A Storage Structure of Geometric Data with Detail Levels

Joon-Hee Kwon[1] and Yong-Ik Yoon[2]
[1] Department of Computer Science
Sookmyung Women's University,
53-12 Chungpa-dong 2-ga, Yongsan-Gu, Seoul, Korea
Tel. +82-02-710-9431, Fax.: +82-02-710-9296
[2] Department of Computer Science
Sookmyung Women's University,
53-12 Chungpa-dong 2-ga, Yongsan-Gu, Seoul, Korea
e-mail : kwonjh24@hotmail.com, yiyoon@sookmyung.ac.kr

Abstract: This paper proposes a new dynamic storage structure and methods for geometric data with detail levels. Using geometric data with detail levels, we can search geometric data quickly. However, the previous structures for detail levels form the bottleneck in the design of database and do not support all types of geometric data with detail levels. Our structure supports all types of geometric data with detail levels. Moreover, our structure does not form bottleneck in the design of database. This paper presents the structure and algorithms for searching and updating of geometric data with detail levels. Experiments are then performed.

1. Introduction

An important requirement in GIS (Geographic Information Systems) is the ability to display numerous geometric data swiftly onto the display window[1]. Geometric data with detail levels enables the swift display of large geometric data. In order to display geometric data swiftly, a storage structure for geometric data, especially a spatial indexing structure is needed.

It turns out that the integrated storage of geometric data with detail levels in the previous spatial indexing structure forms the bottleneck[2]. The approach might be to define a discrete number of levels of detail and store them separately each with its own spatial indexing structure. Though fast enough for interactive applications, it introduces redundancy because some objects have to be stored at several levels[3].

A few spatial indexing structures, that provide some limited facilities for geometric data with detail levels, are known : the Reactive-tree[3, 4], the PR-file[5] and the Multi-scale Hilbert R-tree[1]. However, the structures do not support all types of geometric data with detail levels.

This paper introduces a new storage structure of geometric data with detail levels. In the proposed method, geometric data with a number of levels of detail is integrated into a single storage structure. The integrated structure enables an integrated access of geometric data with detail levels. Moreover, our structure does not introduce redundancy as compared with the previous approaches.

The remainder of this paper is organized as follows. Chapter 2 surveys related works. Chapter 3 describes the overview and the structure. Chapter 4 and Chapter 5 present the algorithms and experiments of the proposed structure. Finally, Conclusions are made.

2. Related Work

2.1 Storage Structure of Geometric Data

In order to handle geometric data efficiently, database system needs an indexing structure that will help it retrieve data items quickly according to their geometric locations. Numerous spatial indexing methods for geometric data are known. Spatial indexing methods are classified in hierarchical methods and hashing based methods[6]. The R-tree[7, 8, 9, 10] and the Quad-tree[11, 12] are based on hierarchical methods. The Grid-file[13] and the R-file[14] are based on hashing based methods.

Among index structures known, the R-tree is the most popular. The R-tree is based on the smallest aligned n-dimensional rectangle enclosing an object. The R-tree is a direct extension of B-trees in k-dimensions. The structure is a height-balanced tree that consists of intermediate and leaf nodes. Data objects are stored in leaf nodes and intermediate nodes are built by grouping rectangles at the lower level.

Let M be the maximum number of entries that will fit in one node and let m $\leq$M/2 be a parameter specifying the minimum number of entries in a node. The R-tree satisfies the following properties.

1. Every leaf node contains between m and M index records unless it is the root.
2. For each index record (I, tuple-identifier) in a leaf node, I is the smallest rectangle that spatially contains the n-dimensional data object represented by the indicated tuple.
3. Every non-leaf node has between m and M children unless it is the root.
4. For each entry (I, child-pointer) in a non-leaf node, I is the smallest rectangle that spatially contains the rectangles in the child node.
5. The root node has at least two children unless it is a leaf.
6. All leaves appear on the same level.

