'97 International Conference Multimedia Databases on Internet
October 10, 1997, Seoul Korea

`A Study of Efficient Access Method based upon the Spatial Locality of Multi-Dimensional Data

Yoon, Seong-young Joo, In-hak Choy, Yoon-chul

Multimedia/Graphics Lab.
Yonsei University, Computer Science Dept.
Seodaemun-gu Shinchon-dong 134
Seoul, Korea, 120-749
E-Mail : yunix@rainbow.yonsei.ac.kr

ABSTRACT

Multi-dimensional data play a crucial role in various fields, as like computer graphics, geographical information system, and multimedia applications. Indexing method for multi-dimensional data is a very important factor in overall system performance. What is proposed in this paper is a new dynamic access method for spatial objects called HL-CIF(Hierarchically Layered Caltech Intermediate Form) tree which requires small amount of storage space and facilitates efficient query processing. HL-CIF tree is a combination of hierarchical management of spatial objects and CIF tree in which spatial objects and sub-regions are associated with representative points. HL-CIF tree adopts "centroid" of spatial objects as the representative point. By reflecting objects' sizes and positions in its structure, HL-CIF tree guarantees the high spatial locality of objects grouped in a sub-region rendering query processing more efficient.

Keywords : spatial indexing, spatial access, GIS, index tree, computer graphics.

1.Introduction

Recently, spatial data processing, widely used in computer graphics, geographical information system, multimedia application, and CAD, has emerged as one of the most important issues in the studies of data manipulation. Indexing method is crucial in terms of determining overall performance in database management system. However, conventional indexing methods which handle one-dimensional alphanumeric data has proven to be inefficient in its application to spatial data whose sizes and positions together should be taken into consideration in indexing system.

HL-CIF(Hierarchically Layered Caltech Intermediate Form) tree is proposed as a solution to this problem and to provide a new dynamic access method for spatial objects. HL-CIF tree is more efficient because it requires smaller amount of storage space and facilitates efficient query processing. HL-CIF tree combines hierarchical management of spatial objects and CIF tree, which associates spatial objects with sub-regions by representative points. HL-CIF tree adopts "centroid"

of spatial objects as the representative point. By reflecting objects' sizes and positions in its structure, HL-CIF tree guarantees the high spatial locality of objects grouped in a sub-region making query processing more efficient.

This paper is organized as follows. Section 2 consists of various spatial indexing methods and defines the properties for efficient spatial indexing method. Section 3 provides detailed explanation of the proposed HL-CIF tree with its theoretical background, definition, and manipulation algorithms. The performance evaluation of the HL-CIF tree compared to R^+-tree and MX-CIF tree is presented in Section 4. In section 5, conclusion and issues for further studies are suggested.

2. Related Studies

2.1 Classification of spatial indexing method

Established spatial indexing methods can be categorized into object mapping/transformation method, object duplication/c'' ›ping method, and object overlapping method according to its underlying principle of handling spatial data.

2.1.1 Object mapping/transformation method

In multi-dimensional data processing, point object can be handled in the same way as one-dimensional data. Any spatial object in N-dimensional space can be represented by a point object in 2N-dimensional space. For the purpose of utilizing this advantage of point data, object mapping/transformation method transforms given spatial objects to point objects. The transformed objects can be manipulated by any of conventional indexing methods such as B-tree. Grid-file and MLGF(Multi-Layer Grid File) are the examples of this method.

Instead of transforming N-dimension to 2N-dimension, space ordering method uses N-dimension to 1-dimension mechanism. It partitions given N-dimensional area into uniformly sized sub regions and imposes a sequential order to them. In terms of ordering algorithms, Hilbert curve and N-ordering are well-known.

Though object mapping/transformation method has a major advantage in that conventional indexing methods like B-tree can be applied directly, it has also a drawback that irregular distribution of spatial objects results in waste of storage space and inefficient query processing.

2.1.2 Object duplication/clipping method

The indexing methods in this category are based upon the policy of dividing given N-dimensional space into random-sized sub-regions. The divided sub-regions are disjoint, and spatial objects any part of which overlap a sub-region are regarded as being contained the sub-region. In

spatial query, using this method, all the objects contained in sub-regions, which overlap given query area, are to be checked if they actually meet the query condition.

This method facilitates efficient query processing by preventing multiple search paths. However, duplicate object storing can be a serious disadvantage when storage space is expensive resource, as is often the case. R^+ tree and mkd tree are representative methods in this category.

