Spatial Query Processing Using Object Decomposition Method*

Yong-Ju Lee
Korea Environmental Technology Research Institute
yjlee@keins.ketri.re.kr

Ho-Hyun Park
Korea Advanced Institute of Science and Technology
hhpark@romeo.kaist.ac.kr

Nam-Hee Hong
Korea Advanced Institute of Science and Technology
nhhong@romeo.kaist.ac.kr

Chin-Wan Chung
Korea Advanced Institute of Science and Technology
chungcw@romeo.kaist.ac.kr

Abstract

We propose a new object decomposition method, called DMBRs, to improve the performance of spatial query processing. This method is suitable for complex spatial objects in real-world geographic applications. The basic idea is that a polygon is recursively divided into two sub-polygons by splitting its MBR until a given constraint is satisfied. To increase the efficiency of the DMBRs method, an extension of an existing spatial indexing structure is presented. Since this new structure can prune a number of false hits quickly, the performance of spatial query processing can be improved. The proposed method is compared with traditional decomposition methods by an analytical study. This comparison shows that our decomposition method outperforms the traditional decomposition methods.

1 Introduction

Queries in spatial databases are usually concerned with massive volumes of data and complex spatial objects. Spatial objects are characterized by extremely irregular geometric components which do not conform to any fixed shapes, and by multi-dimensional data which consist of a large number of coordinates describing the outline of spatial objects. If there are a large number of such complex objects, searching a particular spatial object would be expensive, since a number of geometric computations are required for exact calculations in locating the spatial object. In order to locate a spatial object efficiently, the spatial object, in general, has to be approximated before any geometric computations are applied. For the efficient approximation of spatial objects, many database researchers [1-3] have been considerably interested in the object decomposition. However, while most of them focused on simple spatial objects such as points, lines and rectangles, very little attention was devoted to complex spatial objects.

In this paper, we propose a new object decomposition method for complex spatial objects such as city, road and lake in real-world geographic applications. To increase the efficiency of our proposed method, an existing spatial indexing structure is extended. Under this structure, we derive point, region and spatial join query algorithms. The proposed method is compared with traditional decomposition methods by an analytical study. This comparison shows that our decomposition method is superior to the traditional decomposition methods.

This paper is organized as follows. Section 2 surveys related works. Section 3 proposes a new object decomposition method. Section 4 describes an extension of an existing indexing structure, and algorithms of typical spatial queries for this new structure. In Section 5, we determine an optimal value of the parameter for the proposed method. Section 6 presents a performance comparison between this new method and traditional decomposition methods. Finally, conclusions appear in Section 7.

2 Related work and problem

There are two approaches to approximate spatial objects. The first approach is that a smallest aligned rectangle enclosing an object, a *minimum bounding rectangle(MBR)*, is used to approximate an irregularly shaped spatial object. MBRs allow appropriate proximity query processing by preserving the spatial identification and eliminating many potential intersection tests quickly. For instance, two objects will not intersect if their MBRs do not intersect. Most of approximation methods[4-6] based on traditional spatial access methods are fallen to this approach. The second approach is that a more *accurate approximation* than the MBR, such as a convex container, can be used to approximate a spatial object. This approach is expected to improve the performance of query processing by increasing the quality of the approximation for original objects. Convex approximations[7] and object decomposition techniques[8] are fallen to the second approach.

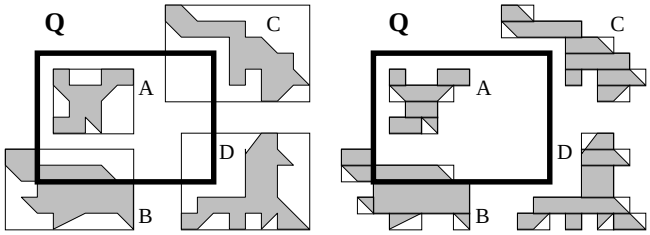
Two well known approximation methods, the *filtering-refinement*[9] and the *object transformation*[10,11], may be considered in the first approach. In the filtering-refinement method, the filter step reduces the entire set of objects to a subset of candidates using their MBRs, and then the refinement step inspects the exact representation of each

* This work was supported by the National Geographic Information Systems Technology Development Project of the Ministry of Science and Technology of Korea.

object of the candidates. Although MBRs provide a fast approximation by existing spatial access methods designed for MBR containers, they are considered as a rather inaccurate approximation since a simple rectangle cannot exactly represent an arbitrary spatial object. By the coarse approximation of this method, the candidates may contain a number of 'false hits' not fulfilling the query condition. Furthermore, the whole candidates have to be transmitted into the refinement step even if they would result in 'false hits.' In the object transformation method, *k*-dimensional spatial objects are transformed to 1-dimensional bitstrings, or *k*-dimensional intervals are transformed to points in 2*k*-dimensional space. Nevertheless, this method also has a rough approximation since its mapping was done under the assumption that spatial objects are MBRs.