2.2 Geometric Data with Detail Levels

The concept of geometric data with detail levels is related to one of the main topics in cartographic research, map generalization, that is, to derive small scale maps(large regions) from large scale maps(small regions)[3]. A number of generalization techniques for geometric data have been developed[15]. The generalization techniques

are a simplification, a combination, a symbolization, a selection, a exaggeration, and a displacement.

However, research on an generalization is going on and it is a non-trivial problem. Therefore, research on a generalization is focused on relatively simple techniques.

2.3 Storage Structure of Geometric Data with Detail Levels

A few storage structures of gemetric data that provide some limited facilities for geometric data with detail levels, that is the Reactive-tree[3, 4], the PR-file[5] and the Multi-scale Hilbert R-tree[1] are known. However, the methods have deficiencies that they support only a selection operation and a simplification operation.

The Reactive-tree : The Reactive-tree assigns an importance value to each spatial data, and each object is stored in a level according to its importance values. An importance value represents the smallest scale map in which the spatial data is still present. Less important objects get lower values while the more important objects get higher values. It is based on the R-tree. In the Reactive-tree, important objects are stored in the higher levels of the tree. The drawback of the Reactive-tree is that it supports only a selection operation of all generalization operations.

The PR-file : The PR-file(Priority Rectangle File) was designed to efficiently store and retrieve spatial data with associated priority number. Each priority number corresponds to a scale in the map. It is based on the R-file. Unlike the Reactive-tree, an object in a PR-file is not stored as an atomic unit. The PR-file makes use of a line simplification algorithm, which will select some of the line segment endpoints from a polyline according to the desired scale. The drawback of the PR-file is that it supports only a simplification operation of all generalization operations and performs poorly with data distribution that is non-uniform.

The Multi-scale Hilbert R-tree : The Multi-scale Hilbert R-tree is similar to the PR-file. The main difference is that geometric objects in a Multi-scale Hilbert R-tree are decomposed and stored as one or more sub-objects in the main data file. It is based on the R-tree, especially in the Hilbert R-tree. For a simplification operation, the storage structure makes use of a line simplification algorithm, that is a modified version of the Douglas-Peuker algorithm[16]. For a selection operation, the storage structure selects objects based on the size. The drawback of the Multi-scale Hilbert R-tree is that it supports only a selection operation and a simplification operation of all generalization operations.

3. Storage Structure of Geometric Data with Detail Levels

3.1 Overview

To represent geometric data with detail levels after all generalization, we can use only the storage structure of geometric data, except for the Reactive-tree, the PR-file, and the Multi-scale Hilbert R-tree. In this paper, we selected the R-tree, because of popularity.

In the approach of the R-tree for geometric data with detail levels, geometric data with detail levels is stored independently, each with its own storage structure. It introduces redundancy because the data in coarse levels has to be stored at detailed levels redundantly.

In our approach, multiple R-trees for each detail levels are integrated into the single structure. Moreover, our approach does not introduce redundancy because an object is stored in the only once and all levels of the object are represented by a composite level value. Figure 1 shows the overall structure. Notice that multiple R-trees are integrated by detail level node.

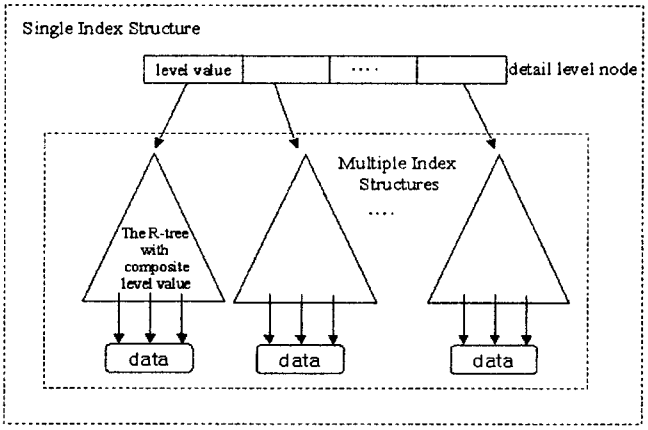

Figure 1. Overall Storage Structure

3.2 Structure

Our storage structure has detail level nodes and the R-trees with the composite level value. A detail level node is the node that has a level value and points the R-tree corresponding to each level value. The R-tree with the composite level value is the tree that adds a composite level value in the R-tree node.

