THE JOURNAL OF NAVIGATION (2007), **60**, 83–93. © The Royal Institute of Navigation
doi:10.1017/S0373463307004043 Printed in the United Kingdom

A Spatial Indexing Approach for High Performance Location Based Services

Bo Huang[1] and Qiang Wu[2]

(*The Chinese University of Hong Kong*)[1]
(*MRF GeoSystems Corporation Calgary*)[2]
(E-mail: bohuang@cuhk.edu.hk)

The rapid development of positioning technology, wireless communication and mobile devices has given rise to the exciting Location Based Services (LBS) thus significantly influencing existing navigational procedures. Motivated by the increasing need to search efficiently through a huge number of service locations (e.g. restaurants, hotels, shops, and more), this paper presents an efficient spatial index QR-tree, a hybrid index structure of Quadtree and R-tree, instead of the exhaustive search to improve the performance in response to user queries. QR-tree consists of two levels: the upper level is a Quadtree residing in the main memory which partitions the data space and the lower level is disk-resident R-trees assigned to the subspaces resulting from the partitioning process. Computational experiments show that the hybrid index structure is able to reduce query response time by up to 30% and achieve significant improvement on data update over the conventional indexing methods, thereby providing an effective option for efficient navigation services.

KEY WORDS

1. Location Based Services. 2. GIS. 3. Spatial indexing. 4. Navigation.

1. INTRODUCTION.

The latest advances in Global Positioning System (GPS) technologies and mobile devices (e.g. Pocket PCs) have created abundant opportunities for Location-Based Services (LBS) (Beatty, 2002; Scott-Young and Kealy, 2002; Huang and Li, 2004). LBS are capable of providing specific spatial information to targeted mobile users that allow users to perform geospatial applications at any time and any place. With the onset of broadband mobile data networks, LBS are expected to play an increasingly vital role and revolutionize people's lifestyles in the modern mobile world.

Keeping pace with the developments within LBS, the GIS community has made a significant shift from pure desktop GIS applications to distributed, Internet/mobile applications. With the advent of mobile technologies, the integration of the communication capability of wireless networks, the accuracy of GPS data, the spatial processing power of GIS, and the portability of mobile devices it is possible to build an ideal foundation for LBS (Huang *et al*., 2005). Thus, GIS functionalities are accessible through the Internet or wireless network to a wide audience consisting also of non-expert users. However, a vast majority of previous researches within the realm of LBS were focused on mobile data collection, mapping and simple information retrieval. Little attention has been paid to those applications requiring efficient retrieval of information from a large spatial database, which comprises both static (e.g. restaurants, hotels and parking lots) and moving objects (e.g. vehicles). Novel methods and techniques are inevitable to improve the performance of LBS, thus ensuring quality service and facilitating users' decision making.

The LBS types can generally be classified by their functionality and utilization of location information. In its simplest form it is a positioning service, informing the user about her/his present location. However, knowing the coordinates is not straightforward in most applications; such a service needs to combine with a digital map linked to the user's location and provide a map service. When the service is augmented with capabilities to search information about real-world physical services and support finding the way to the specified destinations, it is called navigation service, which delivers locational information and geo-processing capability to mobile or static users through a wide range of devices via the Internet and wireless network. In other words, navigation service allows users to search for points of interest of any desired category in accordance with the current location of a mobile user, and presents results on a map with complete navigation guidance and contact details (Virrantaus and Markkula, 2001).

When developing navigation services, the constraints of the mobile computing environment, however, have to be carefully considered. Firstly, the service has to provide different types of location-based information and cover different geographic regions. This may involve queries on large volumes of datasets. Secondly, mobile terminals may submit queries to remote GIS servers to obtain navigation services. However, current mobile networks are plagued by hurdles such as high cost, limited bandwidth and low connection stability. In order to reduce the traffic flow of the Internet, the central GIS server takes charge of the main geo-processing task and returns query results upon requests instead of providing the entire dataset. To deliver query results to mobile terminals within a tolerable latency time, it demands an efficient index structure to quickly retrieve desired navigation information and is thus able to accommodate access from a large number of mobile terminals (Huang and Li, 2004). In addition, navigation services are often used in time-critical circumstances (e.g. in-car navigation system) which require (near) real-time query response and concise guidance information to facilitate decision making. Although real-time wireless navigation services have now been on the market for quite some time, most of these services need to improve their efficiency. Consequently, solving these aforementioned problems and establishing low-cost, reliable, and high-quality navigation service is crucial for LBS.