2.1.3 Object overlapping method

In object overlapping method, only objects fully contained a specific sub-region are stored in the sub-region. However, this restriction imposes most sub-regions to be overlapped each other. Overlapping sub-region implies the existence of multiple search paths in indexing process. Multiple paths cause backtracking in travesal of index tree and result in inefficient query processing[5]. R-tree, R^*-tree, and skd tree are examples of this method.

2.2 R^+-tree

R-tree is an indexing structure proposed by A. Guttmann in 1984. R-tree makes a sub-group with the spatial objects completely contained a specific sub-region. This grouping process is applied recursively until there is no objects remained. The restriction that an object is regarded as being a member of a specific region only if it is completely contained in that region causes inevitably overlaps among sub-regions. This overlaps result in inefficiency in query processing, for there exists multiple paths for a given search.

R^+-tree was devised by T. Sellis, et al. in 1987, to improve efficiency of query processing by getting rid of overlapping sub-region of R-tree. In R^+-tree, every object is regarded as being contained in a specific sub-region if any part of the object intersects the region. And any of two distinctive sub-regions in the same level do not overlap with each other. This restriction successfully gets rid of the possibility of multiple path which was caused by the existence of multiple paths.

The R^+-tree has the following properties.

(1) For each entry (p, RECT) in an intermediate node, the subtree rooted at the node pointed to by p contains a rectangle R if and only if R is covered by RECT. The only exception is when R is a rectangle at a leaf node; in that case R must just overlap with RECT.

(2) For any two entries (p1, RECT1) and (p2, RECT2) of an intermediate node, the overlap between RECT1 and RECT2 is zero.

(3) The root has at least two children unless it is a leaf.

(4) All leaves are at the same level.

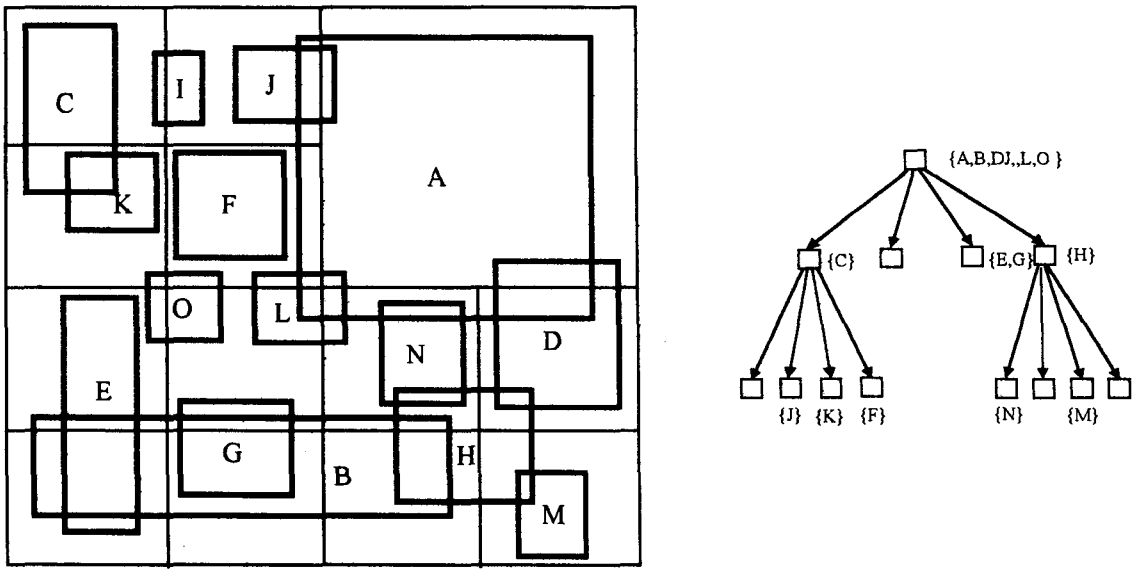
However, a object can be contained more than one sub-regions, and this results in the waste of storage space for duplicate objects. Also, in query processing, a spatial object may be checked more than once, because it can be stored more than one node. It can be a serious drawback in larger scaled spatial database, where it costs far more when in actual geometric data computation than MBR check.

2.3 MX-CIF Tree

In R- tree family indexing methods, whole tree should be traversed to determine in which sub-region a specific spatial object is contained. This approach can be a serious disadvantage, especially when indexing structure is stored in secondary storage devices.

