Indexing Moving Objects Based on 2^n Index Tree

HUANZHUO YE [1 a], HONGXIA LUO [2], KEZHEN SONG [1 b], HUALI XIANG [1 c], JING CHEN [3]
[1] Information School
Zhongnan University of Economics and Law
114, Wuluo Road, Wuhan, Hubei Province
P. R. CHINA

[2] School of Geographical Sciences
Southwest University
2, Tiansheng Road, Beibei District, Chongqing
P. R. CHINA

[3] State Key Laboratory of Information Engineering in Surveying and Mapping
Wuhan University
129, Luoyu Road, Wuhan, Hubei Province
P. R. CHINA

Abstract: - In moving objects information management, Motion state model (MSM) is based on sample method and suits to most applications. Because of its characters, 2^n index tree is proposed to organize the motion data. In order to increase the efficiency of indexing, some issues such as late updating of leaf node splitting and merging, pre-query before the query of objects relation and the reference of motion vectors in index tree are carefully studied. Comparing to other indexing method, 2^n index tree works better with MSM.

Key-Words: - Moving object, 2^n tree, Motion state model, Index, Motion vector

1 Introduction
Wireless communication makes it possible to collect the states of moving objects in real time, thus make it possible to store, retrieve, replay and forecast their dynamic states of position, speed, orientation, etc. Moving objects have temporal attributes as well as spatial attributes. Either temporal or spatial attribute is dramatic in its information management. When these two kinds of attribute are both within the domain of study, things are getting more complicated. Suppose a car equipped with GPS, which data are transmitted through GPRS or CDMA network, how do we manage these spatiotemporal information?

For a single moving object, the motion information is relatively simple to organize even though careful study has to be given to both its spatial and temporal attributes. When there are lots of moving objects and the motion information is of long periods, things get much more complicated, because the quantity of motion data is rather large. To organize a large quantity of data is a good subject to study itself, let alone the data is both spatial and temporal.

In a data center dealing with the motion information of moving objects, such as Intelligent Transport System (ITS), digital battle field, Mobile e-Commerce center, etc., it is necessary to respond quickly to the queries of certain moving objects at certain area in certain period of time with certain condition. This naturally asks for an efficient index to all the motion data of all the moving objects for all periods of time.

2 Related Works
Besides the common indexing method, there are some studies especially for spatial indexing and/or temporal indexing.

2.1 Spatial Indexing
Spatial indexing method is always an important subject in geographic information system (GIS) study. Some efficient and common used spatial indices are Object-Area Index [1], Grid Index [1], QuadTree [2] and its expansion OctTree, R-Tree and its expansion R*-Tree/R+-Tree [1][3][4][5], k-d-Tree and k-d-B-Tree [6][7][8], SS-Tree and SR-Tree [9], Metric Tree and VP-Tree [10], M-Tree [11], etc.

2.2 Temporal Indexing
Many of the existing temporal indices are based on B-Tree or R-Tree. Some indices make use of hashing and other method. Some good algorithms include

Interval B-Tree (IBT) [12], Multi-Version B-tree (MVB-tree) [13][14], Monotonic B+-Tree (MBT) [15], Time Split B-Tree (TSBT) [16], Time Index and Time Index+ [17][18][19], IP-Index [20] and IP*-Index [21], Append-Only Tree (AP Tree) [22], Segment R-Tree [23], Historical R-Tree (HR-tree) [24], SIQ [25], Snapshot Index [26][27], etc.

2.3 Moving Objects Indexing

Moving objects indexing is far more complicated. Besides space partitioning, which is widely used in spatial and temporal indexing, data partioning is also employed. Grid, hashing and tree structure are chosen for different data model.

Hiroshi Nozawa introduces MT structure, ST structure, 3DT and AT structure for different data organization [28]. Hae Don Chon uses Space-Time Grid [29][30]. Trees, esp. R tree and its variances, are commonly used in moving objects indexing, such as PMR quadtree [31][32][33], 3D R-tree and Multi-Version 3D R-tree (MV3R-tree) [34], Trajectory-Bundle tree (TB-tree) and Spatial-Temporal R-tree (STR-tree) [35], Time-Parameterized R-tree (TPR-tree) and R^{EXP} tree [36][37], Q+R tree [38], etc.