Owing to voluminous spatial data and time-consuming geometric algorithms necessitating underlying systems with extended features involving query languages, data models and indexing methods, extensive research has been conducted on the design of efficient index structures to accelerate spatial access (Rigaux *et al.*, 2001). For query processing and optimizing purposes, many indexing methods have been proposed to effectively retrieve spatial objects. These include Quadtree, R-trees, KD-tree, and B-trees etc. Through these spatial indexes, spatial features can be accessed quickly without having to search the entire database. Among these indexing approaches, it is

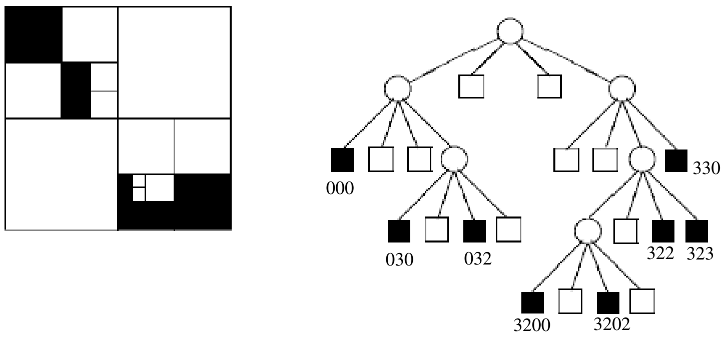

Figure 1. An example of Quadtree.

generally agreed that there is no single approach that can be considered as the best under all circumstances. All of them have inherent strengths and weaknesses. Consequently, we explore an efficient hybrid indexing structure, called improved QR-tree (iQR-tree), based upon the QR-tree (Manolopoulos, 1996) that combines Quadtree and R-tree to improve the performance of database update and queries in navigation guidance.

The remainder of the paper is organized as follows: in the next section, we will describe the background and motivation to adopt QR-tree, then we will illustrate our improvement based on the existing QR-tree. Subsequently, we will experimentally evaluate the proposed techniques and analyze the results. Finally, the paper is concluded with a summary.

2. BACKGROUND AND MOTIVATION.

The spatial index structure aims to reduce the set of objects that need to be processed, thus resulting in lesser time than that consumed by a sequential scan. Ideally, the space indices used should be small and guarantee satisfactory space utilization. Moreover, an optimal index paradigm should support a broad range of operations instead of focusing on the performance over one operation (e.g. searching) at the cost of other operations (e.g. insertion and deletion). Quadtree and R-tree are two of the most popular index structures. Their strengths and weaknesses are as follows.

2.1. *Quadtree*. Quadtree is a non-uniform mesh generation data structure formed by recursively decomposing the space regularly into a maximal set of blocks whose sides are of size power of two (Aref and Samet, 1994). Each node has exactly four children corresponding to the Northwest, Northeast, Southwest, and Southeast quadrants. Non-leaf nodes correspond to non-homogeneous blocks partly inside and partly outside an object, and the leaf nodes correspond to those blocks of the array for which no further subdivision is necessary. Typically, this data structure is employed for accelerating object access in the 2D plane owing to its simplicity and efficiency. Figure 1 gives an example of Quadtree and the corresponding subdivision. From this figure, we can observe that if the objects do not have a regular and symmetric distribution, the tree may exhibit significant *skewness* because Quadtree is not a height-balanced data structure. This will lead to inefficient spatial access in a large dataset.

Another weakness of the Quadtree is that we commonly want to represent a continuous mapping. Normally, we use a Quadtree to represent an approximation of this

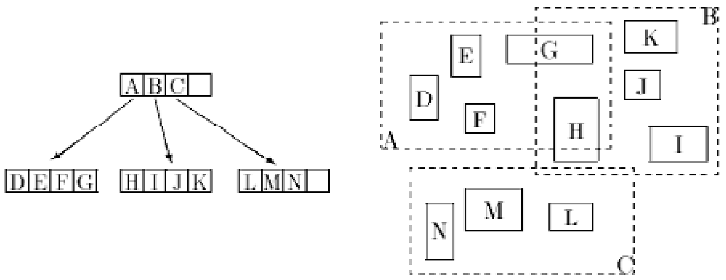

Figure 2. An example R-tree.

continuous mapping, instead of exact mapping, hence facing a difficult trade-off. Using a deeper Quadtree can improve the approximation quality, but, at the same time, makes the structure less efficient, involving both space of storage and time complexity of the operations (Da Fonseca, 2004).