CIF(Caltech Intermediate Form) tree associates a spatial object a priori. MX-CIF tree is a combination of MX (MatriX) tree, which is an index method for point objects and CIF tree. In MX-CIF tree an object is regarded as being contained in a specific quadrant if the object intersects any of the boundaries of the quadrant's child quadrants.

<Figure 2.1> shows an MX-CIF tree built with given set of spatial objects.



<Figure 2.1> MX-CIF Tree

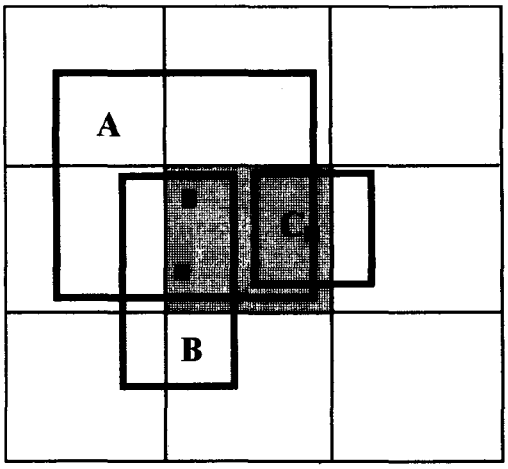
Though MX-CIF tree has an advantage of simplified building procedures, it produces considerable amount of unnecessary object comparisons. Suppose that rectangle L is a query region. Then, query processing procedure with MX-CIF tree has to make a traverse and conduct comparisons whole tree. This can induce a serious drawback of efficient query processing, which is one of major purpose of spatial indexing method.

3. HL-CIF(Hierarchically-Layered Caltech Intermediate Form) Tree

3.1 Basic concepts

HL-CIF tree is a CIF tree with hierarchical management of layered spatial objects. In HL-CIF tree, the spatial object is regarded as being layered according to its size. To associate a object with a sub-region, HL-CIF tree uses "centroid" and the corner points of Minimum Bounding Rectangle of the object. Spatial objects can be categorized to be included in a specific sub-region by its size and location. In HL-CIF tree, a spatial object is regarded as being contained in a specific sub-region if both its centroid and one of four corners of its MBR are included in the sub-region.

<Figure 3.1> shows the containment property of HL-CIF tree. Rectangular objects A, B, and C are regarded as being contained in shaded region because both their centroids and one of four corner points of their MBRs are contained in the shaded region.



<Figure 3.1> The spatial objects contained in a sub-region

The major advantage of this association property is that it guarantees the locality of spatial objects. In HL-CIF tree, any object contained in specific sub-region can not extend to quadrants farther than its neighboring 8 quadrants. In <Figure 3.1>, objects A, B, and C are contained in the shaded region by the definition of HL-CIF tree, and they can not extend to farther than neighboring 8 quadrants. Therefore, assuming that a query is given to retrieve all the objects intersect the shaded region, only the shaded quadrant and 8 neighbors have to be checked.

3.2 Definition of HL-CIF Tree

HL-CIF tree divides given N-dimensional area into 2^N sub-area recursively until specific condition meets. A spatial object is stored in a sub-region if both its centroid and one of its four

corner points are contained in one of four quadrants of the sub-region. That is, a spatial object and its associated node are determined by the its size and location. Thus, different from R-tree family indexing methods, HL-CIF tree has actual object elements in intermediate nodes as well as leaf nodes. HL-CIF tree is not a height-balanced tree. However, it has been reported that "not being height-balanced" do not always means inconsistent average searching time[16]. HL-CIF has a record tuple (R, object-list) where R is a bounding rectangle of all the objects contained in given node and object-list is an array of the very objects.