A detail level node has the form (entries, p_d), where p_d is the pointer pointing to the next detail level node. The entries have the form (d, p_r), where d is the detail level value and p_r is the pointer pointing to a root node of the R-tree.

A non-leaf node of the R-tree has entries of the form (p, MBR, c), where p is the pointer pointing to child nodes in the R-tree node, MBR is the minimal bounding rectangle that covers all rectangles in the lower node's entries, and c is the result of bitwise-OR calculation of all composite level values that is used in the lower node's entries. A leaf-node of the R-tree has entries of the form (id, MBR, c), where id is the pointer pointing to an object, MBR is the minimal bounding rectangle that covers an object, and c is the composite level value, that is the result of the bitwise-OR calculation of all level values that the object appears.

3.3 Properties

Our index is based on the R-tree. Therefore, most of the properties are similar to the properties of the R-tree. However, the following properties are different with the R-tree. Let M_d be the maximum number of entries that will fit in the detail level node.

1. Every node in the detail level nodes contains between 1 and M_d index records.

2. An object is appeared in a tree rooted by a node pointed by entry with the smallest level value that the object is appeared.
3. Each entry of a detail level node is sorted from small level value to large level value.
4. The composite level value of each entry of the R-trees is the result of bitwise-OR calculation of all composite level value that is used in the lower node's entries.

4. Searching and Updating of Geometric Data with Detail Levels

4.1 Searching

The searching algorithm is similar to the searching algorithm of the R-tree, except that it searches the detail level nodes and the composite level values of the R-trees. Firstly, the index tree finds root nodes of the R-tree. And then, for each of those search all nodes until all data corresponding to the level value searched, are found. Figure 2 shows the process that searches data with level 2 in our structure.

Firstly, our structure searches the R-tree corresponding to level 2. The detail level value in the detail level node is set by the following algorithm. The smallest detail level value is 1. As the amount of detail is larger, a detail level value is shifted left. Therefore, the detail level value is '10' in level 2. The detailed algorithm is following as : for each entry (d, p_r), in detail level nodes, check if d is coarser level than the level value '10' or d equals to the level value '10'. Searched R-tree is the trees rooted by node N2 and node N3.

Secondly, our structure searches each entry (p,MBR,c) of nodes in the searched R-tree, where bitwiseAND(c,10) = 10. The detailed algorithm is following as : for each entry (p,MBR,c) of the nodes pointed by the searched R-trees, perform bitwiseAND (c,'10') calculations. If the node is a non-leaf node, perform to the lower nodes recursively. If the node is a leaf node, return results.

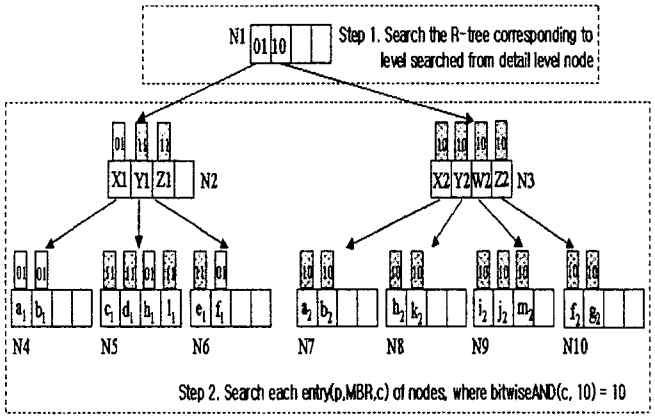

Figure 2. Searching Process

4.2 Updating

To change the structure dynamically, updating is needed. Our structure is a fully dynamic storage structure. In this section, a insertion algorithm and a deletion algorithm are presented for updating our structure.

