# Dynamic and Hierarchical Spatial Access Method using Integer Searching

Kyoosang Cho
Broadband Network Management Development
Sprint Corporation
Overland Park, Kansas 66251
kyoosang.cho@mail.sprint.com

Yijie Han, Yugyung Lee, E.K. Park
Computer Science Telecommunications
University of Missouri at Kansas City
Kansas City, Missouri 641 lo-2499
{han, yugi, ekpark}@cstp.umkc.edu

## ABSTRACT

Dynamic and complex computation in the area of Geographic Information System (GIS) or Mobile Computing System involves huge amount of spatial objects such as points, boxes, polygons, etc and requires a scalable data structure and an efficient management tool for this information. In this paper, for a dynamic management of spatial objects, we construct a hierarchical dynamic data structure, called an IST/OPG hierarchy, which may overcome some limitations of existing Spatial Access Methods (SAMs). The hierarchy is constructed by combining three primary components: (1) Minimum Boundary Rectangle (MBR), which is the most widely used method among SAMs; (2) the population-based domain slicing, which is modified from the Grid File [14]; (3) extended optimal Integer Searching algorithm [4]. For dynamic management of spatial objects in the IST/OPG hierarchy, a number of primary and supplementary operations are introduced. This paper includes a comparative analysis of our approach with previous SAMs, such as R-Tree, R+-Tree and R*-Tree and QSF-Tree. The results of analysis show that our approach is better than other SAMs in construction and query time and space requirements. Specifically, for a given search domain with $n$ objects, our query operations yield $0(\sqrt{\frac{\log n}{\log\log n}})$ compared to $O(\log n)$ of the fast SAM and an IST/OPG hierarchy containing $n$ objects can be constructed in $O(n\sqrt{\frac{\log n}{\log\log n}})$ time and O(n) space.

## Keywords

spatial access method, grid file, dynamic and hierarchical structure, integer searching algorithm.

## 1. INTRODUCTION

Recently, Wireless Mobile Computing and Geographic Information System (GIS) have been widely used. Those applications dynamically deal with geographical positions of mobile users and the surrounding environmental information. In these applications, fast information retrieval from very large spatial databases and efficient geographical computation are largely required. Real world objects are distributed, heterogeneous, and large in size. Much of the information is spatial in nature; objects are dispersed in space and are interacting with each other. The challenging task in the mobile computing application, is to effectively store, retrieve and update interesting and relevant information among vast heterogeneous and distributed spatial datasets.

Mobile computing focuses on mobile device users who are seeking geographical and online information using cellular phones or PDAs. Current user locations are dynamically determined, requested information are retrieved from the Location Dependent Information repository and presented to the user. Location Dependent Queries acquire query results depending on location from where the query originates. As an example, "How far is Kansas City International airport from here"? The answer to this depends on the location from where the query is issued, the answer for the query issued from Lenexa, KS, would be a different one from Lee Summit, MO. Location information of the mobile unit is closely related to local spatial data including the hotels, ATMs, theaters in that area. The Global Positioning System (GPS) would provide information about the position of the mobile unit in terms of longitude/latitude, cellular phone number, sector id or zip codes. Efficient spatial database management is critical for diverse GIS services such as ground traffic, avionics, mobile computing, wireless communication, autonomous navigation, environmental protection, etc. [8].

The major goal of these applications is to provide an efficient repository and methods to facilitates real-time spatial information access, update, and analysis. Spatial objects are represented as points, boxes, polygons, etc. in two dimensional or multidimensional space. The spatial object management deals with the absolute or relative positions of objects and relationships between objects [23]. There are two well known spatial object access methods: point access method (PAM) and spatial access method (SAM). The PAMs primarily perform spatial search on point databases (i.e., only points of objects are available) but SAMs manage more complex objects, such as lines, polygons, or even higher-dimensional polyhedron. We adopt Tree-based SAMs, which use a recursive decomposition of the object space into smaller subspaces. This recursive process traverses a spatial object tree from the root node, through guided internal nodes and to a leaf node, which is either empty or


*CIKM'01,* November 5-10, 2001, Atlanta, Georgia, USA.

pointing to object information in secondary storage. The various tree-based SAMs are different in the decomposition process and the mapping approaches between nodes and regions of the tree. The most well-known tree-based SAMs are the KD-Tree [6] and the R-Tree [12]. A major limitation of these SAM methods is comparison-based access in the relative positions of two objects, which are determined by comparing the coordinates of the objects. This comparison-based searching takes $\Theta(\log n)$ steps for a tree with n objects [26]. To improve the object searching speed, we adopt integer searching algorithm [4] to manage the spatial databases, which yields $O(\sqrt{\frac{\log n}{\log \log n}})$ time.

The goal of this research is to develop a suitable framework for the scalable and dynamic spatial object management. Specifically, in this paper, we evaluate existing techniques and introduce a novel data structure and efficient management for spatial objects. Our approach has three ingredients: (1) the Minimum Boundary Rectangle (MBR), which is widely used method among SAMs; (2) the object population-based space decomposition, which is modified from Grid File [14]; (3) information retrieval and update using optimal Integer Searching algorithm [4]. A multi-layered (hierarchical) structure, called the IST/OPG hierarchy, is modeled by the combination of those ingredients for dynamic and scalable spatial object management. Our IST/OPG hierarchy is flexible, incremental, and dynamic corresponding to the requirements of spatial object management. More importantly, our searching and update operations are better than other SAMs in terms of space and time requirements.

The rest of paper is organized as follows: Section 2 surveys previous spatial object access methods. Section 3 explains our IST/OPG hierarchy. Section 4 analyzes the IST/OPG hierarchy. Section 5 shows the primary and supplementary operations for IST/OPG Hierarchy. Finally, we conclude in Section 6.

# 2. RELATED WORK

## 2.1 Spatial Access Methods (SAMs)

Spatial objects are various and diverse in the form of point, line, rectangle, polygon, etc. Some objects are too complex to model as database elements, and some objects are geographically related to others by overlapping or covering. Thus, the way of representing spatial object affects the performance of spatial access and update as well as the space requirement. The object can be represented by either accurate or approximated representation. Some representation [10, 21, 22] represent spatial object in an accurate way. However, most SAMs adopt some form of approximation because this reduces the storage overhead and simplifies the search and update processes. The typical spatial object approximations include minimum bounding rectangle (MBR) [5,12], minimum bounding circle (MBC) [20, 27], and minimum bounding polygon (MBP) [16].

