Frameworks of Indexing Mechanism for Mobile Applications

Haeng-Kon Kim

Abstract—**While mobile devices own limited storages and low computational resources, the volumes of spatial data are tremendous and spatial operations are time-intensive. Therefore, it is necessary to study a spatial index which is small and efficiently filter out the candidate objects of a spatial operation as well in order to processing spatial operations efficiently in the vector map service based on mobile devices such as PDA, phone, telematics terminals, etc.**

This paper proposes a hash based spatial index for the mobile map service. The index has a simple structure for storage utilization and uses a hashing technique for search efficiency. Our experimental tests show that the proposed index is appropriate for mobile devices in terms of the volume of index, the number of the MBR comparisons, the filtering efficiency and the execution time of spatial operations.

Index Terms—**ITS(Intelligent Transportation System), LBS (Location Based Service) Spatial Index, Mobile Application, Hash, Frameworks.**

I. Introduction

The PDAs lead the popularization of the portable information terminals and their application areas are various such as wireless internet, remote education, wireless game, portable video phone, etc. Especially the mobile map service will be a main application of the PDA in the near future. Therefore, an efficient spatial data retrieval and indexing method for the mobile map service have to be studied.

The volume of spatial data and the computational cost of spatial operations are very tremendous, but on the other hand the mobile devices own a limited memory and a low computational capacity than the PC. Therefore, a spatial index for the mobile devices should be small and achieve good filtering efficiency as well. The existing spatial indices based on the PC, such as R-tree, are not applicable to the mobile devices based on memories. In the mobile device's applications, since the data transmitted from servers always reside in the memory of the mobile devices and disk accesses are not required for data retrievals, the working mechanism of an index for the mobile devices is quietly different from that of the existing indices. The indices based on disks hold a node structure for paged I/O and use clustering techniques in order to reduce the number of disk seeks. On the other hand, because search operations in the indices based on the memory are irrelevant to disks accesses, the node structure and the clustering which are important factors of performance improvements of the existing indices maybe yields contrary results in the mobile devices. Also, the great volume of the existing spatial indices is not appropriate for the mobile devices. For example, the index volumes of R-tree and its variants, which are considered as one of the best spatial indices so far, are about 16%~20% of source data[9].

In addition to that, the storage utilization of each node of the existing indices is less than 70% due to the property of dynamic update. Because the applications of the mobile devices are mostly bounded to retrieval, not update, the property of dynamic update of the R-tree is an obstacle of performance improvements in the mobile devices. The update is generally processed by servers in the mobile map service periodically.

The size of spatial objects in a relation varies extremely. In order to deal with such large objects and at the same time to preserve spatial locality in pages, spatial access methods organize only approximations of objects as an index instead of the exact representation. The minimum bounding rectangle(MBR) of an object is a very common approach for an approximation. A region query is processed in two steps, called filter and refinement step[6]. The filter step finds all candidate objects whose MBR intersects the query region. The refinement step checks whether they really fulfill the query condition for those objects.

Because the refinement step applies the exact representation to check procedures, it is very time-consuming. Therefore, it is had better cut down the number of candidate objects in the filter step if possible.

In this paper, we propose a spatial index structure for the mobile devices, which is so simple and small that shows good storage utilization and filtering efficiency as well. We take the small memory and low computing capacity of the mobile devices into consideration. The proposed spatial index uses multiple-hashing techniques and achieves small number of the MBR comparisons in the filter step of a spatial query.

In order to evaluate the performances of the proposed spatial indexing method and the MBR compression schemes in comparison with R^*-tree, several experiments are conducted on the Sequoia 2000 benchmark data[11]. The items of the performance evaluations are the volume of each spatial index, the number of MBR comparison in the filter step, filtering efficiency and the total operation time. The results of the

Heang-Kon Kim is with the Department of Computer Engineering, Catholic University of Daegu KyungSan, Daegu, 712-702, Korea (corresponding author to provide phone: 053-850-2743; fax: 053-850-2740;
e-mail: hangkon@ cu.ac.kr).

experiments clearly show that we can decide a suitable spatial index for the mobile devices and the proposed methods are also expected to be important technology for the mobile map services to be wide used recently.

