Controlled Decomposition Strategy for Complex Spatial Objects*

Yong-Ju Lee
Information & Computer Center
Korea Environmental Technology Research Institute
9-2 Samsung-dong, Kangnam-ku, Seoul, 135-090, South Korea
yjlee@keins.ketri.re.kr

Dong-Man Lee, Soo-Jung Ryu, and Chin-Wan Chung
Department of Information and Communication Engineering
Korea Advanced Institute of Science and Technology
207-43, Cheongryangni, Dongdaemun, Seoul 130-012, South Korea
{dmlee, sjryu, chungcw}@romeo.kaist.ac.kr

Abstract

The efficient query processing for complex spatial objects is one of most challenging requirements in many non-traditional applications such as geographic information systems, computer-aided design and multimedia databases. The performance of spatial query processing can be improved by decomposing a complex object into a small number of simple components. This paper investigates a natural trade-off between the number and the complexity of decomposed components. In particular, we propose a new object decomposition method which can control the number of components using a parameter. The proposed method is able to fine-tune the trade-off by controlling the parameter. An optimal value of the parameter is explored through experimental measurements. The decomposition method with this optimal value outperforms traditional decomposition methods. The gain by applying the optimal value is more clear as the complexity of spatial objects increases.

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 expected to be required for exact calculations with respect to locating the spatial object. In order to locate a spatial object efficiently, the spatial ob-

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

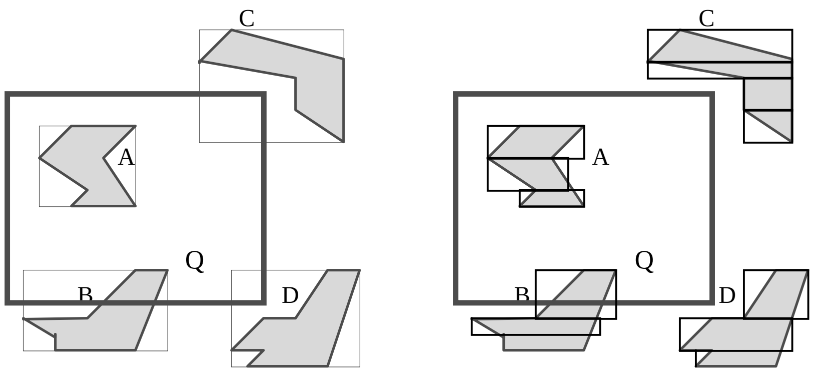
ject, in general, has to be approximated before any geometric computations are applied.

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 the irregularly shaped spatial object. The 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[1-3] 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[4] and object decomposition techniques[5] are fallen to the second approach.

Two well known approximation methods, the *filtering-refinement*[6] and the *object transformation*[7-8], 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 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 $2k$-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 number of simple spatial components such as trapezoids, lead to both a better quality of the approximation and simpler spatial objects. However, the number of decomposed components could result in a query processing overhead, since refinement steps should be tested redundantly by components with the same object identifier. This is illustrated in Example 1.

Example 1 (redundant refinements of decomposed objects):
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 the 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 may be a number of different components labeled with the same identifier, and there are needs for a number of *redundant refinements* on decomposed components. For instance, the objects A and B have to deal with three times redundant refinements, respectively.



(a) existing MBR approach (b) object decomposition approach

Figure 1. Different approximation approaches

End of Example 1.

As described in Example 1, 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 amount of redundant refinements of such complex objects, the efficiency of spatial queries is decreased. Therefore, the development of a new object decomposition method to overcome this problem is essential.

To solve the problem described above, we propose a new object decomposition method which divides a polygon into two sub-polygons recursively by splitting its

MBR until a given constraint is satisfied. This method enables a natural trade-off between the number and the complexity of decomposed components, since the number of components can be controlled by a given constraint. Experimental results show the superiority of our new decomposition method compared to traditional decomposition methods.

This paper is organized as follows. In Section 2, we classify object decomposition techniques by properties of the decomposition. Section 3 describes an algorithm for our decomposition method. Section 4 presents a performance comparison with varying values of parameters through experimental measurements. Finally, conclusions appear in Section 5.

2 Object decomposition strategies

The main goal of the following sections is to propose a new object decomposition method for improving the performance of spatial query processing. Primarily, criteria for evaluating the suitability of decomposition techniques are examined which provide essential requirements for an efficient object decomposition method. Then, a classification of different decomposition techniques is given by properties of the decomposition.

