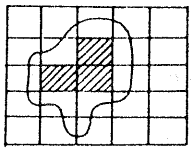


Figure 4: Inner and edge blocks of a region-shaped object

3 Operations and Algorithms for Ladex

Insertion, deletion, modification and search are typical operations for Ladex, each of these operation can be discussed in three cases: Point case, line case and region case. Every operation case has a relevant algorithm.

3.1 Insertion

This operation is to insert a new spatial object's index information in the Ladex structure. Let $M_i x N_i$ be the i-th level partition ($i \in \{1,2,3\}$), and a_i, b_i are the length and width of the i-th level block respectively.

Point Insertion Algorithm:
Input: Point(x,y) and SpIdx.
Output: SpIdx with object identifier and feature information of the point-shaped object inserted.

```
PI1. $B_1 \Leftarrow (y \text{ div } b_1) * M_1 + (x \text{ div } a_1)$;
     $x_2 \Leftarrow x \text{ mod } a_1$;
     $y_2 \Leftarrow y \text{ mod } b_1$;
PI2. $B_2 \Leftarrow (y_2 \text{ div } b_2) * M_2 + (x_2 \text{ div } a_2)$;
     $x_3 \Leftarrow x_2 \text{ mod } a_2$;
     $y_3 \Leftarrow y_2 \text{ mod } b_2$;
PI3. $B_3 \Leftarrow y_3 * a_2 + x_3$; (A third level block can not
     be partitioned further)
PI4. If the identifier of Point(x,y) has been in the
     bucket BK[$B_1$], then step to PI6, otherwise
     allocate a new node and insert the identifier
     and the feature information of Point(x,y) into it;
PI5. Put the new node into the node list in the corresponding
     bucket BK[$B_1$] with feature information "$B_2+B_3$".
PI6. Return.
```

Line Insertion Algorithm:
Input: Line $\{(x_1,y_1),(x_2,y_2),\ldots,(x_n,y_n)\}$ and SpIdx.
Output: SpIdx with the index information of the line-shaped object inserted.
Suppose the points on the inputted line is not distributed sparsely, i.e., for $\forall i \in \{1,2,\ldots,n-1\}$, we have $|x_i - x_{i+1}| \leqslant 1$ and $|yi - y_{i+1}| \leqslant 1$.

```
LI1. $i \Leftarrow 1$;
LI2. If $i = n+1$, then step to LI5;
LI3. Call the Point Insertion Algorithm with the
     parameter Point($x_i$,$y_i$);
LI4. $i \Leftarrow i+1$, go to step LI2;
LI5. Return.
```

Region Insertion Algorithm:
Input: Region $\{(x_1,y_1),(x_2,y_2),\ldots,(x_n,y_n)\}$ and SpIdx.
Output: SpIdx with the index information of the region-shaped object inserted.
A region can be described by its edge. Suppose the points on the edge is dense, i.e., for $\forall i \in \{1, 2,\ldots,n-2\}$, we have $|x_i - x_{i+1}| \leqslant 1$ and $|y_i - y_{i+1}| \leqslant 1$, in addition, $x_1 = x_n$ and $y_1 = y_n$, it is a closed edge.

```
RI1. $i \Leftarrow 1$;
RI2. If $i = n$, then step to RI5;
RI3. Call the Point Insertion Algorithm with the
     parameter Point($x_i$,$y_i$);
RI4. $i \Leftarrow i+1$, go to step RI2;
RI5. Compute the set of inner blocks: $\{B_1,
     B_2,\ldots,B_m\}$, represented as IN_SET;
RI6. If IN_SET $= \varnothing$, then go to step RI9;
RI7. Fetch an element $B_i$ in IN_SET, insert its identifier
     and feature information in a new node, and then put
     the node in the list related to bucket BK[$B_i$];
RI8. IN_SET $\Leftarrow$ IN_SET $- \{B_i\}$, jump to RI6;
RI9. Return.
```

3.2 Deletion

In spatial databases, if an object has been deleted, its information in spatial index should also be deleted, so we can keep the consistency between database and index structure.

Deletion Algorithm:
Suppose the object to be deleted is Obj_x, and the set of first level lattice blocks is B_Set(Obj_x).
Input: Obj_x and SpIdx.
Output: SpIdx.

```
D1. Compute the set B_Set($Obj_x$);
```

```
D2. If B_Set($Obj_x$)=$\varnothing$, then step to D5;
D3. Fetch an element $B_i$ in B_Set($Obj_x$), release the
    node in the node list of the corresponding
    bucket, i.e., S_BK[$B_i$]$\Leftarrow$S_BK[$B_i$]$-$\{$Obj_x$\};
D4. B_Set($Obj_x$)$\Leftarrow$B_Set($Obj_x$)$-$\{$B_i$\}, jump to
    D2;
D5. Return.
```

3.3 **Modification**

Modification is very simple, it comprises two successive steps: Deletion and Insertion. In consider of space, we omit its algorithm.