The way of inserting an object in our storage structure is classified into two cases. Firstly, an object is modified from an object in the coarser level. This is a result of a operation excluding a selection operation. This case is similar to the insertion algorithm of the R-tree, except for processing composite level values. Secondly an object is added from an object in the coarser levels. This is a result of a selection operation. This process is modifying composite level values of objects in the coarser level.

The way of deleting an object in our storage structure is classified with two cases. The first is that an object is deleted completely in our structure. This case is similar to the deletion algorithm of the R-tree, except for processing composite level values. The second is to delete corresponding the level value from the composite level value of an object that is needed to be deleted. To delete the level value, bitwiseAND (c,bitwiseNOT(L)) calculation is performed and propagated to the root node of the modified R-tree, where c is the composite level value of an object to be deleted and L is the level value to be deleted.

5. Implementation and Experiments

5.1 Implementation and Setup

For experiments, we implemented both our storage structure and the R-tree. All programs were written in GNU C++ language on Cygwin running on Windows. Cygwin is the UNIX environment, developed by Red Hat, for Windows[17].

We used synthetic data. The test data sets are consisted of 5 different sets varying the total number of objects, called D1, D2, and D3. The total number of objects is varied from 60,000 to 300,000, that is 60,000, 180,000, and 300,000. The number of detail levels in generated data is 5. The number of objects in detailed level is larger than the number of objects in coarser level. For each level, the number of data is level value by 4,000(12,000, 20,000) in data set D1(D2 and D3 respectively). All data is generated by random generator. Coordinates of generated data are from (0, 0) to (10,000, 10,000). For the number for each level, the number of objects added from coarser levels is the number of all objects in coarser levels.

For window query on test data, we generated 10,000 window queries. The window query size is the 10,000 divided by a level value in each level.

5.2 Experiment

We evaluated the average number of nodes read for search performance comparison and evaluated the total storage capacity for memory capacity comparison.

Figure 3 shows the average number of nodes for test queries on test data. We found that number of nodes read of our structure is approximately equivalent to the number of nodes read of the multiple R-trees.

Figure 4 shows the total storage capacity for each structure on test data. The total storage capacity of the multiple R-trees is the larger than our storage structure, which was resulted from redundancy.

As a result of evaluation on test data and real data, our storage structure shows good performance both in searching

and in memory capacity compared as the R-tree. First, for search performance our index tree is approximately equivalent to the R-trees for each level. Second, for the memory capacity, our storage structure is smaller than the R-tree, which was resulted from not redundancy.

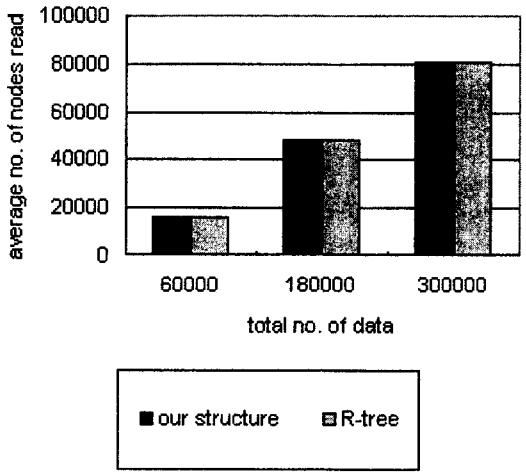

Figure 3. Result of Experiment : Search

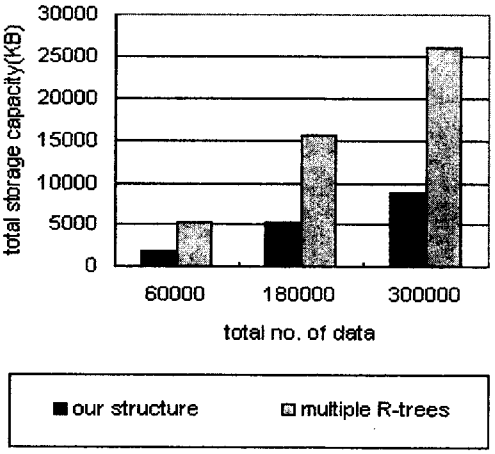

Figure 4. Result of Experiment : Memory

6. Conclusion and Future Works

The ability to access large amount of geometric objects quickly is important. Geometric data with detail levels enables the quick access.