The rest of this paper is organized as follows. In section 2, we investigate the related works on spatial index based on disks and memories respectively. In section 3, we propose a spatial index structure based on memory for the sake of the mobile map services. In section 4, the results of experiments are presented and analyzed. Finally, we give conclusions in section 5.

II. Related Works on Spatial Indices

Spatial indices are used for the efficient processing of the filter step. Many researches on spatial indices have been studied so far. The index in these literatures, however, is almost based on the disks[1], [2], [3], [6], [10], [12]. The R^*-tree called a representative spatial index based on disks is studied in [1].

A node of R^*-tree is designed for one page of disks, and each node tends to cluster the objects located in the extent(Xmin, Ymin, Xmax, Ymax) of the node. Besides, the R^*-tree also shows outstanding query performances on non-uniformly distributed data. The properties of R^*-tree, such as a balanced tree and a large number of fan-out, achieve an efficient data retrieval, but we should endure the low storage utilization reported by several literatures [9]. The low utilization may be a fatal defect in the mobile devices with limited storages. In spite of these drawbacks, the good performance of the R^*-tree is derived from a paged I/O and natural clustering which reduce the number of seek operations.

There have been many works on indices of main memory database systems recently [4], [5], [7], [8], [10]. The T-tree[5] is the binary tree of which each node maintains several number of data and just two number of link fields in order to maximize storage utilization. It can accommodate text data, but it is impossible for T-tree as an index to hold multidimensional spatial data.

The CR-tree[4] based on main memory is the cache-conscious version of the R-tree. To pack more entries in a node, the CR-tree compresses MBRs. While the QRMBR proposed in CR-tree saves memory, it yields low filtering efficiency which is a fatal drawback in the mobile devices as described above.

The literature [13], [14] proposed a clustering-based map compression method which adapts a dictionary to a given dataset. The proposed method achieves lower error than a static dictionary compression such as the one used by the FHM algorithm[15].

There have been many previous works on spatial joins such as join indices[22], join z-ordering[23], and hash join[24]. The most recent works focus on developing spatial index structures like R-tree and its variants[17], [18], PMR Quad-tree[19], Grid Files[16], Seeded-tree[21], and Filter-tree[20].

Furthermore, the spatial indices for moving objects have been increasingly studied recently. 3DR-tree[27] based on R-tree is desirable for trajectory query. TPR-tree[25] attempts to represent the current and future position of the moving objects by a function of the speed and orientation of moving objects. LUR-tree[26] aims at performance improvements of the spatial index for moving objects with the minimization of update operation of the spatial index.

The previous works on the spatial access methods have not studied the index for the mobile devices and not also conducted experiments for performance evaluations. In this paper, we propose a new spatial index structure using a MBR compression scheme and hashing technique for the mobile map service with desirable properties, such as high storage utilization, filtering efficiency and quick response time.

III. A Hash Based Spatial Index

Because the mobile devices own limited storages and low computational resources, the spatial index for the mobile devices should possesses the following properties.

First, small volume and simple structure. R-tree and its variants are comparatively big volume and complicated multi-level indices, and so they are unsuitable for PDA.

Second, quick response time for spatial queries(a simple display operation). Because the main operation of mobile map services is not update but simple search, the index for mobile map service should makes small number of MBR comparisons and high efficiency in the filter phase of basic spatial queries such as point query and region query.

Third, easy to load a region partitioned regularly in order to display full screen image of PDA. Generally speaking, spatial indices with regular decomposition scheme, such as grid file, are able to load easily all objects of a rectangle region, whereas spatial indices with irregular decomposition scheme, such as R*-tree, search multiple node for the sake of bulk loading because each node may be overlapped.

Fourth, easy to manage non-uniformly distributed data. R*-tree is efficient for any data distribution, whereas Grid File delivers a fatally degraded search performance at non uniform data.

Therefore, both R*-tree and Grid File which are representative indices based on disks are unsuitable for PDA spatial indices. We propose a new spatial index to obey above four requirements. This index delivers high storage utilization due to its simple structure, and it also has small number of the MBR comparison due to the hash-based indexing technique. Although it takes regular decomposition, it manages non-uniformly distributed data very well.

A. *Index Structure*

The proposed index hashes the overall space on the basis of X and Y coordinates of each object. The hash functions are as follows.