Convex approximations and *object decomposition* techniques in the second approach have been attempted to improve the quality of the approximation. However, convex approximations using more complex containers require more complex spatial access methods, since complex containers need more parameters than MBRs. Moreover, the use of one container on an original complex object representation cannot decrease the complexity of the spatial object. This means that time-consuming geometric computations have to be applied for deciding the complex objects satisfying the query condition. In contrast, object decomposition techniques, which decompose a complex spatial object into a set of simple spatial components such as trapezoids, lead to both a better quality of the approximation and simpler spatial objects. However, these decomposition techniques generate too many components on complex spatial objects. A number of decomposed components could result in a storage and query processing overhead. This is illustrated in Example 1.

Example 1 (a number of decomposed components):
In Figure 1(a), spatial objects are approximated by MBRs. A typical spatial query may ask for all objects intersecting a user-specified rectangular window Q. In this case, all rectangles that intersect the search region Q are determined in the filter step. Here, objects A, B, C and D belong to the candidate set. In the refinement step, we have to check whether the exact representation of the objects A, B, C and D really intersect the search region Q. At this point, the objects A and B are identified as correct answers of the query, whereas the objects C and D are not. In order to improve the quality of the approximation, object decomposition techniques have given up using one single MBR for every complex spatial object. That is, the original complex objects are decomposed into a set of simple components such as trapezoids. Similar to an existing MBR approach, all decomposed components can be approximated by means of MBRs. Contrary to an existing MBR approach, a good approximation is provided by divided MBRs. As a result of this method, objects A and B belong to the candidate set. Figure 1(b) shows the result of this approximation. However, there are *a number of decomposed components* labeled with the same identifier. For instance, objects A and B have six decomposed components, respectively.



(a) (b)

Figure 1: Different approximation approaches

End of Example 1.

As described in Example 1, traditional object decomposition techniques have a problem on decomposed components. Even worse, the more complex spatial objects are, the more spatial components are produced from the complex objects. Due to a large number of decomposed components of such complex objects, the efficiency of spatial queries will decrease. Therefore, the development of a new object decomposition method to overcome this problem is essential.

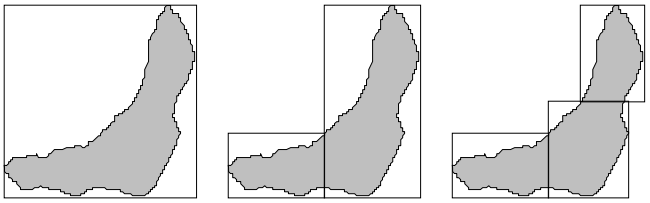
3 Controlled decomposition method

We propose a new object decomposition method called *decomposed minimum bounding rectangles(DMBRs)*. The basic idea is that a polygon is divided into two sub-polygons corresponding to disjoint half regions of its MBR space, then a new MBR, called here a DMBR, for each of those sub-polygons is generated. This operation is performed recursively until every DMBR fulfills a given constraint. The constraint is expressed by the *accuracy of the decomposition(AOD)*. This means that a split is permitted if the size of the resulting DMBR is above a threshold. The threshold is controlled by a parameter *g*: AOD(*g*) requires a split of the DMBR that covers more than 2^{-g} of the MBR space. In order to support the recursive split efficiently, we use the vertical boundary and the horizontal boundary in strictly alternating sequence. The efficiency of this alternating split has been proven in many researches[1,2,12,13]. Our decomposition method is illustrated in Example 2.

Example 2 (controlled decomposition):
Consider a polygon shown in Figure 2(a). This figure shows an MBR enclosing a spatial object. Assume that the threshold size is 25% of the MBR space, i.e., AOD(2). We sub-divide the polygon until the given constraint is satisfied.

At first, the polygon depicted in Figure 2(a) is divided

into two sub-polygons by the middle vertical boundary, then DMBRs for the sub-polygons is generated (see Figure 2(b)). While the DMBR of the left sub-polygon is less than 2^{-2} of the MBR space, the DMBR of the right sub-polygon is bigger than 2^{-2} of the MBR space. Therefore, only the object decomposition on the right sub-polygon is performed recursively against the middle horizontal boundary (see Figure 2(c)). Then, the recursive decomposition terminates since every DMBR covers less than 2^{-2} of the MBR space.



(a) (b) (c)

Figure 2: The process of AOD(2)

End of Example 2.

The algorithm for the controlled decomposition uses a 'divide and conquer' technique. After accepting an array *p* as an input polygon, the algorithm creates two other arrays *p1* and *p2* as two divided polygons. As soon as two arrays are created, this algorithm calls itself for each of the divided polygons, and the next dividing is performed recursively. To increase the efficiency of our decomposition algorithm, the DMBRs of divided polygons are inserted into a *two-dimensional binary tree* that is similar to the LSD tree[12]. Since a polygon generates exactly two divided polygons in our algorithm, the binary tree is appropriate for this kind of representation. In this binary tree, DMBRs and their component identifiers are stored at leaf nodes, and rectangles enclosing sub-polygons are stored at non-leaf nodes. This algorithm is as follows.

