The Multi-scale R-tree

Edward P.F. Chan
Kevin K.W. Chow
Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1
epfchan@emap.uwaterloo.ca

October 24, 1999

Abstract

An important requirement in geographic information systems is the ability to display numerous geometric objects swiftly onto the display window. As the screen size is fixed, the scale of a map displayed changes as the user zooms in and out of the map. Spatial indexes like R-tree variants that are particularly efficient for range queries are not adequate to generate large maps that may be displayed at different scales. We propose a generalization for the family of R-trees, called the *Multi-scale R-tree*, that allows efficient retrieval of geometric objects at different levels of detail. The remedy offered here consists of two generalization techniques in cartography: selection and simplification. Selection means some objects that are relatively unimportant to the user at the current scale will not be retrieved. Simplification means that, given a scale, the display of objects are shown with sufficient but not with unnecessary detail. These two together reduce the time required to generate a map on a screen. A major obstacle to the effectiveness of a Multi-scale R-tree is the proper decomposition of geometric objects required by the simplification technique. To investigate the problem, a Multi-scale Hilbert R-tree is designed and implemented. Extensive experiments are then performed on real-life data and a general and simple design heuristic is found to solve the decomposition problem. We show that, with the proposed design heuristic, the Multi-scale R-tree is a desirable spatial index for both querying and display purposes.

1 Introduction

In applications such as Geographic Information System (GIS), a database comprises a collection of thematic maps and users commonly query geometric objects within a specific region in a database. These queries are typically referred to as *range* or *window* queries and the result is displayed on a screen as a map. Numerous spatial indexes have been proposed in the literature for range queries. See for instance [8, 1, 15, 10]. A good survey on spatial indexes can be found in [7]. With the exception of PR-file and Reactive-tree, existing spatial indexes are primarily concerned with the efficient retrieval of geometric objects within a specific region. They, how-

ever, are not adequate in applications where geometric objects are displayed interactively and in arbitrary scales.

Each map has a scale associated with it. A *scale* is the ratio of distances represented on a map to their true lengths on the Earth's surface. A scale of 1:25000 indicates that a unit of distance on a map corresponds to 25000 of the same unit on the ground. A map of scale 1:x is said to be a *large scale* (*small scale*) if x is small (large, respectively). Informally, objects in a large scale map appear larger when compared to a small scale map.

The *zoom* and *pan* operations are indispensable in displaying the so-called *scaleless* and *seamless* map. As a user zooms out a map, the scale of the displayed map decreases and more geometric objects intersected with the query region. Thus, to display a map with a very small scale, most of the geometric objects in a thematic map are retrieved and displayed. Displaying too many objects will reduce the user's effectiveness in perceiving the relevant information. Moreover, some retrieved geometric objects may become too small to be displayed meaningfully on the screen. For example, in a small scale map, a polyline with several hundred segments may occupy only a pixel on the display, but the system will retrieve all these segments from the secondary storage and then draw them onto the screen. This is not only a waste of CPU time, but also results in many useless secondary disk accesses. This problem becomes more severe for applications where the dataset is large and the map is viewed in real-time. Thus, it is important to have a spatial index that can efficiently store, retrieve and display geometric objects at different levels of detail. This type of index structures are known as *reactive data structures* in the literature [3, 16].

One possible solution is to store the same map in different scales. Thus, a geometric object will be stored as multiple copies in a database; each with a different level of detail. This method not only introduces a high degree of data redundancy, but also has the potential problem of update anomaly. Even if this method is adopted, one needs to know how many scales are required. Another alternative is to store geometric objects in their entirety as usual, and apply a line simplification algorithm, such as the Douglas-Peuker algorithm [5], to obtain the desired scale. This is certainly a more efficient method than the previous approach, as the system needs to store only one copy of the geometric object, which avoids the problem of update anomaly. However, this approach still retrieves more data than necessary.

In this paper, we describe a generalization of the R-tree family called the *Multi-scale (Ms) R-tree* for efficient storage, retrieval and display of geometric objects. A Ms R-tree shares the same indexing structure as its R-tree counterpart. The main difference is that geometric objects in a Ms R-tree are decomposed and stored as one or more sub-objects in the main data file but without any data duplication. In this respect, our design is similar to the PR-file [1]. Storing a geometric object as multiple objects allows the same object be displayed in different scales effectively. However, a major problem with this approach is how to decompose objects in the main data file. To investigate this problem, a *Multi-scale (Ms) Hilbert R-tree* is designed and implemented. Extensive experiments are then performed with real-life data to gain insight into the problem. A general and simple heuristic is found to solve this problem.

Hilbert R-trees, due to its ability of clustering geometric objects well together, are an efficient indexing structure for range queries, spatial joins as well as for nearest neighbor queries [4, 10, 18]. Hilbert R-trees, however, are not designed for displaying geometric objects at different scales. Since a Ms Hilbert R-tree has the same indexing structure as a Hilbert R-tree, it inherits the above-mentioned desirable properties. As a result, the proposed spatial index can be used not only for query processing but also for efficient display. The price for such a design is that when the whole geometric object is retrieved, such as in refinement and/or evaluation steps in query processing, more disk accesses are required.

This paper is organized as follows. The next section surveys related work. Section 3 presents the Ms Hilbert R-tree, and gives algorithms for searching as well as insertion. The experimental results that compare Ms Hilbert R-tree with Hilbert R-tree can be found in section 4. Section 5 gives the conclusions and directions for future research.

2 Previous Work

In view of the need to display maps interactively and in different scales, a number of the so-called reactive data structures have been proposed in the literature. A *reactive data structure* is defined as a geometric data structure with detail levels, intended to support sessions of a user working in an interactive mode [3, 16]. A good survey of this subject can be found in [16]. A Ms R-tree is a spatial index as well as a reactive data structure. In this section, we briefly survey some work that are particularly related to our design. This includes Douglas-Peuker

line simplification algorithm [5], BLG-tree, Reactive-tree [16, 17] and PR-file [9, 1].

2.1 Douglas-Peuker Algorithm

The Douglas-Peuker line simplification algorithm is used to reduce the number of points needed to represent a numerically recorded line, while at the same time, preserve the important features in the original line [5]. The algorithm is based on the idea of *tolerance* and selects a set of suitable vertices to represent the original line. This algorithm accepts a sequence of points which denotes a polyline and a tolerance value as the parameters.

Algorithm Douglas-Peuker (P, t, U)
Input: A sequence ($P_1 \ldots P_n$) of points representing a polyline P and a tolerance value t, where $n>2$ and $t \geq 0$.
Output: U - A simplified version of P.
Method: Initially, the algorithm chooses two points as the $anchor$ and the $floater$, which defines a line segment. In each iteration, the algorithm will adjust the $anchor$ and the $floater$ according to the maximum distance from the intermediate points to the line segment ($anchor$, $floater$). The algorithm will terminate if the $anchor$ is moved to the last point of the original polyline. Each point that has been chosen as an anchor will be stored in U.

1. Let $anchor$ be 1 and $floater$ be n.
2. If $anchor = n$, insert P_n into U and return.
3. For each point P_i, where $floater > i > anchor$, calculate the perpendicular distance from P_i to the line segment (P_{anchor}, $P_{floater}$). Let P_{mid} be a point which has the greatest perpendicular distance to the line segment (P_{anchor}, $P_{floater}$).
4. Let d be the perpendicular distance from P_{mid} to the line segment (P_{anchor}, $P_{floater}$), if P_{mid} exists and d=0 otherwise.
5. If $d > t$, then let $floater = mid$. Go to step 3.
6. If $d \leq t$, then insert P_{anchor} into U. Let $anchor$ be $floater$ and let $floater$ be n. Go to step 2.

The first and last points of P are always in the simplification. The important property of the simplification r generated for a tolerance t is that given a point p in the original line, there is a segment q in r such that the perpendicular distance of p from q is less than t. This simple algorithm turns out to be very effective. Its preeminence has been confirmed in several studies [11, 20, 12, 13].

2.2 Binary-Line Generalization Tree

Given a goemetric object and a tolerance, one can always apply a line simplification algorithm, like the Douglas-Peuker algorithm, to obtain a simplification. In a database environment where an object may be retrieved numerous times with different tolerances, repeated computation on

an object to obtain a simplification may be too inefficient. The Binary-Line Generalization tree (BLG-tree) was developed to overcome this problem [16]. The idea behind this data structure is to store the result of simplification on a geometric object in a binary tree. Consequently, no application of line simplification is required once the tree is constructed for an object, no matter how many times the object is retrieved and displayed. This data structure, together with the Reactive-tree which is going to describe later, form the core of a multi-scale GIS [17].

For a polyline ($P_1, \ldots, P_n$), the line segment (P_1, P_n) represents the most coarse approximation of the polyline. Let P_k be the point which has the largest perpendicular distance from (P_1, P_n), for all the points between P_1 and P_n. P_k will be stored in the root of the BLG-tree along with its perpendicular distance from (P_1, P_n).

The next approximation is formed by two line segments (P_1, P_k) and (P_k, P_n). The root node will have two subtrees: left subtree and right subtree which correspond to (P_1, P_k) and (P_k, P_n), respectively. We repeat the same process to the line segments (P_1, P_k) and (P_k, P_n) until all points in the original polyline are stored in the tree. To retrieve a polyline with a certain tolerance, the algorithm only needs to traverse down the BLG-tree until the required accuracy is met [16].

2.3 Reactive-tree

A Reactive-tree assigns an *importance* value to each geometric object, and each object will be stored in a level according to its importance value [15, 16]. An importance value represents the smallest scale map in which the geometric object is still present. Less important objects get lower values while the more important objects get higher values. Importance values can be assigned by an application program or generated by some mathematical function. During searching, the tree will only retrieve those objects of high enough importance. Thus, the search performance of the system will not be affected by unimportant objects. Recently, an approach that is very similar to the Reactive-tree has been advocated in [6].

Like a R-tree, a Reactive-tree is a multi-way search tree, where each node contains a number of entries. Each node corresponds to one physical disk page. The maximum number of entries that can be stored in a node is denoted as M. Except for the root node and the so-called *pseudo roots* , each node contains between $M/2$ and M entries. The root node has at least two entries, unless it is a leaf node.

Entries are of two types: object entries and tree entries. An object entry has a *minimum bounding rectangle (MBR)*, an importance value, and an object id. An object id is a reference to an object, and a MBR is the smallest axis-parallel rectangle that bounds the object. The importance value is the importance of the object that is referred to by the object entry.

A tree entry has a MBR, an importance value, and a child pointer. The child pointer contains a reference to a sub-tree, and the MBR bounds all the objects in the sub-tree. The importance value of a tree entry is equal to the importance of its child nodes incremented by 1.

