Rend 3D R-tree : 3D R-tree 기반의 이동 객체 데이터베이스 색인구조 연구

임 향 초*, 임기욱**,남지은**,이경오**
*중경우전대학 자동화학과
**선문대학교 전자계산학과
e-mail : renxiangchao_2006@yahoo.com, , rim@sunmoon.ac.kr, njy1204@sunmoon.ac.kr, leeko@sunmoon.ac.kr

Rend 3D R-tree: An Improved Index Structure in Moving Object Database Based on 3D R-tree

Ren XiangChao*, Kee-Wook Rim**, Nam Ji Yeun**,Lee KyungOh**
*Dept. of Automation, Chongqing University of Posts and Telecommunication
**Dept. of Computer Science, Sun Moon University

Abstract

To index the object's trajectory is an important aspect in moving object database management. This paper implements an optimizing index structure named Rend 3D R-tree based on 3D R-Tree. This paper demonstrates the time period update method to reconstruct the MBR for the moving objects in order to decrease the dead space that is produced in the closed time dimension of the 3D R-tree, then a rend method is introduced for indexing both current data and history data. The result of experiments illustrates that given methods outperforms 3D R-Tree and LUR tree in query processes.

Keywords: moving object database, 3D R-Tree, spatiotemporal, indexing

1. Introduction

With the development of the related technologies such as mobile computing, Global Positioning System, GIS, etc., database needs store and manage large quantities of physical objects with spatiotemporal information in physical world, and their spatial location or scope will change continuously with time flies, which brings a big challenge to the spatiotemporal database. The application of spatiotemporal database spreads over many domains as transportation (such as vehicle supervision), weather monitoring, military, LBS etc. In such applications, a dynamic index is often built to improve the evaluation of spatial queries.

Much work has been recently conducted in the domain of indexing spatiotemporal data and several spatiotemporal access methods have been proposed, which can be separated into two major categories: those indexing the past positions of moving objects(3D R-Tree [1], HR-Tree [2], TB-Tree, STR-Tree [3]), and those indexing the current and predicted movement of objects [4].The typical history query that is supported by the server terminal is district query ,which include time query that is " find all mobile objects passing through district S at time t, and window query that is find all mobile objects passing through district S between time [t_1,t_2].

The 3D R-tree [1] considers time as an extra dimension and represents 2D rectangles with time intervals as three-dimensional boxes. This tree can be the original R-tree or any of its variants [7]. 3D R-tree can manages history information effectively and has a better performance on window query than other index structure. However, the original 3D R-tree can only index history data (offline data) that is the data in condition of interval value of data item are confined and require data to be closed in time dimension. The 3D R-tree avoids of the differences of spatial query and temporal query; when object keep stationary in some time interval, it will format a cube, but for the stationed in a long time, the appearance of many cubes will make the indexing quality drop heavily. This paper attempts to reconstruct the MBR (minimum bounding rectangle) to improve the indexing quality and rend the 3D R-tree in order to index the current data.

2. Related work

The research target of moving object database is to extent database technique and makes it not only be used to represent various moving objects in database, but also used to do the query processing related to object position. Nowadays, among amount of indexing structures supporting query processing for object position, 3D R-tree as one member of R-tree family has a simple structure and has a good performance for spatial query.

The problems of dead space and overlap exist in almost all indexing structure based on classic R-tree family, proposed by [6]. How to decrease or eliminate them as much as possible is the key. For 3D R-tree, its indexing key is MBR(minimum bounding rectangle) of data ,if the time interval [t_1, t_2] gets a much longer, the data MBR will gets longer too ,then the dead space will increase much more,

which obviously makes a bad effective on its indexing performance.
3D R-tree employs a unified indexing structure in which only one spatial data structure needs to be implemented and the factor of time is eliminated because spatio-temporal operators are implemented as three-dimensional queries and retrieved using the three-dimensional index. But at the mean time, the simple combination for 1D and 2D area and the closed time characteristic of this mechanism result to the 3D R-tree can only used to index the history query.

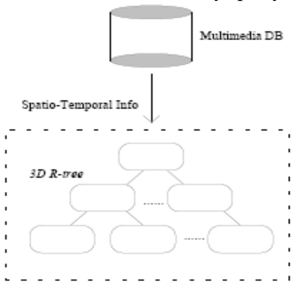

Figure 1.a unified indexing scheme of actors. [1]

The 3D R tree approach assumes that both ends of the inte rval [t_1, t_2] of each rectangle are known and fixed. If the end t ime t_2 is not now, this approach does not work well. If we wa nt make the 3D R tree can index the current data, it involves a time *now*, what is *now*, refer to [5]. Simplify, we can see no w as a near future. In this paper, I divide the time range into t wo parts, one is [t_1, now], the terminal time value is unknown ; the other is [now, t_2], the terminal time value is known.