2.1 Evaluation criteria

In order to achieve the best decomposition strategy for spatial query processing, it is necessary to take into account a number of requirements concerning object representations. The following properties are important requirements for developing a new efficient object decomposition method.

• *Minimizing the number of components*

High redundancy induced by a large number of components results in a storage and query processing overhead. Due to the high amount of redundancy incurred by traditional object decomposition techniques, the storage overhead is unacceptable and the performance of spatial queries is inefficient. Therefore, an important goal of developing a new object decomposition method is to minimize the number of generated components.

• *Simplification of complex objects*

In the case of complex spatial objects, a major factor determining query performance is the performance of the refinement step. This is due to the fact that the more complex spatial objects are, the more time-consuming geometric algorithms are required for inspecting exact representations. The performance of the refinement step can be improved by decomposing complex spatial objects into a set of simpler components. In this respect, the simplification of complex objects is an essential requirement in the query processing of complex spatial objects.

• *Good container approximation*
It is of crucial importance to minimize the 'dead space' between a spatial object and its container. Our one premise for the development of a new decomposition method is to use MBRs as containers for the components. So the proposed object decomposition method must supply components that can be well approximated by MBRs.
• *Good run-time performance*
Whenever a new object is inserted into the database, a decomposition of that object must be performed. An algorithm, producing an optimal number of components but requiring exponential order run-time, is unacceptable. Thus, decomposition algorithms with run-time of low order complexity have to be provided.
• *Small amount of storage*
Until now, we only focused on the efficient processing of spatial queries. Nevertheless, it is an important issue to limit the amount of additional storage required for the redundant components.

Most of decomposition techniques proposed up to now are considered to be incapable of fulfilling above criteria completely. Most decomposition techniques [9] known from the field of computational geometry had been developed under requirements different from those arising in spatial query processing. These techniques try to address an optimization with respect to one of the above criteria, e.g., minimizing the number of components, but ignoring other aspects such as the run-time performance. Decomposition techniques in spatial query processing reveal an extreme unbalance between the number and the complexity of their components. While the MBR approximation represents no redundancy with a complex spatial object, object decomposition techniques generate a large number of very simple components such as trapezoids.

In order to fulfill the above criteria totally, the goal of an object decomposition method has to be newly defined. This consideration leads to the development of a new object decomposition method that tunes a trade-off between opposing criteria such as the number and the complexity of decomposed components. In practice, traditional object decomposition methods generate too many components. The problem is that these methods introduce redundancy in an uncontrolled way. Thus, the redundancy of components should be controlled in the object decomposition method. If the redundancy can be controlled, it is possible to fine-tune the trade-off between the retrieval time required for the filter step and the refinement step. The optimal value can be selected according to a cost model.

2.2 Classification of decomposition techniques

There are many object decomposition techniques in use for representing spatial objects. The basic principle of these techniques is a recursive decomposition such as 'divide and conquer' methods. We classify the object decomposition techniques according to the following three properties of the recursive decomposition: *condition* of decomposition, *number* of partitions, and *containers* of components. This classification yields five strategies as in Table 1. We discuss a brief description with respect to five decomposition strategies and attempt to capture their particular properties concerning evaluation criteria outlined in Section 2.1.

Table 1. Classification of decomposition techniques

Strategy	Properties of Decomposition			Indexing Structures
	Condition	Number	Containers	
No	no redundancy	1	MBR	R-tree[10] R^+-tree[11] R^*-tree[12]
Reqular	regular grid	2^d	a set of fixed grids	quad-tree[13] B^+-tree with z-value[7]
Variable Grid	grid and object shape	variable	variable cells	edge-quadtree[14] PM quadtree[14]
Structural	object structure	*n*	MBRs	Cell-tree[15] TR^*-tree[16]
Controlled	controllable parameters	variable (controllable)	decomposed MBRs	will be discussed

d: the number of decompositions which split a region into two equal-sized sub-regions recursively

n: the number of decomposed components

(1) No decomposition (MBR)

Each spatial object is placed in a container, i.e., an MBR. This approach avoids redundant object representations. For preserving the spatial locality and exploiting the spatial clustering, MBRs of spatial objects can be efficiently managed by spatial access methods such as R-tree, R^+-tree and R^*-tree. Since an MBR causes a coarse approximation, however, a whole complex object has to be transmitted to the refinement step even though its result is a false hit. In order to evaluate this complex object, the refinement step requires time-consuming computational geometry algorithms. Figure 2(a) shows an MBR that enclosed a complex spatial object.