The minimum bounding rectangle (MBR) represents a spatial object by the smallest rectangle that besieges the object with two points (x and y coordinates of an object): {(x-start, y-start), (x-end, y-end)}. The minimum bounding circle (MBC) represents a spatial object by the smallest circle which covers the object. Considering the line calculation for MBP and floating point overhead for MBC, the MBR-based approximations are characterized by an intermediate degree of complexity and accuracy between MBC and MBP. Our approach is based on MBR which is used more frequently than other approximations.

Several SAM schemes [8] include transformation (object mapping), overlapping regions (object bounding), clipping (object duplication), and multiple layers [17]. First of all, the transformation method maps a PAM object to higher dimensional point or transforms it into a SAM object. Then, the SAM object is represented as a set of one-dimensional intervals by means of space filling curves [16]. Some problems of the PAM approaches can be resolved through the transformation method. The major limitations of this method are computational overhead required for transformation, duplicated multi-dimensional searching, and requirements of potential changes of relationship between objects.

The overlapping method uses different data buckets for mutually overlapped regions [8]. This method efficiently accesses objects using bucket, where objects are assigned. The R-tree [12] and R*-Tree [5] belong in this category. The performance of R*-Tree is superior to R-Tree [8] because the R*-Tree effectively restricts the search space using splitting and reinserting operations. Therefore, the searching area is limited to some of overlapped regions. However, the splitting algorithm requires additional overhead because continuously sorting object along each axis and partitioning the overloaded region into several subregions are required. Thus, there is a trade-off between updating cost and access time.

The clipping methods such as the R+-Tree [24] and Cell-tree [11] resolve overlapped objects by using mutually disjointed bucket regions. For the overlapped objects, the R+-Tree excludes them from the search space but inserts a duplicated version of object information into all the overlapped regions. Although this method reduces the overhead of searching, deficiency still exists in deletion [8].

The multiple layer methods include the multi-layer Grid File [25, 14] and the multi-layer R-File [15]. The multi-layer Grid File excludes the overlapping regions by creating another layer of Grid File. However, the creation gradually reduces the storage utilization. To overcome this problem, the hyperplane may be recursively split by using buckets like R-Tree method, but the overlapped buckets still exist and some buckets may overflow, which results in inefficiency.

The SAM operations include object searching, insertion, and deletion operations. In [12], splitting and merging operations were introduced for overloaded and thin regions. These operations improve efficiency in the searching [12]. Our approach supports these SAM operations and some relational query operations introduced in [23], such as disjoint, meet, overlap, covered-by, etc.

## 2.2 Integer Searching Algorithm

Most comparison-based searching methods, like Binary Search Tree, compare the searching value with values represented in tree nodes until the process finds a matching node or reaches a leaf node of tree. Since the value comparison performs through a traverse path of a tree, it takes O(log n) time.

Boss proposed an efficient method of searching an integer, which uses bit wise operations instead of comparison operations [9]. In his method, a tree is designed to represent integers. Thus, the bits of integers are partitioned over the multiple levels of the tree, and only fixed bits of a search-

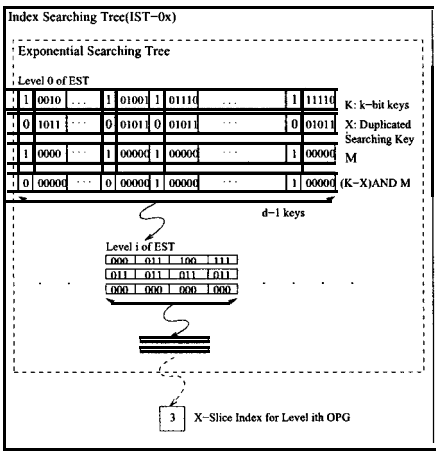


**Figure 1: Index Search Tree**

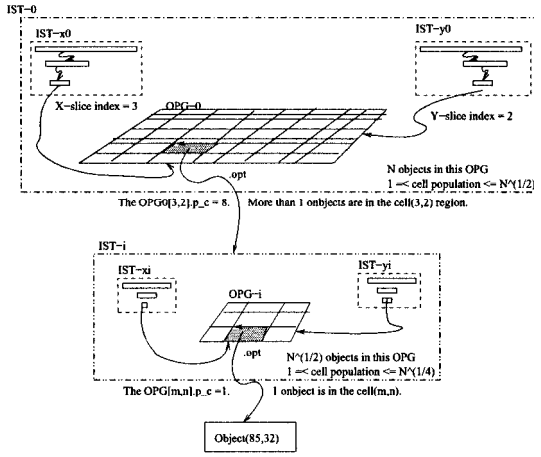


**Figure 2: The IST/OPG Hierarchy**

ing value are compared at each level. As an example, for an 8-bit word, this method first considers 4 bits from left, which are 7th to 4th bits and proceeds to the next level for the remaining bits only if there is a match. Boas's algorithm [9] improved the searching time to $O(\log \log m)$ using a stratified tree structure with m leaf nodes, where $m \geq n$. Amir et *al.* also demonstrated an efficient searching method based on controlling the size of the stratified tree and the distribution of integer extant [1].

As the computer memory is fixed by the word size, the RAM modeling [2] supports sub-logarithmic searching without multiplication. Recently, Andersson and Thorup's [4] designed an optimal Integer Searching algorithm by building an exponential search tree for word sized integers. In the exponential search tree (Figure 1), the root has a degree $\theta(n^{1/k})$ for a selected number $k$, and their children use a local S-structure to store $d$ integers, unlike Boas's algorithm, which checks a half of the word size or unchecked bits . The local S-structure can be built in $O(d^{k-1})$ time and space. Through a recursive decomposition of the searching space, their searching algorithm yields $O(\sqrt{\frac{\log n}{\log \log n}})$ in a fully dynamic linear space. In addition, Their original exponential tree structure [2, 3] requires the amortizing cost in time. Recently, they introduce a new de-amortization approach that is optimal or near optimal [4].

# 3. THE IST/OPG HIERARCHY

Real world objects are always large in size, form in any shape, and evolve dynamically. In order to deal with such objects, we should represent their dynamic features, efficiently retrieve them when needed, and incrementally update them as they are evolving. There are difficulties in storing typical spatial object datasets in main memory and retrieving relevant information from unlimited object spaces. Furthermore, updating information on continuously evolving object is the most challenging task.