2.2. *R-tree*. R-tree is a spatial access method which splits space with hierarchically nested and possibly overlapping boxes. The tree, which handles objects by means of their conservative approximation, is however, height-balanced. The simplest approximation of an object's shape is the Minimum Bounding Rectangle (MBR). Each node of the tree corresponds to exactly one disk page. In this tree, it is not the objects that are stored, but their MBRs. Internal nodes contain entries of the form (R, child-ptr), where R is the MBR that encloses all the MBRs of its descendants and child-ptr is the pointer to the specific child node. Leaf nodes contain entries of the form (R, object-ptr) where R is the MBR of the object and object-ptr is the pointer to the object's detailed description.

Although R-tree has been proven to be a very efficient structure, it still has some weaknesses that may deteriorate the overall performance. Firstly, since MBRs may overlap, additional disk accesses may be required, because multiple paths may have to be examined from root to leaves when answering a range query. As the population of the stored object increases, overlapping also increases. Secondly, we may be forced to examine several levels of the tree, although there are no (or a few) objects satisfying the query. In addition, the performance of the structure may vary substantially under varying orders of object insertion. Finally, as the number of nodes increases, the tree possibly grows taller, insertion of new objects require more disk accesses and more CPU time, in order to specify the most promising tree path to store an object (Manolopoulos, 1996). Figure 2 is an example of R-tree.

2.3. *Comparison of Quadtree and R-tree*. Quadtree and R-tree do have their respective strengths under different operations and circumstances. Firstly, the time required for constructing the Quadtree index is much less than R-tree, as the clustering time in the R-tree for a large dataset is relatively expensive. The operation for selecting the best MBR and inserting R-tree nodes one by one takes a substantial portion of this creation time. In the case of the Quadtree, the cost is considerably less due to its simple tessellation principle, especially on processing point objects without the need to split the objects. Nevertheless, Quadtree is not suitable for indexing large number of objects due to its skewness. For navigation service, there may be hundreds of thousands of objects distributed throughout the city and stored in a spatial database, it is impossible to index so many objects with Quadtree because the height of

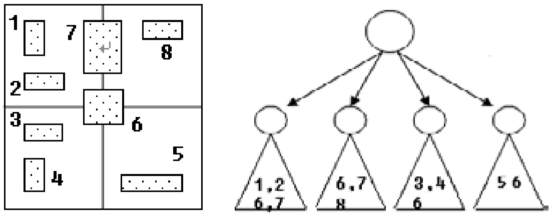

Figure 3. An example of QR-tree.

Quadtree increases significantly with a large dataset and thus results in considerable storage consumption comparing with R-tree and adversely affects the query performance. Although many reports indicate that R-tree is capable for large datasets and consistently outperforms Quadtree in an average case for all kinds of queries, it is still inefficient for navigational services. This is so since the large number of objects spread over the entire city leading to high overlapping of MBRs may take a long time to process the queries, especially for PDAs which have limited computational power.

As both the data structures have their inherent shortcomings, they are not able to independently satisfy the computational requirement on (near) real-time response in LBS. Hence, it is necessary to develop an appropriate approach in order to improve their performance. A hybrid index structure which combines Quadtree and R-tree seems to be a possible solution.

3. THE PROPOSED INDEX STRUCTURE.

There are various hybrid trees proposed by a lot of researchers serving as spatial indexes to effectively manipulate large volume of spatial objects in the GIS arena, such as HR-tree, RB-tree etc. One of them is a combination of Quadtree and R-tree, called QR-tree, which was developed by Manolopoulos (1996). This access method is a hybrid approach in the sense that it combines certain aspects of both bounding region (BR) based and space partitioning (SP) based data structures in a single data structure.

QR-tree comprises two parts: a main memory high-level structure and a disk resident low-level structure. R-trees are employed for the low-level structure. The main-memory structure is a complete Quadtree which consists of a few levels only and each Quadtree node is associated to a whole R-tree. Thus, the Quadtree is used mainly to carry out rough level partitioning of the data space and assign R-trees to portions of it. R-trees index space objects for each subspace that the partition produces. This means each R-tree associates itself with each leaf node (subspace) of the Quadtree respectively. The data structure is shown in Figure 3. For simplicity, the depth of Quadtree is 1 and its leaf nodes are depicted by circles, the triangles represent the R-trees that index the objects enclosed by Quadtree leaf nodes (Manolopoulos, 1996).

