Database Support for Multiresolution Terrain Visualization

Kai Xu

School of Information Technology and Electrical Engineering
The University of Queensland
Brisbane, Queensland 4072, Australia

kaixu@itee.uq.edu.au

Abstract

Using Multiresolution Terrain Model (MTM) is a common approach to improve the performance of visualizing large terrain data. With the constant increase in the size of terrain data, it is becoming less feasible to have all the data in main memory during visualization. The data exchange between main memory and secondary storage becomes a bottle-neck of terrain visualization, especially for operations like selective refinement. However, this problem has received little attention so far in the context of multiresolution terrain visualization. The need to access the terrain database at multiple levels of detail (LOD) is a fundamental requirement to support secondary-storage multiresolution terrain visualization. However, conventional spatial database access methods are typically based on single resolution only. A new indexing structure, LOD-quadtree, is proposed to support selective refinement on tree-structured MTMs. The new indexing method manages the terrain data in an x-y-error three-dimensional space. In this way, it incorporates the LOD information into the spatial index and at the same time provides efficient locality support. The new indexing method improves visualization performance by accessing the terrain database efficiently at different levels of detail and restricting the data retrieval within specified area. As the LOD-quadtree has no specific requirements for MTM hierarchy structure, it can be used to support different types of tree-structured MTM without modifying them.[1]

Keywords: Multiresolution, terrain visualization, secondary storage, spatial index.

1 Introduction

Terrain data, with its characteristic large size, is always a challenge for visualization. A dataset with millions of polygons is not unusual. Even state-of-the-art workstations cannot handle it comfortably if a large terrain is visualized at full resolution. This may not be necessary due to limited resolution of the display device. A *Multiresolution Terrain Model* (MTM) is proposed to capture a wide range of terrain approximations (meshes) from an original terrain data set (Garland 1999). Each approximation represents the data at a different resolution and can be used to reconstruct the terrain for different viewing requirements.

Generally, there are three types of MTM (De Floriani, Puppo and Magillo 1999): pyramidal (layered), incremental (evolutionary, historical) and tree-structured models. Pyramidal models consist of a small number of pre-computed approximation meshes with different LOD. Incremental models code each step of MTM construction, either the insertion of a refinement method or the inverted removal step of a decimation method. Approximation mesh is reconstructed during the visualization. A much larger number of possible approximations can be obtained from incremental models. An incremental model can be turned into a tree-structured model by the identification of hierarchical independencies in the single incremental steps. While the first two types of MTMs allow only reconstruction of approximation meshes cover the entire domain, tree-structured models also enable approximation meshes covering part of the domain. Tree-structured models are best suited for terrain visualization applications since often only a small subset of the whole dataset is required by users.

Once an MTM is constructed, the operation to extract an approximation mesh for user-specified view conditions is known as *selective refinement*. The user specifies the area he/she is interested in (Region Of Interest, or ROI) and the resolution (Level Of Detail or LOD) required. The selective refinement algorithm reconstructs a terrain approximation (mesh) from the MTM according to these parameters.

Many selective refinement algorithms have been proposed for tree-structured MTMs when datasets are small enough to fit into the main memory (De Floriani, Magillo and Puppo 1998). A typical selective refinement algorithm starts with a coarse approximation and then refines it gradually. In each step, a small part of the mesh is refined. If the part is within the ROI and its LOD is not sufficient, it is replaced by more detailed data. If the part is not within ROI or it is within ROI but its LOD is sufficient, it is not further refined. This process repeats until LOD of all parts of mesh within the ROI is equal to or greater than the level specified by the user.

As the size of terrain data increases, it is increasingly difficult to hold all the data in main memory during visualization. The data exchange between main memory and secondary storage becomes a bottle-neck of selective refinement. However, this problem has received little attention so far in the context of multiresolution terrain visualization. Selective refinement algorithms for main memory will cause a large number of disk I/O operations if applied directly to secondary storage data. When the main-memory selective refinement algorithm is performed on secondary-storage data, one disk I/O operation is needed every time a small part of the mesh is refined. The ROI is made up of many small parts. In most cases, each

point within ROI is checked individually, i.e. each point is a "part". Besides, each part may require a number of updates, thus the total number of the disk I/O operations performed during selective refinement is very large (a more detailed example is provided in Section 3). So the need to access a terrain database at multiple levels of detail (LOD) becomes a fundamental requirement to support efficient selective refinement in secondary-storage multiresolution terrain visualization (*LOD support*). While there exist many spatial data access methods (Gaede and Gunther 1998), most conventional spatial database access methods are typically based on single resolution only. In addition, the need to reduce the data exchange requires locality support of the spatial index. Thus the data exchange can be further reduced by restricting data retrieval within the ROI (*ROI support*).