In order to deal with large spatial databases, we build a hierarchy, called an IST/OPG hierarchy, supported by two primary structures: Index Search Tree (IST) and Object Point Grid (OPG). The main idea of the IST/OPG hierarchy is to recursively decompose object space both in horizontal and vertical modes. An IST/OPG hierarchy, which is formed through the recursive vertical and horizontal decomposition processes, is composed of a set of ISTs (Index Search Trees) and OPGs (Object Point Grids) (Figure 2).

Figure 2 shows our IST/OPG hierarchy. We now demonstrate how the IST/OPG hierarchy is used to search an object. Consider an object $O_{(85,32)}$. First, at the level 0, we access $IST_x0$ and $IST_y0$ to search the indices of $OPG_0$. From $IST_x0$ we obtain 3 for the given x value, 85, and 2 for the y value, 32, from $IST_Y0$. Then, we access the object space indicated by the indices $(OPG_0[3,2])$ and check the population of the object space $(OPG_0[3,2].p\_c)$. Assume that $OPG_0[3,2].p\_c$ is 7. Since $OPG_0[3,2].opt > 1$, we further traverse the IST/OPG hierarchy for the object $O_{(85,32)}$. Similarly, at the lower level, $i$, we obtain 2 and 1 as the OPG indices from $IST_xi$ and $IST_yi$, respectively. Finally we can obtain the $O_{(85,32)}$ pointer from the object space $(OPG_I[2,1])$.

The IST/OPG hierarchy, resided in the main memory, represents approximated information (pointers) of detailed objects, resided in the secondary storage. The Integer Search Tree (IST), applied Andersson and Thorup's algorithm [4], deals with large amount of object information and plays a significant role in our fast and dynamic searching. Using the IST/OPGs, we dynamically decompose a large object space into dynamic hierarchical sub-structures. Compared to the performance of existing R-Tree, O(logn) time and O(n) space, our approach achieves $0(\sqrt{\frac{\log n}{\log \log n}})$ time and O(n) space.

## 3.1 The Extension of MBR: Multi-dimensional Attributes Representation

There are many ways to represent spatial objects. In mobile computing and GIS domain, the most crucial information on spatial objects must be the location of spatial objects. The location information can be represented using its absolute position in an object space. We use Minimum Bounding Rectangle (MBR), which is one of widely used SAMs [23], to represent an object and for a typical object retrieval and a relational search according to some search criteria. MBR represents the location of object with a boundary information of object such as a starting point( $[S_x, S_y]$) and ending point ($[E_x, E_y]$). In many cases, the starting point is sufficient for object retrievals. However, considering a situation that more than two objects locate in the same starting point, the ending point is additionally required to distinguish them. In this way, we efficiently resolve

the limitation of R-Tree family approach, such as difficulty of "overlapping" and "covered-by" quires in &SF-Tree [28].

In this paper, we limit our object model with two dimensional attributes such as the starting and ending points. However, our model is flexible and dynamic, so that additional attributes of object can be represented by simply expanding dimension (see [7]). For instance, an object shape or type information might be useful information in spatial object retrieval. Consider a typical query of GIS, "find Indian restaurants for me." The intelligent mobile computing system may answer the query with the Indian restaurants located within walkable distance from the location of the user. For this kind service, the mobile database system first retrieves Indian restaurant represented as a type, identifies the user's location, selects appropriate restaurants according to the relative distances between the selected restaurants and the user and finally generates a list of the restaurants. Our approach can support this kind of computing by expanding the dimension of data model for representing the additional attribute values. Individual object attribute can be represented as an independent dimension of the IST/OPG hierarchy. Thus, distributed and parallel processing can be used to store, update, and retrieve additional attributes of spatial objects.

## 3.2 Horizontal Space Decomposition: Object Point Grid (OPG)

The data distribution, the number of object spaces and the number of objects in an object space (object space population), are directly related to the spatial database performance. If the distribution of objects is skewed, the population of some of searching spaces might be huge, requiring many levels of structure and results in performance degradation of query process. The Grid File method [13, 14, 19] decomposes the object space into an orthogonal grid of columns and rows. The resulting spaces may have different shapes and sizes, and each space associates with a bucket, and a bucket may associate with one or more spaces to store objects on a disk page. To guarantee less than two disk accesses for an exact match query, the grid itself need to be kept in main memory, represented as an array for each dimension. Our OPG model is designed followed by the Grid File method [13, 14, 19]. In this paper, we use a term, *object* space (OS), to represent a universe containing spatial objects. An OPG is constructed by a horizontal decomposition of an object space and is structured as a grid of $x$ and $y$ coordinates. A cell decomposed from the object space is called *object subspace* (OSS). Our criteria for the horizontal decomposition are the number of object spaces (i.e., $n^{\frac{1}{2}}$) and the object population of an object space (an object space in a level 1 may have maximum $n^{(\frac{3}{4})^l}$ object points). Each dimension has $n^{\frac{1}{4}}$ partitions and a grid has $n^{\frac{1}{4}}$ by $n^{\frac{1}{4}} = n^{\frac{1}{2}}$ subspaces. Figure 3 shows the horizontal decomposition of an object space with 64 objects, partitioned according to an object population threshold value of $n^{\frac{1}{2}}$. Using the MBR method, 64 objects are represented as rectangles (Figure 3-(a)). In this example, the starting points of the rectangles are used for the horizontal partition (Figure 3-(b)). After decomposing the object space, the level 1 of the OPG has $n^{(\frac{3}{4})^{l-1}\cdot\frac{1}{2}}$ subspaces and each subspace has $n^{(\frac{3}{4})^l}$ object points. We denote the $i^{th}$ level OPG as $OPG_i$, and with x and y indices in the $OPG_i$ as $I_{xi}$ and $I_{yi}$. The subspace is