3 2^n Tree Indexing

Every moving object has its location and orientation, which must be in a certain space. So, moving objects indexing is closely related to spatial indexing. On the other hand, all the motion data, such as position, orientation, speed, acceleration, etc., of moving objects are changing with time. Hence, the organization of motion data must take temporal indexing into consideration. Furthermore, motion data management is subject to motion data representation, i.e. moving object model. It is model dependent.

Motion State Model (MSM) [39] is based on sampling method. Comparing to other moving object models, MSM is able to be applied to 3D space while many models only deal with 2D space. Besides, MSM takes orientation of objects into model as well as position, while most other models can only manage objects' position. Therefore, MSM is chosen for moving object model, based on which 2^n index tree is studied.

As described by MSM, the motion state of a moving object can be represented by a vector with infinite elements as

$$\begin{aligned}(&t, x, x', x'', \ldots, x^{(n)}, \ldots,\\ &y, y', y'', \ldots, y^{(n)}, \ldots,\\ &z, z', z'', \ldots, z^{(n)}, \ldots,\\ &\varphi, \varphi', \varphi'', \ldots, \varphi^{(n)}, \ldots,\\ &\omega, \omega', \omega'', \ldots, \omega^{(n)}, \ldots,\\ &\kappa, \kappa', \kappa'', \ldots, \kappa^{(n)}, \ldots)\end{aligned}$$

in which (x, y, z) is the position and $(\varphi, \omega, \kappa)$ is the orientation of the moving object. And (x', y', z') is the moving speed while $(\varphi', \omega', \kappa')$ is the rotating speed. Similarly (x'', y'', z'') is the moving acceleration and $(\varphi'', \omega'', \kappa'')$ is the rotating acceleration, etc. In fact, in any application the motion state vector has limited elements, in which only part of them are needed to be indexed, which can be noted as indexed vector $\boldsymbol{c}(c_0, c_1, c_2, \ldots, c_{n-1})$ (c_0 is time stamp).

3.1 2^n Index Tree

$\boldsymbol{c}(c_0, c_1, c_2, \ldots, c_{n-1})$ can be considered as a point in R^n. It is always possible to find a range $[r_{i,\,min}, r_{i,\,max})$ and let all $c_i \in [r_{i,\,min}, r_{i,\,max})$. A hyperrectangular S in space R^n can be found as boundary of the domain of the indexed vector $\boldsymbol{c}$. The range of S in dimemsion i is $[r_{i,\,min}, r_{i,\,max})$. By making S as the root node and continuously dividing in the middle of each dimension, a 2^n tree structure can be established. The first division in each dimension of root node divides the root node into 2^n congruent small hyperrectangulars, which form the level one nodes and can be numbered as S_{k_1} (k_1=0, 1, 2, ..., 2^n-1). The following divisions of nodes level one makes nodes level two, which can be numbered as $S_{k_1k_2}$ (k_1, k_2=0, 1, 2, ..., 2^n-1), and so on so forth.

From the root node, each parent node contains pointers to its child nodes, each child node contain a pointer to its parent node. So that it is easy to navigate between different nodes. Of course, root node has no parent pointer and leaf nodes have no child pointers.

Because the motion of moving object is not necessarily uniform, the distribution of indexed vector is not even in S. Therefore, some blank nodes exist, which need not to be recorded. And the relative pointer in its parent node is not necessary.

The leaf nodes of index tree contain the motion state vectors. There are two ways to include the motion state vectors into leaf nodes, one is to record the vectors themselves, another one is to record the references to thoes vectors.

It is not difficult to write out the operation algorithm of 2^n index tree, such as establishing index tree, locating, adding and deleting motion state vector, etc. However, some issues are worth discussing.

3.2 Late Updating

Every time when a new motion state vector is added into the index tree, the leaf node which contains that vector should be checked. If the number of vectors it contains is more than the maximum number q_{max}, it should be split. Similarly, when a vector is deleted, the leaf node that contains the vector and its siblings should be check. If the node contains vectors less than the minimum number q_{min}, and the number of vectors in its parent node is less than q_{max}, it should be merged. But this checking process needs time. Especially when the system is busy in recording the vectors from large amount of moving objects, it is inefficient to check the node every time when a new vector is added. Sometimes then adding and deleting happens in the same period of time, it will make the node split and merged repeatedly. So, it is necessary to perform late updating, i.e. to check the number of vectors contained in the leaf node periodically rather than to do it every time of adding and deleting. There are different policies to perform the checking.