We can summarize properties of this approach with respect to the evaluation criteria described in the previous section: it has no redundancy and reveals a strong disadvantage caused by the coarse approximation, and requires expensive computational geometry algorithms.

(2) Regular decomposition

The data space is divided into cells by a regular grid. The container of a spatial object is represented by a set of cells that it intersects. The set of cells over regular grids can be represented by bitstrings, i.e., z-value, representing the recursive grid partitioning of the data space. These bitstrings can be efficiently stored by using an onedimensional access method such as the B-tree. However, this approach includes a large number of fixed cells, and provides no flexibility for the decomposition of the original object. This is due to the strict condition of the partitioning process given by a predefined grid resolution. In this approach, the performance of the filter step is improved by forming a set of fixed cells, but the performance of the refinement step is not improved since the complexity of the object is not generally decreased using this approach. Figure 2(b) shows the grid representation as an example of the regular decomposition.

Summarizing, main properties on evaluation criteria are: it provides a good container approximation, but cannot decrease the complexity of the object. Furthermore, it has a number of components, and requires a large amount of storage.

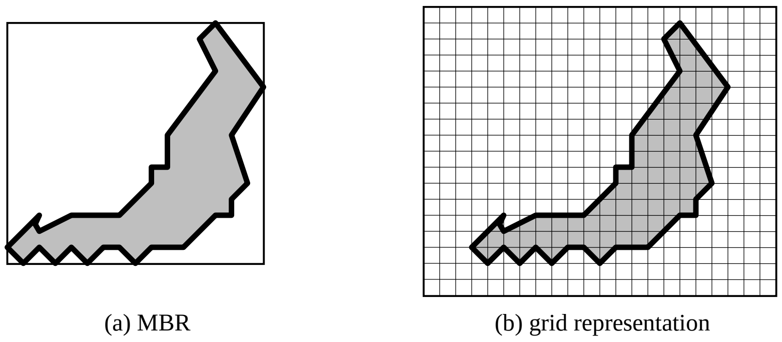


Figure 2. No and Regular decomposition

(3) Variable Grid decomposition

A further approach is taken by grid based methods which are not restricted to a predefined grid resolution, but taking into account of the object location and shape. Similar to the edge-quadtree, a region is sub-divided into four squares repeatedly until a square is obtained that contains a single curve that can be approximated by a single straight line (see Figure 3(a)). This means that no overall minimum cell resolution can be guaranteed. The resolution depends on the shape of the object. Thus, the decomposition process over complex spatial objects can typically produce a large amount of cells of a very small area. However, the refinement step, contrary to Regular decomposition using a fixed grid resolution, is tuned by the occurrence of very simple objects represented by each grid cell.

The properties of this approach can be summarized as follows: the refinement step can be tuned by simple components. However, the amount of redundancy essentially depends on the shape of the object, and can arbitrarily grow.

(4) Structural decomposition

The term 'structural' expresses that the decomposition is oriented on the boundary of a polygonal object. The structural decomposition provides a high degree of choices for component types and decomposition algorithms. Typical types of decomposed components are convex polygons, trapezoids and triangles. These components, similar to the MBR approach, are managed by a spatial access method by placing them into containers, e.g., MBRs. Choosing a proper decomposition algorithm improves the performance of both the filter step and the refinement step. The main drawback is given by a large number of components. Figure 3(b) shows the trapezoids representation as an example of the structural decomposition.

As a consequence, the important properties of this approach are that it leads to a good container approximation as well as simpler spatial objects. Due to a large number of components, however, the storage overhead and the efficiency of spatial queries are unacceptable.