In order to support the 3D R-tree to index the active data, 2+3 R tree is one possible solution, the 2+3 R tree approach is a variation of an original idea proposed in [9], as long as the end time t_2 of an object interval is unknown, it is indexed by the front 2D R-tree, keeping the start time t1 of its position along with its id. When t_2 becomes known, then:

1. The associated entry is migrated from the 2D R-tree to the 3D R-tree.
2. A new entry storing the updated current location is inserted into the 2D R-tree.

However, the above method requires both trees need to be searched, which increase the overhead and insert cost.

3. The implementation

In this paper, a time period update method is introduced to decrease the dead space. Fist of all, some conditions are the prerequisite:
Assumption 1: The moving objects will move in a fixed space.
Assumption 2: The data update is processed through a stable update cycle, which means every moving step of the objects just changes according to the time update cycle. This also means the moving object has the same speed when moving.

Compared with rational database, moving database has a large number of data because it continuously accumulates data according to time update. So the object is spatiotemporal object.
Definition 1: object is a spatiotemporal object that changes (still or move) according to time update and marked only by a series (id, t, c, x, y). Id is the label code of moving object; t is the time that data produce; c is the MBR of the object in indexing structure; x, y is the space coordinates.

In this paper, a rend method that combine the two trees into a whole index structure is introduced, the later one is used to index the current data, the front one uses improved 3D R-tree for indexing the history data. The current data and history data are stored independently. When some a spatiotemporal data become history data, transfer [t_1, now] to [t_1, t_2], then processing form the front tree to later tree, and for the insert processing, if the node that is chosen to insert overflows, considering the split in the time axis priority.

Indexing for 3D R-tree, the stored data item is trajectory item, which is a segment that formatted by the moving of object between [t_1, t_2], if the interval time between the t_1 and t_2 is too long, the dead space will increase heavily, in order to solve this problem As the prerequisite given above, assuming a moving object O (id, t, c, x, y), its basic update cycle is T, It updated at time t_2 to get the point p1(id,t_1,c_1,x_1,y_1), then updated again to reach the point p2(id,t_2,c_2,x_2,y_2), in traditional 3D R-tree method, we can get the two segment op1 and op2 and the corresponding MBR, R1 and R2. Now that the velocity is stable and has a basic update cycle T, so the basic update number is n_1= t_1/T and n_2= (t_2-t_1)/T; Then we can get N= (n_1+n_2) numbers update spot and will reconstruct MBR of every basic update spot. The demonstration is showed in following figure 1 and figure 2:
Assume n_1=1; n_2=2; leaf node data capacity of the established 3D R-tree is 2

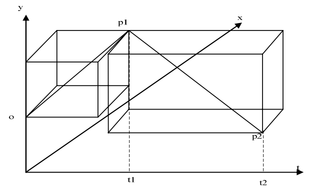

Figure 2: the original MBR construct method of 3D R-tree

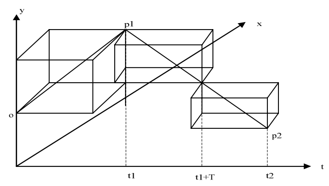

Figure 3: construct MBR of leaf node at update period T

Compare with the two conditions above, we can see that the rectified method can reduce unnecessary search which comes from dead space when the query is issued. However, at the same time, the method that uses sampling point to construct MBR to replace of realistic trajectory will add the overhead for the query processing, in the next part, Rend 3D R-tree's data structure will be introduced and combined with front method, at the end , we will give the compared experiment result.

The basic idea of Rend 3D R-tree is to employs the rend method that constructs a combined indexing mechanism. As we all know, there are two kinds of spatiotemporal data items: active data and history data. Here, we will rend the original 3D R-tree to two trees; R1 tree that remains the characteristics of 3D R-tree is used to index the history; the other tree R2 is used to index store active data. The R2 tree data structure is denoted by series (id, c, t_1, t_2), t_2=now; R1 tree data structure is also denoted by series (id, c, t_1, t_2,), but the value of t_2 is known, c is the MBB of the corresponding leaf node of moving object.