1. Check when system idles;
2. Check when the number of added or deleted vectors reaches certain amount;
3. Check in a certain period of time.

No matter which policy will be taken, it is necessary to mark the leaf node whenever a adding or deleting operation happens, and clean the mark after checking the node.

3.3 Pre-query

It is very easy to find a moving object satisfying certain condition with the help of 2^n index tree. But things get complicated if query relates to two or more moving objects, for example, to find out the moving objects pair whose distance between each other is less than certain amount. Such relationship query is time-consuming. But if a pre-query process can be taken, the query time will be much less.

The original idea of pre-query is to filter out the objects that are obviously not in the result set. This will help to reduce the calculation. For example, if "the closest distance" between objects in a period of time and the relative moving objects are queried, the filter process can be as follows,

a) Find out all the satisfied leaf nodes;
b) For each leaf node, if there is more than one vector within it, then all the vectors are chosen. Calculate the filter distance d_{Filter} as the shortest leaf node diagonal in position dimensions plus the longest distance that a moving object can move in the period cover by that leaf node. That is,

$$d_{Filter} = \min(d_{PositionDiagonal,i} + \Delta t_i \times speed_{\max})$$

in which, $d_{PositionDiagonal,\ i}$ is the diagonal of leaf node i in position dimensions, Δt_i is the period covered by leaf node i.

c) If there is no leaf node which contains more than one vector, then the filter distance d_{Filter} is,

$$d_{Filter} = \min(d_{PositionMax,i,j} + \max(\Delta t_i, \Delta t_j) \times speed_{\max})$$

in which, $d_{PositionMax,\ i,\ j}$ is the longest distance between leaf node i and leaf node j in position dimensions, Δt_i is the period covered by leaf node i, and Δt_j is the period covered by leaf node j.

d) For every leaf node k, calculate the shortest compare distance $d_{Min,\ k}$ with other leaf nodes,

$$d_{Min,k} = \min(d_{PositionMin,k,h} - \max(\Delta t_h, \Delta t_k) \times Speed_{\max})$$

in which, $d_{PositionMin,\ k,\ h}$ is the shortest distance between leaf node k and leaf node h in position dimensions, Δt_k is the period covered by leaf node k, and Δt_h is the period covered by leaf node h. If $d_{Min,\ k} > d_{Filter}$, then leaf node k will be discarded.

In program implementation, a table of leaf node can be made while finding satisfied leaf node in step 1. Every time, when a new satisfied leaf node is found, the longest distance between this node and the nodes already in the table $d_{PositionMax,\ i,\ j}$ can be calculated and filled into the table, as well as the shortest distance $d_{Min,\ k}$, and the longest distance that a moving object can move in the period cover by that leaf node. And every time when a new leaf node is added to the table, the latest filter distance d_{Filter} can be updated. When all satisfied leaf nodes are found, d_{Filter} is calculated out, and according to the table, unnecessary leaf node can be filtered.

The longest and shortest distance between leaf nodes is based on the distance in different position dimension. According the splitting method of 2^n index tree, there are four possible situations for two