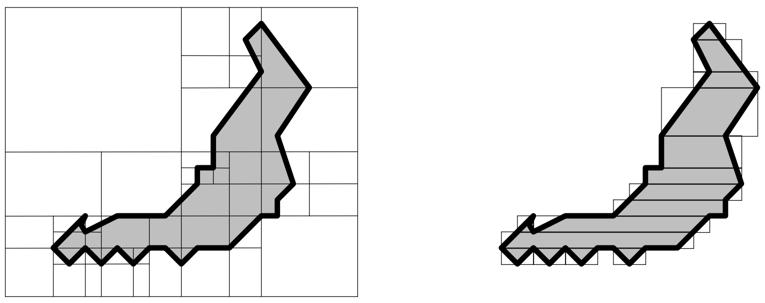

(a) edge-quadtree representation (b) trapezoids representation

Figure 3. Variable Grid and Structural decomposition

(5) Controlled decomposition

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 the left and the right half regions of its MBR space, then a new MBR, called here 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. There are some relationships among decomposition strategies in this section. Both No and Regular decomposition are the limiting cases of Controlled decomposition. The No decomposition is equivalent to AOD(0). Considering an MBR with resolution 2^{d} (i.e., the conceptual grid has 2^{d} pixels), the Regular decomposition is equivalent to AOD(d). Example 2 illustrates the process of Controlled decomposition.

Example 2 (controlled decomposition strategy):

Consider a polygon shown in Figure 4(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, using the vertical and horizontal boundaries in a strictly alternating sequence.

By the vertical boundary, at first, the decomposition result of Figure 4(a) is depicted in Figure 4(b). Since an DMBR of the right sub-polygon is bigger than 2^{-2} of the MBR space, the right sub-polygon is sub-divided into two polygons against the horizontal boundary (see Figure 4(c)). Then the recursive decomposition terminates since every DMBR covers less than 2^{-2} of the MBR space.

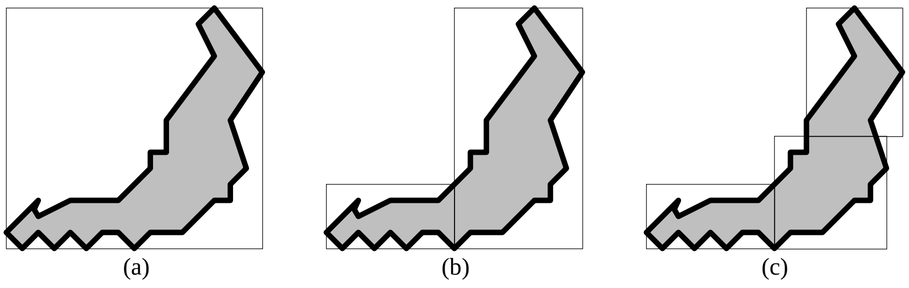


Figure 4. The process of AOD(2)

End of Example 2.

The DMBRs strategy has a parameter that controls the amount of redundancy for each object. It is shown that redundancy could be controlled by the parameter. 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. In Section 4, an optimal value of the parameter for DMBRs will be explored through experiments.

To summarize, this approach makes that it is possible to fine-tune among the number of components, the complexity of components, the container approximation and the amount of storage by controlling the parameter in object representations. The run-time performance is also suitable for most complex objects.

3 Controlled decomposition algorithm

An algorithm which divides a polygon must deal with many different cases, such as those shown in Figure 5. Sometimes several sub-polygons result from dividing a polygon. In this respect, we need an organized way to deal with all these cases. Our algorithm travels around the polygon vertices v_1, v_2, …, v_n, at each step examining whether a line connecting two successive vertices intersects against the middle splitting boundary. By this intersection test, two intersection points will be selected. However, some intersections might occur right at the vertices of the polygon such as Figure 5(b), or coincide with edges of the polygon such as Figure 5(c). One way to solve these cases is to simply ignore points that fall on the splitting boundary. In the particular case of Figure 5(d), the splitting boundary is made by a line connecting two smallest values of *y* (or *x*) axis among intersection points. A solid line in the figure shows this splitting boundary.

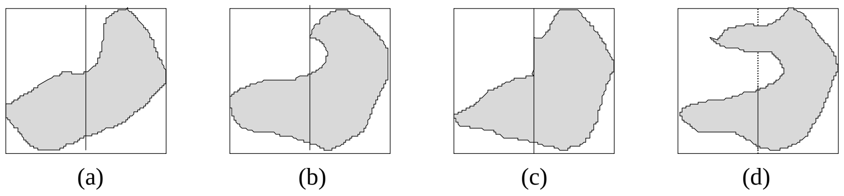


Figure 5. Examples of dividing a polygon

The following decomposition algorithm uses a 'divide and conquer' technique by splitting an MBR recursively until every DMBR fulfills a given AOD(*g*) constraint. 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 against the next splitting boundary recursively. To increase the efficiency of our decomposition algorithm, the DMBRs of divided polygons are inserted into a *two-dimensional(2D) binary tree* that is similar to the LSD tree[17]. 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.