Object entries can be stored in any level of a Reactive-tree, in contrast to the R-tree, where object entries can only reside in the leaf nodes. An internal node can have object entries, tree entries, or both. Entries in the same node must have the *same* importance value, and all nodes in the same level are of the *same* importance. Nodes with higher importance values are stored in higher levels of the tree. Since the importance of a tree entry is equal to the importance of its sub-tree incremented by 1, therefore, along any path from the root to a leaf node, the importance values are ordered consecutively in a decreasing order. For example, if the root node of a Reactive-tree has an importance value of 10, then every node in level one will have an importance value of 9. The importance of the nodes in level two will be 8, and so on. The nodes in the lowest level will have importance value of 1.

In a Reactive-tree, an object entry can be stored in a non-leaf node. This implies that the tree cannot be optimally balanced, because not all leaf nodes are in the lowest level of the tree. Thus, the tree's height is not at its minimum. However, since the more important objects can be accessed with less cost in a Reactive-tree, searching can be very efficient if the more important objects are accessed more frequently. A Reactive-tree is particularly efficient if the number of objects has a 'hierarchical distribution' over the importance values. That is, the number of objects in an importance level is 1 or 2 orders of magnitude larger than the number of objects at the next higher importance level [16]. Figure 1a shows a Reactive-tree for a set of objects which has a hierarchical distribution. There is 1 object of importance 3, 3 objects of importance 2 and 27 objects of importance 1. The tree's height is about 3 and is well-filled and balanced.

If the data do not have a hierarchical distribution, the tree can become unbalanced. Figure 1b shows a Reactive-tree with a set of spatial data that do not have a hierarchical distribution. In figure 1b, there are 27 objects of importance 3, 3 objects of importance 2 and 1 object

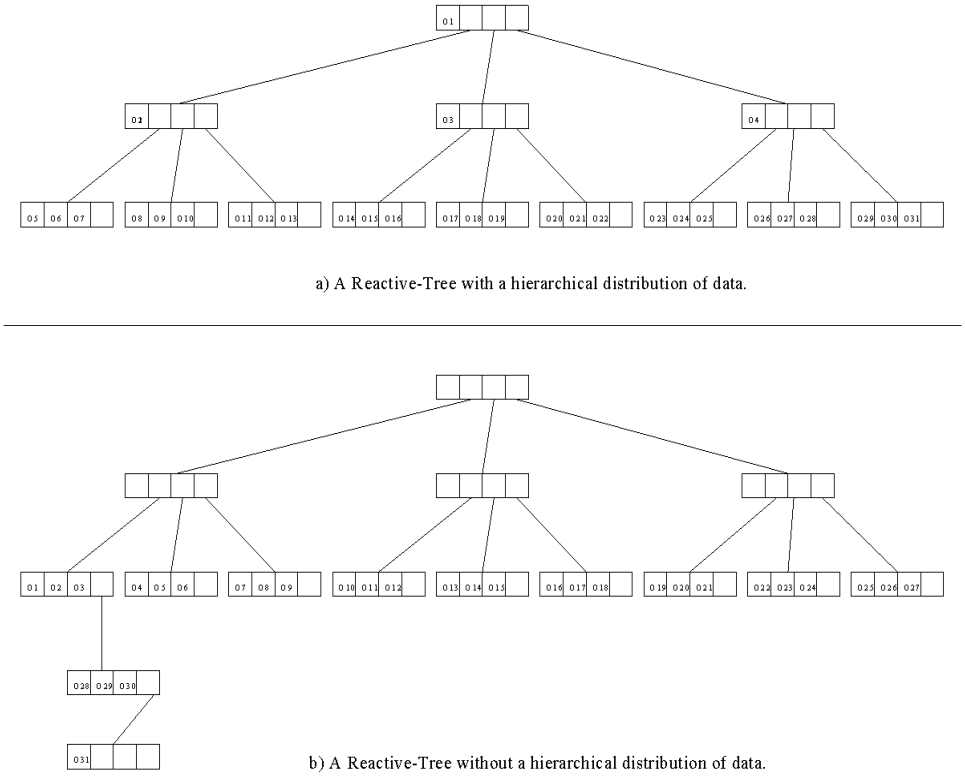


Figure 1: Two Reactive-trees with same number of data but different distribution over the importance value

of importance 1. The Reactive-tree in figure 1b has a height of 5, while the tree in figure 1a has a height of 3. Moreover, none of the objects in the Reactive-tree in figure 1b are stored in a higher level than any object in figure 1a. Thus, searching in the Reactive-tree is very inefficient if the data do not have a hierarchical distribution.

Since there is a one-to-one correspondence between the level and the importance value, the number of importance values cannot be too large or the tree will become too tall. Another restriction is that the importance values must be ordered consecutively.

In short, the performance of the Reactive-tree not only is data dependent but also relies on how well the program assigns importance values to geometric objects. To create a good importance value generator, the designer must have some prior knowledge about the data and the application. In most cases, it is difficult for the designer to have this information in advance.

2.4 Priority Rectangle File

The Priority Rectangle File (PR-file) was designed to efficiently store and retrieve geometric objects in arbitrary scales. It is based on the design of R-File [9]. Unlike the Reactive-tree, an object in a PR-file is not stored as an atomic unit. The PR-file makes use of a line simplification algorithm, which will select some of the line segment endpoints from a polyline according to the desired scale. For each vertex, the PR-file will assign a *generalization index* to indicate the smallest scale in which the vertex is selected. Vertices with the same generalization index value from the same object are clustered together to form a new geometric access structure. Each vertex is assigned with a sequence number to indicate its position in the original object. To obtain an object of a larger scale, the PR-file will merge vertices of the object at a smaller scale with other vertices in the larger scales. The merged vertices are obtained by sorting on their sequence numbers.

Although a geometric object in a PR-file is partitioned into different parts, to simplify our discussion, let's still refer to each part of the object as an 'object'. Each object has a *priority* number, which is equal to the generalization index value of the vertices within the object. Each priority number corresponds to a scale in the map. A lower priority number corresponds to a smaller scale map, while a higher priority number corresponds to a larger scale map. For a relatively small scale map, queries will only return those objects of relatively low priority; objects of higher priority are omitted. For example, if priority number 2 corresponds to a 1:100 map, a query for a 1:100 map will only retrieve those objects which have a priority number 1 or 2. Objects with priority 3 or higher will not be retrieved. If an object of priority 1 and an object of priority 2 contain vertices that originally come from the same object, these vertices will be merged together and then sorted by their sequence numbers to obtain the original object. To facilitate the search process, *data blocks* and *directory blocks* are arranged in certain way so as to minimize the block accesses. Interested readers please refer to [1] for detail of insertion and deletion as well as splitting and merging of data blocks and directory blocks.

In a PR-file, a geometric object is broken into multiple objects and each object is assigned with a priority value. Each priority value corresponds to a map scale. For each scale, objects which have a higher priority value than the priority corresponding to the map's scale will not be retrieved. Objects which have higher priority values than the required priority contain the unnecessary detail and are not retrieved from the disk. This differs from a Reactive-tree in

which the whole object is retrieved if its important value meets the retrieval criteria. In a Reactive-tree, the simplification is generated with the help of a BLG-tree while in a PR-file it is generated by merging points with the same object id.

For a Reactive-tree, there is a one-to-one correspondence between the tree's level and the importance of the objects. Thus, a Reactive-tree can become grossly unbalanced if the objects do not have a 'hierarchical distribution' over the importance value. A PR-file overcomes the problem by using a directory split operation that can dynamically adjust to the different distribution of data. In a PR-file, directory entries of different priorities can be stored in the same node, and hence, there is no one-to-one correspondence between the tree's level, and the priority of the object. Nevertheless, the resulting tree is not, in general, height-balanced.

Although a PR-file has some advantages over a Reactive-tree, it has its own drawbacks. In order to achieve its desirability, the overall structural design as well as insertion and deletion algorithms are rather complex. Moreover, a PR-file is based on a R-file, and hence, it inherits the R-file's problems. More specifically, objects that fall on the center line and the center point of a map have to be taken care of in a different way. As geometric objects are partitioned into objects, and these objects are retrieved and regenerate by merging the sets of points involved. These all imply higher storage and processing cost. Another problem of the PR-file is the size of its directory entry. A directory entry in a PR-file has a MBR, a reference, a cell border and a priority number. In contrast, a tree entry in a Reactive-tree has a MBR, a reference and an importance value. The larger directory entry implies that there is lower fan-out in a PR-file, which can make a PR-file less efficient. Finally, the effectiveness of a PR-file hinges on the proper assignment of priority values to objects. This important problem, however, is not addressed by the authors.

3 Multi-scale Hilbert R-tree

Members of R-tree family consist of an index and a main data file which stores the actual geometric objects. See for instance [8, 19, 2, 10]. To illustrate how a Ms R-tree is designed, we shall concentrate on a variant called *Ms Hilbert R-tree* in the rest of the discussion. Hilbert R-trees have an advantage over other R-tree variants in its ability to cluster objects well in a 2D space [10]. Although a specific variant is used as an illustration, it should be clear from the

subsequent discussion that the techniques employed are applicable to other variants as well. In fact, the *selection* and *simplification* techniques discussed later not only are applicable to other R-tree members but also to file structures like the PR-file as well.

In an environment where geometric objects are retrieved and displayed in arbitrary scales, it is essential to retrieve the right number of data points so as to minimize secondary accesses. Clearly, it is imperative that the *simplification* generated is visually identical to the original object when it is displayed on the screen. In a Ms Hilbert R-tree, to minimize disk accesses, points in objects are partitioned and stored as multiple pieces. These pieces are totally ordered with the *higher* level represents simplifications in a *smaller* scale map. A simplification of a geometric object at a larger scale map can be obtained from the simplification at a smaller scale map by adding in more points from pieces at lower levels. The partitioning and the total ordering enable efficient retrieval of points to generate a simplification based on a given resolution. The term *resolution* here refers to the units per pixel when displaying a map onto a screen. Let us look at an example. Assume a geometric object is denoted by a polyline in Figure 2. This polyline is denoted by a sequence of 11 points, where P1 and P11 are (125,306) and (189,306), respectively. In a large scale map, every detail of this object may be required to be displayed. However, in a small scale map, say when the resolution is larger than half its width, displaying every detail becomes unnecessary. Instead, certain subset of points should be selected and displayed with no visual difference than if all points were used.

As an illustration, let us partition the points into two sets:$\{P1, P4, P8, P11\}$ and $\{P2, P3, P5, P6, P7, P9, P10\}$, with the first set at the highest level. Given a resolution t and assume further that all points in the second set are within distance t from the polyline formed by the points in the first set. The *simplification* r generated with points in the first set is adequate and thus the second set is not required. On the other hand, if there is some point in the second set the distance of which from r is greater than or equal to t, we may want to generate the whole object instead. This can be done by merging the two sets together, provided we keep track of the position of points as in the original object.

Ms R-trees decompose and store geometric objects as multiple sub-objects in the main data file. The data points in the sub-objects partition points in the original object. This is the same as what the PR-file proposes. However, Ms R-trees differ from PR-files in other respects and the differences highlight how Ms R-trees overcome some of the problems of PR-files.