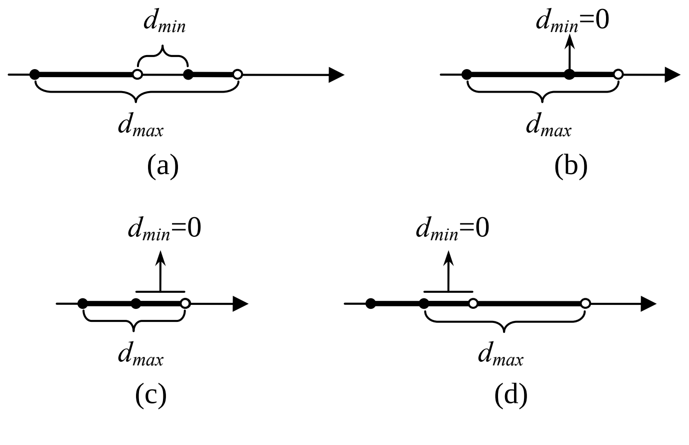

Fig. 1: Longest and shortest distance calculation in projection on one dimension

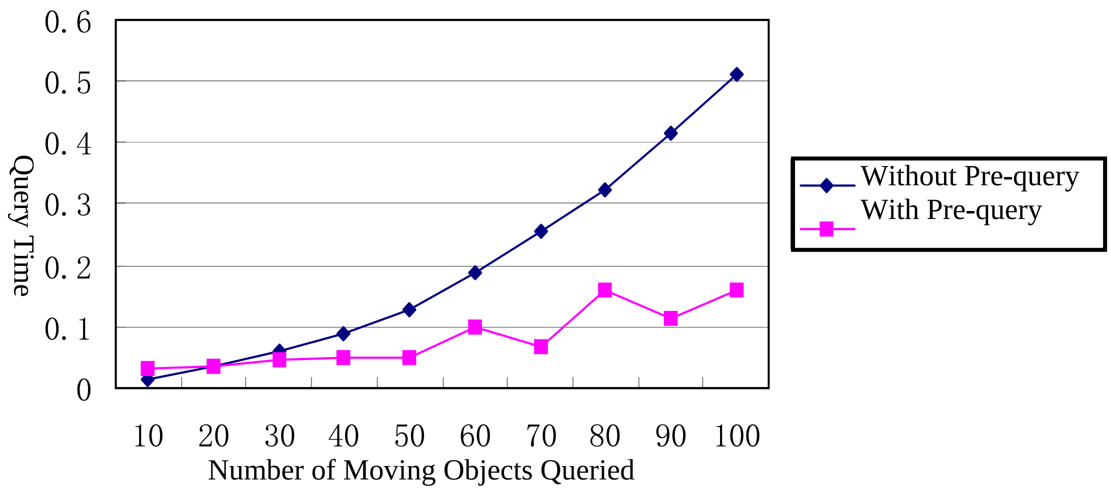

Fig. 2: Query time with and without pre-query process

different nodes projected on one dimension as shown in Fig. 1. In this way, the longest distance $d_{u,\,max}$ and shortest distance $d_{u,\,min}$ in dimension u can be easily calculated. Thus, the longest and shortest distance between two nodes can be calculated as,

$$d_{PositionMax} = \sqrt{\sum_u d^2_{u,\max}}$$

$$d_{PositionMin} = \sqrt{\sum_u d^2_{u,\min}}$$

The effect of pre-query varies in different situations. When there are few moving objects, or the query has nothing to do with motion state, pre-query does not help much, in some cases, it even make the query time longer. But when there are lots of moving objects and the query is somewhat about the motion state, pre-query can heavily reduce the query time needed. Even so, however, the efficient of pre-query is also up to the distribution of motion vectors in index tree. The percentage of filtered moving objects is not fixed in all queries. Fig. 2 illustrates this clearly, it shows time needed for queries of the shortest distance with and without pre-query process within different numbers of moving objects.

3.4 Motion Vector Reference in Index Tree

Logically, the motion vectors are contained in the index tree. But in implementation, they can be stored into a table and be referenced by leaf node of index tree. To distinguish different motion vectors, time stamps in the vectors as well as the moving objects' IDs are needed. However, in any node of 2^n index tree, the time period is specified. And the vectors falling into it must be within that period. If the time stamps in the vectors are recorded directly as part of the reference, it will bring redundancy.

If the largest number of vectors of the same object within one leaf node is p, then only $\lceil \log_2 p \rceil$ bits is needed and enough to distinguish all the vectors with different time stamp. For the same object, the motion vectors in one leaf node can be numbered with unsigned integer according to their time sequence. This time number works together with object ID can distinguish all the motion vectors in one leaf node.

For read and write efficiency, the time coding should be multiple of byte. On whether and how to use time coding in vector reference, both storage cost and read/write efficiency should be taken into consideration. For example, in MS Windows system, if time stamp is an instance of COleDateTime(), which uses 12 bytes, and in any leaf node, the vectors from the same object will not exceed 256, which can be represented by one byte, then the storage cost of index tree with time coding is less than 27% of that without time coding even when the number of moving objects reaches 16 million. This is very important when the index tree is loaded into memory when system works. It will reduce the memory space needed and reduce the page demand times.

4 Compare 2^n Tree Indexing with Other Indexing Methods

There are lots of indexing methods introduced in section 2. Each method is good in this way or that, and fits for certain application. With motion state model, 2^n tree indexing works well.

4.1 Compare with R-tree

R-tree is popularly employed indexing, no matter in spatial indexing, temporal indexing or moving object indexing which is both spatial and temporal. Most studies of index based on R-tree are focused on moving trajectories, which are represented by linear function. This representation is simple, certain and unique, thus can be easily managed by least boundary box. On the contrary, MSM is based on motion state vector, which offers different LODs representation of motion state thus is uncertain and not unique, so that can not be managed by least boundary box.