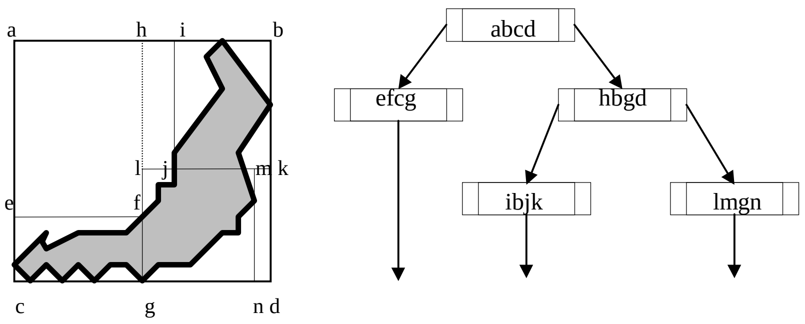
```
Algorithm: Decomposition (p, d)
  Input: A series of polygon vertices p=($v_1$, $v_2$, …, $v_n$), where polygon edges are
         from $v_i$ to $v_{i+1}$ for $i=1,2,...,n-1$ and from $v_n$ to $v_1$. A boolean variable d,
         where d is toggled on the way that divides the region to effect the alternating
         tests on the vertical and horizontal boundaries.
  Output: A new 2D binary tree.
  Method:
  D1. [Find MBR or DMBR coordinates]
        Find minimizing and maximizing values of x and y coordinates from the array
        p of polygon vertices. In case of MBR, initialize 2D binary tree.
  D2. [Check termination condition]
        If DMBR space is bigger than $2^{-g}$ of the MBR space, then do step D3
        through D5 for the array p, otherwise terminate the recursive program.
  D3. [Divide a polygon into two sub-polygons]
        Make a splitting boundary in the center of an input polygon. If polygon edges
        are less than the splitting boundary then add in the array p1, otherwise add in
        the array p2. In case that polygon edges intersect with the splitting boundary,
        find the intersection point and add to both array p1 and array p2.
    D4. [Build 2D binary tree]
        Keep track of the current pointer of the tree. An DMBR related p1 is inserted
        into the left node of the current node, and an DMBR related p2 is inserted into
        the right node of the current node.
    D5. [Call Decomposition algorithm recursively]
        Call Decomposition algorithm for each of array p1 and array p2.
End of Algorithm
```

The following example illustrates our decomposition algorithm.

Example 3 (processing steps of algorithm):
Consider the decomposition strategy illustrated in Example 2. The processing steps of our decomposition algorithm are shown in the following.

- D1. Find an MBR from the polygon shown in Figure 4(a), then initialize a root node in 2D binary tree
- D2. The MBR space is bigger than 2^{-2} of the MBR space
- D3. Dividing the polygon into two sub-polygons by a vertical splitting boundary
- D4. In 2D binary tree, two DMBRs for sub-polygons are inserted into the left and right nodes of the root node
- D5. Call *Decomposition* algorithm for the left sub-polygon
  - D1. Find an DMBR from the left sub-polygon
  - D2. The DMBR is less than 2^{-2} of the MBR space
- D5. Call *Decomposition* algorithm for the right sub-polygon
  - D1. Find an DMBR from the right sub-polygon
  - D2. The DMBR is bigger than 2^{-2} of the MBR space
  - D3. Divide the sub-polygon into two sub-polygons by a horizontal splitting boundary
  - D4. In 2D binary tree, two DMBRs for new sub-polygons are inserted into the left and the right nodes of the current node
  - D5. Call *Decomposition* algorithm for the above sub-polygon in the right half
    - D1. Find an DMBR from the above sub-polygon
    - D2. The DMBR is less than 2^{-2} of the MBR space
  - D5. Call *Decomposition* algorithm for the below sub-polygon in the right half
    - D1. Find an DMBR from the below sub-polygon
    - D2. The DMBR is less than 2^{-2} of the MBR space

The set of DMBRs and its corresponding binary tree are shown in Figure 6(a) and 6(b), respectively.



(a) a set of DMBRs (b) 2D binary tree

Figure 6. DMBRs and corresponding 2D binary tree

End of Example 3.

4 Experimental results

Experimental analyses are conducted for the following reasons:
(1) to determine an optimal value of parameter *g*, G_{opt} ,
(2) to simulate effects of evaluation criteria with varying *g* values.
Both the determination of an optimal *g* value and the experimental measurement of evaluation criteria are studied in this chapter. For this experimental study, we implemented the point query, the region query and the spatial join query. The implementation is made in C language on a Sun SPARCstation 20 running on SunOS Release 5.4.