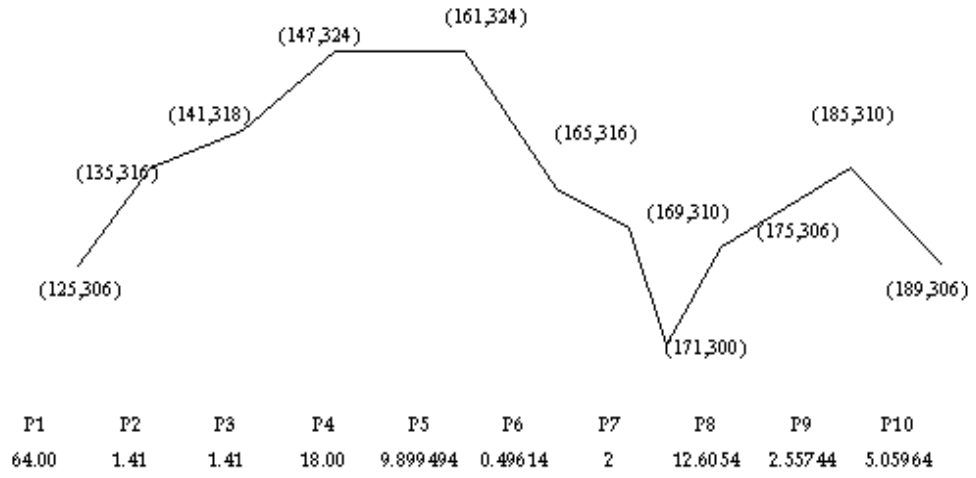


Figure 2: A polyline and the result returned by the MDP algorithm

Ms R-trees have the same index structure as their corresponding R-tree variants; they differ only in the way objects in the main data file are stored. Thus, Ms R-trees are always height-balanced. The insertion and deletion algorithms of Ms R-trees, compared to those of PR-files, are much simpler and easier to implement. In a PR-file, objects that fall on the center line and the center point of a map are taken care of in a different way. Ms R-trees do not have this problem as all objects, no matter where they are, are treated in a uniform manner. Directory entries in PR-files require the cell border be recorded as part of the entry. As this is not needed in Ms R-trees, this implies fan-out of PR-file's directory is lower than that of Ms R-tree's. Finally, we study the important problem of how to decompose objects into sub-objects and give algorithms for their construction. We also perform extensive testing with real-life data to verify the desirability and drawback of Ms R-trees.

3.1 Design Overview

We are primarily concerned with three kinds of geometric objects: points, polylines and polygons. A point is denoted by (x, y) while a polygon and a polyline are represented by a sequence of points in the form $(v_1, v_2, v_3, \ldots, v_n)$. The position of a point in a polygon or polyline is called its *sequence number*.

The Ms Hilbert R-tree employs two techniques in cartography: *selection* and *simplification*. Selection means that for a small scale map, some objects which are relatively unimportant will not be displayed. In our case, the importance of an object is determined by its size relative to

the current resolution. Simplification means objects will be displayed with less detail in a small scale map than in a large scale map.

3.1.1 Selection

A Reactive-tree selects objects based on an importance value assigned to each object. Objects of high importance values are stored higher up in the tree and will be selected even in a small scale map. In a Ms Hilbert R-tree, a simple technique is used to disqualify selection of objects. If the MBR of an entry in an index node is less than or equal to certain threshold, the search on that subtree is terminated and a point centered at the MBR is returned. This simple and inexpensive technique achieves the objective desired. As this technique will benefit the retrieval and display of objects and is easily adaptable to Hilbert R-trees with no cost, it is incorporated into the Hilbert R-tree in the experiments conducted in Section 4. To distinguish it from the original Hilbert R-tree, let's call this index structure the *Hilbert R-tree with selection*. Thus the experimental result reported in this work is for Ms Hilbert R-trees and Hilbert R-trees with selection. From now on, to simplify the notation, we refer to the Hilbert R-tree with selection as just the Hilbert R-tree.

3.1.2 Simplification

Ms Hilbert R-trees use a modified version of the Douglas-Peuker algorithm to simplify polylines and polygonal objects. In section 2.1, there is a description of the Douglas-Peuker algorithm. The algorithm takes a polyline, or a polygon, and a *tolerance* distance to compute a simplified line, or a simplified polygon.

The Douglas-Peuker algorithm is simple and easy to use, but it may not be efficient for some applications. For instance, in the case where we want to compute simplified lines for a polyline over a set of tolerance values, we will have to run the algorithm once for each tolerance value. Given that the algorithm has the worst case running time of $O(n^2)$, it is not efficient for such an application.

The Modified Douglas-Peuker (MDP) algorithm below takes a polyline object and returns an array D. D has the same size as the number of points in the polyline P. Each value $D[i]$ in the array D represents the largest tolerance value where the point P_i will be selected. Thus, to obtain the simplified version of P for a tolerance value t, the algorithm only needs to select each

point P_i in P where $D[i] \geq t$. Hence, once the array D is obtained, we can use it to compute a simplified line for any scale. There is no need to re-run the Douglas-Peuker algorithm for each scale. The function of MDP is similar to that of BLG-tree except that a tree is not needed.

Figure 2 shows a polyline and the array returned by the MDP algorithm.

Algorithm MDP (P, D, s, e, p)
Input: A sequence ($P_1 \ldots P_n$) of points representing a polyline P. D is the array storing the computed tolerance for each point in the sequence. When this is called, $D[s]$ and $D[e]$ have been computed. We want to compute the tolerance values for those points in-between P_s and P_e. p is the maximum perpendicular distance from the previous call.
Output: An array D which stores the largest tolerance values from which the points in P will be selected.
Method: The algorithm will recursively divide a polyline into two pieces by the point which is the furthest away from the segment (P_s, P_e). If P_i is the point which is the furthest away from the line with a perpendicular distance of dis, $D[i]$ will be the lesser value of dis and p.

1. For each point, P_i, where $s > i > e$, calculate the perpendicular distance from P_i to the line segment (P_s, P_e).
2. Let P_{mid} be the point which has the greatest perpendicular distance dis to the line segment (P_s, P_e).
3. Let $D[i] = min(dis, p)$ and set $p = min(dis, p)$.
4. If $mid > s + 1$, call **MDP** (P, D, s, mid, p).
5. If $mid < e - 1$, call **MDP** (P, D, mid, e, p).
6. Return.

For a polyline P, the first and last points are assumed to be the most important and are drawn whenever P is displayed. The tolerance value assigned to these two points is the maximum distance max between any pair of points in the sequence. The tolerance values for other points are computed by calling MDP(P, D, 1, n, max), where P is assumed to have $n > 2$ points.

Given a tolerance t, the *simplification* r returned by the MDP algorithm consists of those points P_i such that $D[i] \geq t$. If $t > max$, no point is returned by the MDP algorithm. This is different from the Douglas-Peuker algorithm where the first and last points are always returned. If $t \leq max$, r enjoys the same property as the simplification returned by the Douglas-Peuker algorithm.

Lemma 3.1 *Let $t \leq max$ and let r be the simplification returned by the MDP algorithm. For each point P_j in P, there is a segment q in r such that the perpendicular distance of P_j from q is less than t.*

[Proof]: Define a binary tree of segments as follows. The tree is constructed by tracing the recursive calls on the algorithm MDP. If P_{mid} is the point chosen in step 2 of MDP algorithm for the segment (P_s, P_e), then (P_s, P_{mid}) and (P_{mid}, P_e) are the *sons* of the (P_s, P_e) in the binary tree. Define the *assigned tolerance* for segment (P_s, P_e) in the binary tree be the minimum of $D[s]$ and $D[e]$. The assigned tolerance will determine if the segment is in the simplification.

Given a tolerance t, the simplification returned by the MDP algorithm is precisely the set of lowest segments in the tree with assigned tolerance greater than or equal to t. Let us show that the simplification is in fact within t. Clearly every point in the simplification is within t. Let P_j be a point in P but not in the simplification. That is, $D[j]<t$. Then there exists the lowest ancestor edge (P_s, P_e) such that $s > j > e$ and its assigned tolerance is just greater than or equal to t. Such an ancestor segment guarantees to exist. The segment (P_s, P_e) is in the simplification. The sons of (P_s, P_e) have an assigned tolerance t', where $t'<t$. By step 1, the perpendicular distance of P_j from (P_s, P_e) is less than or equal to t'. Thus, P_j is within the tolerance t. □

For a polygon P, simplifications are generated as follows. A pair (s, e) of vertices the distance of which is the largest among all pairs of vertices is first determined. Let the maximum distance be max. P is retrieved only if the resolution is less than or equal to max. Two sequences of points from P, one that starts with s and ends with e and the other that starts with e and ends with s, are constructed. The tolerance values for the first and last points in both sequences are initially assigned with max. The tolerance values for the rest are computed by calling MDP twice with the two sequences as the input.

3.2 Data Organization and Algorithms

3.2.1 Data Structures

A Ms Hilbert R-tree is exactly the same as a Hilbert R-tree except that a data object may *not* be stored as one object. A Ms Hilbert R-tree of k *scales* if geometric objects in the file are stored in k scales. The k scales are defined by k ranges of values and are totally ordered with the highest level as level 1.

For a Ms Hilbert R-tree with k scales, where $k> 1$, the tolerance values for each scale is defined by k-1 numbers: $t[1],\ldots,t[k-1]$ where $t[1] > t[2] > \ldots > t[k-2] > t[k-1]$. The ranges for k scales are $[t[1],\infty)$, $[t[2],t[1])$,..., [0, $t[k-1]$), respectively. The scale or range $[t[1],\infty)$

is said to be the *highest* or the *first* while $[0, t[k-1])$ is the *lowest* or the k^{th}. A point is said to be in one of the k scales if its computed tolerance value falls into the corresponding range. A Ms Hilbert R-tree of 1 scale is precisely the Hilbert R-tree. In a Ms Hilbert R-tree, vertices of an object are organized according to the given k scales. The scales must be specified before an index tree and a data file are constructed.

The MDP algorithm will run once for every polyline or polygonal object. The result will be stored in an array denoting the (largest) tolerance value for each point in the object. For each point P_i, it is assigned to a scale in which $D[i]$ belongs. For example, if the polyline P in Figure 2 is going to be stored in a Ms Hilbert R-tree of 4 scales, and the tolerance ranges for 4 scales are defined by the following numbers: 100, 9 and 1, then the points are partitioned into following sets, from the highest scale to the lowest: $\emptyset$, {P1,P4, P5, P8, P11}, {P2, P3,P7, P9, P10}, {P6}.