Algorithm 1: Decomposition (*p*, *d*)
Input: A series of polygon vertices $p=(v_1, v_2, \ldots, v_n)$, where polygon edges are from v_i to v_{i+1} for i=1,2,...,n-1 and from v_n to v_1. A boolean variable *d*, where *d* is toggled on the way that divides the region to effect the alternating tests on the vertical and horizontal boundaries.
Output: A new two-dimensional binary tree.

find MBR or DMBR coordinates from *p*.
case MBR: computer MBR space.
DMBR space = MBR space.
initialize a root node in the two-dimensional binary tree.
case DMBR: compute DMBR space.
if DMBR space > MBR space/ 2^g ,
then make a middle vertical (or horizontal) boundary.
for each polygon edge in array *p*,
if the edge lies in the left (or above) of the middle boundary,
then endpoints of the edge are added into array *p1*.
if the edge lies in the right (or below) of the middle boundary,
then endpoints of the edge are added into array *p2*.
if the edge intersects against the middle boundary,
then find the intersect point and the point is added into both *p1* and *p2*.
end-for
/* build two-dimensional binary tree */
DMBR coordinates related *p1* is inserted into the left node of the current node.
DMBR coordinates related *p2* is inserted into the right node of the current node.
/* call decomposition algorithm recursively */
$d = \neg d$
call Decomposition (*p1*, *d*)
call Decomposition (*p2*, *d*)
else terminate this program.
End of Algorithm 1.

The controlled decomposition algorithm has a parameter that controls the number of components for each object. At a low value of the parameter, the number of components can be minimized, but this decomposition provides a rather poor approximation of the object. On the other hand, the accuracy of the approximation can be better at a higher value, but the linear increase in the number of components can be observed. From this observation, we can conclude that there is a balanced ratio between the number of components and the accuracy of the approximation. The optimal value of the parameter will be explored through experimental measurements in Section 5. In Section 6, our method will be compared with traditional decomposition methods.

4 Spatial query processing based on object decomposition

For the use of the two-dimensional binary tree, an extension of an existing indexing structure is proposed in this section. Using this new structure, we will discuss algorithms of spatial query processing based on object decomposition.

4.1 Two-step indexing structure

The success of the object decomposition approach depends on the ability to narrow down quickly the set of components that are affected by spatial queries. In order to decide which components are relevant for a particular geometric test, we need an efficient indexing structure that organizes a set of components of one object. A number of spatial indexing

structures based on MBRs have been developed. The most promising group includes the R-tree and their variations, e.g., R^+-tree and R^*-tree. However, these structures are considered unsuitable for organizing the decomposed components, since components with the same identifier are distributed on the secondary storage independently. An arbitrary distribution of these objects over the secondary storage leads to high access cost during query processing.

The topic of this section is the design of a spatial indexing structure for spatial query processing based on object decomposition. However, it is not the intention of this paper to discuss in detail which indexing structure is the most suitable to organize decomposed components. We propose instead an extension of the existing indexing structure, which integrates two indexing structures for original objects and their decomposed components. As the existing indexing structure, the most popular R-tree is selected for storing MBRs. But our indexing structure can be easily extended to other R-tree variations.

Figure 3 depicts our *two-step indexing structure* schematically. We distinguish two parts: (1) the first level is the indexing structure for original objects, called here *Ro-tree*, which is a straightforward modification of the R-tree; and (2) the second level is the two-dimensional binary tree, called here *Rd-tree*, which is designed to reduce the number of main memory operations and to store decomposed components. The Ro-tree consists of leaf and non-leaf nodes. MBRs of original objects are stored in the leaf nodes. Each leaf node is supplemented by a pointer to the Rd-tree and by an object identifier of the original object. Non-leaf nodes are built by grouping rectangles at the lower level. The Rd-tree produced by Algorithm 1 is attached to the corresponding leaf node of the Ro-tree.

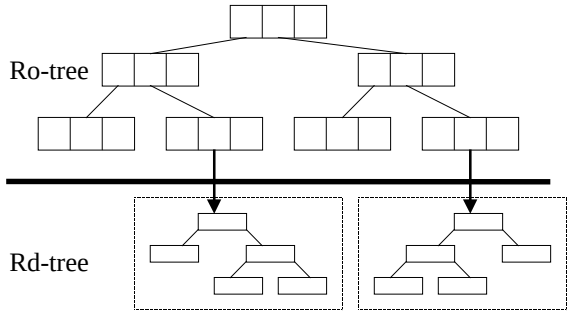


Figure 3: Two-step indexing structure

4.2 Two-step processing of spatial queries

Since spatial database systems are used in very different application environments, there exists currently no standard set of spatial queries fulfilling all requirements of spatial applications. Thus it is necessary to provide a small set of basic spatial queries which are supported by database query facilities. Basic spatial queries that have been addressed by other researchers in the past can be summarized as follows:

- *Point query*: Given a query point *P* and a set of objects *M*, the point query yields all objects of *M* geometrically containing *P*.
- *Region query*: Given a polygonal query region *R* and a set of objects *M*, the region query yields all objects of *M* sharing points with *R*. A spatial case of the region query is a *window query*. The query region of the window query is given by a rectangle.
- *Spatial join query*: Given two sets of objects S_1 and S_2, a spatial join query yields all pairs of objects (s_1, s_2), $s_1 \in S_1$, $s_2 \in S_2$ whose spatial components intersect. More precisely, for each object $s_1 \in S_1$, we have to look for all objects in S_2 intersecting with s_1.