- $H_x(x) = \mathrm{int}[(x - X_{min})/(X_{max}-X_{min})*N_x]$
  N_x is the number of buckets of X axis
- $H_y(y) = \mathrm{int}[(y - Y_{min})/(Y_{max}-Y_{min})*N_y]$
  N_y is the number of buckets of Y axis

The hash table is two dimensional structure and the second(or subsequent) hashing is executed to prevent a bucket overflows as shown in Fig. 1. Even though the data distribution is non-uniformed and skewed, therefore the reasonable response time is guaranteed without severe delay for spatial queries.

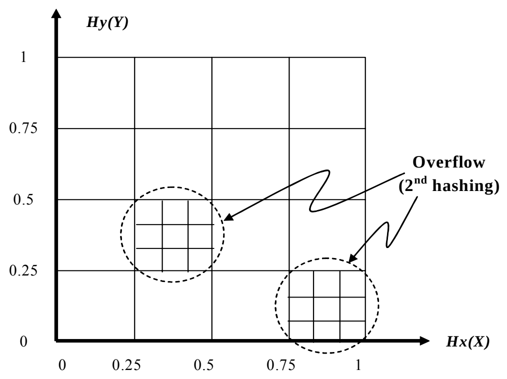

Fig. 1. Hash Table

The second hash function is similar to the first hash function except for the min and max values of the extents. The min and max values in the second function are changed the extent of entire data space into the extent of the overflowed bucket. The third or more hashing is executed successively until no longer overflows. We assume that the Nx and Ny of second or more hashing are a half of those of first hashing respectively. The capacity(*M*) of a bucket is determined by the combinations of the whole number of the objects, the desired volume of the index and the desired search time. The volume of the index is in reverse proportion to the *M* and the number of the MBR comparison operations in the filter step.

The Nx, Ny and M are the consideration points in the index design. Large Nx, Ny and small M bring about low performance due to excessive subsequent hashing and the redundancy of objects caused by multiple assignment policy[16]. Also, small Nx, Ny and large M bring about low performance due to extended search space. That is to say, when the values of Nx, Ny and M are almost equal, the performance of the proposed index is enhanced.

The 100 percent cell utilization can be possible in the text data and uniform distributed spatial data, but it is not probable in the non-uniformly distributed spatial data due to frequent overflow and sub hashing. Therefore, we assume the cell utilization to be about 65 ~75%. The cell utilization is defined as follows.

- Cell_Utilization ≈ #_of_Obj / [(Nx*Ny)*M]
  where Nx, Ny and M are equal.

The header of the index consists of an extent of entire map and the number of buckets of both X and Y axes as shown as in Figure 2.

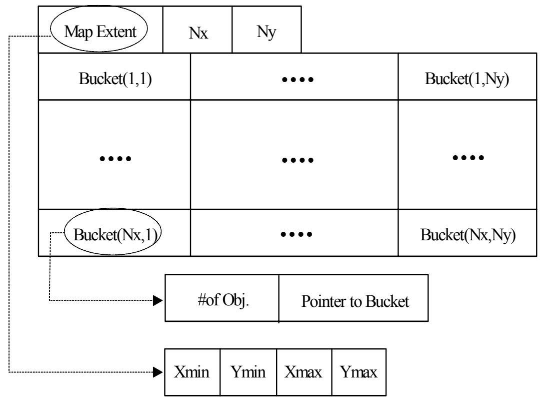

Fig. 2. Header Format

A bucket entry holds the number of objects and the pointers to indicate each object in data file. The pointer is classified by the number of objects as follows.

- [# of Obj. = 0] Empty Bucket Condition :
  Pointer = NULL;
- [# of Obj. 1~M] Normal Condition :
  Pointer to indicate each bucket
- [# of Obj. > M] Overflow Condition :
  Pointer to sub hash table

Each bucket holds MBRs and Pointers of own objects as follows

- <MBR of Object, Pointer to Object> * # of Obj.

One of the most important characteristics of GIS data in view of the mobile device is that the volume of spatial data is tremendous. To filter efficiently candidate objects, spatial indices usually use MBRs organized by the coordinates of low left corner and upper right corner as the approximation of each spatial object. The 16 byte MBR is generally used for two-dimensional key because a coordinate of each axis takes a 4 byte float number. The existing spatial indices are usually too big to use in the mobile devices. The MBR keys also occupy almost 80% of their indices. Therefore, we focus on the MBR compression scheme for the sake of a small index.