Thus, in a Ms Hilbert R-tree of k scales, objects are stored as k sub-objects in the main file. Or equivalently, objects in the file are said to be stored in k scales. A sub-object is denoted by a tuple (*pts*, *seq#*, *next*, *scaleExist*), where *pts* and *seq#* are arrays of points and sequence numbers, respectively. They are of the same size and are the points of the sub-object and their sequence numbers. *next* points to the sub-object in the next level. *scaleExist* is a boolean array of size k indicating which scale is empty. *scaleExist* is non-empty exactly at the highest level. It is used in retrieval to determine if the *next*, if it exists, points to which lower scale. It is worth noting that the highest level sub-object *always* exists even if there is no point in the sub-object as it always contain a non-empty *scaleExist*.

Given a tolerance value or resolution t, to generate the simplification of an object within t, determine the scale s in which t is in and retrieve all non-empty sub-objects in s and its higher scales. Once they are retrieved, the points are merged to obtain the simplification. Let us illustrate with the polyline P above. Suppose the current resolution is 5. As 5 falls into the third scale, then the sub-objects in the first, second and third scale of P are retrieved. The simplification obtained contains all points except point P6. To facilitate the retrieval, the k sub-objects are stored in different files and they are linked together as a single linked list. Figure 3 shows a leaf node of the index in a Ms Hilbert R-tree of 3 scales and its data pages that store the geometric objects.

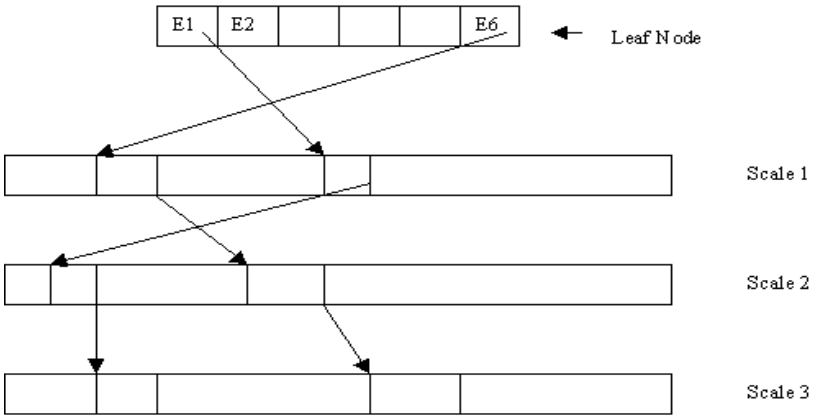


Figure 3: A leaf node and its data pages

3.2.2 Scales Selection

The effectiveness of a Ms Hilbert R-tree depends on the simplification algorithm used and the choice of scales. The choice of scales depends on many factors such as screen size, characteristics of objects in the thematic map as well as the scale in which they were digitized. Thus, the major problem of using the Ms Hilbert R-tree (in fact, the same for PR-file) is to find a general and simple rule for determining the scales that can store and retrieve objects efficiently. Extensive experiments have been performed on a set of thematic maps. A general and simple heuristic is found and is described in Algorithm ScalesGeneration. We are going to demonstrate its effectiveness in Section 4.

Given a set of objects in a thematic map, there are many ways to partition points in objects into different scales. Objects in the same file could be stored in the same scales or in different scales. However, as objects in a window query are going to be displayed in the same scale, partitioning objects according to the same scales would be a better choice in general. Other important factors are the average number of segments in an object, the average object size and the resolution at *maxpack*. The average number of segments, denoted as $segment_{number}$ is just the average number of segments of an object in the thematic map. The average object size, denoted as obj_{size} is just the average width of an object in the thematic map. Let the screen size to display the map be of certain size, say $w*h$, measured as units of pixels. Define *maxpack* as the operation to display all objects in the map, with area size $mw*mh$ units, onto the screen so that no more zoom out operation is allowed. Intuitively, the result of *maxpack* is the smallest scale map for the objects involved. Then the resolution at *maxpack*, denoted as $res_{maxpack}$ is

defined as the $max\{mw/w, mh/h\}$.

Algorithm ScalesGeneration ($segment_{number}$,obj_{size}, $res_{maxpack}$)
Input: $segment_{number}$ is the average number of segments in an object, obj_{size} is the average width of an object and $res_{maxpack}$ is the resolution when maxpacking all objects in the thematic map.
Output: k-1 numbers define the k scales for the Ms Hilert R-tree for a thematic map.
Method: A general and simple heuristic for determining k scales for a Ms Hilbert R-tree. First decide the number of levels needed. Then find the ranges of values.

1. Determine the *number of scales* for the thematic map as follows: If the $segment_{number} \leq 6$, just store it as a Hilbert R-tree and return null. If $10 \geq segment_{number} \geq 7$, the number of scales is 3. If $segment_{number} > 10$, set the number of scales to 4.

2. Having determined the number k of scales, find the scales as follows. As indicated in subsection 3.2.1, k scales are defined by k-1 numbers and they are determined as follows: Let *quotient* be $min(obj_{size}, res_{maxpack})$. The initial *quotient* value denotes the resolution of the smallest scale map in which an object is displayed. Repeatedly divide *quotient* by 4 to obtain the k-1 numbers. Return the k-1 numbers.

For instance if $min(obj_{size}, res_{maxpack}) = 100$, then the 4 scales are defined by the following 3 numbers: 25,6.25,1.5625.

This algorithm is used in Section 4 to construct the Ms Hilbert R-trees in our experiments.

3.2.3 Algorithms

The insertion and deletion algorithms are very similar to those in the Hilbert R-tree. The main difference is the way geometric objects are stored in the main data file. In a Ms Hilbert R-tree of k scales, objects are stored in k scales. Thus, objects are first decomposed into k sub-objects and insert into the k scales before the index entry is created and inserted.

Insertion and deletion

Algorithm Insert (O)
Input: O-the geometric object that is inserted into a Ms Hilbert R-tree of k scales.
Output: The main data file and the index are updated.
Method: O is first decomposed into at most k sub-objects. The sub-objects are inserted from the lowest scale to the highest. The highest scale always contains a non-empty sub-object with a Boolean array *scaleExist* indicating which scale has a non-empty sub-object. Except for the highest scale, a sub-object is empty if it has zero point.

1. Decompose O into at most k non-empty sub-objects. Points in sub-objects are partitions of points in O. Points in O are partitioned by first computing the tolerance value for each point using the MDP algorithm. Depending on the type (either polyline or polygon), points are partitioned into the given k scales according to the algorithm outlined in Section 3.1.2.

2. Insert the non-empty sub-object starting from the lowest level to the highest so that *next* in a sub-object points to the non-empty sub-object at the next scale. Let p be the pointer to the sub-object at the highest level.

3. Insert the leaf entry (p, mbr) into the index, where mbr is the MBR for the object O.

The insertion algorithm into the index is the same as in the Hilbert R-tree. The deletion algorithm is similar to the insertion algorithm.

Searching

The searching algorithm in the Ms Hilbert R-tree is similar to the searching algorithm of the Hilbert R-tree, except that it accepts one more parameter res. The parameter res is the resolution of the current display. In our implementation, the *threshold* for selection in leaf and non-leaf entries are set to $2*res^2$ and $4 * res^2$, respectively.

Algorithm Search (N, $rect$, res)
Input: N is a node in the index, $rect$ a query window and res the resolution in the current display.
Output: Retrieve simplifications of objects that intersect with $rect$. The simplification returned is within the tolerance res.
Method: Apply the selection technique to index tree. Then search the qualified leaf entries and only retrieve those sub-objects that are within the res from the main file.

1. For each entry in node N, if the $MBR \leq threshold$, the entry is disqualified and a point at the entry of the MBR is returned.
2. Recursively call **search** for those nodes pointed at by the qualified non-leaf entries.
3. For those qualified leaf-entries, retrieve the sub-objects at scale i and up, where i is the scale in which res is in. Merge points in sub-objects retrieved and return the simplified object.

4 Performance Evaluation

In this section, we are going to evaluate the performance of the Ms Hilbert R-tree against the Hilbert R-tree. The comparison will be based on various categories, including *the response time*, *the total number of segments drawn*, *the total number of bytes read from the secondary storage* and *the number of seeks*. As both trees use the same selection technique, the number of points drawn is the same in both case.

A number of randomly generated window queries are used in the experiments. *Response time* is the total time for each window query. This includes the time for searching the index, retrieving objects from the disk and drawing objects onto the display. As the response time could be influenced by factors outside our control, in all experiments, two identical tests were conducted and the average is taken as the response time. *Total number of segments drawn* is the total number of line segments drawn for each window query. *The number of bytes read from the secondary storage* is the total number of bytes read from the main data file. *The number of seeks* is the number of times objects in the main data file are retrieved back to main memory. If geometric objects are stored as atomic objects and if no selection is performed, the number of

seeks in a window query is the same as the number of objects intersected by the window query. However, for Ms Hilbert R-trees where simplification and selection are used, this represents the number of secondary accesses to the main data file.

Data Set	Type	No. of Objects	Ave. no. of Segments	Size as Text File (in Mbytes)	Construction Time (in sec.)	
					HRT	Ms HRT
Building	Polygon	8894	9.03	2.97	42.3	65.9
Roads	Polyline	13589	11.15	5.31	82.1	118.3
Drainage	Polyline	15610	27	13.9	120.3	290.5
Vegetation	Polygon	4579	81	12.3	79.4	340.4

Figure 4: Test Data Sets Information

Data Set	File Size (in Mbytes)
Building	1.42
Road	2.82
Drainage	7.01
Vegetation	5.76

Figure 5: Hilbert R-tree Data File Summary

We implemented both Hilbert R-tree and Ms Hilbert R-tree, as well as a display program with zoom and pan operations in Java. All experiments are run with Sun's Java Workshop 2.0 on a 300MHz PII 96 Mbyte memory machine under Window 95. As the index structures are the same in both trees, they are made main-memory resident all the time. The geometric objects are retrieved from the disk whenever they are needed. There is no buffering on the main data file.

4.1 Test Data

The four sets of real-life data used in the experiments are provided by the Faculty of Environmental Studies of the University of Waterloo. The area covered has the size of 60000 * 57000 units. These data sets correspond to different thematic maps on the Region of Waterloo and they are:

1. Building: Buildings in the Waterloo Region.

Data Set	Ave. Object Width	Scales	File Size in Various Scales (in Mbytes)	File Size (in Mbytes)
Building	40	10	0.9	2.11
		2.5	0.55	
		0	0.66	
Road	650	15	0.97	3.9
		4	0.32	
		1	0.58	
		0	2.03	
Drainage	978	15	1.48	8.99
		4	1.15	
		1	2.12	
		0	4.24	
Vegetation	2626	15	0.99	6.9
		4	1.09	
		1	1.71	
		0	3.11	

Figure 6: Ms Hilbert R-tree Data File Summary

2. Road: The road system in the Region. It includes all major highways, main as well as side streets.

3. Drainage: The waterways in the region. It includes streams, creeks, rivers and lakes in the region.

4. Vegetation: Various types of vegetation in the Region.

Information on these data sets are summarized in Figure 4. These data sets include both polylines and polygons and have distinct characteristics. The building data set is relative small and simple and has the lowest average number of segments while the drainage and vegetation are large and complex data sets. The road data set is somewhere in-between the two extremes. The construction time is the time in building the index tree and the main data file from the raw data file. The raw data file is a text file recording the coordinates of points in objects. A disadvantage of Ms Hilbert R-tree is the insertion cost. As shown in the figure, the time to construct a Ms Hilbert R-tree in Java ranges from 50% more to 300% more compared with a Hilbert R-tree. The construction time increases with the average number of segments in an object. This is primarily due to the higher cost in running the MDP algorithm to obtain the tolerance for each point.