In this paper, we propose a new spatial indexing method, the *LOD-quadtree*, to support selective refinement on tree-structured MTMs. The new indexing method manages the terrain data in an x-y-error three-dimensional space. In this way, it incorporates the LOD information into the spatial index and at the same time provides efficient locality support. The new indexing method improves visualization performance by accessing the terrain database efficiently at different levels of detail and restricting the data retrieval within specified area. In other words, both ROI and LOD are supported in this new spatial data access method. The LOD-quadtree can be used for various types of existing tree-structured MTM without modification. The remainder of this paper is organized as follows. In Section 2, related work is reviewed. Selective refinement using tree-structured MTM is briefly reviewed in Section 3. A new three-dimensional space, x-y-error space, is introduced in Section 4 to manage the terrain data. The new spatial index, LOD-quadtree, is described in Section 5. In Section 6, visualization performance of the new indexing method is compared with other spatial indexing methods currently available. Section 7 concludes the paper.

2 Related work

According to the data structures used, there are two types of tree-structured MTMs: MTM based on Regular Square Grid (RSG) and MTM based on Triangulated Irregular Network (TIN) (Xu and Zhou 2002). RSG-based MTMs have been adopted for terrain visualization for quite some time. However, there is relatively less work for TIN-based MTM (Xu and Zhou 2002). The methods discussed in this paper focus on TIN-based tree-structured MTM.

There are mainly two types of spatial indexing methods available now to support selective refinement for TIN-based tree-structured MTM visualization. In the first type, the spatial index is created during the process of MTM construction (Magillo and Bertocci 2000) (Hoppe 1998). As the construction (simplification) algorithms of MTM are quite different from each other, the index structures they create are also quite different. In Magillo and Bertocci's work (2000), the simplification algorithm divides the whole dataset into smaller parts so that each part can fit into main memory and be simplified separately. As a result, the MTM is made up of a number of smaller MTMs that can be displayed independently or together. The selective refinement algorithm then retrieves only the parts that intersect with the ROI from secondary storage. This indexing method provides some support for ROI, but it has no support for accessing the data at different LOD. In Hoppe's work (1998), a quadtree index is built into a MTM called progressive meshes (Hoppe 1996). The method starts with partitioning the data in the original model into grids. Then simplification is performed within each grid. Next, every four adjacent grids are merged into one larger grid and then simplified. This process repeats until there is only one grid left or other condition is met. To guarantee that grids of different LOD can match each other when reconstructing the approximation mesh, the simplification performed inside each node is forced to preserve the boundary points. The build-in quadtree index provides good support for ROI. However, the quadtree index used here provides very limited support to accessing data at different LOD.

The second type of method adds a spatial index designed specifically for visualization purposes after the MTM construction. Hardly any work has been done so far in this area. In one of them (Kidner, Ware, Sparkes and Jones 2000), the terrain data is indexed with a quadtree similar to the PMR-quadtree (Nelson and Samet 1986) after the MTM construction. However, this method only works with the pyramidal (layered) terrain model (De Floriani, Puppo and Magillo 1999) and cannot be extended easily to support tree-structured MTMs.

Adding new indexing structure after the MTM is constructed is considered to be a more promising approach. Building the index during the MTM construction will always affect the simplification. As in the work by Hoppe (1998), extra constrains are added to ensure that all the boundary points of every grid have to be preserved，which results in a larger size of MTM. Additionally, these methods are MTM specific. One indexing method can only work with MTM created by certain type of construction method. Adding new indexing structure after the MTM is constructed avoids these problems.