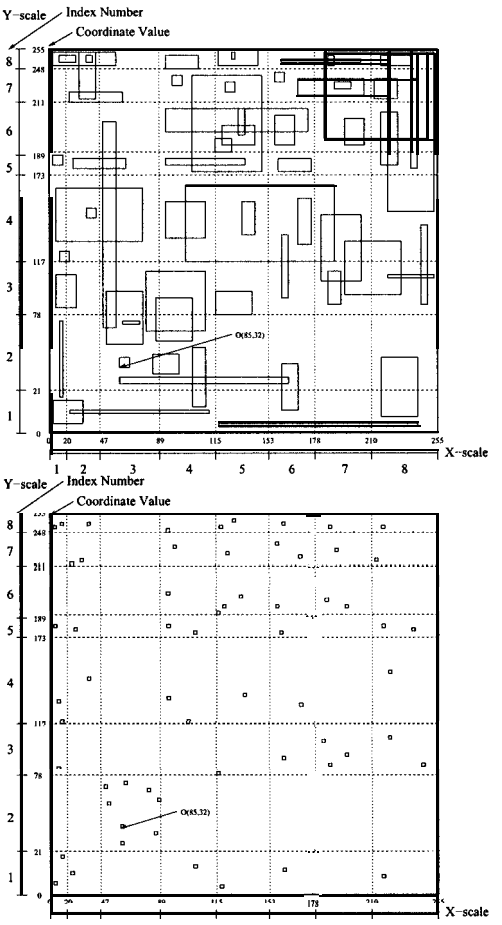


**Figure 3: Object Space Decomposition: (a) MBR of Object Space (b) Starting Points of MBR**

denoted by OPG[$I_{xi}$, $I_{yi}$]. After a complete decomposition of object space in both vertical and horizontal modes, the object subspace in the leaf node of an IST/OPG hierarchy points to a single object.

Now, we describe the properties of the OPG.

- An object space at level $l$ in the IST/OPG hierarchy has $n^{(\frac{3}{4})^{l-1}\cdot\frac{1}{2}}$ subspaces.
- Each *object subspace* of $l$th level *object space* in the IST/OPG hierarchy has at most $n^{(\frac{3}{4})^l}$ object points.
- Each *object subspace* at leaf nodes in the IST/OPG hierarchy has either 0 or 1 object point.

**LEMMA** 3.1. *In the IST/OPG hierarchy,* an *object subspace* contains *at* most $n^{(\frac{3}{4})^l}$ *object points, where 1 is* a *level.*

**Proof:** Use induction. At $l = 0$, there are n objects. Assume at the level $k$ there are $n^{(\frac{3}{4})^k}$ elements in the subspace S. Now S is divided into subspaces ($m = n^{(\frac{3}{4})^k}$) and each subspace contains no more than $m^{\frac{3}{4}}$ objects. Therefore, there are at most $n^{(\frac{3}{4})^{k+1}}$ objects in a subspace. ■

**LEMMA 3.2.** An *object* space *at the level 1* in *the IST/OPG hierarchy has* $n^{(\frac{3}{4})^{l-1}\cdot\frac{1}{2}}$ subspaces.

**Proof:** At the level 0, there is 1 subspace. At the level 1, there are $n^{\frac{1}{4}} * n^{\frac{1}{4}} = n^{\frac{1}{2}}$ subspaces. At the level *1,* there are $n^{(\frac{3}{4})^{l-1}\cdot\frac{1}{4}} \cdot n^{(\frac{3}{4})^{l-1}\cdot\frac{1}{4}} = n^{(\frac{3}{4})^{l-1}\cdot\frac{1}{2}}$ subspaces. ■

LEMMA 3.3. *The total space* requirement for *OPGs is O(n).*

Proof: According to Lemma 3.2, there are $\sum_i n^{(\frac{3}{4})^i \cdot \frac{1}{2}} \leq O(n)$ subspaces. ■

Every *object* space is annotated with data elements, $<\mathcal{O}, \mathcal{N}, \mathcal{L}>$ consisting of Object Variables $(\mathcal{O})$, Neighbor Variables $(\mathcal{N})$ and an Object Space Level $(\mathcal{L})$.

- $\mathcal{O}$ represents object variables: a population counter$(p_c)$ indicating the number of object(s) in an object space, and an object pointer$(opt)$ has a pointer to either an object or the next level IST/OPG space.
- $\mathcal{N}$ represents relation variables: four neighbor variables $\{x\_l, x\_r$, y-u, y-d$\}$ where x-2 is the x index of left side (adjacent) object space, x-r is the x index of right side (adjacent) object space, y-u is the y index of upper side (adjacent) object space and $y\_d$ is the y index of down side (adjacent) object space. These neighbor variables represent logical neighboring relationships between object spaces because an object space index is assigned by the inserting order of objects not by their physical locations. The OPG changes dynamically, so that the indices of the object space may not be consecutive. The neighbor variables are useful to find the neighbors in the OPGs.
- $\mathcal{L}$ represents the level of an object space in the IST/OPG hierarchy. Since the lower OPGs are smaller than the higher OPGs, their object population is fewer than the higher OPG's.

While the Grid File approach [14] uses disk page to store object information, our approach stores object points using the IST/OPG hierarchy in the main memory. The major difference between our OPG and the Grid File method is the forming of structure in the main memory, while our OPG points to the object in secondary storage, the Grid File points to the disk page which includes more than one object information .

## 3.3 Vertical Space Decomposition

The vertical decomposition is to partition the object space in The level 0 into several disjointed object subspaces at level 1, these spaces at the level 1 are further partitioned into subspaces at the level 2, and so on, until each leaf of the hierarchy points to a single object in secondary storage. Our criteria for the vertical decomposition is the object population of an object space. As a hierarchy level is increased, the object population is decreased. This means our object traversing is through a path of the IST/OPG hierarchy incrementally moving down toward to a specific object. The vertical decompositions incorporate with horizontal decompositions to form a multi-layered hierarchy.

LEMMA 3.4. *The height of IST/OPG hierarchy is* O(loglogn).

Proof: According to Lemma 3.1, each object space at the level $h$ has $n^{(\frac{3}{4})^h} \leq 1$which imply $h$ = O(log log n). ■

LEMMA 3.5. *For a given object* space, *the path* from *the root to the leaf of* an *IST/OPG hierarchy* is unique.

## 3.4 Index Search Tree (IST)

To speed up the performance of the SAMs (i.e., logn), we have extended Andersson and Thorup's integer searching algorithm [4] for our multilevel and multidimensional object structure. As mentioned previously, our model can dynamically expand the IST dimension for additional object attributes. In this paper, as our model is limit to represent two dimensional attributes of an object (starting point and ending point), two independent ISTs are used for representing each dimension: one for x coordinate and another for y coordinate of object points.