For the Hilbert R-tree, the main data file is summarized in Figure 5. For the Ms Hilbert R-tree, the scales are generated with the *scalesGeneration* algorithm in Section 3.2.2. The display window size is 1000*800 pixels. Thus when *maxpacking* a map, the resolution ($res_{maxpack}$) is about 72 unit per pixel. The information is summarized in Figure 6. The average width is obtained by randomly sampling 500 objects from the corresponding file. A Ms Hilbert R-tree incurs some extra storage cost when compared to a Hilbert R-tree. The total storage is about 20% to 50% more. The extra storage is due primarily to storing the sequence number of points in various scales as well as pointers to the next scale. Note that all these files are stored in binary format, not in text format.

Query Set	No. Of Queries	Size Range (as a fraction of the whole map)
QS1	500	.0005 to .001
QS2	200	.002 to .01
QS3	100	.01 to .0333
QS4	50	.02 to .1
QS5	50	.05 to .25
QS6	30	.1 to .5
QS7	20	.3 to 1

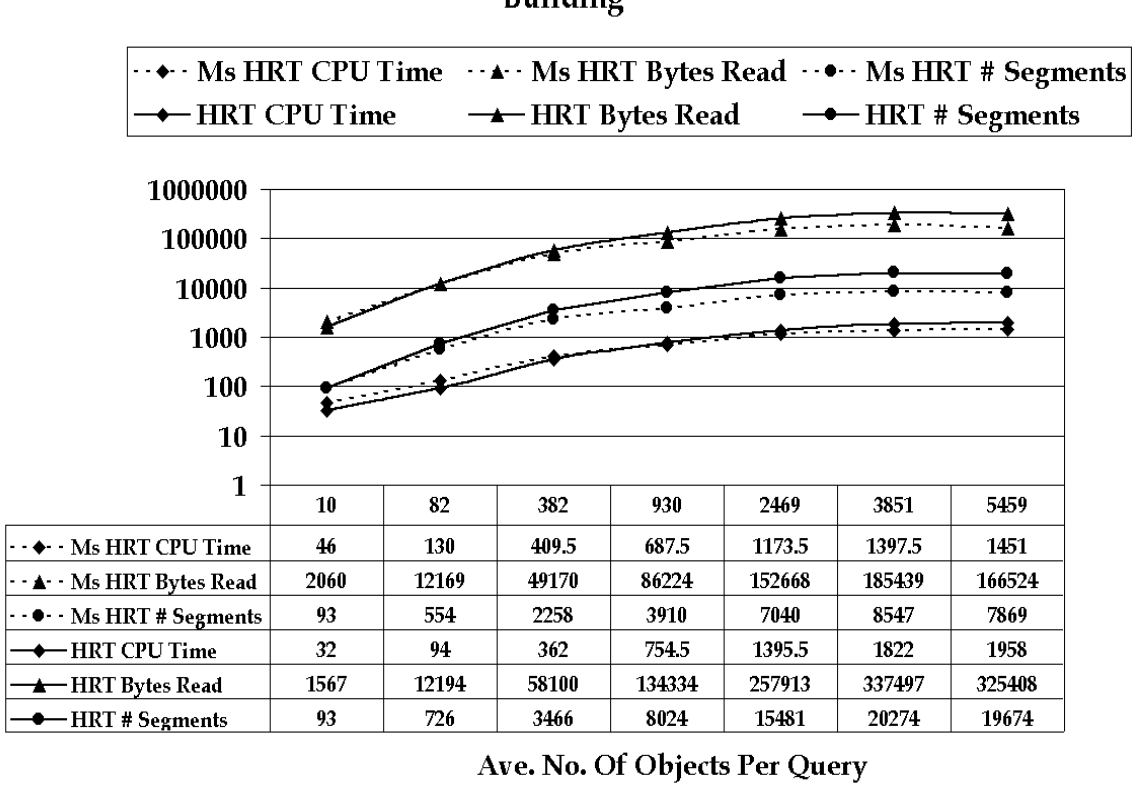
Figure 7: The Query Sets

4.2 Correctness Tests

To verify the correctness of the Ms Hilbert R-tree, it is first compared visually with the Hilbert R-tree in different scales to see if there are any noticeable differences. The resolution passed into the window query algorithm is the current display resolution. There is no noticeable difference that we can detect when we pan and zoom the two maps. Some snapshots in various scales are shown in Figures 22, 23 and 24.

4.3 Test Queries

To evaluate the effectiveness of a Ms Hilbert R-tree, a number of window queries of various sizes are randomly generated. To facilitate the presentation, these queries are grouped into seven query sets denoting queries of various sizes. The query sets are summarized in Figure 7.



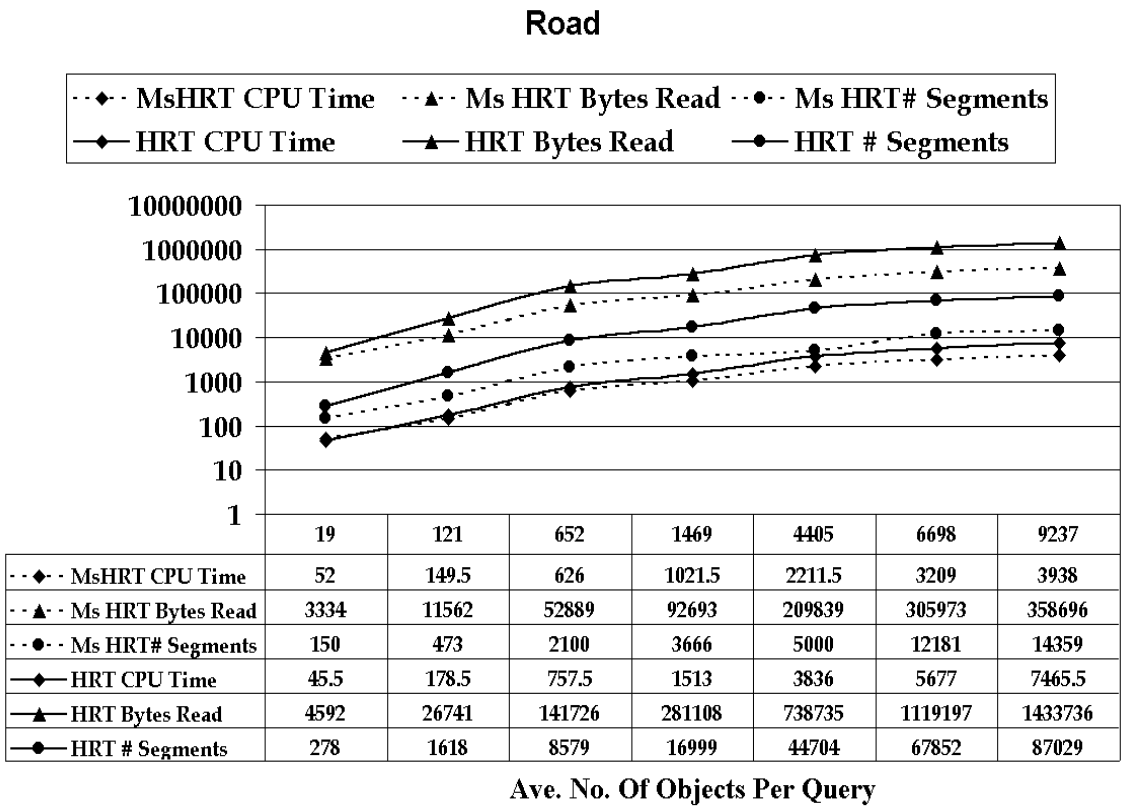
	10	82	382	930	2469	3851	5459
Ms HRT CPU Time	46	130	409.5	687.5	1173.5	1397.5	1451
Ms HRT Bytes Read	2060	12169	49170	86224	152668	185439	166524
Ms HRT # Segments	93	554	2258	3910	7040	8547	7869
HRT CPU Time	32	94	362	754.5	1395.5	1822	1958
HRT Bytes Read	1567	12194	58100	134334	257913	337497	325408
HRT # Segments	93	726	3466	8024	15481	20274	19674

Figure 8: Building Data Set Results

The *Size Range* column records the range of the sizes of queries in the set as a fraction of the area covered by all objects in these thematic maps. These query sets represent large window queries (*QS7*) to small window queries (*QS1*). In general, a smallest window query, on average, intersects up to a dozen of objects whereas a large window query could encompass half of the total area.

4.4 Results

The test results on the four data sets are shown in Figure 8, 9, 10 and 11. The results for Hilbert and Ms Hilbert R-tree are showed in the graph as solid and dotted lines, respectively. The diagrams summarize results on response time, number of bytes read as well as number of segments drawn onto the display. The x-axis represents an average window query in each query set. The number shown on the x-axis is the average number of objects that intersect with a window query in the set. As the query sets represent a smallest to a largest window query on