According to assumption 2, the data of moving objects updates every basic period time T, so for the update of the time interval [t_1, now], we just transfer now to defined value $t_2=t_1+N\times T$ (N is positive integer). Update current data item in R2, and then reinsert the new data item into the index structure R1.

Update algorithm:

Step 1: find the leaf node including moving point o_id from R1 tree, if not find, end. // using the same algorithm of R-tree//

Step 2: set now=t_2; t_1=now; t_2=t_1. //after time period T, update the new data item in R2//.

Step 3: implement the reinsert R1 algorithm and reinsert R2 algorithm.

Step 4: if update, repeat step1 at the time "*now*", else if, finis h.

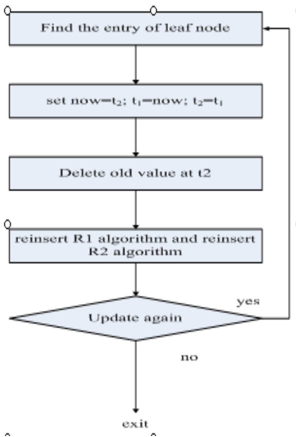
Find the entry of leaf node
set now=t₂; t₁=now; t₂=t₁
Delete old value at t2
reinsert R1 algorithm and reinsert R2 algorithm
Update again
yes
no
exit

Figure 4: update algorithm

We need twice reinsert in this process. The insertion operation probably involves the issue of split. When the MBB of nodes changes, May including root and leaf nodes, split occurs. Here, a problem is existed, the original split algorithm of 3D R-tree often causes the low page utilization of old node after splitting because of biasing to insert into the new node. We will demonstrate the rent method that split the old node to two nodes distract in different trees R1 and R2 along time axis, then improve the index ability.

Reinsert algorithm (R1, R2)

Step 1: if the number of root node is less than 2, transfer into step 3

Step 2: at time slice ΔT, find the leaf node into which the object is inserted.// the same sub-tree choose algorithm of R-tree.//

Step 3: if node overflows, rend the node into two new nodes at update moment, the front for R1; the later for R2.

Step 4:twice reinsert operations at t_{end} and $t_{end}+\Delta T$。

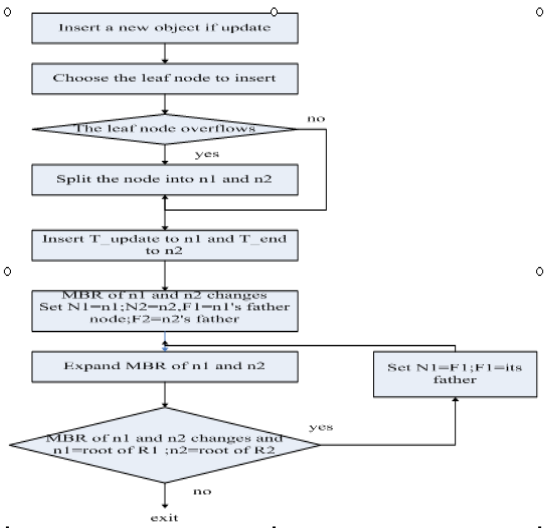
Insert a new object if update
Choose the leaf node to insert
The leaf node overflows
no
yes
Split the node into n1 and n2
Insert T_update to n1 and T_end to n2
MBR of n1 and n2 changes Set N1=n1;N2=n2,F1=n1's father node;F2=n2's father
Expand MBR of n1 and n2
Set N1=F1;F1=its father
MBR of n1 and n2 changes and n1=root of R1 ;n2=root of R2
yes
no
exit

Figure 5: reinsert algorithm

Assume that we have established a combing Rent 3D R-tree structure and N=1. R1 is used in [t_1,$t_{end}-\Delta T$], R2 is used in [$t_{end}-\Delta T$, now]. If data updates at tend after basic period T, we will reinsert the value of tend to the end part of the R1 and the value of the point $t_{end}+\Delta T$ to now for R2. At the mean time, delete the moving point that reaches terminal spot from R2.

4. Experiment evaluation

In this section, we will compare the rent 3D R-tree with 3D R-tree for history query and LUR (lazy-update) R-tree for current query. LUR tree is used to index the current position of moving objects and doesn't store the history data, once updating, it deletes the old value as well as insert the new value. We used GSTD [8] generator to generate the datasets for the experiments. The initial distribution of moving objects is Gaussian and the movement of objects is random. The datasets contain regions with density 0.2. All the experiments were performed on a Pentium IV 2G machine with 512MB memory, running Windows XP. For all experiments, the disk page size is set to 2k bytes.