While the Integer Searching algorithm yields 0( $\sqrt{\frac{\log n}{\log \log n}}$) searching speed in integer, we decompose the attribute of spatial object by each dimension by the order instead of the value. For existing 2 dimensional objects, each ISTs, $IST_x i$ and $IST_y i$ only deal with its own dimension. The $IST_x i$ only works on the $x$ value of objects, while $IST_y i$ represents the y value of objects. The cell (a cross-section of x and y slices) in the OPG can be traversed with the indices of $x$ and y coordinates, $I_x i$ and $I_y i$ (Figure 2).

An IST is composed of two parts: the top part consists of an exponential search tree [4] and the bottom part (leaf nodes) points to the object spaces in the OPG. A cell in the IST is called "Index Search Space (IS) ." Every Index Space is annotated with data elements, $<\mathcal{K}, \mathcal{I}, \mathcal{L}>$ consisting of Object Key $(\mathcal{K})$, Object Index (2) and Object Space Level $(\mathcal{L})$.

- $\mathcal{K}$ represents object keys specifying attribute values of object. An IST at the level i in an IST/OPG hierarchy has $n^{(\frac{3}{4})^l}$ keys.
- $\mathcal{I}$ represents an Object Index specifying an object pointer to an object space in the OPG.
- $\mathcal{L}$ represents the level of Index Search Space $(IS)$in the IST/OPG hierarchy. The lower IST has smaller Index Search Space and fewer object population than the higher IST.

LEMMA *3.6. An IST at the level 1 of an IST/OPG hier*archy has $n^{(\frac{3}{4})^l}$ *object points.*

**Proof:** According to the IST/OPG hierarchy definition, since ISTs represent OPGs indices, the ISTs space requirement for representing OPG index are equivalent to the OPGs'. According to Lemma 3.1, this is true. ■

LEMMA 3.7. Accessing *the lth level IST takes* $O(\sqrt{\frac{\log n^{(\frac{3}{4})^{l-1} \cdot \frac{1}{2}}}{\log \log n^{(\frac{3}{4})^{l-1} \cdot \frac{1}{2}}}})$ *time.*

**Proof:** According to [4] and Lemma 3.6, this is true. ■

LEMMA *3.8. The total space requirement for ISTs is* O(n).

**Proof:** According to the IST/OPG hierarchy definition, since ISTs represent OPGs indices, the ISTs space requirement for representing OPG index are equivalent to the OPGs. According to Lemma 3.3, this is true. ■

# 4. ANALYSIS OF IST/OPG HIERARCHIES

We have developed a spatial database model, the IST/OPG hierarchy, to support an efficient mapping between real world objects and the object information in a large spatial database. The IST/OPG hierarchy supports (1) an optimal query time, i.e., $O(\sqrt{\frac{\log n}{\log \log n}})$, (2) a construction time requirement, i.e., $O(n\sqrt{\frac{\log n}{\log \log n}})$, (3) a space requirement, i.e., O(n), and (4) a dynamic and flexible data structure for an incremental update by adding objects, attributes or attribute values.

We have described the decomposition of IST/OPG hierarchy in vertical and horizontal modes (Figure 2). In order to

efficiently decompose an object space, we maintain an object population threshold per each object space in a level $l$, i.e, $n^{(\frac{3}{4})^l}$. The decomposition is recursively performed according to the threshold, $n^{(\frac{3}{4})^l}$. The recursion factor starts from the object space population n to the object space population 0 or 1. We believe that our population-based recursive decomposition is very effective in handling large spatial object databases. We dynamically and incrementally build an IST/OPG hierarchy.

**LEMMA** 4.1. *An object subspace* in a level $l$ ($l \geq 1$) of an *IST/OPG hierarchy,* has $n^{(\frac{3}{4})^l}$ *objects.*

Proof: According to Lemma 3.1 and 3.6, this is true. ■

**LEMMA** *4.2. The required time to* search an *object* in a level *1* of *the IST/OPG* hierarchy is $0(\sqrt{\frac{\log n^{(\frac{3}{4})^{l-1}\cdot\frac{1}{2}}}{\log\log n^{(\frac{3}{4})^{l-1}\cdot\frac{1}{2}}}})$.

**Proof:** According to Lemma 3.7, this is true. ■

**THEOREM 4.1.** *Searching* an *object in the IST/OPG* hierarchy *for n object points* is $O(\sqrt{\frac{\log n}{\log\log n}})$ *time.*

**Proof:** According to Lemma 4.2, we can access an object in a level *1* in $O(\sqrt{\frac{\log n^{(\frac{3}{4})^{l-1}\cdot\frac{1}{2}}}{\log\log n^{(\frac{3}{4})^{l-1}\cdot\frac{1}{2}}}})$. Then, searching an object in any level of IST/OPG hierarchy ($1 \leq l \leq h$, where $h$ is the height of the IST/OPG hierarchy) can be computed by

$$\sum_{t=1}^{h}\sqrt{\frac{\log n^{(\frac{3}{4})^{t-1}\cdot\frac{1}{2}}}{\log\log n^{(\frac{3}{4})^{t-1}\cdot\frac{1}{2}}}} = \sum_{t=1}^{O(\log\log n)}\sqrt{\frac{\log n^{(\frac{3}{4})^{t-1}\cdot\frac{1}{2}}}{\log\log n^{(\frac{3}{4})^{t-1}\cdot\frac{1}{2}}}}$$

$$\leq \sum\sqrt{\frac{(\frac{3}{4})^{t-1}\cdot\frac{1}{2}\log n}{\log\log n}} = O(\sqrt{\frac{\log n}{\log\log n}}).$$

Therefore, we have $O(\sqrt{\frac{\log n}{\log\log n}})$ searching time same as Andersson's linear space searching time. ■

For a given finite set of object points of cardinality n and partition threshold $\delta = n^{\frac{3}{4}}$, there exists a unique IST/OPG hierarchy. We now describe the time and space requirements for a construction of an IST/OPG hierarchy.

**THEOREM 4.2.** *The* construction time *for* an *IST/OPG hierarchy with n objects is* $O(n\sqrt{\frac{\log n}{\log\log n}})$.