In general, the spatial query processing consists of two steps: *filter step* and *refinement step*. These steps can be expressed in the query algorithm as follows:

Candidates = Filtering(MBRs)
Result = Refinement(Candidates)

The input parameter of Filtering is MBRs of spatial objects. The Filtering prunes the search space by means of using a spatial indexing structure. The Refinement evaluates exactly the query condition for objects filtered in Filtering. In the following, we will examine basic spatial queries based on this two-step processing.

4.2.1 Point query

The result of a point query consists of all stored spatial objects containing a given query point. The search algorithm of the Ro-tree is used for the filter step. This algorithm is analogous to that of the traditional R-tree. However, the refinement step of the R-tree is very costly if spatial objects are complex, since this approach is provided a bad approximation and applied time-consuming computational geometry algorithms for the complex spatial objects. In our approach, DMBRs of the Rd-tree is used before applying the refinement step. The use of the DMBRs can eliminate a number of false hits quickly and lead to simpler computational geometry algorithms. That is, regarding a small number of simple components filtered by DMBRs, the exact evaluation (i.e., '*point-in-polygon*' test) can be performed.

One method of the point-in-polygon test is to construct a line between a point in the question and a point known to be outside the polygon. Then we count how many intersections of the line with the polygon boundary occur. If there are an odd number of intersections, then the point in the question is inside. An even number indicates that it is outside. When the point of the intersection is the vertex where two sides meet, we must look at the other endpoints of the two segments which meet at this vertex. If these points lie on the same side of the constructed line, then the point in the question counts as an even number of intersections. If they lie on opposite sides of the constructed line, then the point is counted as a single intersection.

Algorithm 2 shows point query procedures in the Rd-tree. First, we find all index records covering a query point

P=(Px, Py) using the search algorithm of the Ro-tree. For each of these records, we invoke the following algorithm.

Algorithm 2: Point_Query (T)
Input: An Rd-tree rooted at node T.
Output: An *Oid* covering P.

if T is not a leaf node,
  then if T covers P,
    then call Point_Query(left(T))
      call Point_Query(right(T))
  else if T covers P,
    then count the number of intersections between a test line and the component.
      if the count is odd,
        then return(*Oid*).
End of Algorithm 2.

4.2.2 Region query

A region query yields all spatial objects intersecting a given query window. Similar to the point query, this query processing is based on the Ro-tree and the Rd-tree. As described in Example 1, traditional decomposition techniques do not handle window queries efficiently, since there are a number of decomposed components. To avoid this drawback, we have proposed the DMBRs approach which can control the number of components. By the trade-off between the number and the complexity of components, the exact evaluation (i.e., '*polygon-in-rectangle*' tests) can be performed efficiently in our approach.

For the polygon-in-rectangle test, our algorithm performs some initial tests on polygon edges to determine whether intersection tests are really necessary. First, edges are checked to see if they are within the window, so the polygon can be trivially accepted as the answer. Otherwise, *region checks* are applied. For instance, in Figure 4, if x coordinate boundaries of a query window are at *Xmin* and *Xmax* and y coordinate boundaries are at *Ymin* and *Ymax*, both endpoints of edge e_1 have y coordinates greater than *Ymax* and thus lie in the region above the window. This means that the edge needs not an intersection test. Similarly, we need not intersection tests on edges in regions below *Ymin*, to the left of *Xmin* and to the right of *Xmax*.

If both endpoints of an edge don't lie in a region above, below, to the left or to the right of the window, the polygon can be either accepted or rejected (e.g., both edge e_5 and edge e_7 are this case, but only edge e_5 intersects the window). The algorithm selects an endpoint of the edge that insures the outside of the window. If this endpoint lies in the region to the left of the window, then we examine whether this edge intersects the left boundary of the window. For instance, edge e_5 and edge e_7 are tested against the boundary *ab* of the window. Similar to the left of the window, we can investigate this *intersection test* with respect to the right, above, and below of the window.