The MBR compression scheme proposed in this paper takes the hybrid representation scheme that makes use of the merit of not only reasonable storage utilization but also good filtering efficiency. The low left corner of the compressed MBR is represented by relative position, but the upper right corner of the compressed MBR is represented by the lengths(width, height) of MBR as follows.

```
・Compressed MBR =
        Xmin(2byte, relative position)
        Ymin(2byte, relative position)
        Width(1byte, real or quantized)
        Height(1byte, real or quantized)
```

In order to represent the width and height of MBR by means of 1 byte respectively, a following method is introduced. To begin with, if the length of MBR is smaller than threshold value(β), the length is represented by actual value just as it is. If not, the length is represented by a quantized value.

The β is determined by quantization level. The β is 255-*n*, where *n* is quantization level. Therefore, the unit length(δ) of a quantum is [length of bucket's extent / *n*] and so the length by quantized value is [β + (length of MBR) / δ].

When we use the proposed MBR compression schemes, several decompression algorithms are generally necessary for executing the spatial queries. However, we transform a given query input coordinates into the compressed MBR coordinate system instead of decompression of compressed MBR and so the overheads caused by compression are easily lessen.

B. Procedures for Spatial Operation

In the following, we introduce the procedures of a point query and a region query used frequently in the map service applications. The input parameters of the point query procedure are the header pointer of the index and X, Y coordinates pointed by mouse. The return values are selected objects.

```
Procedure Point_Query( )
   Input : Header, Point
   Output : selected objects
Read Extent from Header
X = Hx(Point.x);
Y = Hy(Point.y);   // Hash X,Y
Read Information of Bucket(X,Y)
CASE( # of Obj. )
   [ 0 ] :   "Not Found"
   [ > M ] :   Bucket_Header = Pointer_to_Bucket;
            Point_Query(Bucket_Header, Point)   // Recursive Call
   [ 1~M ] :S = Pointer_to_Bucket;
      FOR( all Obj. ∈ S )         // from 1 to # of Obj.
                  IF( Obj.MBR ∩ Point ≠ ∅)
               Result += Obj.
end of Procedure
```

The input parameters of the region query procedure are the header pointer of the index and a query region which consists of Xmin, Xmax, Ymin and Ymax. The return values are also selected objects like point query.

```
Procedure Region_Query( )
   Input : Header, Region
   Output : selected objects
Read Extent from Header
X_low = Hx(Region.X_min); X_max = Hx(Region.X_max);
Y_low = Hy(Region.Y_min); Y_max = Hy(Region.Y_max);
FOR( X = X_low ~ X_max )
   FOR( Y = Y_low ~ Y_max )
   {
     Read Information of Bucket(X,Y)
     CASE( # of Obj. )
       [ 0 ] : Continue
       [> M] : Bucket_Header = Pointer_to_Bucket;
   Region_Query(Bucket_Header, Region)
        // Recursive Call
       [1~M] : S = Pointer_to_Bucket;
          FOR( all Obj ∈ S )          // from 1 to # of Obj.
             IF( Obj.MBR ∩ Region ≠ ∅)
                             Result += Obj.
   }
end of Procedure
```

The point and region query in the index is resolved by simple hashing. In particular, to load all the objects located in a rectangle area for simple display is straightforward because the index decomposes data space regularly. In addition to that, the query for skewed areas is easily executed by smaller number of recursive calls. The impacts of the skews will be examined in chapter 4.

IV. Performance Evaluation

A. Environments for Performance Evaluation

The test data for performance evaluation are Sequoia 2000 which is widely used as benchmark data. The data set consists of several polygons in which each of them represents objects of 38 layers such as GIRAS Land use, Land cover, etc. Fig. 3 display these layers with overlapped and non-uniformed distribution.