3.4 **Search**

The search operations report the set of objects that satisfy a given query condition or predicate with the help of spatial index.

Point Search Algorithm:

Description: Retrieve objects near a given point in space.
Input: Point(x,y), SpIdx and query precision level PL_i, note PL_i corresponds to the i-th level blocks ($i \in \{1,2,3\}$).
Output: The set R_Set which contains all objects that satisfy the given spatial condition.

```
PR1. Compute the block number $B_1$ according to
     Point(x,y), and then get the feature infor-
     mation $B_2$ and $B_3$ (depends on $PL_i$), R_Set$\Leftarrow$
     $\varnothing$;
PR2. If S_BK[$B_1$]=$\varnothing$, then step to PR5;
PR3. Fetch an object $Obj_x$ in S_BK[$B_1$], compare
     the feature information of $Obj_x$ with that of
     Point(x,y) according to $PL_i$, if they match,
     then we do R_Set$\Leftarrow$R_Set$\cup$\{$Obj_x$\};
PR4. S_BK[$B_1$]$\Leftarrow$S_BK[$B_1$]$-$\{$Obj_x$\}, jump to
     PR2;
PR5. Return.
```

Line Search Algorithm:

Description: Retrieve objects on or near a given line.
Input: Line-shaped object Obj_x, SpIdx and precision level PL_i($i \in \{1,2,3\}$).
Output: The set R_Set which contains objects that satisfy the given spatial condition.

```
LR1. Compute block numbers and feature infor-
     mation (according to $PL_i$) of those blocks
     overlapped by $Obj_x$, and put the results in
     the set B_Set ($Obj_x$), which has the format:
     \{$B_1$_$INFO_1$, $B_2$_$INFO_2$, ..., $B_n$_$INFO_n$\},
     where $B_i$ is the bucket number, and $INFO_i$
     stasnds for relevant feature information,
     and then we do R_Set$\Leftarrow\varnothing$;
LR2. If B_Set($Obj_x$)=$\varnothing$, then step to LR8;
LR3. Fetch an element $B_i$ in B_Set($Obj_x$);
LR4. If S_BK[$B_i$]=$\varnothing$, then go to LR7;
LR5. Fetch an element $Obj_y$ in S_BK[$B_i$], compare
     the feature information of $Obj_y$ with that of
     $Obj_x$ according to $PL_i$, if they match, then
     we do R_Set$\Leftarrow$R_Set$\cup$\{$Obj_y$\};
LR6. S_BK[$B_i$]$\Leftarrow$S_BK[$B_i$]$-$\{$Obj_y$\}, jump to
     LR4;
LR7. B_Set($Obj_x$)$\Leftarrow$B_Set($Obj_x$)$-$\{$B_i$\}, jump to
     LR2;
LR8. Return.
```

Region Search Algorithm:

Description: Retrieve objects contained in or intersect with a given region.
Input: Region-shaped object Obj_x, SpIdx and precision level PL_i($i \in \{1,2,3\}$).
Output: The set R_Set which includes objects that satisfy the given spatial condition.

```
RR1.  Compute block numbers and feature infor-
      mation (according to $PL_i$) of the inner/edge
      blocks overlapped by $Obj_x$, and put the re-
      sults in inner block set IN_Set/edge block
      set B_Set, R_Set$\Leftarrow\varnothing$;
RR2.  If B_Set($Obj_x$)=$\varnothing$, then step to RR8;
RR3.  Fetch an element $B_i$ in B_Set($Obj_x$);
RR4.  If S_BK[$B_i$]=$\varnothing$, then jump to RR7;
RR5.  Fetch an element $Obj_y$ in S_BK[$B_i$], compare
      the feature infor mationof $Obj_y$ with that of
      $Obj_x$ according to $PL_i$, if they match, we do R
      Set$\Leftarrow$R_Set$\cup$\{$Obj_y$\};
RR6.  S_BK[$B_i$]$\Leftarrow$S_BK[$B_i$]$-$\{$Obj_y$\}, go to RR4;
RR7.  B_Set($Obj_x$)$\Leftarrow$B_Set($Obj_x$)$-$\{$B_i$\}, jump to
      RR2;
RR8.  If IN_Set=$\varnothing$, then step to RR10;
RR9.  Fetch an element $B_j$ in IN_Set, R_Set$\Leftarrow$R_Set
      $\cup$S_BK[$B_j$], IN_Set$\Leftarrow$IN_Set$-$\{$B_j$\}, jump
      to RR8;
RR10. Return.
```

4 Performance Analyses and Results

4.1 **Time and Space Complexity**

Time Complexity: In this paper, we only analyze the time complexities of Point Insertion, Line Insertion and Line Search algorithms.