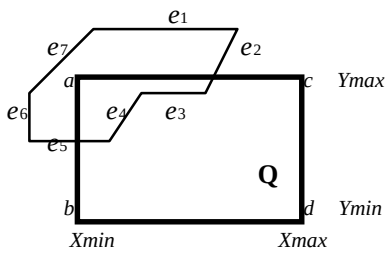


Figure 4: Polygon-in-rectangle test

Algorithm 3 shows region query procedures in the Rd-tree. First, we find all index records whose MBRs overlap Q=(*Xmin*, *Xmax*, *Ymin*, *Ymax*) using the search algorithm of the Ro-tree. For each of these records, we invoke the following algorithm.

Algorithm 3: Region_Query (T)
Input: An Rd-tree rooted at node T.
Output: An *Oid* overlapping Q.

if T is not a leaf node,
  then if T is covered by Q,
    then return(*Oid*).
  if T overlaps Q,
    then call Region_Query(left(T))
      call Region_Query(right(T))
  else if T is covered by Q,
    then return(*Oid*).
  if T overlaps Q,
    then for each edge of the component,
      if the edge lies within Q,
        then return(*Oid*).
        else apply region checks.
      if the edge isn't trivially rejected by the region checks,
        then apply intersection test.
          if an intersection is detected,
            then return(*Oid*).
            else take the next edge.
    end-for
End of Algorithm 3.

4.2.3 Spatial join query

A spatial join query combines spatial objects from two or more relations according to their geometric attributes. In this section, our discussion is restricted to the intersection join for two spatial relations which correspond to polygonal areas. Although join processing has been studied in the literature extensively (see [14] for a survey), these approaches designed for traditional join processing can be hardly used for the spatial join without modifications. Only the approach of nested loops can be used without any modifications[15]. Thus, the nested loops approach will serve as an initial starting point.

However, the approach of nested loops is very inefficient

for spatial join processing because of the high number of loops. In particular, the refinement step of this approach can become the bottleneck if spatial objects consist of a large number of vertices. In this respect, our approach is to accelerate the expensive step of spatial join processing by preceding filter steps which reduce the number of vertices investigated in the refinement step. MBRs and DMBRs are used for these filter steps. After pruning components that are clearly not fulfilling the join condition by these filters, '*polygon-in-polygon*' tests are performed by the *plane-sweep technique*[16] from the area of computational geometry.

The plane-sweep technique sorts the vertices of two components in a preprocessing step according to their *x* coordinates. Then a vertical line (i.e., sweep line) sweeps the data space from left to right. The sweep line stops at every vertex, where the status of the plane-sweep is updated. This sweep line status stores the edges which intersect the sweep line, and these edges are sorted according to their *y* coordinates at the sweep line position. For every vertex, the corresponding edges are inserted into or deleted from the sweep line status. While the insert operation is performed, the considered edge is tested for the intersection with its new neighbors in the sweep line status. For the delete operation, the former neighbors of the edge are tested for the intersection.

For accelerating the polygon-in-polygon test, DMBRs can be used for restricting the search space. That is, we only have to check edges in the plane-sweep algorithm which intersect the intersection rectangle of the DMBRs. For instance, edge e_1 and edge e_5 in Figure 5 do not need to be processed by the plane-sweep since they can not intersect an edge of the other polygon. By a linear scan through each of the two components, we can exclude all edges not intersecting this rectangle.

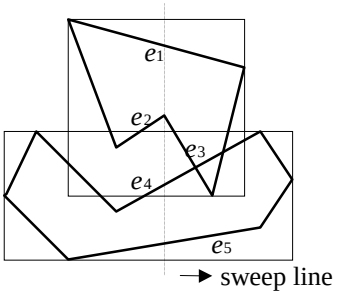


Figure 5: Example for the plane-sweep algorithm

Algorithm 4 shows spatial join query procedures in the Rd-trees. First, we find all index records whose MBRs overlapped between two spatial objects (i.e., *O1* and *O2*) using the search algorithm of the Ro-tree. For each of these records, we invoke the following algorithm.

Algorithm 4: Spatial_Join_Query (*T1*, *T2*)
Input: An Rd-tree rooted at node *T1*, and another Rd-tree rooted at node *T2*.
Output: An *Oid* whose components overlapped between *O1* and *O2*.

if *T1* is not a leaf node,
  then if *T1* overlaps MBR of *T2*,
    then call Spatial_Join_Query(left(*T1*), *T2*)
      call Spatial_Join_Query(right(*T1*), *T2*)
if *T2* is not a leaf node,
  then if *T2* overlaps DMBR of *T1*,
    then call Spatial_Join_Query(*T1*, left(*T2*))
      call Spatial_Join_Query(*T1*, right(*T2*))
  else exclude edges that not intersect the intersection rectangle of the DMBRs.
    apply plane-sweep technique.
    if an intersection is detected,
      then return(*Oid*).
End of Algorithm 4.

5 Determination of an optimal *g* value

In this section, our goal is to evaluate which *g* value of the controlled decomposition method introduced in Section 3 leads to an optimal performance in spatial query processing. For this purpose, the two-step indexing structure can be used. The performance of the first step (i.e., the Ro-tree) is mainly determined by accesses to the secondary storage and comparisons within the directory as well as the data pages. The performance of this Ro-tree, which is used for handling MBRs of spatial objects, is practically independent of the object decomposition. Therefore, this performance is not considered in this test series.