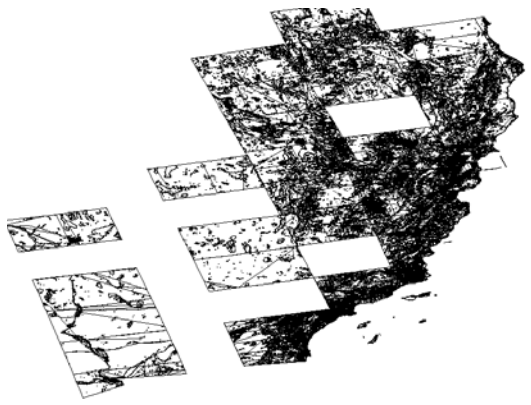

Fig. 3. Test Data Sequoia 2000

The H/W platform is Compaq iPAQ PDA with 32 Mbyte memories for data area and the developing tool is Microsoft embedded visual C++. The point query and region query are carried out 1000 times respectively.

B. Volume of Index

Fig. 4 graphs the volumes of the two indices, R*-tree and the proposed index, with the proposed MBR compression scheme. The node size of R*-tree is 512 bytes and the bucket capacity of top level of the proposed index is 50, the capacities of second and more levels are assigned a value decreased by 5 compared with that of previous level. The quantization level of the compressed MBR is 50. The Nx and Ny of the proposed index are both 50. The M_INDEX means the proposed index in Fig. 4. The C_MBR also means the compressed MBR in Fig. 4.

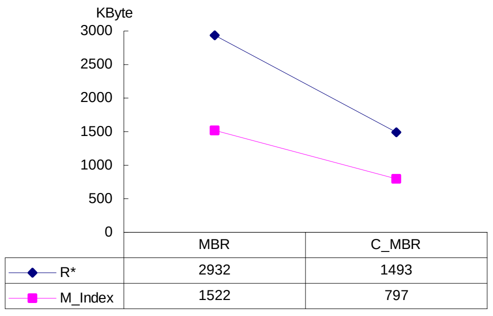

Fig. 4. Size of Spatial Indices

The following results are obtained from Fig. 4. First, the M_INDEX outperforms the R*-tree in the storage utilization aspect when an identical MBR representation scheme is used by each index. While the M_INDEX has very simple structure, the R*-tree has complicated multi-level structure and the property of the tree-balance which makes R*-tree yield the low storage utilization.

Second, compared with the volume of spatial index according to the MBR compression schemes, Fig. 4 tells us that C_MBR also achieves a good compression effect.

C. Number of MBR Comparison in Filter Step

Fig. 5 depicts the number of the MBR comparison operations which means search performance in the filter step of point query. The M_INDEX generally outperforms the R*-tree. More precisely, the M_INDEX with either C_MBR requires almost 50% of MBR comparison of R*-tree.

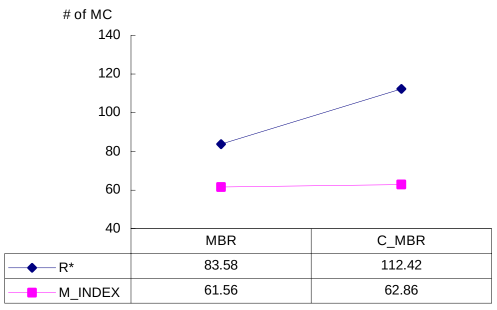

Fig. 5. Number of MBR Comparison Operations

D. Filtering Efficiency of MBR Compression Scheme

The aim to use spatial indices is to execute efficiently filter step as well as to minimize the candidate objects participated in the refinement step. If the low computational resource of the mobile devices is taken into consideration, the number of the candidate objects is strongly related to overall execution time of spatial operations. Table 2 summarizes the average number of candidate objects after filter step of point and region query. The areas of the query region are 0.1%, 0.4 and 1% of the whole data space respectively. The experimental results indicate that the number of candidate objects is almost irrelevant to the kind of spatial indices, but the number of candidate objects depends on the MBR compression schemes. Even though C_MBR uses the quantization technique, the number of the candidate objects in C_MBR is almost identical with that of normal MBR because this quantization is only used for big objects in C_MBR.

TABLE 2. NUMBER OF CANDIDATE OBJECTS

Query Region / MBR Rep.	PointQuery	0.1%	0.4%	1%
MBR	3.2	187.4	521.3	1038.9
CMBR	3.5	201.3	553.8	1052.0

E. Query Execution Time

In the following, the performances of M_INDEX are evaluated. Table 3 describes the average execution time of the spatial queries. Generally, M_INDEX improves the performance of about 15% and more percents in comparison with R*-tree. The results obtained from experiments coincide with what was expected. Table 3 also shows that the query execution time have nothing to do with the area of query region mostly.