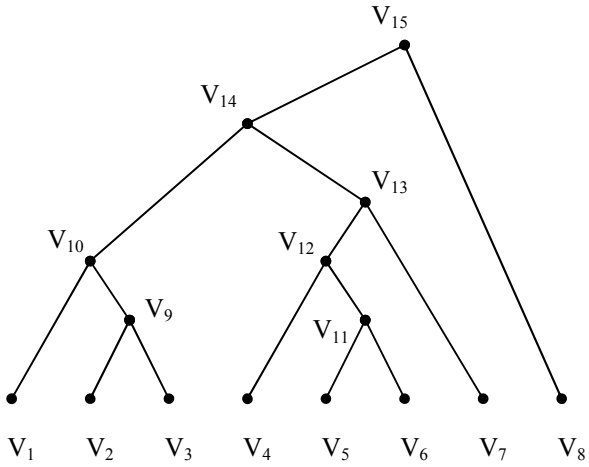
3 Selective refinement using tree-structured MTM

As mentioned before, a typical main-memory selective refinement algorithm starts with a coarse approximation and then refines it gradually. In each step, a small part of the mesh is refined according to two conditions: the *ROI condition* and the *LOD condition*.

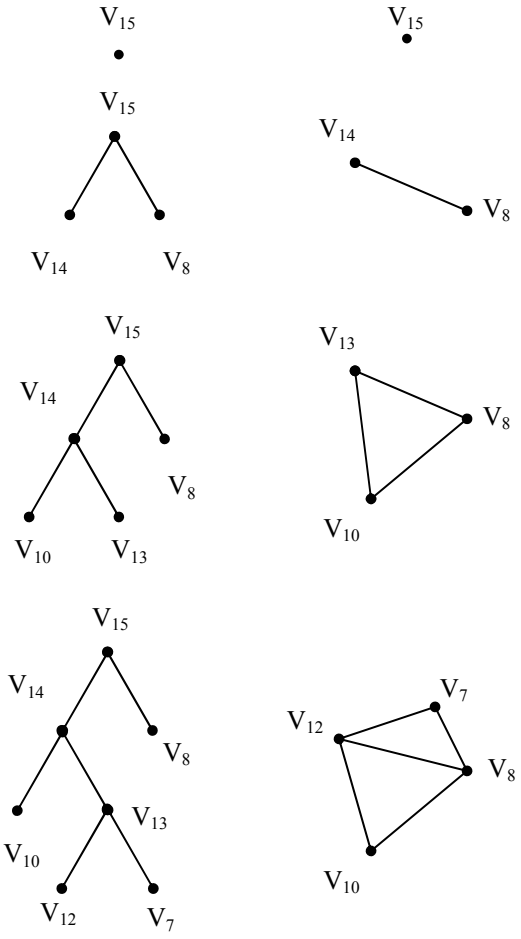
- The ROI condition defines a portion of terrain to be visualized. It is a Boolean function defined on each point of an MTM that returns TRUE if and only if at least one triangle having this point or any of its descendant point as one of the three vertices intersects with the ROI specified by the user.
- The LOD condition defines the level of detail of the mesh to be extracted. It is a Boolean function defined on each point of an MTM that returns TRUE if and only if its level of detail is considered sufficient for user-specified visualization.

Therefore, a selective refinement on secondary-storage terrain data can be regarded as a query with a conjunction of a LOD condition and a ROI condition. The ROI condition is typically a spatial window query (i.e., to find all points within a given polygon). The LOD condition is equal to a query to find all the points whose LOD is within certain interval. The LOD of points in MTM is usually measured by their *approximation error*. The approximation error metric used in our test datasets is the vertical distance between a point in the approximation mesh and its corresponding point in the original model. This is one of the most widely used error metrics for terrain visualization (Lindstrom and Pascucci 2001).