**Proof:** According to Theorem 4.1, the time for inserting an object to the IST/OPG hierarchy is $O(\sqrt{\frac{\log n}{\log\log n}})$. Therefore, the constructing a IST/OPG hierarchy with n objects takes $\sum_1^n O(\sqrt{\frac{\log n}{\log\log n}}) = O(n\sqrt{\frac{\log n}{\log\log n}})$. ■

**THEOREM** 4.3. *The* space requirement *for* an *IST/OPG hierarchy with n objects is O(n).*

Proof: According to Lemma 3.8, the space requirement for IST structure in an IST/OPG hierarchy is O(n). According to Lemma 3.3, the space requirement for OPG structure in an IST/OPG hierarchy is O(n). Therefore, the space requirement for an IST/OPG hierarchy with n objects is O(n). ■

# 5. OPERATIONS FOR IST/OPG HIERARCHIES

Two types of operation, primary operations (Search, Insert, and Delete) and supplementary operations (Split and Merge), are developed for manipulating objects stored in the IST/OPG hierarchy. The primary operators directly manage objects stored in the IST/OPG hierarchy. A set of supplementary operations, Split and Merge, is to maintain balanced structure of the IST/OPG hierarchy. As the IST/OPG hierarchy is dynamically changed by adding new objects or deleting objects, some object spaces in the hierarchy might be overflowed while others are underflowed. Splitting overflowed slides is required to maintain the optimal access time So that each object space maintains the number of objects, less than the threshold $(n^{(\frac{3}{4})^l}$, where 1 is a level). Unlike the split operation, the Merge-Spaces algorithm maintains a equally distributed balanced data structure by merging the adjacent underflowed object spaces into an object space.

## 5.1 Primary Operations

Given $x$ and y coordinates of an object, the Search operation scans down through the IST/OPG hierarchy and finds an object space pointing to the object information $(O_{(x,y)})$ in secondary storage. This operation searches recursively down through a path of the IST/OPG hierarchy until the object pointer is found or it reaches a leaf of the hierarchy.

First, we find the indices of starting and ending points of a given object. At a level, *i,* of the IST/OPG hierarchy, the leaf node of $IST_x$, $IST_{xi}$, indicates the starting point of x-index, $I_{xi}$, and y-index, $I_{yi}$. Second, we directly access the subspace in the $OPG_i$ with $I_{xi}$ and $I_{yi}$. The object space, $OPG_0[I_{xi}, I_{yi}]$, has two types of object variables; (1) a **population** counter and (2) four neighbor variables. Third, we check the population counter, $(OPG_0[I_{xi}, I_{yi}].p\_c)$. Here we have three possible cases: (1) When the subspace population is zero, p-c = 0, which indicates no object in this search result; (2) When the subspace contains one object point, p-c = 1, the search processing is completed by returning the pointer, which points to the real object in secondary storage; (3) Otherwise, we continue to search down to the IST/OPG hierarchy with the pointer to lower level of the IST/OPG hierarchy.

```
Algorithm Object-Search (IST_i, v_x, v_y) {
  // Find the x and y object space indices
  I_xi = Find(IST_xi, v_x); I_yi = Find(IST_yi, v_y);
  //Case of Population = 0: there is no indicated obiect.
  if (OPG[I_xi.p_c, I_yi] == 0 ) return null;
  //Case of Population = 1: return object address.
  else if ( OPG[I_xi.p_c, I_yi] == 1 )
     return OPG[I_xi.opt, I_yi];
  //Case of Population>1: recursive call to lower level IST/OPG.
  else return Object_Search(opt.OPG[I_xi, I_yi], v_x, v_y);
  //OPG[I_xi, I_yi].opt points to lower level of IST/OPG.
}
```

The Object-Insert Operation is used to update the spatial database with new object information. This operation inserts the real object information $(O_{(x,y)})$ into the secondary storage, and then sets the object pointer *(opt)* to $O_{(x,y)}$ into the IST/OPG structure. According to the x and y coordinates of an object, we find their OPG indices $[I_{xi}, I_{yi}]$ from a given $IST_i$s. After that, the insert operation checks the population of $OPG_i[I_{xi}, I_{yi}]$. When its population is 0, Object-Insert operation inserts the object into the object space according the following three steps: First, the population of the object space is initialized. Second, its object space variables are updated. Third, its relational variables are set. When the population of the object space is 1, a new object is inserted to an existing object space. After increasing local population variables, a new sub level IST/OPG $(IST_j$s and

$OPG_j$) is created, and both existing and new objects are inserted into a new level of the IST/OPG hierarchy. In case of p-c $\geq 1$, a recursive call is performed with the lower level IST/OPG, $OPG[I_{xi}, I_{yi}].opt$, and Split_Space operation will be invoked to maintain the threshold ($> \sqrt{n^{(\frac{3}{4})^l}}$).

```
Algorithm Object-Insert (ISM_i, O_(x,y)) {
  ++n.ISM_i;// Increase local population.
  //Find the x and y space indices.
  v_x = MBR(x, O_(x,y)); v_y = MBR(y, O_(x,y));
  I,, = Find(IST_xi, v_x); I,, = Find(IST_yi, v_y);
  //Case of Population = 0
  if ( OPG[I_xi, I_yi] == null ) {
    //Set the object pointer to the address of the object
    OPG[I_xi, I_yi].opt = &O_(x,y);
    OPG[I_xi, I_yi].p_c = 1;
    //Initialize and increase pop-counter of x and y spaces.
    if ( OPG[I_xi, 0] == null ) OPG[I_xi, 0].p_c = 0;
    if ( OPG[0, I_yi] == null ) OPG[0, I_yi].p_c = 0;
    ++OPG[I_xi, 0].p_c; ++OPG[0, I,,] .p_c;
    //Set the neighbor space variables.
    OPG[I_xi, I_yi].x_l=OPG[I_xi, 0].x_l;
    OPG[I_xi, I_yi].x_r=OPG[I_xi, 0].x_r;
    OPG[I_xi, I_yi].y_u=OPG[0, I_yi].y_l;
    OPG[I_xi, I_yi].y_d=OPG[0, I_yi].y_d;
  }
  //Case of Population = 1
  else if (OPG[I_xi, I_yi].p_c == 1) {
    ++OPG[I_xi, I_yi].p_c;
    ++OPG[I_xi, 0].p_c; ++OPG[0, I_yi].p_c;
    ,/Creates New IST and OPG structures
    temp-opt = MAKE(IST, OPG);
    ,/Inserts existing and new object into new IST
    Object-Insert(temp-opt, OPG[I_xi, I_yi].opt);
    Object_Insert(temp_opt, O_(x,y));
    OPG[I_xi, I_yi].opt = temp-opt;
  }
  //Case of Population > 1
  else if{
    Object-Insert (OPG[I_xi, I_yi].opt, O_(x,y));
    ++OPG[I_xi, I_yi].p_c;
    ++OPG[I_xi, 0].p_c; ++OPG[0, I_yi].p_c;
    Split_Space(*IST, v_x, v_y);
  }
}
```