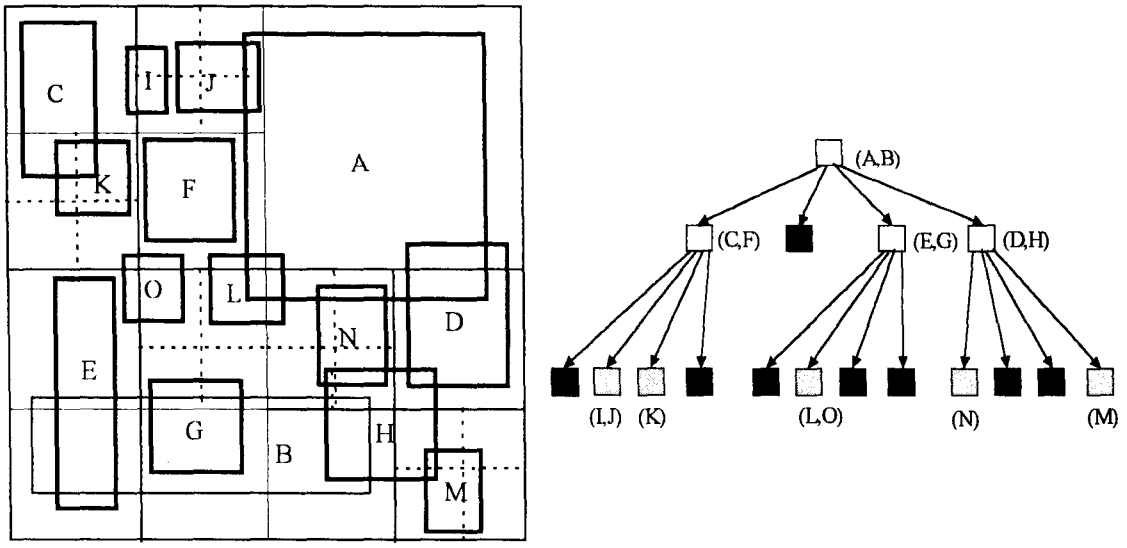
HL-CIF tree has the following properties.

(1) Every non-leaf node in the tree has 4 child nodes.

(2) Each node has the record structure (R, object id-list) where R is a bounding rectangle of all the object contained in the node and object-id-list is an array of spatial objects contained in the node.

(3) At any two nodes, (R1, object-id-list1) and (R2, object-id-list2), R1 and R2 can overlap with each other, but there are no common elements in object-id-list1 and object-id-list2.

(4) Every non-leaf node includes a spatial object only if both its centroid and one of its four corner points are contained in any of the node's child quadrants.



<Figure 3.2> HL-CIF Tree

<Figure 3.2> shows an example of HL-CIF tree built with given spatial objects. The letters in the parenthesis beside a node indicate the element objects of the node. Because the building algorithm for HL-CIF tree is very simple and intuitive, an index tree can be built without any complicated building algorithm.

3.3 Searching and Modifications in HL-CIF Tree

The search procedure in HL-CIF tree consists of 3 simple steps. First, retrieve all the quadrants intersecting given query region. Second, retrieve all the neighbor quadrants of the quadrants. Third, determine whether the objects contained in retrieved quadrants actually intersect given query region.

```
Search(R,pRoot, depth)
// R : query region.
// pRoot : start node to be searched.
// depth : current depth of the tree.
{
    RECT   tempR;

    // Get entire map size.
    int mapSize := GetMapSize();
    // Check if neighbor quadrants intersect
    given query region.

    tempR := R + SIZE/depth;

    if (CheckIntersect( tempR,pRoot)==TRUE)
        CheckObject( R, pRoot);
    else
        return;

    // Check children nodes recursively.
    if (pRoot != LEAF_NODE )
    {
        Search(R,pRoot->NW,depth+1);
        Search(R,pRoot->NE,depth+1);
        Search(R,pRoot->SW,depth+1);
        Search(R,pRoot->SE,depth+1);
    }
}
```

```
Insert ( pRoot, R )
{
 RECT   target;

  // Determine quad in which
  // given object is contained.
  Target := ComputeQuad(pRoot, R );

  // Add R in the selected quadrant.
  AddObject (target, R );
}
```

```
Delete( pRoot, R )
{
  RECT source;

  // Determine quad in which
  // given object is contained.
  source := ComputeQuad(pRoot, R );

  // Remove R in the selected
  RemoveObject ( source, R );
}
```

<Figure 3.3> Search, Insert, and Delete Algorithm in HL-CIF Tree

Insertion and deletion operations can be conducted with no complicated algorithm. For insertion, first, determine in which quadrant the object to be inserted is contained. And just add it to the quadrant. Deletion procedures can be inferred in the same vein as the insertion procedure.

The algorithms of search, insert, and delete are shown in <Figure 3.3>.

4. Performance Evaluation

The parameters of performance evaluation are the size of spatial objects and the size of query region. The indices for evaluation are page fault frequency, clustering degree, and overall processing time.