TABLE 3. QUERY EXECUTION TIME(UNIT : MS)

Query / MBR Rep.	Point Query	0.1%	0.4%	1%
R*-tree (C_MBR)	69.1	543.8	881.2	1215.2
M_INDEX (C_MBR)	46.8	451.8	782.9	1103.0

F. Data Loading Time with Various Data Distribution

Table 4 summarizes the time to load the results of point and region queries of M_INDEX and R*-tree without refinement step. The query region is occupied with 1% of entire data space. Each query is performed in random area and skewed area respectively. We define the skewed area as where to hash 3 or more times subsequently. The results clearly show that the M_INDEX generally outperforms the R*-tree for the simple display to load all the objects in a rectangle region. Far from one's anticipation, the M_INDEX carries out region queries well in case of skewed area. Moreover, the performance of point query of M_INDEX is also superior to that of R*-tree regardless of data distribution.

TABLE 4. DATA LOADING TIME(UNIT : MS)

	M_INDEX (C_MBR)	R*-tree (C_MBR)
Random Area(1%)	313.5	427.9
Skewed Area(1%)	1805.2	1814.8
Point(Random)	32.0	53.7
Point(Skewed)	44.3	56.2

G. C MBR : Impacts of Bucket Capacity

In case of sequoia 2000 benchmark data and above condition, the proper numbers of Nx, Ny and *M* are about 50. Fig. 6 and 7 show that this seems to be a realistic assumption. Fig. 6 depicts the volume of M_INDEX with various bucket capacities(*M*). The Nx and Ny are assumed to be 50. As *M* increases, the volume of M_INDEX with C_MBR decreases.

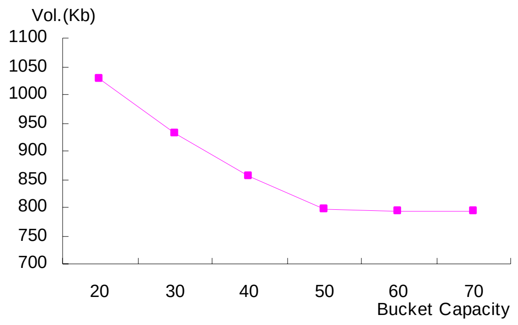

Fig. 6. Volume of M_INDEX

Fig. 7 depicts the number of MBR comparison in filter step with various *M*. The number of MBR comparison in filter step shows different aspects. As M decreases from 50 to 20, the number of MBR comparison of the region query rapidly increases. Also, as M increases from 50 to 70, the number of MBR comparison of the region query increases to some degree. In case of point query, there is nearly no deviation of the number of MBR comparison except 70 bucket capacity. Therefore, when the Nx, Ny and M are 50, the volume of index and the number of MBR comparison are improved as expected. As the purpose of this paper is concerned, we will leave the detailed discussion of cell utilization and bucket capacity for further researches.

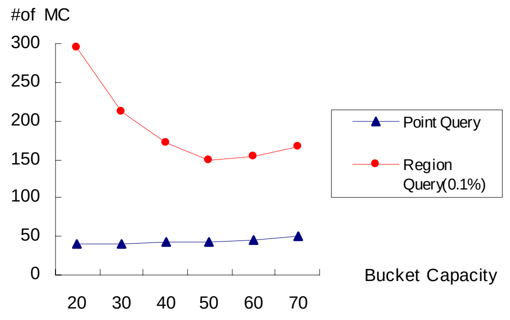

Fig. 7. Number of MBR Comparison

V. Conclusions

In this paper, a new spatial index is proposed. We would like to propose a reliable spatial index for the mobile map service. The requirements of this are high storage utilization, quick response time and easy simple display. The proposed index has simple structure for storage efficiency and uses a hashing technique, which is direct search method, for search efficiency.

The experimental results indicate that the R*-tree, one of the most efficient spatial index based on disk, is possible to be inefficient in memory based mobile device system. On the contrary, the proposed index outperforms R*-tree due to the proposed index's high storage utilization and retrieval efficiency. The proposed MBR compression scheme requires small storages and achieves high filtering efficiency. The proposed index consumes about 50% less memory space in comparison with R*-tree, and the number of MBR comparison in filtering step of the compressed MBR is about 50% less than that of R*-tree. In the MBR compression aspects, the spatial index with the compressed MBR requires about 50% smaller than the spatial index with normal MBR.