Although QR-tree offers a better index structure based on the two original index structures, it is not devoid of weaknesses. From Figure 3 we can observe that objects 6 and 7 intersect with the quadrant subdivision, hence they have to be split and assigned to several R-trees. When performing insertion or deletion, it needs to access multiple R-trees to complete the operation, which will slow down the operation. An improved QR-tree namely iQR-tree is, therefore, proposed in this paper. In this index

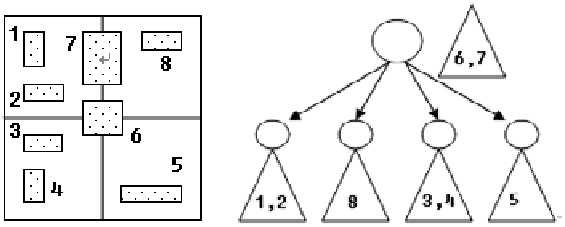

Figure 4. An example of iQR-tree.

structure, the R-tree associates itself with the Quadtree's leaf nodes as well as non-leaf nodes. This implies that when an object intersects with the quadrant subdivision, its parent will be examined to determine whether it can contain the entire object. Otherwise, the previous step is kept and so on, until root is reached. It is defined that a spatial object *P* belongs to a subspace *S* where *P* is entirely located in *S* and *S* is the smallest subspace which entirely envelops *P*. The data structure is shown in Figure 4. In this way, objects that are relatively far from each other are stored into different tree indexes in most cases and thus reducing the overlap between MBRs. Since a specific object is associated with only one index the effect of the insertion order in performance is reduced. As each operation is limited in a least populated set of dataspace, access is faster and the average response time of the queries is reduced.

4. EXPERIMENT AND ANALYSIS

4.1. *Dataset Description.* Experiments were designed to validate our expectation that the improved QR-tree, iQR-tree, is superior to the original R-tree for various datasets. This needs quantitative investigation for various object types and distribution to ascertain the conditions under which this new structure performs better than the original R-tree with less disk page access and CPU time in the cases of insertion, deletion, and retrieval. Therefore the experiment datasets include simulated as well as real datasets.

The simulated dataset consists of four datasets, each one containing 10,000 points or line segments with the uniform and skew distribution, respectively. For the skew distribution, the points or line segments concentrate in certain areas and distribute sparsely in other areas.

The real datasets are the business locations in Calgary with over 10,000 points and polygon objects, including restaurants, hotels, shopping malls, bus stops, car rental agencies, cinemas, parking lots, schools, ATMs, financial institutions, gas stations, golf courses, health care, police services, and toll booths etc. Each tuple represents one business site which contains the following attributes: ID, Name, point (or polygon), address, type, postcode, phone number etc. Figure 5 shows the section of Calgary with business locations.

4.2. *Implementation of the Index Structure.* We implemented the iQR-tree and the original R-tree using VC++ 6.0 in conjunction with PostgressSQL database. In our program, the Quadtree is used to roughly partition the data space and store in main memory. The R-trees maintain their logical tree structure and were implemented as an index table where each node of the R-tree corresponds to a row in the index table. The R-trees for all Quadtree nodes are stored in a same index table. Each