Spatial index structures are, in most case, very large and normally stored in secondary storage devices as like disk and tape. In paged storage scheme, frequent page faults apparently result in poor efficiency in query processing. Page fault frequency shows necessary page access for processing a spatial query. For a given query area, large page reference can be a serious drawback in efficient query processing. It is obvious that accessing secondary storage device is far more expensive than operation in CPU and main memory.

Clustering degree analysis provides an index for the effectiveness of indexing method in terms of reflecting locality of spatial objects. Higher clustering degree means that given indexing structure optimizes pages and objects necessary to be checked for query processing, which resluts in better query efficiency.

Page fault frequency and clustering degree together determine the overall processing time. Thus, overall processing time can be regarded as a decisive index for performance comparison.

The evaluation uses 2 randomly-generated data sets; a set of objects with the size of 0.01 percentile of total area, and that of 0.1 percentile. A data set consists of 10,000 spatial objects. For convenience, the MBR of given objects are regarded as the object itself. Applied query regions have sizes of 0.01, 0.1, 1, 5, 10 percentile of total area. Each query processing result is an average value calculated after 10 repeated applications of distinctive query region to each data set.

This evaluation is performed under the environment of Microsoft Windows 95 running on IBM-PC equipped with Intel-Pentium 150Mhz. Testing codes are programmed in C++ language, and compiled with Microsoft Visual C++ V.4.0 compiler.

4.1 Page fault frequency

The suppositions that each node of indexing tree is stored in a page, and that each node of indexing tree contains approximately same amount of objects, lie on the estimating page fault frequency.

<Table 4.1> shows the fact that page fault frequencies increase proportionally to the size of data objects for all of the indexing methods. MX-CIF tree shows relatively higher frequency in smaller data sets. It is due to the fact that MX-CIF tree has to check all the parent nodes to determine whether an object meets given query.

Object size (%)	Indexing method	Size of Query region (%)				
		0.01	0.1	1	5	10
0.01	**HL-CIF**	**3.57**	**4.15**	**5.68**	**8.59**	**13.62**
	MX-CIF	4.12	4.33	5.73	8.81	14.33
	R^+	2.42	3.31	6.61	12.39	17.82
0.1	**HL-CIF**	**3.66**	**4.16**	**5.35**	**9.19**	**13.69**
	MX-CIF	4.22	4.58	5.67	8.82	14.29
	R^+	2.51	3.47	6.44	11.0	16.64

<Table4.1> Page Fault Frequency Result

4.2 Clustering degree

Clustering degree plays a role in estimating how well a given indexing method reflects the locality of spatial objects. When query area intersects a sub-region, all the objects contained in the sub-region are regarded as candidate ones possibly fulfilling the query, and are checked whether they actually fulfill. The ratio of all the objects in sub-regions intersecting a given query region compared to the objects meeting the query is a clustering degree. Higher clustering degree indicates better clustering efficiency, which, in another words, is better query processing efficiency.

<Table 4.2> shows higher clustering degree of HL-CIF tree among the indexing methods for all size of data objects. MX-CIF tree shows poor degree as predicted by its definition where all the objects intersecting a boundary of given quadrant are to be stored in the parent quadrant of it.

Object size(%)	Indexing method	Size of Query region (%)				
		0.01	0.1	1	5	10
0.01	**HL-CIF**	**1.10**	**3.86**	**15.52**	**33.24**	**45.73**
	MX-CIF	0.54	1.92	10.0	25.81	40.94
	R^+	1.06	2.73	8.65	17.24	30.71
0.1	**HL-CIF**	**2.49**	**5.00**	**14.99**	**31.93**	**45.76**
	MX-CIF	0.91	1.95	7.68	21.97	36.99
	R^+	2.38	4.06	9.82	20.59	32.91

<Table 4.2> Clustering Degree Result

4.3 Processing time

Processing time can be interpreted as a combined result of page fault frequency and clustering degree. R^+-tree shows relatively higher increase ratio in processing time, which can be explained by the unnecessary objects comparisons caused by not guaranteeing spatial locality and duplicated comparisons caused by its innate property. MX-CIF tree presents higher processing time especially in smaller sizes of query region. This is because MX-CIF tree checks the unnecessary objects, nevertheless they are spatially far from given query region only because they intersect quadrant boundary.