In summary, it seems reasonable to conclude that the proposed spatial index structure is appropriate for the spatial index in the mobile devices with small memory space and low processing capacity.

Furthermore, the index is expected to be useful for mobile map service, ITS(Intelligent Transportation System), LBS(Location Based Service) to have been increasingly studied recently.

References

[1] N. Beckmann, H.P. Kriegel, R. Schneider, B. Seeger, "*R*-tree : An Efficient and Robust Access Method for Points and Rectangles*", Proc. of Int. Conf. on ACM SIGMOD, 1990, pp. 322-331.
[2] A. Guttman, "*R-trees: A dynamic index structure for spatial searching*", Proc. of Int. Conf. on ACM SIGMOD, 1984, pp. 47-57.
[3] E.G. Hoel, H. Samet, "*A Qualitative Study of Data Structures for Large Line Segment Databases*", Proc. of Int. Conf. on ACM SIGMOD, 1992, pp. 205-214.
[4] K.H. Kim, S.K. Cha, K.J. Kwon, "*Optimizing multidimensional index trees for main memory access*", Proc. of Int. Conf. on ACM SIGMOD, 2001.
[5] T.J. Lehman, M.J. Carey, "*A Study of index structures for main memory database management system*", Proc. of Int. Conf. on VLDB, 1986, pp. 294-303.
[6] H. Lu, B.C. Ooi, "*Spatial Indexing : Past and Future*", IEEE Data Engineering Bulletin, Vol. 16, No. 3, 1993, pp 16-21.
[7] J. Rao, K.A. Ross, "*Cache conscious indexing for decision-support in main memory*", Proc. of Int. Conf. on VLDB, 1999, pp. 78-89.
[8] J. Rao, K.A. Ross, "*Making B+-trees cache conscious in main memory*", Proc. of Int. Conf. on ACM SIGMOD, 2000, pp. 475-486.
[9] H. Samet, "*The Design and Analysis of Spatial Data Structures*", Addison Wesley, 1990, pp. 507.
[10] A. Shatdal, C. Kant, J.F. Naughton, "*Cache conscious algorithms for relational query processing*", Proc. of Int. Conf. on VLDB, 1994, pp. 510-521.
[11] M. Stonebraker, J. Frew, K. Gardels, J. Meredith, "*The SEQUOIA 2000 Storage Benchmark*", Proc. of Int. Conf. on ACM SIGMOD, 1993, pp. 2-11.
[12] K.Y. Whang, R. Krishnamurthy, "*The Multilevel Grid Files – a Dynamic Hierarchical Multidimensional File Structure*", Proc. of Int. Conf. on Database Systems for Advanced Applications, 1991, pp. 449-459.
[13] S. Shekhar, Y. Huang, J. Djugash, "Dictionary Design Algorithms for Vector Map Compression", Proc. of Data Compression Conf(Abstract), 2002, pp. 471.
[14] S. Shekhar, Y. Huang, J. Djugash, C. Zhou, "Vector Map Compression : A Clustering Approach", Proc. of 10th ACM int. Symposium on Advances in GIS, 2002, pp. 74-80.
[15] P.W. Wong, J. Koplowitz, "Chain Codes and Their Linear Reconstruction Filters", IEEE Trans. On Information Theory, Vol. 38, No. 2, 1992, pp. 268-280.
[16] X. Zhou, D. J. Abel, David Truffet, "Data Partitioning for Parallel Spatial Join Processing", Proc. of Int. Conf. on SSD, 1997, pp. 178-196.
[17] T. Brinkhoff, H.-P. Kriegel, R.Schneider, B. Seeger, "Efficient Processing of Spatial Joins Using R-trees", Proc. ACM SIGMOD Conf., 1993, pp. 237-246.
[18] T. Brinkhoff, H.-P. Kriegel, R.Schneider, B. Seeger, "Multi-Step Processing of Spatial Joins", Proc. ACM SIGMOD Conf., 1994.
[19] E.G. Hoel, H. Samet, "Data-Parallel Spatial Join Algorithms", Int. Conf. On Parallel Processing, 1994, pp. 227-234.