The performance of the next step is determined by the time spent for the Rd-tree, which is designed to store decomposed components. The processing time of the Rd-tree is measured by the task of handling decomposed components in the main memory. That is, the performance of the Rd-tree is determined by the time spent for comparisons within the directory of the Rd-tree and computational geometry algorithms for those components. As this performance strongly depends on the value of parameter *g*, we explicitly measured them using various spatial objects.

We used three different spatial objects to get expressive and realistic results on the performance of the object decomposition. To be as general as possible, these spatial objects were chosen from real digitized data used in existing geographic information systems. Figure 6 depicts the analyzed spatial objects and Table 1 lists their characteristics. For describing characteristics of the spatial objects, we provide the number of vertices, the area of a spatial object and its MBR, and its cover characterizing the accuracy of the MBR approximation. The cover is

presented by the area of the spatial object normalized to the area of the corresponding MBR.

(a) Park (b) Lake (c) Korea

Figure 6: Analyzed spatial objects

Table 1: Characteristics of analyzed spatial objects

Spatial Object	Num. of Vertices	Area		Cover (%)
		Object	MBR	
Park	83	700	1634	43
Lake	206	472	3105	15
Korea	229	1431	3456	41

Queries that we performed are classified into point queries, window queries and spatial join queries. For spatial objects presented in Figure 6, Table 2 presents the average time required for the evaluation of one single query. The time values are given in seconds. For a clear evaluation, they are divided into the query time for 25, 50, 75 and 100 spatial objects. Due to the space limitation, the full set of results obtained is not presented in this table. The trends discussed below were observed in all experiments. In the table, we have shadowed the best performing values for each type of queries.

Table 2: Average time per a query (in second)

	Point Query				Region Query				Spatial Join Query			
g \	25	50	75	100	25	50	75	100	25	50	75	100
Park												
0	0.06	0.11	0.16	0.22	0.01	0.03	0.05	0.06	0.10	0.21	0.31	0.41
2	0.01	0.02	0.03	0.04	0.01	0.02	0.02	0.03	0.04	0.08	0.12	0.16
3	0.00	0.01	0.02	0.03	0.00	0.01	0.02	0.02	0.02	0.05	0.08	0.11
4	0.01	0.02	0.02	0.03	0.01	0.02	0.03	0.03	0.03	0.06	0.09	0.12
6	0.00	0.02	0.03	0.04	0.01	0.02	0.03	0.04	0.03	0.06	0.08	0.11
Lake												
0	0.12	0.24	0.35	0.47	0.04	0.07	0.10	0.13	0.45	0.91	1.37	1.82
2	0.04	0.08	0.12	0.16	0.02	0.03	0.04	0.05	0.12	0.24	0.36	0.48
3	0.02	0.04	0.07	0.09	0.02	0.04	0.05	0.07	0.07	0.15	0.23	0.30
4	0.03	0.06	0.09	0.12	0.03	0.06	0.08	0.11	0.06	0.12	0.17	0.23
6	0.05	0.10	0.14	0.19	0.06	0.10	0.14	0.18	0.08	0.15	0.22	0.29
Korea												
0	0.14	0.28	0.41	0.55	0.03	0.06	0.09	0.13	0.36	0.71	1.06	1.40
2	0.03	0.06	0.09	0.10	0.02	0.04	0.05	0.07	0.05	0.10	0.14	0.18
3	0.01	0.03	0.04	0.07	0.01	0.03	0.03	0.04	0.04	0.08	0.11	0.14
4	0.01	0.02	0.03	0.04	0.01	0.02	0.03	0.04	0.03	0.07	0.10	0.14
6	0.01	0.03	0.04	0.06	0.01	0.03	0.05	0.06	0.05	0.09	0.14	0.18

Results in Table 2 suggest that the reasoning of the existence of an optimal *g* value is valid. The query performance of the no decomposition (i.e., *g*=0) and the high decomposition (i.e., *g*=6) is considerably worse than the middle decomposition (i.e., from *g*=2 to *g*=4). When *g* is 0, the performance is particularly time-consuming due to the high complexity of objects which are not decomposed. The performance degeneration corresponding to the high *g* value is strongly caused by a large number of components. From this test, the optimal *g* value can be obtained around *g*=3.

6 Comparison of decomposition methods

The proposed controlled decomposition method is compared with traditional decomposition methods through an analytical study in this section. There are many object decomposition methods in use for representing spatial objects. The basic principle of these methods is a recursive decomposition such as 'divide and conquer' techniques. We classify the object decomposition methods according to the following three properties of the recursive decomposition: *condition* of decomposition, *number* of partitions, and *containers* of components. This classification yields five classes as in Table 3.