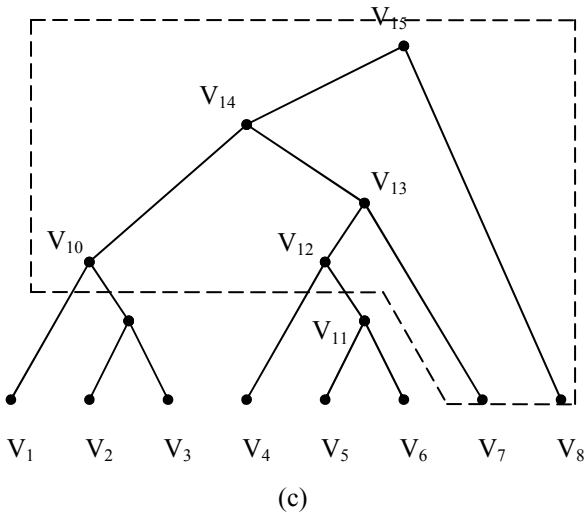
The detail of a typical selective refinement process is given in Figure 1 with a simple example.



(a)



(b)



(c)

Figure 1: Selective refinement

Suppose there is a simple tree-structured MTM whose hierarchy structure is shown in Figure 1(a). V_1 to V_8 are the points from original terrain model (i.e., with no approximation error). All other points are derived from them using some simplification algorithms, such as the quadric error metrics (Garland and Heckbert 1997) used in our test datasets, to minimize step-wise approximation

error. A user issues a request to display part of this terrain represented at certain LOD. Figure 1(b) shows every step of selective refinement. The left column shows part of the MTM used to reconstruct the approximation mesh; the right column shows the resulting approximation mesh. The approximation mesh starts with only one point, V_{15}, which is the root of the MTM hierarchy. Assuming that the LOD of V_{15} is not sufficient (i.e., the approximation error of V_{15} is greater than the user-specified error), V_{15} is refined and replaced by two points: V_{14} and V_8. According to the refinement, the approximation mesh is now a line. Assume that the LOD of V_8 is sufficient, but not V_{14}. V_{14} is further refined and replaced by V_{10} and V_{13}, after which the approximation mesh turns into a triangle. Suppose V_{10} is outside the ROI, so it is not refined any further; V_{13} is within ROI and its LOD is still not sufficient. V_{13} replaced by V_{12} and V_7. The final approximation mesh is made up of two triangles.

The selective refinement could potentially cause a large number of I/O operations as each point refinement needs to retrieve the data of its two child points. In this example, four retrievals (one for each step) are performed to retrieve seven points, which is not efficient. To improve the performance, one alternative is to pre-retrieve the data that is necessary for selective refinement. Figure 1(c) shows the part of MTM that is used for selective refinement. If this part of the MTM can be pre-retrieved, it can save a number of disk I/O operations and improve the selective refinement performance. To achieve this, the indexing method needs to provide both ROI and LOD support. While ROI based selection can be supported by existing spatial data access methods such as quadtree or R-tree indexes(Xu and Zhou 2002), there is no information in such spatial indexing mechanisms to care for LOD at the same time. Therefore there is a need for a new indexing structure that contains both ROI and LOD information. Such an indexing method requires a space that incorporates spatial and level of detail information.

4 X-Y-Error space

Terrain data is two-and-half-dimensional data and a terrain model can be defined as a set of points (x, y, f(x,y)) where x and y are the point coordinates (De Floriani, Marzano and Puppo 1996). Based on this, terrain data is typically managed in x-y two-dimension space. However, only the spatial information (for ROI support) is included in such a space, but not the level of detail information for LOD support. A new three-dimensional space, *X-Y-Error space*, is introduced to incorporate both ROI and LOD information. The X and Y dimensions have the ROI information; the Error dimension, which represents the approximation error in MTM, has the LOD information. An example of X-Y-Error space is given in Figure 2.

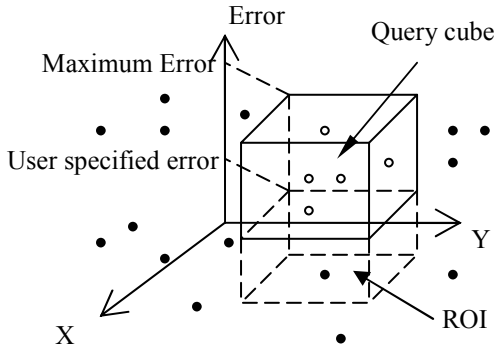


Figure 2: X-Y-Error space and query cube

In the previous example, the points necessary for selective refinement (need to be pre-retrieved) are within the part of MTM indicated by the dashed line in Figure 1(c). Obviously, most of these points are within the ROI. Since selective refinement always starts from the root, which has the maximum approximation error, and stops where the approximation error of points is no greater than the user specified error, we can use the error range between the user-specified error and the maximum error to approximate the error range of data for selective refinement.