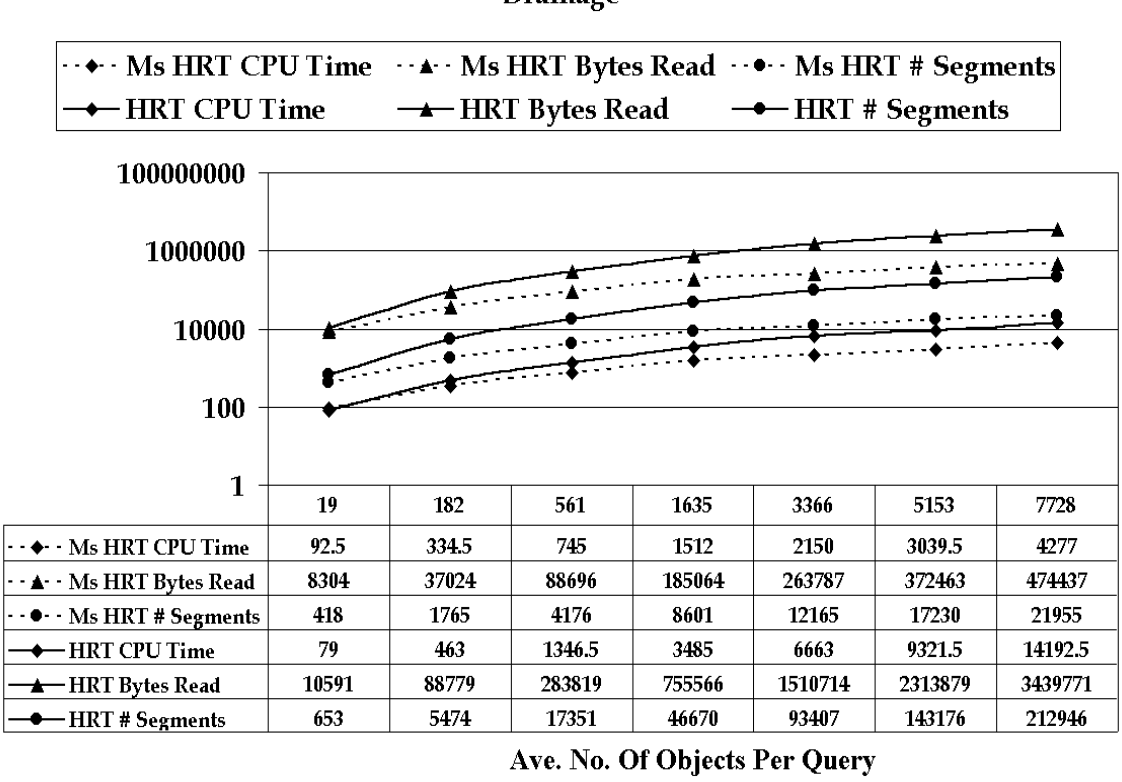
	19	121	652	1469	4405	6698	9237
MsHRT CPU Time	52	149.5	626	1021.5	2211.5	3209	3938
Ms HRT Bytes Read	3334	11562	52889	92693	209839	305973	358696
Ms HRT# Segments	150	473	2100	3666	5000	12181	14359
HRT CPU Time	45.5	178.5	757.5	1513	3836	5677	7465.5
HRT Bytes Read	4592	26741	141726	281108	738735	1119197	1433736
HRT # Segments	278	1618	8579	16999	44704	67852	87029

Figure 9: Roadway Data Set Results

these thematic maps, these graphs show how a Ms Hilbert R-tree performs compared with a Hilbert R-tree over all possible window queries. It is worth noting that the y-axis of all graphs are in *logarithmic scale*. There are several general observations made on these results.

Firstly, except for some cases where the window query size is small, the Ms Hilbert R-tree performs consistently much better than the Hilbert R-tree. Secondly, the curves in the Ms Hilbert R-tree plateau off much faster than the Hilbert R-tree. Lastly, the improvement is most impressive with vegetation and drainage data sets, followed by road and then by building data set.

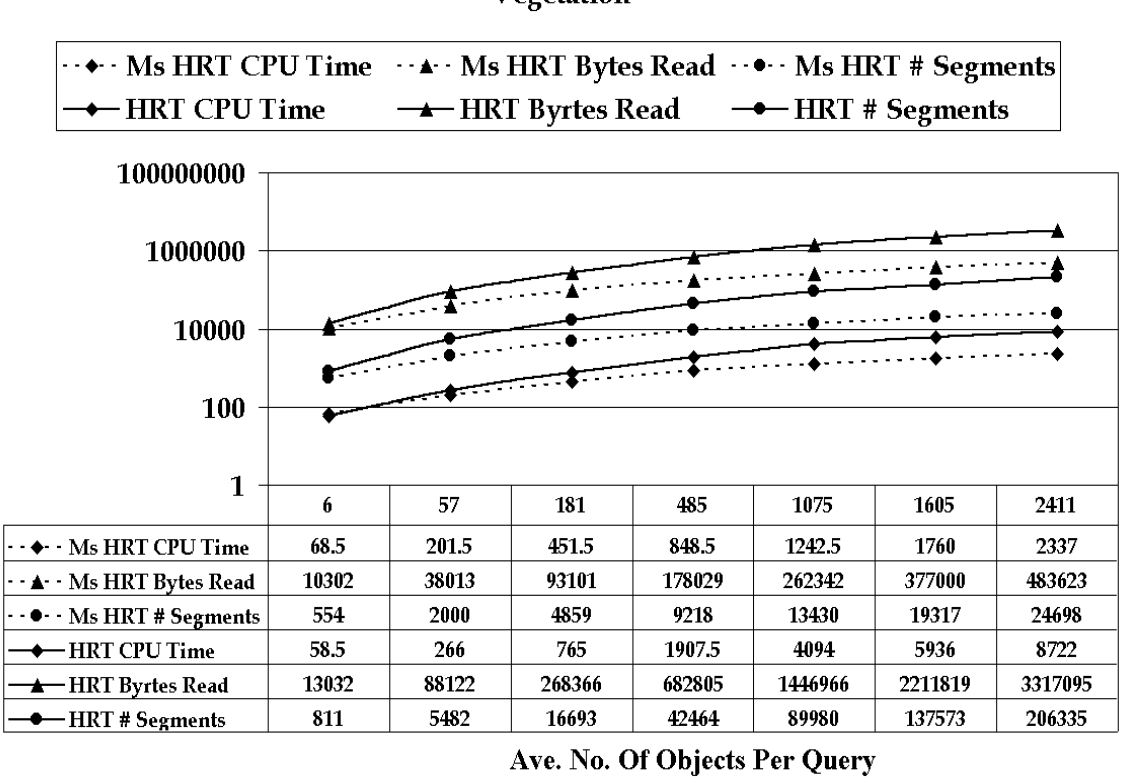
Let us now look at the numbers more closely. Figure 12 shows the ratio of number of bytes read by an average query with a Hilbert R-tree to that of a Ms Hilbert R-tree. This represents the number of bytes read by a Hilbert R-tree for a byte read by a Ms Hilbert R-tree in an average query. Figures 13 and 14 show the same ratio for number of segments drawn and response time, respectively.



	19	182	561	1635	3366	5153	7728
Ms HRT CPU Time	92.5	334.5	745	1512	2150	3039.5	4277
Ms HRT Bytes Read	8304	37024	88696	185064	263787	372463	474437
Ms HRT # Segments	418	1765	4176	8601	12165	17230	21955
HRT CPU Time	79	463	1346.5	3485	6663	9321.5	14192.5
HRT Bytes Read	10591	88779	283819	755566	1510714	2313879	3439771
HRT # Segments	653	5474	17351	46670	93407	143176	212946

Figure 10: Drainage Data Set Results

Except for the building data set with $QS1$, a Ms Hilbert R-tree has a better performance with respect to bytes read and segments drawn in all queries and data sets. The performance improves as the window query size increases. For number of segments drawn, a Hilbert R-tree requires at least as many as in a Ms Hilbert R-tree and up to 9-10 times more in the worst case. For number of bytes read, a Ms Hilbert R-tree again has a better performance except for building data set with $QS1$. In that case, a Ms Hilbert R-tree requires to read about 33% more than a Hilbert R-tree. In all other cases, a Hilbert R-tree requires up to 7 times the number of bytes read by a Ms Hilbert R-tree. These results demonstrate the superiority of the Ms Hilbert R-tree in minimizing data retrieved. The desirability of the Ms Hilbert R-tree is further supported by results on response time, as shown in Figure 14. Except for $QS1$ on all data sets, and for $QS2$ and $QS3$ on the Building data set, the time required to display an average window query is longer with a Hilbert R-tree than with a Ms Hilbert R-tree. It ranges from 15% up to 4 times longer to display an average window query result with a Hilbert R-tree. It should be



	6	57	181	485	1075	1605	2411
Ms HRT CPU Time	68.5	201.5	451.5	848.5	1242.5	1760	2337
Ms HRT Bytes Read	10302	38013	93101	178029	262342	377000	483623
Ms HRT # Segments	554	2000	4859	9218	13430	19317	24698
HRT CPU Time	58.5	266	765	1907.5	4094	5936	8722
HRT Byrtes Read	13032	88122	268366	682805	1446966	2211819	3317095
HRT # Segments	811	5482	16693	42464	89980	137573	206335

Figure 11: Vegetation Data Set Results

pointed out that the time required to display smaller window queries is small (less than half a second).

In a Ms Hilbert R-tree, objects are decomposed into a number of sub-objects to try to reduce the secondary accesses. If objects are decomposed into more sub-objects, the amount of data will be reduced but the number of seeks will increase. Thus the k scales chosen will severely affect the performance of a Ms Hilbert R-tree. Figure 15 shows the average number of seeks per object retrieved in each query set. As expected, in a small window query where more detail of an object is displayed, the number of seeks is much larger than one. For middle and large queries, however, the number of seeks required by a Ms Hilbert R-tree is about or less than one. The exceptionally small numbers in the large window queries for the Building data set are primarily due to the selection technique employed in a Ms Hilbert R-tree and because the building objects are small.

In sum, the Ms Hilbert R-tree has a better, and in some cases, significantly better perfor-

Query Set	Building	Road	Drainage	Vegetation
QS1	0.76	1.38	1.28	1.26
QS2	1.00	2.31	2.40	2.32
QS3	1.18	2.68	3.20	2.88
QS4	1.56	3.03	4.08	3.84
QS5	1.69	3.52	5.73	5.52
QS6	1.82	3.66	6.21	5.87
QS7	1.95	4.00	7.25	6.86

Figure 12: Bytes Read Comparison

Query Set	Building	Road	Drainage	Vegetation
QS1	1.00	1.85	1.56	1.46
QS2	1.31	3.42	3.10	2.74
QS3	1.53	4.09	4.15	3.44
QS4	2.05	4.64	5.43	4.61
QS5	2.20	8.94	7.68	6.70
QS6	2.37	5.57	8.31	7.12
QS7	2.50	6.06	9.70	8.35

Figure 13: Number of Segments Drawn Comparison

mance than the Hilbert R-tree when displaying objects in various scales. The only cases where the Ms Hilbert R-tree has a consistently worse performance is when the query window size is small, such as those in query set $QS1$. As $QS1$ represents very small window queries, the amount of data retrieved from the disk and the time to display such a small window is small. Thus the extra cost required by the Ms Hilbert R-tree is negligible. On the other hand, as window size increases, it takes much more secondary accesses and much longer time to display objects; the saving with the Ms Hilbert R-tree is significant, especially when the map is displayed interactively. It is expected that the improvement is even more noticeable if the window query result is shipped via a network.