Table 3: Classification of decomposition methods

Class	Properties of Decomposition			Spatial Access Methods
	Condition	Number	Containers	
C1	no redundancy	1	MBR	R-tree, R^+-tree, R^*-tree
C2	regular grid	2^d	a set of fixed grids	quad-tree, B^+-tree with z-value
C3	grid and object shape	variable	variable cells	edge-quadtree, PM quadtree
C4	object structure	*n*	MBRs	Cell-tree, TR^*-tree
C5	controllable parameters	controll-able	DMBRs	two-step indexing structure

2^d : the number of grids (i.e., resolution)
n: the number of decomposed components

In order to determine which method performs best in terms of the performance of spatial query processing, it is necessary to analyze each of these classes with respect to criteria for evaluating the suitability of decomposition methods.

6.1 Analysis of decomposition methods

For improving the performance of spatial query processing, the performance of both the filter step and the refinement step must be considered. The performance of the filter step considerably depends on the quality of the spatial object approximation by the container used to filtering issues. The *approximation quality* is defined as the amount of area covered by the container not by the object itself.

Minimizing the amount of that area will directly and proportionally improve the filtering performance. The performance of the refinement step depends on the number of refined objects as well as on their complexity. Minimizing the number of objects to be refined is the task of the filter step. Thus the *object complexity* is the issue to be examined in the refinement step. A simplification of the refined objects using the object decomposition technique may lead to better performance of the refinement step. However, the main drawback is given by *a number of components*. As criteria for evaluating decomposition methods, therefore, we will consider the number of components, the quality of the approximation and the simplification of objects. Table 4 indicates parameters which are common to the forthcoming analysis.

Table 4: Parameters of the analysis

Parameter	Description
d	the number of split-levels which decompose a region into two equal-sized sub-regions
p	the perimeter of a spatial object
n_v	the number of polygon vertices

• *The number of components(NC)*
In case of C1, NC obviously is one. NC of a grid-like representation such as C2 and C3 depends on a function of the resolution (i.e., the number of split-levels) and the size of the object (i.e., its perimeter). Walsh[17] and Samet[18] proved that NCs of C2 and C3 were less than or equal to $\frac{3}{2}pd$ and $p \cdot 2^{\lfloor (d+1) \rfloor}$, respectively. NC of C4 may be $O(n_v)$ since this NC is produced by the plane-sweep technique, i.e., the vertices are passed and handled with increasing y-coordinates. NC of C5 is determined by the number of split-levels, i.e., $O(2^d)$, since a polygon is split into two divided polygons recursively.

• *The quality of the approximation(QA)*
QA is improved by minimizing the deviation of the approximation from an original object. This deviation is measured by the false area of the approximation. The QA can be expressed by the following formula:

$$QA = \frac{A_{obj}}{A_{appr}}$$

where A_{obj} and A_{appr} denote the area of a spatial object and of its container, respectively. In general, it is difficult to predict the area of the spatial object by the analytical approach. A preliminary approach of counting QA is to use the value that is inversely proportional to NC. This is due to the fact that QA improves as NC increases.

• *The simplification of objects(SO)*
SO can be given by the following formula:

$$SO = \frac{c_{ref}}{C_{obj}}$$

where C_{obj} and c_{ref} represent the complexity (i.e., the number of polygon vertices) of an original object and of a refining object, respectively. In C1, c_{ref} is the same with C_{obj}. c_{ref} of C2 is analogous to that of C1, since the access method based on z-value (e.g., B-tree) returns original object identifiers for the refinement step. In C3 and C4, an original object is decomposed into a set of simple components such as straight lines or trapezoids, thus c_{ref} of these classes leads to a small constant which can be ignored. c_{ref} of C5 may be presented as the number of polygon vertices divided by the number of components, i.e., the average number of component vertices, $\frac{n_v}{2^d}$.

Table 5 summarizes results from this analysis. The schematic relationship between d and the corresponding cost is shown in Figure 7. As d increases, the cost of NC increases exponentially. On the other hand, as d is decreased, the cost of QA and SO approaches their maximal values. To minimize the total cost, a balance among NC, QA and SO must be taken by selecting a proper d.

Table 5: Results of complexity analysis

Class	NC	QA	SO
C1	$O(1)$	$O(1)$	$O(1)$
C2	$O(pd)$	$O(\frac{1}{pd})$	$O(1)$
C3	$O(p \cdot 2^d)$	$O(\frac{1}{p \cdot 2^d})$	$O(\frac{1}{n_v})$
C4	$O(n_v)$	$O(\frac{1}{n_v})$	$O(\frac{1}{n_v})$
C5	$O(2^d)$	$O(\frac{1}{2^d})$	$O(\frac{1}{2^d})$

Figure 7: Cost Estimation

6.2 Determination of the best strategy
In decomposition methods, both QA and SO should be considered seriously since the performance of the filter step and the refinement step is obviously determined by QA and SO. Table 5 shows that QA and SO of C1 have relatively

high cost even though NC has low cost. That is, C1 cannot be expected to become the best solution by our cost estimation. Thus we will exclude C1 strategy from the performance evaluation.