Based on this observation, a three-dimensional rectangle, *query cube*, is introduced in X-Y-Error space to pre-retrieve the data for selective refinement. The query cube is defined by the ROI and LOD of selective refinement, as shown in Figure 2. If the points within the query cube are pre-retrieved (indicated as white dots), there is a high probability that these points will be used in selective refinement.

Generally, there are two types of indexing methods to manage point data in multiple-dimension space: one is space-centric, the other is data-centric. One of the most common space-centric indexes is the region quadtree (Samet 1984). However, a three-dimensional region quadtree does not work well in this case (see Section 6 for test results). The reason for this is that the points in the MTM do not distribute evenly in the x-y-error space. Table 1 shows the distribution of the points in the error dimension in our test dataset which has approximately 110,000 points in total. The error metric used here is the vertical-distance error metric, as mentioned in Section 3, given in metres.

Error range	Point number	Percentage
0	55859	50.9365%
0-1	51564	47.0200%
1-2	1546	1.4098%
2-3	358	0.3265%
3-4	170	0.1550%
4-5	62	0.0565%
5-6	47	0.0429%
6-7	28	0.0255%
7-8	10	0.0091%
8-9	2	0.0018%

9-10	5	0.0046%
10-11	0	0.0000%
11-12	2	0.0018%
12-13	2	0.0018%
13-14	1	0.0009%
14-15	4	0.0036%
15-16	2	0.0018%
16-17	0	0.0000%
17-18	0	0.0000%
18-19	0	0.0000%
19-20	0	0.0000%
20-21	0	0.0000%
21-22	0	0.0000%
22-23	0	0.0000%
23-24	0	0.0000%
24-25	0	0.0000%
25-26	2	0.0018%

Table 1: Point distribution in error dimension

As we can see from Table 1, the points in the MTM are highly skewed within the 0 to 1 error range (97.9565%). The reason is: about half of all the points in the MTM are from the original terrain model and the approximation errors of these points are zero; the simplification algorithm that creates the MTM tries to minimize the approximation error introduced at each step; the approximation errors of points in the MTM only increase quickly as the simplification is done to higher than a certain level and there are few points left at that stage. Finally the original terrain model is simplified to a single point, which is the root point of the MTM and has the maximum approximation error. So the data is highly skewed in the error dimension.

The quadtree, as a space-oriented indexing method, doesn't perform well in this highly-skewed case. Data-centric indexing methods, such as the R*-tree (Stonebraker, Sellis and Hanson 1986) and the K-D-B-tree (Robinson 1981), are designed to adapt skewed data distribution. However, these indexing methods need to actually store and maintain an index structure, whereas the quadtree-based index methods can use Z-ordering (Orenstein and Merrett 1984) without explicitly storing the index structure.

5 LOD-quadtree

5.1 Description

Now we propose a new indexing structure, *LOD-quadtree*, that adapts to the highly skewed data distribution in MTM and still can employ the compact structure of Z-ordering. It can provide support for both ROI and LOD because it is based on the X-Y-Error space. Intuitively, the LOD-quadtree is a combination of a two-dimensional region quadtree and a one-dimensional adaptive K-D-tree (Bentley and Friedman 1979) and works similarly to a three-dimensional region quadtree. In the x-y dimensions, as the data is not skewed for terrain data, the LOD-quadtree partitions space as a two-dimensional region quadtree. In the error dimension, as the data is highly skewed, the LOD-quadtree uses a splitting method similar to the one-dimensional adaptive K-D-tree: the data is split in such a way that each half has the same number of points. A three-dimensional region quadtree decomposes a space into eight equal-sized subspaces recursively (Figure 3 (a)). In part of the universe where the data is skewed, the decomposition stops at very deep level; whereas for the part of the universe where the data is sparse, the decomposition stops at much higher level. The resulting index tree is highly unbalanced and this will cause poor performance when retrieving the data for selective refinement. The LOD-quadtree decomposes the universe according to the data distribution in the error dimension (Figure 3(b)) and this gives it a more balanced index tree.