The Object-Delete operation deletes a spatial object information $O_{(x,y)}$ from the IST/OPG hierarchy and secondary storage. The required steps are similar to Object-Search and Object_Insert. After completing the delete operation, Merge-Spaces operation might be invoked to maintain well balanced IST/OPG hierarchy. Refer to [7] for the details.

## 5.2 Supplementary Operations for Balanced IST/OPG Hierarchies

As the consequence of inserting new object information to spatial object databases, some object space may be overflowed (population > threshold, $\sqrt{n^{(\frac{3}{4})^l}}$). Among the X or Y object spaces, an object space with a bigger object population is selected for the Split. As the next step, a new object space is created in $OPG_i$, the smallest unassigned number is chosen (0 < new space index number $\leq \lceil \sqrt{n^{(\frac{3}{4})^l}} \rceil$), and the index number is inserted to $IST_i$. Next, the object points are partitioned into two groups. The object points in a group move to the new object space by deletion and insertion operations., $\frac{OPG[I_{xi}, I_{yi}].p\_c}{2}$, which is larger than squire root of local population ($\sqrt{n^{(\frac{3}{4})^l}}$). The existing and new object spaces may contain $\lceil \frac{\sqrt{n^{(\frac{3}{4})^l}}}{2} \rceil$ and $\lfloor \frac{\sqrt{n^{(\frac{3}{4})^l}}}{2} \rfloor$.

```
Algorithm Split_Space(*IST, v_x, v_y) {
  I_xi = Find(IST_xi, v_x); I , , = Find(IST_yi, v_y);
  //1. X space population is larger than y space population.
  if ( OPG[I_xi, 0].p_c > Threshold(IST_i)) {
    Get the new I_xi by searching IST_xi;
    Find the first object's x coordinate from IST_xi;
    while (number-of-traverse < Threshold(IST_i)/2 )
      Traverse next object in IST_xi by x value:
    while (i <= OPG[I_xi, 0].p_c ) {
      Traverse nest-obiect in IST_xi;
      Get v_x and v_y value of the O_(x,y) to move;
      Object_Delete(IST_i, v_x, v_y); Object_Insert(IST_i, O_(x,y));
    }
  }
  //2. Y space population is larger than x space population.
  else if ( OPG[I_yi, 0].p_c > Threshold(IST_i)) {
    Get the new I_yi by searching IST_yi;
    Find the first object's y coordination from IST_yi;
    while (number-of-traverse < sqrt(n.IST_i + 1/2 )
      traverse to next-object in IST_yi by y value;
    while (0 < OPG[I_yi, 0].p_c ) {
      Traverse next-object in IST_yi;
      Get v_x and v_y value of the O_(x,y) to move;
      Object_Delete(IST_i, v_x, v_y); Object_Insert(IST_i, O_(x,y));
    }
  }
}
```

By merging thin object spaces into one, we could have balanced storage utilization. As the consequence of deleting object information from spatial object databases, some object space may be underflowed (population << threshold, $\sqrt{n^{(\frac{3}{4})^l}}$). Therefore, two smallest spaces among four neighbor spaces ($x\_l$, x-r, y-u, y-d) are selected to merge.

```
Algorithm Merge-Spaces (*IST, v_x, v_y ) {
  //below function will return the order of minimum values.
  k = Min_of_Order( p_c.OPG[x_l.OPG[I_xi, I_yi], 0],
      OPG[x_r.OPG[I_xi, I_yi], 0].p_c,
      OPG[0, y_l.OPG[I_xi, I_yi]].p_c,
      OPG[0, y_d.OPG[I_xi, I_yi]].p_c )
  switch(k) {
    case 0: {//this case we merge x with x_left space.
      while (OPG[I_xi, 0].p_c){
        Get smallest object in OPG[I_xi, 0]
        //fallowing deletion-and-insertion will move the object.
        Object_Delete(IST_i,O_(x,y)); Object_Insert(IST_i,O_(x,y));
      }//end while.
    }//end case 0.
    case 1: {//this case we merge x_right with x space.
      while (OPG[OPG[I_xi, I_yi].x_r, 0].p_c){
        Get smallest object in OPG[OPG[I_xi, I_yi].x_r, 0];
        Object_Delete(IST_i,O_(x,y)); Object_Insert(IST_i,O_(x,y));
      }//end while.
    }//end case 1.
    case 2: { //this case we merge y with y-down space.
      while (OPG[0, I_yi].p_c ){
        Get smallest object in OPG[0, I_yi];
        Object_Delete(IST_i,O_(x,y));Object_Insert(IST_i,O_(x,y));
      }//end while.
    }//end case 2.
    case 3: {//this case we merge y-up with y space.
      while (p_c.OPG[0, OPG[I_xi, I_yi].y_u]) {
        Get smallest object in OPG[0, OPG[I_xi, I_yi].y_u];
        Object-Delete (IST_i, O_(x,y)); Object-Insert (IST_i, O_(x,y));
      }//end while.
    }//end case 3.
  }//end switch
}
```

We introduce the supplementary operations for dynamically changing IST/OPG structure. The Split-Space splits over-populated object spaces to maintain the population threshold. On the other hand, the Merge-Spaces merges two under-populated spaces. As a result, we can reduce the operation time in IST and the size of OPG in the main memory. Using these operations, the data structures can be maintained in a well distributed form. In addition, since we can run these supplementary operations in offline process,

the degradation of the overall IST/OPG performance can be minimized.