4.1 Measurement of evaluation criteria

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 7 depicts the analyzed spatial objects and Table 2 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 7. Analyzed spatial objects

Table 2. 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

To simulate the effect of the object decomposition, we examined *the number of components, the quality of the container approximation, the complexity* and *the relative storage requirement* for various values of parameter *g*. Table 3 contains test results for spatial objects presented in Figure 7.

Table 3. Decomposition results for various *g* values

g / Name	0	1	2	3	4	5	6	7
Number of Components								
Park	1	2	5	9	15	28	53	94
Lake	1	6	10	14	18	23	37	60
Korea	1	3	4	9	17	27	51	99
Quality of Approximation								
Park	2.33.	1.89	1.79	1.54	1.41	1.31	1.20	1.17
Lake	6.58	5.53	4.60	3.31	2.74	2.47	2.10	1.78
Korea	2.42	2.17	1.86	1.65	1.39	1.34	1.25	1.20
Complexity (Average Number of Vertices)								
Park	83	43	19	12	9	6	5	4
Lake	206	37	23	17	14	12	9	7
Korea	229	79	60	29	17	12	8	6
Relative Storage Requirement								
Park	1.00	1.10	1.42	1.85	2.52	3.96	6.71	11.24
Lake	2.43	2.98	3.36	3.77	4.19	4.74	6.26	8.78
Korea	2.70	2.92	3.03	3.59	4.46	5.57	8.21	13.51

Results show that the number of generated components increases exponentially as the *g* value increases. This occurs because a large number of smaller components are needed to represent the spatial object with the required accuracy. The quality of the container approximation and the complexity improve as the *g* value increases. Specifically, the improvement in low *g* values is more rapidly than high *g* values. The storage requirement for the object decomposition is incurred by the number of components. Although the reason for the introduction of an object decomposition is to speed up spatial query processing, the object decomposition in a high *g* value leads to a high amount of storage.

From our test, we learned that a best *g* value cannot be obtained by increasing the *g* value due to the penalty of the more storage requirement. Although the approximation quality and the complexity become more effective by higher *g* values, a storage overhead is caused by the large number of components. On the other hand, while the number of components and the storage requirement are more efficiently handled by lower *g* values, bad approximations and high complexities are measured. So, it is desirable to find a *g* value taking into account a balanced ratio between the low and the high values.

4.2 Determination of an optimal *g* value

Our goal is to evaluate which value of a parameter *g* leads to an optimal performance in spatial query processing. For this purpose, the *2D binary tree* introduced in Section 3 is used. The performance of the 2D binary tree is determined by the time spent for comparisons within the directory of the tree and computational geometry algorithms for decomposed components. As this performance strongly depends on the value of parameter *g*, we explicitly measured the processing time for various *g* values using the implementation of different spatial queries.

Queries that we performed are classified into point queries, window queries and spatial join queries. Table 4 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. In the table, we have shadowed the best performing values for each type of queries.

Table 4. 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 4 suggest that the reasoning of the existence of G_{opt} 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.

To be more precise, the query results are presented in figures for spatial objects introduced in the last section (see Figure 8 and 9). The figures depict the following information: the horizontal axis presents g values from 0 to 7, and the vertical axis gives the time for performing queries. The time is given in seconds. Since all figures for the query results cannot be presented due to the space limitation, the number of spatial objects used is fixed to 50 for point queries, 100 for region queries and 75 for spatial join queries. The results presented are typical, and the trends discussed below were observed in all experiments.