We proposed a new effective storage structure geometric data with detail levels. Our storage structure is a fully dynamic structure. We presented the structure and algorithms. We implemented our structure and performed evaluation. The results showed the good search performance and low memory capacity.

The contribution of this paper is following. First, the proposed method supports results of all types of geometric data with detail levels efficiently. Until now, any spatial index structure supporting of all types of geometric data with detail levels efficiently does not exist. Second, as compared with the R-tree, our structure decreases memory capacity and reduces search performance.

For future works, we will perform experiments on more various types of parameters on test data.

References

[1] P.F.C Edward, K.W.C Kevin, "On Multi-Scale Display of Geometric Objects", *International Journal on Data and Knowledge Engineering,* 40(1), pp.91-119, 2002.

[2] S.C.Guptill, "Speculations on seamless, scaleless cartographic data bases", *Proceedings of Auto-Carto*, 9, pp.436-443, 1989.

[3] P.V. Oosterom, "The Reactive-tree : A Storage Structure for a Seamless, Scaless Geographic Database" , *Proceedings of Auto-Carto,* 10, pp.393-407, 1991.

[4] P.V. Oosterom, V. Schenkelaars, "The development of an Interactive multi-scale GIS", *International Journal of Geographic Information Systems,* pp.489-507, 1995.

[5] B. Becker, H-W. Six, and P. Widmayer, "Spatial Priority Search : An Access Technique for Scaless Maps", *Proceedings of ACM SIGMOD,* Denver, Colorado, pp.128-137. 1991.

[6] V. Gaede, O. Gunther, "Multidimensional Access Methods" , *ACM Computing Surveys*, 30(2), pp.170-231, 1998.

[7] A. Guttman, "R-trees : A Dynamic Index Structure for Spatial Searching", *Proceedings of the ACM SIGMOD International Conference on Management of Data,* pp.47-54, Boston, Ma., 1984.

[8] T. Sellis, N. Roussopoulos, and C. Faloutsos, "The R+-tree : A Dynamic Index for Multidimensional Objects" , *Proceedings of the 13th International Conference on VLDB*, pp.507-518, Brighton, England, 1987.

[9] N. Beckmann, H.P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree : an Efficient and Robust Access Method for Points and Rectangles", *Proceedings of ACM SIGMOD International Conference on Management of Data,* pp.322-331, Atlantic City, NJ, 1990.

[10] I. Kamel, C.Faloutsos., "Hilbert R-Tree : An Improved R-Tree Using Fractals", *Proceedings of 20th VLDB,* Santiago de Chile, Chile, pp.500-509, 1994.

[11] H. Samet, "The Quadtree and Related Hierarchical Data Structure", *ACM Computing Surveys,* 16(2), pp.187-260, 1984.

[12] H. Samet, R.E. Webber, "Storing a collection of polygons using quadtrees" , *ACM Transactions. Graph,* 4(3), pp.182-222, 1985.

[13] J. Nievergelt,, H. Hinterberger, and K.C. Sevcik, "The Grid file : An adaptable, symmetric multikey file structure", *ACM Transactions on Database Systems,* 9(1), pp.38-71, 1984.

[14] A. Hutflesz, H-W. Six, and P. Widmayer, "The R-file : An Efficient Access Structure for Proximity Queries" , *Proceedings of IEEE 6th International Conference on Data Engineering,* pp.372-379, Los Angeles, CA, 1990.

[15] K.S. Shea and R.B.McMaster, "Cartographic generalization in a digital environment: When and how to generalize", *Proceedings of Auto-Carto*, 9, Baltimore, pp.56-67, 1989.

[16] D.H. Douglas, T.K.Peucker, "Algorithms for the Reduction of Points Required to Represent a Digitized Line or its Caricature", *Canadian Cartography*, 10, pp.112-122, 1973.

[17] http://www.cygwin.com.