4.5 Display Window Changed

The scales used in the experiments in the previous section is derived based on the assumption that the size of the window display is 1000*800 pixels. A related question is how the change in the window size affects the performance. Series of similar experiments were performed by displaying the image onto a 500*400 pixel window. The results are summarized in Figures 16, 17 and 18. The numbers in these figures show the ratio of response time, bytes read, segments

Query Set	Building	Road	Drainage	Vegetation
QS1	0.91	0.88	0.85	0.85
QS2	0.79	1.19	1.38	1.32
QS3	0.88	1.21	1.81	1.69
QS4	1.15	1.48	2.30	2.25
QS5	1.23	1.73	3.10	3.29
QS6	1.31	1.77	3.07	3.37
QS7	1.37	1.90	3.32	3.73

Figure 14: Response Time Comparison

Query Set	Building	Road	Drainage	Vegetation
QS1	2.60	2.21	2.79	3.67
QS2	1.82	1.63	1.93	2.39
QS3	1.46	1.48	1.59	2.09
QS4	1.04	1.12	1.18	1.59
QS5	0.64	0.81	0.81	1.13
QS6	0.49	0.78	0.73	1.08
QS7	0.28	0.64	0.60	0.93

Figure 15: Number of Seeks per Object in a Query

drawn with the Hilbert R-tree to that of the Ms Hilbert R-tree.

The performance of the Ms Hilbert R-tree has improved slightly in these tests compared to the case where the window size is larger. As discussed in the previous section, the Ms Hilbert R-tree performs worse for small window queries. Or equivalently, the Ms Hilbert R-tree performs worse when displaying a large scale map. This is due primarily to less simplification can be done on the objects. When the display window size reduces by half, the resolution in a query doubles. That is with the same window query, the map displayed is a smaller scale map when displayed in a smaller window. Thus more simplification can be performed and fewer details are shown. These all benefit the Ms Hilbert R-tree. Conversely it can be expected that when the window size is made larger, the performance of the Ms Hilbert R-tree will degrade slightly. Nevertheless, it can be expected that the Ms Hilbert R-tree is still a better choice for displaying geometric objects in a spatial database even if the display window size is double in size. The change in size of display window does not seem to have a significant impact on the Ms Hilbert R-tree performance.

Query Set	Building	Road	Drainage	Vegetation
QS1	0.88	0.92	0.92	1.06
QS2	1.05	1.29	1.57	1.80
QS3	1.18	1.56	2.57	3.03
QS4	1.32	1.71	3.35	3.63
QS5	1.28	1.92	3.34	3.58
QS6	1.37	2.20	3.88	3.83
QS7	1.36	2.24	3.91	4.06

Figure 16: Response Time Comparison for a 500*400 pixel Window

Query Set	Building	Road	Drainage	Vegetation
QS1	0.88	1.91	1.55	1.54
QS2	1.28	2.73	3.03	2.84
QS3	1.53	3.38	5.24	4.93
QS4	1.85	3.86	6.27	6.10
QS5	1.91	4.18	6.73	6.18
QS6	1.94	4.56	7.73	6.95
QS7	2.09	5.16	8.02	7.06

Figure 17: Number of Bytes Read Comparison for a 500*400 pixel Window

4.6 Whole Objects Retrieval

As the index of a Ms Hilbert R-tree is identical to a Hilbert R-tree, it can be used not only for display purpose, but also for query processing in which the whole geometric object is retrieved. In such an application, however, there will not be as efficient as using the Hilbert R-tree. To retrieve the whole object in a Ms Hilbert R-tree, the resolution or tolerance passes into the query must be set to zero. This forces the search algorithm to retrieve the whole object.

Figure 19 summarizes the retrieval time (but not display) whereas Figures 20 and 21 compare the number of bytes read from the secondary storage and the number of seeks performed, respectively. The time denote the ratio of the average time in retrieving objects in a window query using a Ms Hilbert R-tree to that of using a Hilbert R-tree. Similarly for number of bytes read and the number of seeks.

In general, the time in retrieving the whole object in an average query using a Ms Hilbert R-tree ranges from 50% more to 115% more when compared to a Hilbert R-tree. The amount of data read from disk is about 15% to 32% more than a Hilbert R-tree. The number of seeks performed in a Ms Hilbert R-tree is s times that of a Hilbert R-tree, where s is less than the scale of the tree. The larger the average size of objects in a thematic map, the closer s

Query Set	Building	Road	Drainage	Vegetation
QS1	1.15	2.75	1.93	1.79
QS2	1.68	4.12	3.96	3.38
QS3	2.00	5.17	6.99	5.94
QS4	2.40	5.86	8.39	7.40
QS5	2.44	6.31	8.92	7.47
QS6	2.46	6.86	10.22	8.41
QS7	2.57	7.68	10.52	8.52

Figure 18: Number of Segments Drawn Comparison for a 500*400 pixel Window

is to the scale of the tree. All road, drainage and vegetation objects are of scale 4; and the average number of seeks to retrieve the whole object increases with the average size of these thematic maps. This is due to the fact that the number of empty sub-object in the main data file decreases as the average size of the object increases. Recall that some of the k sub-objects of a geometric object could be empty in a Ms Hilbert R-tree.

Query Set	Building	Road	Drainage	Vegetation
QS1	1.80	1.83	1.74	1.50
QS2	2.05	2.16	1.86	1.58
QS3	2.03	2.03	1.91	1.57
QS4	1.90	2.03	2.04	1.53
QS5	1.91	2.07	2.15	1.53
QS6	1.94	1.96	1.94	1.36
QS7	1.84	1.93	1.91	1.57

Figure 19: CPU Time Comparison for Whole Object Retrieval

Query Set	Building	Road	Drainage	Vegetation
QS1	1.31	1.25	1.20	1.15
QS2	1.32	1.27	1.21	1.16
QS3	1.32	1.27	1.20	1.16
QS4	1.32	1.27	1.30	1.16
QS5	1.31	1.27	1.21	1.16
QS6	1.32	1.27	1.21	1.16
QS7	1.32	1.27	1.21	1.16

Figure 20: Number of Bytes Read Comparison for Whole Object Retrieval

Query Set	Building	Road	Drainage	Vegetation
QS1	2.60	2.68	3.37	4.33
QS2	2.65	2.69	3.35	3.95
QS3	2.62	2.67	3.32	3.95
QS4	2.62	2.64	3.32	3.94
QS5	2.61	2.62	3.31	3.93
QS6	2.62	2.63	3.30	3.94
QS7	2.62	2.63	3.30	3.94

Figure 21: Number of Seeks Comparison for Whole Object Retrieval

5 Conclusion

We proposed a fully dynamic spatial indexing structure which is able to store and display geometric objects in different scales with no data duplication. The new indexing structure is a generalization of the R-tree family called the *Multi-scale R-tree (Ms R-tree)*. The Ms R-tree uses two techniques in cartography: selection and simplification. Selection means some objects which are relatively small will not be displayed. This is basically accomplished by setting a threshold on the size of MBR's in the index. Simplification means objects will be displayed in less detail in a small scale map. Simplification is realized with a line simplification algorithm and by partitioning points in an object into one or more levels.

A major obstacle in using a Ms R-tree is how to decide on the number as well as the scales used. To evaluate the proposed index structure, we implemented both the Hilbert and the Ms Hilbert R-tree in Java and tested with a number of real-life thematic maps. Extensive experiments were then performed on the Ms Hilbert R-tree and an algorithm for constructing scales for a thematic map was proposed in this work. The algorithm is simple and yet seems to be general enough for a wide variety of thematic maps. For the real-life data tested, the suggested scales perform consistently well over all but small window queries. The Ms Hilbert R-tree is particular valuable when the map and/or the number of segments or points in a geometric object is large. It is interesting to see if this algorithm is applicable to the PR-file as well.

One drawback of the Ms Hilbert R-tree is its high insertion cost, which could range from 50% more to several time more than the Hilbert R-tree. The insertion cost increases with the number of segments per object in a thematic map. As a Ms Hilbert R-tree has the same index as in a Hilbert R-tree, it can be used in spatial query processing [4, 18]. However, our experiments

show that it requires up to a third more data retrieval and could take twice as long to retrieve an object than a Hilbert R-tree when implemented in Java. If efficient spatial query processing is an important requirement, data duplication may be required. A related question is if and how the scale information stored in a Ms R-tree be used in refinement and/or evaluation steps in spatial query processing.

References

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

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

[3] Bos, J. Van Den, Naelten, M. Van, and Teunissen, W., "IDECAP interactive pictorial information system for demographic and environmental planning applications," *Computer Graphics Forum, 3*, 1984, pp.91-102.

[4] Brinkhoff, T., Kriegel, and Seeger B., "Efficient Processing of Spatial Joins Using R-Trees," *Proceedings of ACM SIGMOD*, Washington, D.C., 1993, pp.237-246.

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

[6] Horhammer, M. and Freeston, M., "Spatial Indexing with a Scale Dimension," *Proceedings of Symposium on the Design and Implementation of Large Spatial Database*, Hong Kong, China, pp. 52-71, 1999.

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

[8] Guttman, A., "R-trees: A Dynamic Index Structure for Spatial Searching," *Proceedings of ACM SIGMOD International Conference on Management of Data*, Boston, Ma.,1984, pp. 47-57.

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

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

[11] Marino, J.S., "Identification of Characteristic Points Along Naturally Occurring Lines: An Empirical Study," *Canadian Cartography, Vol. 16*, pp. 70-80, 1979.

[12] McMaster, R.B., "The Geometric Properties of Numerical Generalization," *Geographical Analysis, Vol. 19(4)*, pp. 330-346, October 1987.

[13] Monmonier, M.S., "Towards a Practical Model of Cartographic Generalization," *Proceedings of Auto Carto*, London pp. 257-266, 1986.

[14] Oosterom, P.V. and van den Bos, J., "An Object-Oriented Approach to the Design of Geographic Information Systems," *Proceedings of Symposium on the Design and Implementation of Large Spatial Database*, Santa Barbara, CA, pp. 255-269, 1989.

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

[16] Oosterom, P.V., *Reactive Data Structures for Geographic Information Systems*, Oxford university Press Inc. 1993.

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

[18] Roussopoulos, N., Kelley, S. and vincent, F., "Nearest Neighbor Queries," *Proceedings of ACM SIGMOD*, San Jose, CA, 1995, pp.71-79.

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

[20] White, E.R., "Assessment of Line-Generalization Algorithms Using Characteristic Points," *American Cartographer, Vol. 12(1)*, 1985, pp.17-27.