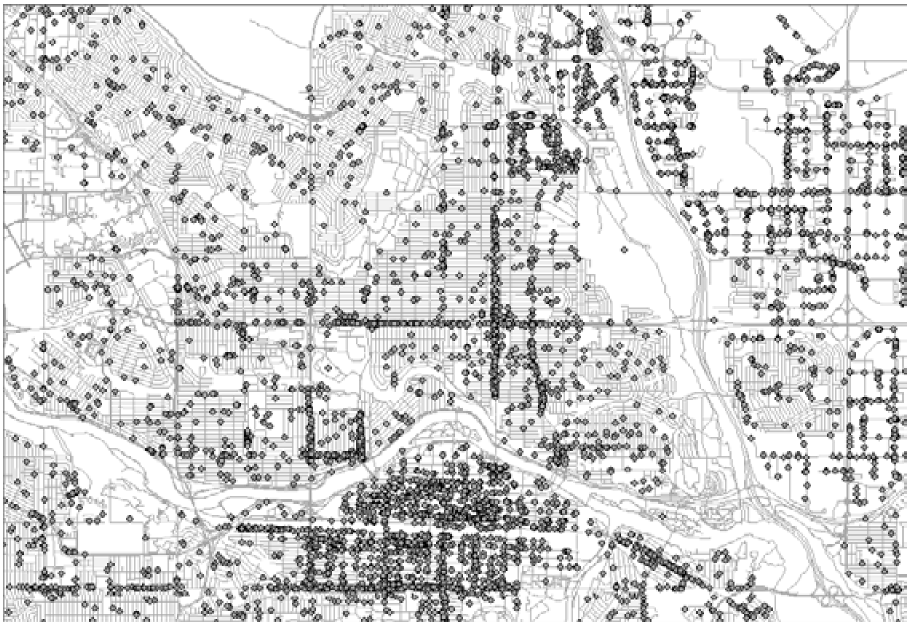

Figure 5. Business locations in Calgary.

row consists of an attribute namely "Partition", which indicates the quadrant subdivision to which the node belongs. In this manner the R-tree for a certain Quadtree subdivision can be retrieved based on the "Partition" value of the nodes. This R-tree is one of the R-trees in the index table.

When inserting an object with its MBR, we first append it to the data table, obtain its ID, and then call a function to locate its subdivision in the Quadtree; the corresponding Quadtree node may be a root node, a non-leaf node at any level or a leaf node. Based on the obtained subdivision, the correct R-tree is retrieved from the index table and the root of the R-tree is captured. Subsequently, navigation is performed down the tree to the most promising leaf, adding the MBR and ID to this node, and finally the R-tree if necessary.

In this process, a basic question arises with respect to the number of Quadtree levels required. The object statistics is assumed to follow a uniform distribution and each quadrant subdivision is assumed to contain equal objects. For an R-tree, at least m^{d+1} objects and utmost M^{d+1} objects can be indexed, wherein m and M are the minimum and maximum entries allowed for each node. Given $M=8$, and $m=3$, first we examine the capacity of an R-tree with 2 as the depth value. In this case, it can index at least 27 objects and at most 512 objects. Also, the total nodes of a Quadtree is $N=(4^{d+1}-1)/3$. Now, we must determine the depth for both Quadtree and R-tree in order to estimate their efficiency on both the tree construction and queries. Considering the real dataset which contains about 10,000 objects of Calgary, the depth of the Quadtree is chosen as 3, the total number of nodes of the Quadtree is chosen to be 85. Each subdivision is assumed to contain 118 objects on average, although the R-tree can index a maximum of 512 objects with the depth of 2. That is, if the height of Quadtree is 3, it may lead to high space redundancy. Thus depth $=2$ is

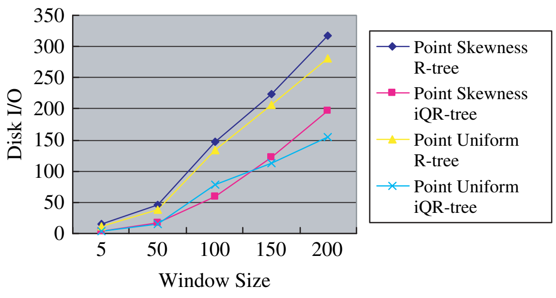

Figure 6. Performance comparison using the simulated point dataset.

appropriate and the total number of nodes is 21. Each subdivision may contain 477 objects on average, and they can be indexed by R-tree with the depth of 2. In this way, the height of both the Quadtree and R-tree is small, and the efficiency of the proposed data structure can be guaranteed.

4.3. *Performance Testing.* Spatial features usually span a wide feature space. However, users may be interested in only viewing or querying a portion of the feature space instead of the whole space. The performance of extracting parts of the space is an important feature for an indexing mechanism. Two fundamental query types are window query and point query. Window query finds all objects that have at least one common point within the query window, while point query locates the Nearest Neighbour (NN) or K Nearest Neighbours (K-NN) to the query point. In the case of navigation, we may expect to find the nearest gas station in the east or find three nearest restaurants when we are driving on the street.

Various experiments were performed to test iQR-tree on three important kinds of spatial operations: query, insertion, and deletion. As accessing secondary storage (e.g. hard disk) costs about twenty times that of main memory and takes the major part in all kinds of operations, the frequency of disk input/output (I/O) is the main concern and it can be taken as a benchmark to test the efficiency of the original R-tree and iQR-tree. Figures 6–9 show the performance comparisons between iQR-tree and R-tree with different datasets.