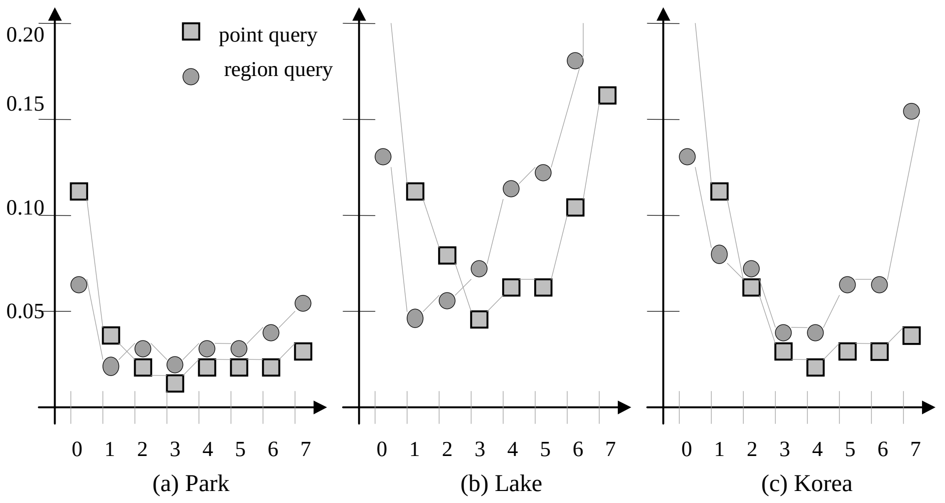


Figure 8. Query processing time for point queries and region queries

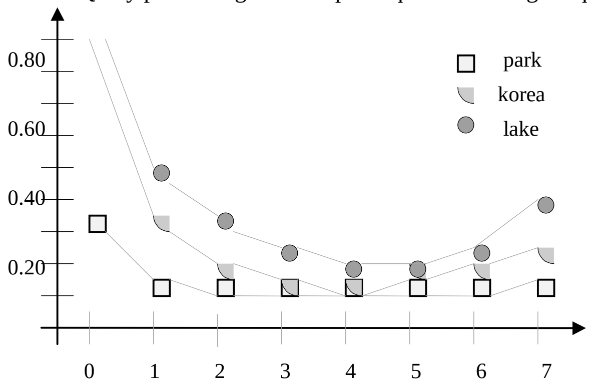


Figure 9. Query processing time for spatial join queries

The shape of graphs reveals that the existence of G_{opt} is apparent. As expected, the G_{opt} from performance curves corresponds to *g* values in the range of [3,4]. The query performance with these *g* values almost always beats the performance with the other *g* values. The more complex the objects are, the more clear the gain of these *g* values appears. For instance, complex spatial objects such as `Lake' and `Korea' show a significant gain by applying the *g* values.

Recall that No decomposition is equivalent to *g*=0, and Regular, Variable Grid and Structural decomposition are related with the large number of components, i.e., the high *g* value. As discussed earlier, only Controlled decomposition can control the parameter *g*. Caused by the reduced amount of components and still low complexity of components, Controlled decomposition is superior to other strategies.

5 Conclusions

We have proposed a new object decomposition method, called DMBRs, which can control the number of decomposed components using a parameter. The proposed method is expected to outperform traditional decomposition methods due to its ability to tune the trade-off among evaluation criteria. An optimal value has been investigated by experimental analyses. For most queries and arbitrary types of objects, the optimal value of the parameter *g* can be obtained around 3. The gain by applying the optimal value is more clear as the complexity of spatial objects increases.

More work is needed to see what happens in situations not explored by these experiments. In particular, it is desirable to analyze the effect of digitized maps used daily in real-world applications. For our experiments, the selected queries are point queries, region queries and spatial join queries. More general types of spatial queries have to be examined in the future.

References

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

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

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

[4] 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.

[5] 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.

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

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

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

[9] F. Preparata, M. Shamos, Computational Geometry: An Introduction, Springer-Verlag, New York, 1985.

[10] A. Guttman, "R-trees: A Dynamic Index Structure for Spatial Searching," Proc. ACM SIGMOD, 1984, pp. 47-57.

[11] T. Sellis, N. Roussopoulos, and C. Faloutsos, "The R^+-tree: A Dynamic Index for Multi-dimensional Objects," Proc. 13th Very Large Data Bases, Conf. Sep. 1987, pp. 507-518.

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

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

[14] H. Samet, "Hierarchical Spatial Data Structures," Proc. of 1st Symp. on Large Spatial Databases, Lecture Notes in Computer Science 409, Springer-Verlag, 1989, pp. 193-212.

[15] O. Gunther, "The Design of the Cell tree: An Object-oriented Index Structure for Geometric Databases," Proc. Data Engineering Conference, 1989, pp. 598-605.

[16] R. Schneider, H.P. Kriegel, "The TR*-tree: A New Representation of Polygonal Objects Supporting Spatial Queries and Operations," Proc. 7th Workshop on Computational Geometry, Bern, Switzerland, Lecture Notes in Computer Science 553, Springer-Verlag, 1991, pp. 249-264.

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