The main drawback of traditional decomposition methods has been proven to generate too many components. This means that NC should also be considered seriously. In C2, C3 and C4, NC has relatively high cost since in practical cases p and n_v dominate d. Although these classes have relatively low cost to QA or SO, they cannot be expected to become the best solution since they reveal an extreme unbalance among QA, SO and NC.

From the above observation, we may conclude that the better performance is expected by C5 strategy even though it does not have the least value in each of the criteria as compared with other classes. This is due to the fact that only this strategy can control the parameter d. (Recall that our decomposition method has a parameter that controls the number of components.) By controlling d, C5 strategy can tune the trade-off between opposing shaped curves in Figure 7.

7 Conclusions

We have proposed a new object decomposition method, called DMBRs, to improve the performance of spatial query processing, and an extension of an existing indexing structure, called two-step indexing structure, to increase the efficiency of the DMBRs method. Then we have derived point, region and spatial join query algorithms under this new structure. The proposed method has been compared with traditional decomposition methods by an analytical study. Our method is superior to the traditional decomposition methods due to its ability to tune the trade-off among evaluation criteria.

There has been no analytical study up to now with respect to the comparison of object decomposition methods. We have provided an analytical approach for this study. However, the study reported here is preliminary. It would be desirable to extend the analytical study to a more general cost model. The experimental verification for this study is also required in the future.

References

[1] J. Nievergelt, H. Hinterberger, "The Grid File: An Adaptable, Symmetric Multikey File Structure," ACM Transactions on Database Systems, Vol. 9, No. 1, March 1984, pp. 38-71.

[2] H. Samet, "The Quadtree and Related Hierarchical Data Structure," ACM Computing Survey, Vol. 16, No.2, June 1984, pp. 187-260.

[3] J. A. Orenstein, F. A. Manola, "PROBE Spatial Data Modeling and Query Processing in An Image Database Application," IEEE Transactions on Software Engineering, Vol. 14, No.5, May 1988, pp. 611-629.

[4] O. Guenther, A. Buchmann, "Research Issues in Spatial Databases," ACM SIGMOD RECORD, Vol. 19, No. 4, 1990, pp. 61-68.

[5] B. C. Ooi, Efficient Query Processing in Geographic Information Systems, Lecture Notes in Computer Science 471, Springer-Verlag, 1990.

[6] R. H. Gueting, "An Introduction to Spatial Database Systems," VLDB Journal, Vol. 3, No. 4, 1994, pp. 357-399.

[7] T. Brinkhoff, H. P. Kriegel, and R. Schneider, "Comparison of Approximations of Complex Objects Used for Approximation-based Query Processing in Spatial Database Systems," in Proc. 9th Int. Conf. on Data Engineering, Vienna, Austria, 1993, pp. 40-49.

[8] H. P. Kriegel, H. Horn, and M. Schiwietz, "The Performance of Object Decomposition Techniques for Spatial Query Processing," Proc. of 2nd Symp. on Large Spatial Databases, Lecture Notes in Computer Science 525, Springer-Verlag, 1991, pp. 257-276.

[9] J. A. Orenstein, "Redundancy in Spatial Databases," Proc. of ACM SIGMOD, 1989, pp. 294-305.

[10] J. A. Orestein, "Spatial Query Processing in An Object Oriented Database System," Proc. of ACM SIGMOD, 1986, pp. 326-336.

[11] C. Faloutsos, W. Rego, "Tri-Cell: A Data Structure for Spatial Objects," Information Systems, Vol. 14, No. 2, 1989, pp. 131-139.

[12] A. Henrich, H. W. Six, and P. Widmayer, "The LSD Tree: Spatial Access to Multidimensional Point and Non-point Objects," Proc. 15th Very Large Data Bases Conf., Amsterdam, Aug. 1989, pp. 45-53.

[13] A. Hutflesz, H. W. Six, and P. Widmayer, "The R-File : An Efficient Access Structure for Proximity Queries," Proc. 6th Data Engineering Conf., Los Angeles, CA, 1990, pp. 372-379.

[14] P. Mishra, M. H. Eich, "Join Processing in Relational Databases," ACM Computing Surveys, Vol. 24, No. 1, 1992, pp. 63-113.

[15] T. Brinkhoff, H. P. Kriegel, R. Schneider, and B. Seeger, "Multi-Step Processing of Spatial Joins," in Proc. ACM SIGMOD International Conference on Management of Data, Minneapolis, Minnesota, USA, 1994.

[16] M. I. Shamos, D. Hoey, "Geometric Intersection Problems," 17th Annual Symposium on Foundations of Computer Science, IEEE, 1976.

[17] T. R. Walsh, "On the Size of Quadtrees Generalized to d-dimensional Binary Pictures," Computers and Mathematics with Applications 11, 11, November 1985, pp. 1089-1097.

[18] H. Samet, The design and Analysis of Spatial Data Structures, Addison-Wesley Pub., 1990.