Some R-tree based indices also manage the state of moving objects. For example Q+R tree [38]. The state it manages is the last collected objects' state or the current state. It does not show the motion process in a period of time.

Besides, R-tree is very complicated with space partitioning, nodes overlay, etc. It is not as simple as 2^n tree in motion state vectors indexing.

4.2 Compare with Spatial-Temporal Grid

Spatial-temporal grid (STG) and 2^n tree are both based on space partitioning. STG can be treated as a special kind of 2^n tree, in which all leaf nodes have the same size and their splitting/merging operation happen at the same time. Locating motion vector in STG is simpler than that in 2^n tree. The cell that the motion vector falls in can be directly calculated from the vector itself, thus need not searching from the root node like in 2^n tree.

The problem comes with storage. If every cell in STG is allocated with the same space, it may bring lots of waste of space when the vector distribution is not even. If each cell is allocated with different storage space as it needs, it loses the advantage of direct locating of victors. And the numbering of cells also makes it difficult for random access.

4.3 Compare with Quadtree

Obviously quadtree is not suitable for high dimension vectors. However, one of the solutions is to establish *n*-1 quadtrees for *n*-dimensional index vectors. This may help in some certain scenarios. But generally, it brings more storage cost and more query time.

5 Acknowledgment

This research has been supported by China Scholarship Council and Startup Research Fund for Fetched in Intelligence of Zhongnan University of Economics and Law.

References:

[1]. J. Gong, *Fundamental of Geographic Information System*, Science Press (Beijing), 2001

[2]. R. A. Finkel, J. L. Bentley, Quad Trees – A Data Structure for Retrieval on Composite Keys, *Acta Informatica*, pages 1-9, 1974.

[3]. N. Beckmann, Hans-Peter Kriegel, The R*-tree: an efficient and robust access method for points and rectangles. *Proceedings of ACM SIGMOD International Conference on the Management of Data*, Atlantic City, NJ, May 1990

[4]. T. Sellis, N. Roussopoulos and C. Faloutos, The R+-tree: a dynamic index for multi-dimensional objects. *Proceedings of the 13th VLDB Conference*, Brighton 1987. pages 507-518.

[5]. A. Guttman, R-tree: a dynamic index structure for spatial searching. *Proceedings of ACM SIGMOD International Conference on the Management of Data*. 1984

[6]. D. B. Lomet and B. Salzberg, The hB-tree: a multiattribute indexing method with good guaranteed performance. ACM Transactions on Database Systems, 1990, 15(4): 625-658

[7]. J. T. Robinson, The K-D-B-tree: a search structure for large multidimensional dynamix indexes. *Proceedings of SIGMOD Conference 1981*. pages 10-18

[8]. J. H. Friedman, J. L. Bentley and R. A. Finkel, An algorithm for finding best matches in logarithmic expected time. *ACM Transaction on Math. Software (TOMS)*, 1977, 3(3): 209-226.

[9]. N. Katayama, S. Satoh, The SR-Tree: An index structure for high-dimensional nearest neighbor queries. *Proceedings of 16th ACM SIGMOD*, page 369-380. 1997.

[10]. P. N. Yianilos, Data structure and algorithm for the nearest neighbor search in general metric spaces. *Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. Austin-Texas, Jan. 1993

[11]. P. Ciaccia, M. Patella and P. Zezula, M-tree: an efficient method for similarity search in metric space. *Proceedings of the 23rd VLDB Conference*, Athens Greece, 1997

[12]. M. A. Nascimento, Efficient indexing of temporal database via B^+-trees. Ph.D dissertation, 1996

[13]. B. Salzberg, V. Tsotras, A comparison of access methods for temporal data. *ACM Computing Surveys*, 1999, 31(2): 158-221

[14]. B. Becker, S. Gschwind, T. Ohler, B. Seeger, P. Widmayer, An asymptotically optimal multiversion B-Tree. VLDB Journal 1996, 5(4): 264-275

[15]. R. Elmasri, The time index and the monotonic B^+-tree. *Proceedings of Theory Design and Implimentation in Temporal Databases*, 1993. pp. 433-456