Experiment processing is following: now that we have a basic update period T, we tested the 10 update point [T, 10T], and set T=10s; the spatial universe is a square that consists N=10,000 simulated points. For the history query, at every update time, produce two query asking time slice query and window query randomly, both of them have the same probability. Set the number of timestamp included in time interval to 1.From figure 3, we can see that as the update interval time get longer, the leaf node dead of 3D R-tree increase respondent, then it processes a large number of ineffective compare computing to judge whether there are overlapping or not, which result to the increment of I/O access. Compared with LUR-tree for current query, as the update time moves, once the object get out of its MBR, MBR expands to contain it in the condition of short distance between them or delete-insert operation, which result to LUR-tree will gradually get the worse performance, our experiment proved this and at the later part of updated time, Rend 3D R-tree has a better performance.

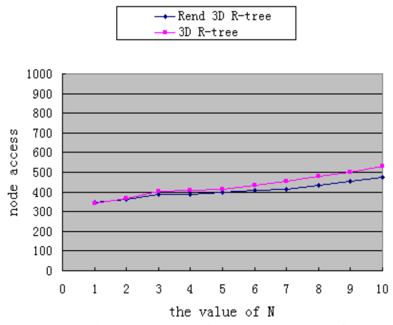

Figure 3.comparison of Rend 3D R tree and 3D R-tree for history query

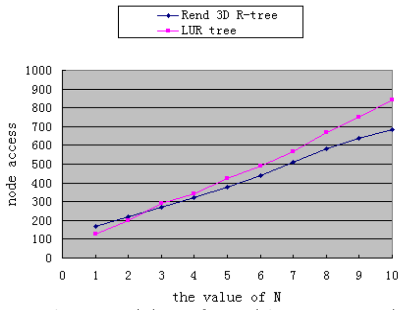

Figure 4.comparision of Rend 3D R-tree and LUR-tree for current query

5. Conclusions

This paper has proposed a combined index structure Rend 3D R-tree based classic 3D R-tree. In the prerequisite condition of the periodic update, we give a solution to decrease the much more dead space if interval time is too long for original 3D R-tree. A rend method has been introduced to establish the combined index structure to index both history and current position. The experiment proved that the proposed method had a better performance for history query with original 3D R-tree as well as current query compared with LUR-tree.

Acknowledgments:

This research was supported by the MIC (Ministry of Information and Communication), Korea, under the ITRC (Information Technology Research Center) support program supervised by the IITA (Institute of Information Technology Advancement) (IITA-2007-C1090-0701-0020)

References

[1] Theodoridis, Y., Vazirgiannis, M., and Sellis, T.: Spatio-temporal indexing for Large Multimedia Applications. In Proceedings of the 3rd IEEE Conference on Multimedia Computing and Systems, Hiroshima, Japan, 1996

[2] Nascimento, M.A., and Silva, J.R.O.: Towards historical R-trees. In Proceedings of the 13thACM Symposium on Applied Computing (ACM-SAC'98), 1998

[3] Pfoser D., Jensen C.S., and Theodoridis, Y.: Novel Approaches to the Indexing of Moving Object Trajectories. In Proceedings of the 26th International Conference on Very Large Databases, Cairo, Egypt, 2000.

[4] Saltiness, S., Jensen, C.S., Leutenegger, S.T., and Lopez, M.A.: Indexing the Positions of Continuously Moving Objects. TIMECENTER Technical Report, TR-44, 1999

[5] J. Clifford, C.E. Dyresom, T. Isakowitz, C.S. Jensen and R.T. Snodgrass: "On the Semantics of 'now'", ACM Transactions on Database Systems, Vol.22, No.2, pp.171-214, 1997.

[6] Guttman, A. (1984) R-tree: A dynamic index structure for spatial searching. In SIGMOD '84, Proceedings of the ACM SIGMOD Conference. ACM Press.

[7] R-trees Have Grown EverywhereYANNIS MANOLOPOULOS, ALEXANDROS NANOPOULOS ,APOSTOLOS N. PAPADOPOULOS Aristotle University of Thessaloniki, Greece and YANNIS THEODORIDIS University of Piraeus, Greece ACM Computing Surveys, Vol. V, No. N, Month 20YY

[8]GIST: a generalized search tree for secondary storage [CP/OL], http://gist.cs.berkeley.edu/gist.html.

[9]. A. Kumar, V.J. Tsotras and C. Faloutsos: "Designing Access Methods for Bitemporal Databases", IEEE Transactions on Knowledge and Data Engineering, Vol.10, No.1, pp.1-20, 1998.