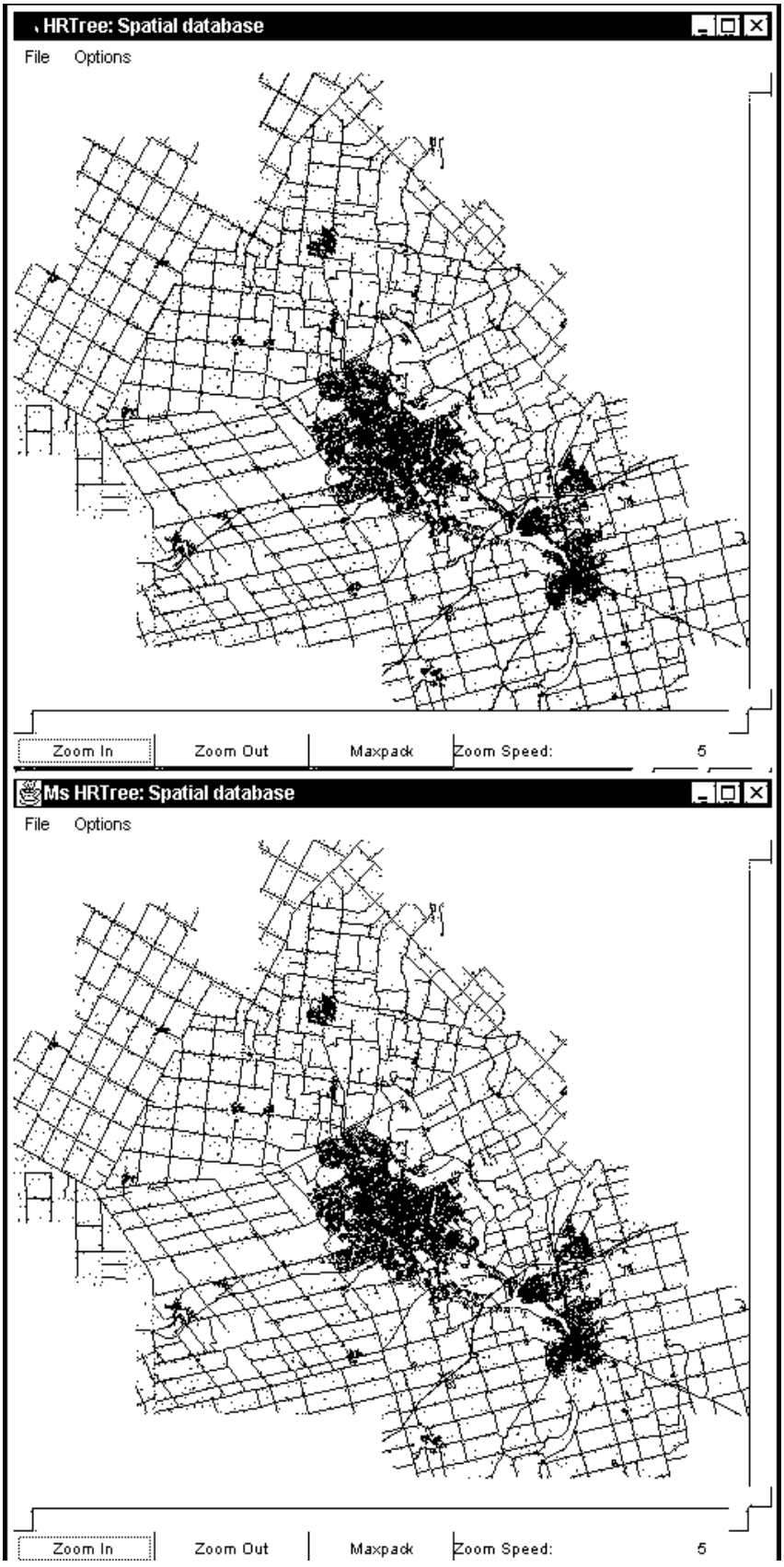


Figure 22: Hilbert & Ms Hilbert R-tree Maxpack on Road and Building

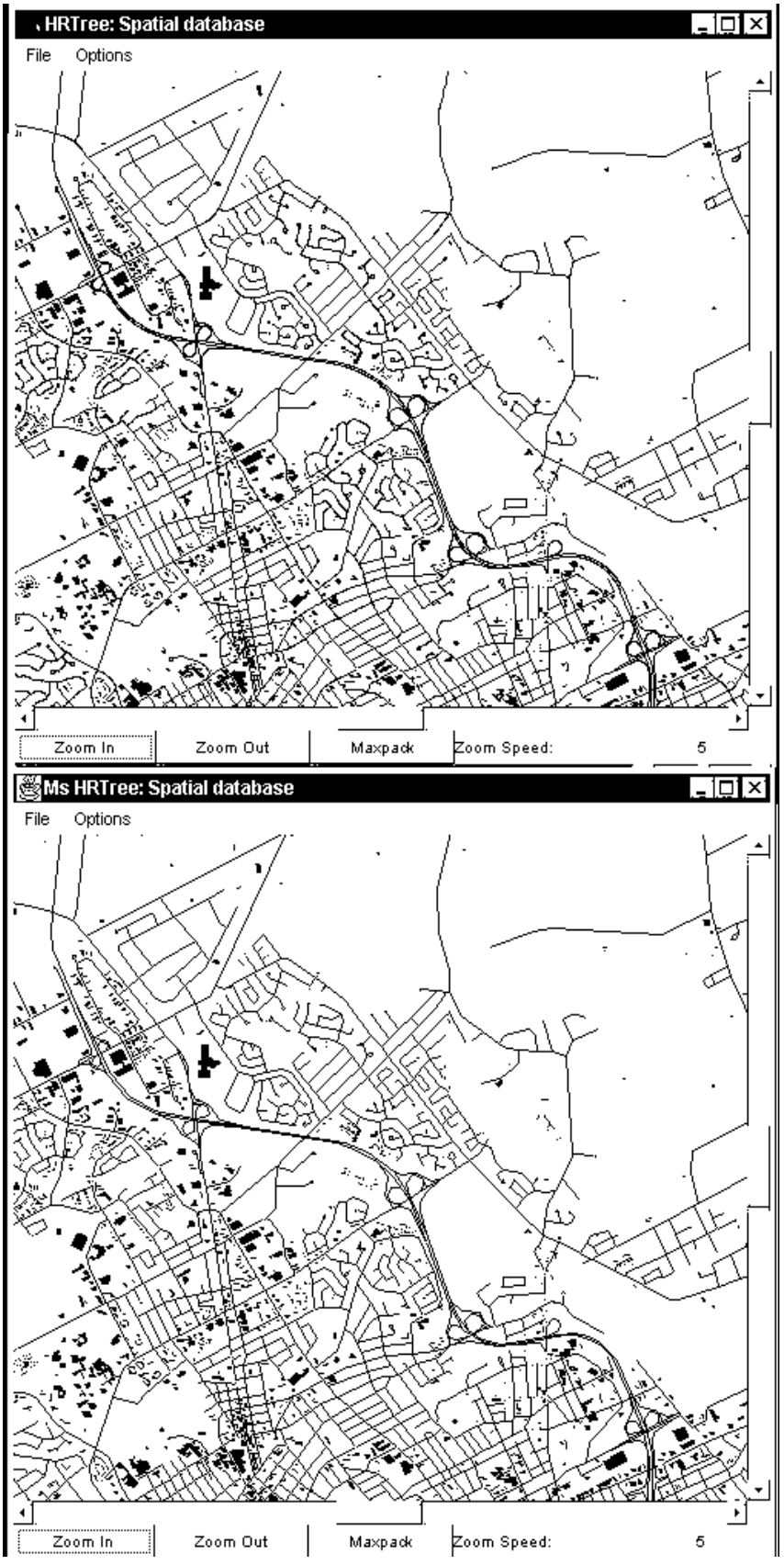


Figure 23: Hilbert & Ms. Hilbert R-tree Zoom in on Road and Building

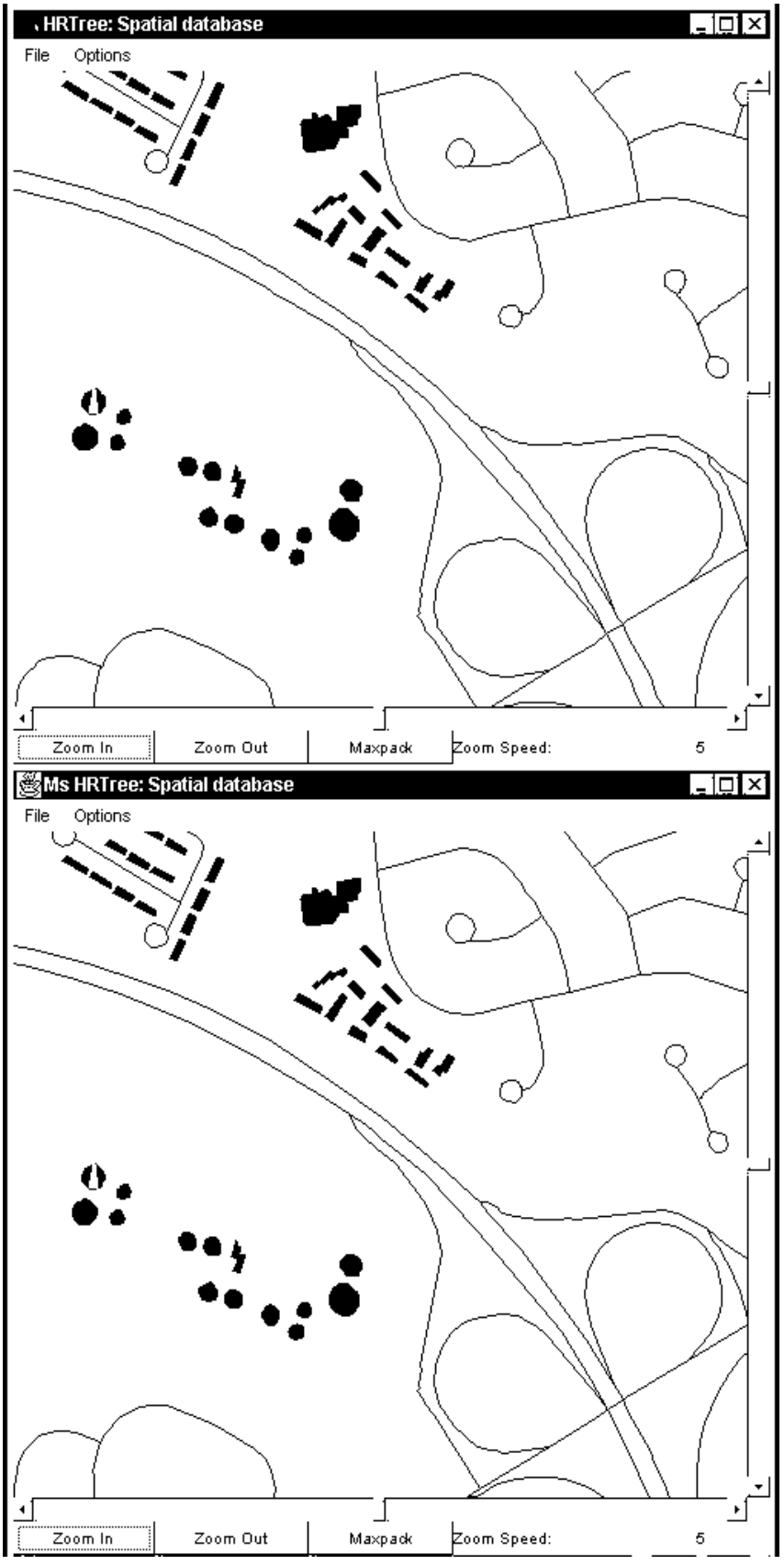


Figure 24: Hilbert & Ms Hilbert R-tree Further Zoom in on Road and Building