Figure 6 illustrates the gains of iQR-tree over the R-tree in disk access against different query window size using the point dataset in either uniform distribution or skew distribution. With the increase of the query window size, the performance of iQR-tree decreases.

Figure 7 demonstrates the performance of iQR-tree for line segment dataset. Comparing the simulated uniform and skew line segment datasets, it is observed that iQR-tree can support efficient spatial data access with uniformly distributed line segments and may degenerate quickly in the case of skew distribution. The relative performance of line segments with skew distribution is very close to that of R-tree when the window size is about 200. In the worst case scenario, iQR-tree may totally lose its strength. This implies that if the objects concentrate in a particular area, the partitioning operation cannot reduce the overlap of MBRs.

Figure 8 shows that the performance gain in disk access for window queries ranges from 5% for large window size to 50% for small window size. For small query windows the gain is more significant and thus iQR-tree is suited for such queries. With the increase of window size, the performance of iQR-tree is close to R-tree.

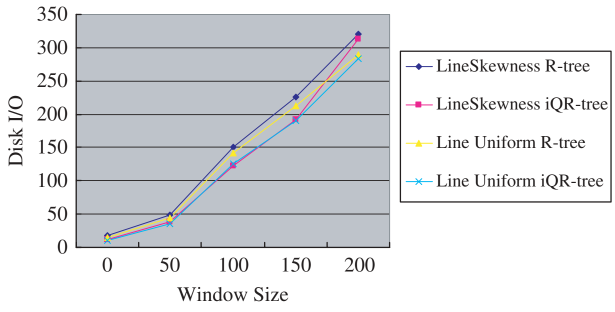

Figure 7. Performance comparison using the simulated line segment dataset.

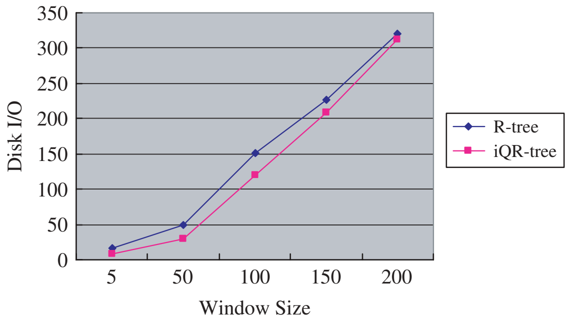

Figure 8. Performance comparison using the Calgary business location dataset.

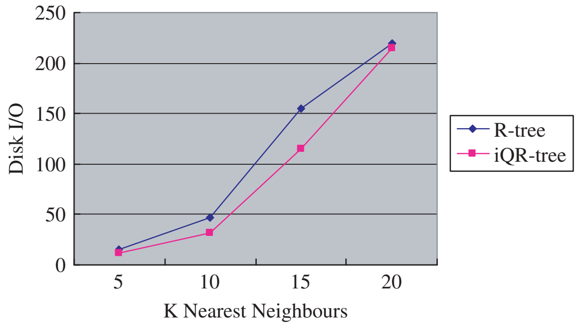

Figure 9. Disk access for K-NN query using the Calgary business location dataset.

Considering the size of the query window as 10, the equivalent range in real world is about 400 m. This kind of range queries may often occur on a diurnal-basis for a PDA or smart phone holder. For example, one may want to find a shop that is on sale within 400 m from his/her current position. In this case, the central GIS server should perform a small size window query and provide the list of all on sale shops located in this area. Due to the low depth for each R-tree of a small region, the overlapping MBRs and multi-path searching can be reduced, thus the central GIS server is able to respond to the queries immediately and accommodate accesses from

Table 1. Comparison of iQR-tree with R-tree on storage space cost.

Dataset	Uniform Point Distribution	Skew Point Distribution	Uniform Line Segment Distribution	Skew Line Segment Distribution	Business Location
iQR-tree	1415	1270	1416	1264	1454
R-tree	2137	1719	1507	1323	1876
iQR-tree/R-tree Ratio	66%	74%	94%	96%	78%

Table 2. Insertion of 100 objects and the disk access per insertion.

Dataset	Uniform Point Distribution	Skew Point Distribution	Uniform Line Segment Distribution	Skew Line Segment Distribution	Calgary Business Location
iQR-tree	4·5	5·1	4·6	5·8	5·3
R-tree	7·8	8·2	5·9	6·5	9·2
Ratio	58%	62%	78%	89%	57%

Table 3. Deletion of 100 objects and the disk access per deletion.