(1) Point Insertion

The complexity of this algorithm occurs in the computation of buckets and its feature information. Suppose a division and a module can be counted as a multiplication operation respectively. From step PI1, PI2 and PI3, we can see that only 11 multiplications are needed even if the given query condition is at the highest precision level. A few of comparing operations exist in step PI4. Usually, every bucket has less than 20 objects, then average is not larger then 10. So, in most cases, there are 10 comparisons at most in this step. Let a multiplication time be T_m, and a comparison time be T_c, the time complexity of Point Search algorithm could be expressed by the formula: $PT = 11T_m + 10T_c$. Obviously, PT is very small, therefore, this algorithm is very quick.

(2) Line Insertion

From step LI3 in this algorithm, we can see, a point insertion time (PT) is needed here. For there exists n points, so the time for this algorithm is: $LT = n * PT$, i.e., $LT = 11n * T_m + 10n * T_c$. If we use PT as measure, the time complexity of this algorithm is O(n).

(3) Line Search

The time consumed in step LR1 is evidently not great than LT. Let T_{cc} be the time needed for a comparison related to precision control. In fact, in application, the average number of objects a bucket holds is not great than 10, so, from step LR5, the time spent here is not more than $10\, T_{cc}$. Let the element number of B_Set(Obj_x) at step LR1 be m initially. the time for this algorithm can be evaluated as follow: $LRT = LT + m * 10 * T_{cc} = n * PT + 10m * T_{cc}$. If the query precision is at first level, then $T_{cc} = 0$, in this situation, the time complexity is only O(n) . Obviously, $m \leqslant MxN$, usually, $m << n$ and $T_{cc} < PT$, so the time complexity of this algorithm is O(n).

Space Complexity: From figure 3, we can see that Ladex must keep a table with MxN buckets and many a lot of lists that vary in length. Hence, Ladex structure needs a lot of storage space. However, because only the object identifier and its feature information are stored in the structure, the storage needed by Ladex is significantly less then that occupied by the large scale database itself. In a sense, the storage overhead of Ladex can be ignored in most applications.

4.2 Experimental Results

Ladex is established on Map Databse Management System MDB funded by Chinese Electronic Industry Ministry. In MDB, users can define, store, access and edit conventional data and complex spatial data in a uniform format. Conventional data and complex spatial data are tightly coupled at physical level. We integrate Ladex with B^n-tree (An improved B^+-tree) in MDB, and the goal of quick spatial retrieval is achieved successfully. The following is the performance evaluation approach.

The resolution of the tested maps is 10240x10240 in pixel. In fact, in many applications, a data space with resolution of 10240x10240 in pixel has less then 10000 objects. We designed a tool automatically generate 100 maps with 100 geographic objects, 100 maps with 1000, 100 maps with 5000 and 100 maps with 5000 geographic objects respectively. Each map has different number of point-shaped objects (such as bridges and towers), different number of line-shaped objects (railways, roads and so on) and different number of region-shaped objects (such as plant fields and residential areas). Every spatial object contain multiple conventional attributes (such as order number, longitude, latitude, etc.), the average is about 12. The Evaluation indices are as follows (excluding the drawing time):

(1) T_{Q1}: Average time retrieving all objects in a whole map.
(2) T_{Q2}: Average time retrieving all objects in a region with arbitrary shape that takes one quarter of a map in size.
(3) T_{Q3}: Average time retrieving one object in a map.
(4) T_C: Average time creating Ladex for a whole map.
(5) T_{D1}: Average time deleting Ladex for a complete map.
(6) T_{D2}: Average time deleting index information of all objects in a region with arbitrary shape that occupy a quarter of a map in size.
(7) T_{D3}: Average time deleting one object's index information in Ladex.

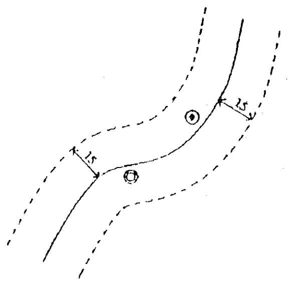
Table 1 shows the experimental results at the 2nd precision level in a CompaQ 486/33 personal computer. From the results, we can see that most of the performance indices are ideal, especially the performance for search operation. This implies that Ladex meets the needs of quick interactive spatial query.