[16]. D. Lomet, Transaction time database. *Proceedings of Theory, Design and Implementation in Temporal Databases*, 1993. pp. 388-417

[17]. V. Kouramajian, The time $index^+$: an incremental access structure for temporal databases. *Proceedings of 3rd International Conference on Knowledge and Management*, 1994. pp. 296-303

[18]. R. Elmasri, The time index: an access structure for temporal data. *Proceedings of 16th Very Large Database Conference*, Australia, 1990. pp. 1-12

[19]. R. Elmasri, Efficient implementation techniques for the time index. *Proceedings of 7th International Conference on Data Engineering*, 1991. pp. 102-111

[20]. L. Lin, Indexing value of time sequences. *Proceedings of 5th International Conference on Information and Knowledge Management (CIKM)*, Maryland, USA, 1996. pp. 223-232

[21]. Guoming Du, Study of Process Algorithm of Series Data and its Application in Urban Water Supply Network System, Ph.D Dissertation of Wuhan University, 2002.

[22]. H. Gunadhi, and A. Segev. Efficient indexing methods for temporal relations. Transactions of Knowledge and Data Engineering, 1993, 5(3): 496-509

[23]. C. P. Kolovson. Indexing techniques for historical databases. *Proceedings of Theory, Design and Implementation in Temporal Databases*, 1993. pp. 418-432

[24]. M. Nascimento, J. Silva. Towards historical R-trees. ACM SAC, 1998

[25]. A. Nanopoulos. Indexing time-series databases for inverse queries. *Proceedings of 1998 International Conference on Database and Expert System Applications*, Vienna, Austria, 1998. pp. 551-560

[26]. G. Kollios, and V. J. Tsotras. Hashing methods for temporal data. IEEE Transaction on Knowledge and Data Engineering. 2002, 14(4): 902-919

[27]. V. J. Tsotras. The snapshot index, an I/O optimal access method for timeslice queries. Information System, 1995, 3(20): 237-260.

[28]. H. Nozawa, N. Saiwaki, S. Nishida. Spatio-temporal indexing methods for moving objects for highly interactive environment. *Proceedings of 1999 IEEE International Conference on Systems, Man, and Cybernetics*, 1999. IEEE SMC '99 Conference, 12-15 Oct. 1999, Volume: 6. Pages: 7–12

[29]. H. D. Chon, D. Agrawal, A. El Abbadi. Using space-time grid for efficient management of moving objects. *MobiDE*, pp. 59-65, May 2001

[30]. H. D. Chon, D. Agrawal, A. El Abbadi. Query processing for moving objects with space-time grid storage model. *Proceedings of the International Conferenceon Mobile Data Management*, 2002

[31]. H. Samet. Spatial data structure. VLDB'97 Tutorial, Greece, 1997

[32]. R. Ding, X. Meng. A quadtree based dynamic attribute index structure and query process. *Proceedings of International Conference on Computer Networks and Mobile Computing*, pp. 446-451, 2001

[33]. J. Tayeb, O. Ulusoy, and O. Wolfson. A quadtree-based dynamic attribute indexing method. The Computer Journal, pp. 185–200, 1998

[34]. Y. Tao and D. Papdias. Mv3r-tree: a spatio-temporal access method for timestamp and interval queries. *Proceedings of the 27th International Conference on Very Large Databases*. 2001

[35]. D. Pfoser, C. S. Jensen, and Y. Theodoridis. Novel approaches in query processing for moving objects. *Proceedings of the 26th International Conference on Very Large Databases (VLDB)*, 2000

[36]. S. Saltenis, C. S. Jensen, S. T. Leutenegger, M. A. Lopez. Indexing the positions of continuously moving objects. *Proceedings of ACM SIGMOD Conference 2000*

[37]. S. Saltenis, C. S. Jensen. Indexing of moving objects for location-based services. *the 18th International Conference on Data Engineering (ICDE'02)*, pp. 463-472, 2002

[38]. Y. Xia, S. Prabhakar. Q+Rtree: efficient indexing for moving object databases. *Proceedings of the 8th International Conference on Database Systems for Advanced Applications (DASFAA 2003)*, pp. 175 -182, 2003

[39]. H. Ye, J. Gong, J. Pan, Y. Chen, Representation, Indexing and Retrieval of Moving Objects, *Proceedings of Storage and Retrieval Methods and Applications for Multimedia 2004*, SPIE Vol.5307: 158-166.