Dataset	Uniform Point Distribution	Skew Point Distribution	Uniform Line Segment Distribution	Skew Line Segment Distribution	Calgary Business Location
iQR-tree	5·3	3·8	3·4	6·2	4·9
R-tree	9·4	8·5	4·5	11·4	9·4
Ratio	56%	45%	75%	54%	52%

many clients. Consequently, the advantage of iQR-tree for small size window queries is very important, especially for fast moving users.

Figure 9 shows the performance of iQR-tree in K-NN queries. As parameter K increases, the query range also increases and more R-trees will be involved to obtain the K nearest neighbours. Fortunately, we are usually interested in the nearest neighbour in our daily life, thus iQR-tree can maintain its strength practically.

Besides the disk access, another measure for comparison is the storage space of the indexing structure in disk, called space cost. The smaller the space cost, the better the performance. We also use a similar relative model to define space cost as $SC(QR)/SC(R) \times 100$, where SC stands for the space cost measured by the number of tuples in each index table. Table 1 shows the comparison of space cost for each dataset between iQR-tree and R-tree. It shows that iQR-tree can save up to 34% of the storage space for the point dataset.

The iQR-tree constantly outperforms the R-tree while considering the performance of insertion and deletion for data update. In general, it may reduce 30% disk access than the original R-tree, especially in the point dataset. From Tables 2 and 3, we can find that when applying iQR-tree to the point database, it exhibits great improvement on reducing disk access and save about half of the processing time as opposed to R-tree.

5. CONCLUSION.

With the advent of wireless network and the increasing popularity of portable digital devices, LBS have presented a lot of opportunities in the mobile communication market, although LBS business is still in its evolutionary stages. Given the present technical limitations such as data access speed, it is imminent that LBS adopt efficient data access methods. Such efficient data access methods can ensure the deliverance of relevant and timely information to targeted consumers at the time and place of their choice.

We have demonstrated, in this paper, an improved spatial index structure, iQR-tree, which combines the popular Quadtree and R-tree, for service location queries and update. In this hybrid index structure, the Quadtree serves as an index of a collection of R-trees; a single R-tree is replaced by a number of R-trees under a divide-and-conquer philosophy. Experimental results illustrate that iQR-tree outperforms R-tree in all operations such as insertion, deletion, and searching. In particular, it can reduce query response time by up to 30%. This structure can be employed for efficiently storing and manipulating massive spatial objects in navigational services.

ACKNOWLEDGEMENT

The funding support from National Science and Engineering Research Council (NSERC) of Canada is gratefully acknowledged.

REFERENCES

Aref, W. and Samet, H. (1994). *A Window Retrieval Algorithm for Spatial Databases Using Quadtrees*, Matsushita Information Technology Laboratory, Panasonic Technologies Inc.

Beatty, C. (2002). Location-Based Services: navigation for the masses, at last! *The Journal of Navigation*, **55**, 241–248.

Da Fonseca, G. D. (2004). QUAD-GRAPHS, http://www.cs.umd.edu/~fonseca/ quadgraphs.pdf. Accessed September 5, 2004.

Huang, B. and Li, H. G. (2004). Developing location-aware navigation guides that use mobile Geographic Information Systems, *Transportation Research Record*, 1879, 108–113.

Huang, B., Xie, C. L., and Li, H. G. (2005). Mobile GIS with enhanced performance for pavement distress data collection and management, Photogrammetry Engineering & Remote Sensing, **71(4)**, 443–452.

Manolopoulos, Y. (1996). QR-tree-a hybrid spatial data structure, *Proceedings of the 1st International Conference on Geographic Information Systems in Urban, Regional and Environmental Planning*, Samos Island, Greece, pp. 3–7.

Rigaux, P., Scholl, M. O. and Voisard, A. (2001). *Spatial Database with Application to GIS*, Morgan Kaufmann, US.

Scott-Young, S. and Kealy, A. (2002). An intelligent navigation solution for land mobile location-based services, *The Journal of Navigation*, **55**, 225–240.

Virrantaus, K. and Markkula, J. (2001). Developing GIS-supported location-based services, *Proceedings of the 2nd International Conference on Web Information Systems Engineering* (*WISE'01*), Vol. 2, Kyoto, Japan, 3-6 December, pp. 66–75.