5 Advantages of Ladex

Ladex is a new spatial index mechanism quite different from R^+-tree. it has the following advantages:
(1) The number of buckets is fixed, the structure is simple and operation algorithms are efficient;
(2) Can manage spatial objects with irregular shape;
(3) Support various query types, and the query range can be non-rectangle in shape. For instance, retrieval of all towns with 15 kilometers along a road (see figure 5) and search of all plant areas between 200 and 250 meters of altitude (refer to figure 6).

Operations	Indices	100 objects	1000 objects	5000 objects	10000 objects
Search	T_{Q1}	0.80	2.52	5.40	9.20
	T_{Q2}	0.42	1.21	2.50	4.70
	T_{Q3}	0.10	0.13	0.14	0.18
Insertion	T_C	5.23	58.67	311.38	595.40
Deletion	T_{D1}	0.46	1.07	1.52	1.95
	T_{D2}	2.28	3.35	5.44	7.29
	T_{D3}	0.12	0.14	0.16	0.19

Table 1: Performance Indices (in second)



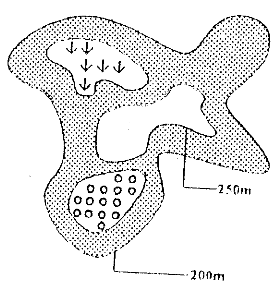


Figure 5: a line query Figure 6: a region query

(4) Can control query precision level. Different applications require different precisions. For example, in a military GIS system, accurate results are often needed; In an interactive tour information system, user can give a query condition on a touch screen with his or her finger. In this case, the query precision level at the first or second level is enough, therefore further computation is omitted, so the performance is improved.

6 Conclusions

Ladex is a new spatial index mechanism, it supports index of spatial objects in irregular shape, and can control query precision level. The structure is simple. The fundamental algorithms are of high performance. We integrated Ladex with B^n-tree to improve the efficiency of spatial retrieval. In our Map Database Management System MDB, Ladex has changed the query format significantly. Users do not need to master complex SQL database language, they can use mouse to select objects or draw a query range in arbitrary shape on the screen, and immediately get the information of the desired objects. If users operate through touch screen, they can get what they want with their fingers. This is really "What you want, what you get" without any ad hoc language.

Future work could focus on: (1) In application, find an approach to determine the optimal number of the first level blocks; (2) Study the relationship between the number of partition levels and spatial search efficiency; (3) Find a better way to expres feature information; (4) Extend Ladex mechanism to 3 and above dimensional space.

References

Aref, W. G., and Samet, H., 1991. Optimization strategies for spatial query precessing. Proc. of VLDB, pp. 81—90.

Beckmann, N., Kriegel, H. P., Schneider, R., and Seeger, B., 1990. The R^*-tree: an efficent and robust access method for points and rectangles. ACM SIGMOD, pp. 322—331.

Faloutsos, C., Sellis, T., Roussopoulos, N., 1987. The R^+-tree: A dynamic Index for Multi-Dimensional Objects, Proc. of the 13th VLDB conf., pp. 507—518.

Guttman, A., 1984. R-trees: a dynamic index structure for spatial searching. Proc. ACM SIGMOD, pp. 47—57.

Hinrichs, K., and Nievergelt, J., 1983. The grid-file: a data structure to support proximity queries on spatial objects. Proc. of the WG'83 pp. 100—113.

Ibrahim, K., and Christos, F., 1994. Hilbert R-tree: An Improved R-tree Using Fractals, Proc. of the 20th VLDB Conf. pp. 500—509.

Lu, H., and Ooi, B. C., 1993. Spatial Indexing: Past and future. IEEE D ta Engineering, 16(3): 16—21.

Medeiros, C. B., and Pires, F., 1994. Database for GIS, In SIGMOD Record.

Monehouse, S., 1992. The ARC/INFO Geographic Information System, Computers and Geosciences: An international Journal, 18(4).

Ousterhout, J. K., Hamachi, G. T., Mayo, R. N., Scott, W. S., and Taylor, G. S., 1984. Magic: A VLSI Layout System. In 21st Design Automation Conference, pages 152—159, Alburquerque, NM.

White, M., 1981. N-trees: Large Ordered Indexes for Multi-Dimensional Space. Application Mathematics Research Staff, Statistical Research Division, U.S. Bureau of the Census.