Object size(%)	Indexing method	Size of Query region (%)				
		0.01	0.1	1	5	10
0.01	**HL-CIF**	**118.5**	**126.7**	**200.9**	**348.0**	**647.9**
	MX-CIF	190.0	188.6	257.5	414.3	687.8
	R^+	127.6	163.9	340.4	666.8	965.9
0.1	**HL-CIF**	**132.5**	**153.3**	**213.3**	**386.1**	**643.2**
	MX-CIF	278.5	300.4	352.1	506.9	762.6
	R^+	142.4	210.5	423.3	758.9	1146.0

<Table 4.3> Processing Time Result

HL-CIF tree is reported to be the most efficient in terms of processing time. As expected, HL-CIF tree prevents unnecessary comparison by retrieving most possible candidate objects for a query. This efficiency is induced by the fact that HL-CIF has the best clustering degree as stated above section.

5. Conclusion

In spatial indexing structures, required storage space and efficiency in query processing are regarded as two major tradeoffs. Most established methods achieve high utilization of storage space at the cost of query processing efficiency, and vice versa. R-tree and R^*-tree promote better storage efficiency. But, allowing multiple search paths causes inefficiency in query processing. Though R^+-tree can get rid of the multiple search paths in R-tree, it requires more space for storing objects than R-tree.

MX-CIF tree achieves high storage utilization and agreeable query efficiency with the principle that every spatial object and its associated sub-region are determined a priori. However, its associating policy induces inevitable drawbacks of unnecessary page references and object comparisons.

HL-CIF tree resolves the drawbacks of MX-CIF tree by adopting locality-reflected policy in associating a spatial object with a sub-region. HL-CIF tree, by minimizing unnecessary page references and false object comparisons, produces better query processing throughput as well as minimized storage space requirement.

References

[1] H. Samet, The Design and Analysis of spatial Data Structures, Addison-Wesley, 1989.

[2] H. Samet, Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS, Addison-Wesley, 1990.

[3] A.Guttman, "R-Trees : A Dynamic index structure for spatial searching",Proc. ACM SIGMOD International Conference on Management of Data, Boston, MA.,pp47-57, 1984.

[4] T. Sellis, N. Roussopoulos and C. Faloutsos, " The R^+ Tree : A Dynamic Index for Multidimensional Object", Proc. VLDB, pp507-pp518,1987.

[5] C. Faloutsos, I. Kamel, High Performance R-trees, IEEE Data Engineering Bulletin, Vol. 16, No. 3, pp 28- pp33, Sep 1993.

[6] N. BeckMann, H. P. Kriegel, R. Schneider, and B. Seeger, " The R*-tree : An Efficient and Robust Access method for Points and Rectangles", Proc. ACM SIGMOD 1990, pp.58-67, Mar 1990.

[7] G. Kedem, The quad-CIF tree : A Data structure for hierarchical on-line algorithms, Proc. Of the 19th Design Automation Conference, Las Vegas,pp352-357, June 1982.

[8] H. Lu and B. C. Ooi, "Spatial Indexing: Past and Future", IEEE Data Engineering Bulletin, Vol. 16, No. 3, pp 16- pp21, Sep 1993.

[9] Beng Chin Ooi: Efficient Query Processing in Geographic Information Systems. Lecture Notes in Computer Science No. 471, pp22-pp59, Springer, 1990.

[10] M. N. Gahegan, An Efficient use of quadtree in a GIS, International Journal of GIS, Vol.3, No.3, pp201-214, 1989.

[11] O. Gunter,, J. Bilmes, "Tree-based access methods for spatial databases : Implementation and performance Evaluation",IEEE Transactions on Knoledge and Data Enginnering, Vol. 3, No. 3, pp 342-pp356, Sep 1991.

[12] T. J. Ibbs and A. Stevens, Quadtree storage of vector data, International Journal of GIS, Vol.2, No.1, pp43-56, 1988.

[13] B. C. Ooi, K. J. McDonell, and R. Sacks-Davis, Spatila k-d tree : An Indexing mechanism for spatial database, Proc. Of the 11th International Computer Software and Applications Conf.(COMPSAC), Tokyo, pp433-438.,October 1987,

[14] Hwang, B. Y., Byun, B. K. and Moon, S. C., "Efficient Access Method for Multidimensional Complex Objects in Spatial Databases : BR Tree", Journal of Microprocessing and Microprogramming, Vol. 32, No.1-5, pp.765-772, 1991.

[15] Michael Freeston, A General Solution of the n-dimensional B-tree Problem, Proc. of the 1995 ACM SIGMOD, Vol. 24, No. 2,pp80-91, June 1995.