# 6. CONCLUSIONS

We introduced the IST/OPG hierarchy for a dynamic management of spatial objects. The hierarchy is constructed by combining Minimum Boundary Rectangle (MBR), which is the most widely used method among SAMs; the population-based domain slicing, which is modified from the Grid File [14]; extended optimal Integer Searching algorithm [4]. Using an efficient and dynamic data structure and management, we have overcome some limitations of existing SAMs. We also developed a number of operations for the IST/OPG hierarchy. From our comparative analysis with other SAMs such as R-Tree, Rf-Tree and R*-Tree and QSF-Tree, we have proved that our approach is better than others in the construction and query time and space requirements. Specifically, for a given search domain with n objects, our query operations yield $O(\sqrt{\frac{\log n}{\log \log n}})$ compared to O(log n) of the fast SAM and an IST/OPG hierarchy containing n objects can be constructed in $O(n\sqrt{\frac{\log n}{\log \log n}})$ time and O(n) space.

# 7. REFERENCES


[1] Amir, A., Efrat, A., Indyk, P., and Samet, H. *Efficient regular data structures and algorithms for location* and *proximity problems.* manuscript (www.graphics.stanford.edu/ alon/regdata.html).

[2] Andersson, A. Sublogarithmic searching without multiplications. In Proc. 36th IEEE Symposium on Foundations of Computer Science, 1995, pp. 655-663.

[3] Andersson, A. Faster deterministic sorting and searching in linear space. In Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science, October 1996, pp. 135-141.

[4] Andersson, A., and Thorup, M. Tight(er) worst-case bounds on dynamic searching and priority queues. In *Proceedings of the 32nd* Annual *ACM Symposium* on *Theory of Computing(STOC* '00), ACM Press, New York, 2000, pp. 335-342.

[5] Beakmann. N., Kriegel, H., 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, May 23-25, 1990, pp. 322-331.

[6] Bently J. L. Miltidimensional binary search trees use for associative searching. Communications *of the ACM 18(9),* 1975, pp. 509-517.

[7] Cho, K. *"A Study* on Spatial *Access Method Based on* Integer Searching *Algorithms, "* M.S. Thesis, Dept. of CST, University of Missouri Kansas City, December 2000.

[8] Gaede, V., and Gunther, 0. Multi-dimensional Access Methods. *ACM* Computing Surveys, 30(2), June 1998, pp. 170-231.

[9] Boas, V. P., Kaas, R., and Zijlstra, E. Design and Implementation of an Efficient Priority Queue. Mathematical *Systems Theory* 10, Springer-Verlag New York Inc. 1977, pp. 99-127.

[10] Gunther, O., and Bilmes, J. Tree-based access methods for spatial databases: implementation and performance evaluation. *IEEE* Transactions on *Knowledge* and Data Engineering, 3(3), Sept. 1991, pp. 342-356.

[11] Gunther, 0. The Design of the Cell Tree: An Object-Oriented Index Structure for Geometric Databases. *Proc. 5th* International Conference on Data *Engineering,* 1989, pp. 508-605.

[12] Guttman, A. R trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD* Conference ACM, New York. 1984. pp. 47-57.

[13] Henrich, A., and Six. H. How to Split Buckets in Spatial Data Structures. In *Geographic* Database Management *Systems* Gambosi., G., Scholl., M., and Six, H.W. eds., Springer-Verlag, Berlin, 1991. pp. 212-244.

[14] Hinrichs, K. Implementation of the grid file: Design Concepts and experience. BIT 25, 1985, pp. 569-592.

[15] Hutflesz, A., Six, H. W., and Widmayer, P. The r-file: an efficient access structure for proximity queries. Proceedings *of 6th IEEE International* Conference on *Data* Engineering, 1990, pp. 372-379.

[16] Kamel, I., and Faloutsos, C. Hilbert R-Tree: An Improved R-Tree Using Fractals. Proc. 20th International Conference on Very Large Data Bases, 1994. pp. 500-509.

[17] Kriegel, H.- P., Horn, H., and Schiwietz, M. The performance of object decomposition techniques for spatial query processing. In *Proc.* 2nd Symposium on *Large Spatial* Databases, Lecture *Notes in Computer* Science, 525, 1991, pp. 257-276.

[18] Kuan, J., and Lewis, P. A study on data point search for HG-trees. *SIGMOD Record,* 28(1), March 1999, pp. 90-96.

[19] Nievergelt, J., Hinterberger, H., and Sevcik, K.C. The grid file: an adaptable, symmetric multikey file structure. *ACM Trans.* on database *systems* 9(1), 1984, pp. 38-71.

[20] Oosterom, P. "Reactive *Data Structures for Geographic Information Systems,"* Ph.D. Thesis, Dept. of CS, Leiden University, December 1990.

[21] Orenstein, J. A., and Merrett, T. A class of data structures for associative searching. In Proceedings *of SIGART-SIGMOD* 3rd Symposium *on Principles of* Database *Systems,* Waterloo, Canada, 1984, pp. 181-190.

[22] Orlandic, R. A High-Precision Spatial Access Method Based on a New Linear Representation of Quadtrees. Proceedings *of 1st* Conference on Information and *Knowledge* Management *CIKM-92,* Baltimore, MD, 1992, pp. 499-50s.

[23] Papadias, D., Theodoridis, Y., Sellis, T., and Egenhofer, M. J. Topological Relations in the World of Minimum Bounding Rectangles: A Study with R-Trees, *Proce. ACM SIGMOD Int. Conf.* On Management *of Data,* 1995, pp. 92-103.

[24] Sellis, T., Roussopoulos, N., and Faloutsos, C. The R+-Tree: A Dynamic Index For Multi-Dimensional Objects. *Proceedings of the 13th VLDB* Conference, Brighton 1987, pp. 507-518.

[25] Six, H., and Widmayer, P. Spatial searching in geometric databases. Proceedings *of the 4th* International Conference on Data Engineering, Los Angeles, 1988, pp. 496-503.

[26] Wang, W, Yang J., and Muntz R. PK-Tree: A Spatial Index Structure for High Dimensional Point Data, In *Proceeding of 5th International* Conference on Foundation *of* Data Organization *(FODO98),* Japan, November, 1998.

[27] White, D. A., and Jain, R. Similarity indexing with the ss-tree, Proceedings *of the 12th ICDE,* Feb. 1996. pp. 516-523.

[28] Yu, B., Orlandic, R. and Evens, M. Simple QSF-Trees: An Efficient and Scalable Spatial Access Method. *CKIM* '99, Kansas City, MO, Nov. 1999, pp. 5-14.