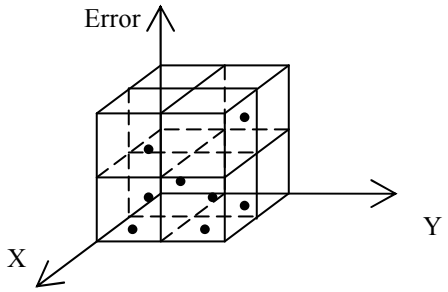


Figure 3(a): Three-dimensional region quadtree

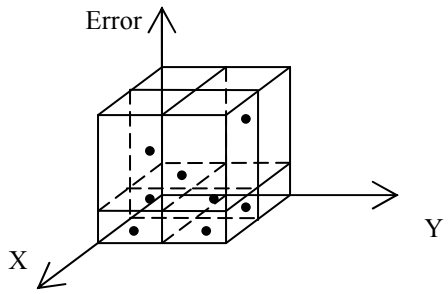


Figure 3(b): LOD-quadtree

The following is the formal description of the LOD-quadtree. In the LOD-quadtree, all data is stored in the leaf nodes. A leaf node is of the form:

(Bid)

where Bid is the pointer referring to the database block where the actual data points belonging to this node are stored.

Each internal node is of the form:

(Err_plane, Child_list)

Where the Err_plane is the position of a plane in the X-Y-Error plane, called *error plane*. Error plane is perpendicular to the error axis and partitions the space into two subspaces in such a way that each subspace contains the same amount of points. The Child_list is a list of pointers to the child nodes of this node. Each internal node

has eight child nodes because each internal space is divided into eight subspaces.

The implementation of the LOD-quadtree uses Z-ordering. Each data point in MTM has an associated Z-value similar to the one used for three-dimensional region quadtree. The positions of error planes are stored separately. The number of error planes is equal to the number of internal nodes of a LOD-quadtree, which could be very large. Searching such a large dataset will obviously slow down the retrieval for selective refinement.

In the effort to reduce the number of error planes, it is found that the positions of error planes at the same level are actually very similar. Based on this observation, a *global error plane* is introduced to reduce the number of error planes. So the subspaces at the same level can share one error plane instead of having a different error plane for each. Figure 4 illustrates this.

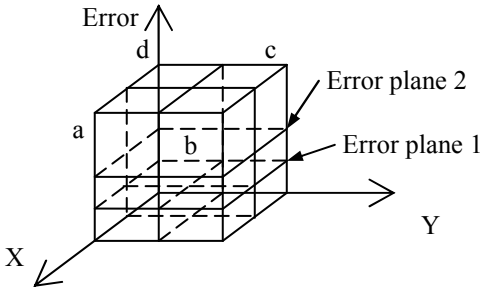


Figure 4: Global error plane

In Figure 4, the data space is first partitioned with Error plane 1, which is a normal error plane. Then, the four subspaces in the upper half of the data space (subspace a, b, c, and d) are further partitioned using the same error plane – Error plane 2 (global error plane). Using a global error plane can reduce the number of error planes greatly: there are 4^n subspaces at level n; these 4^n subspaces can share one global error plane instead of having 4^n error planes.

Similar to region quadtree, the LOD-quadtree is not a balanced tree. It would be highly unbalanced only if the data is skewed in the x-y dimensions, which is not a usual case for MTM data. One feature of LOD-quadtree is that it has no specific requirement for the MTM hierarchy. So it can support different types of existing tree-structured MTMs without any modification.

5.2 Searching

The searching algorithm for the LOD-quadtree is the same as the one for the region quadtree except for the way it partitions the space. In the region quadtree, each space is partitioned equally. In the LOD-quadtree, each space is partitioned equally in the x-y dimensions. However, the space is partitioned according to the position of the error plane stored at each node.

For selective refinement, the pre-retrieval algorithm uses LOD-quadtree to find an approximation space of the query cube of selective refinement and retrieves the points within this space.

5.3 Construction

The construction algorithm for the LOD-quadtree is again the same as the one for the region quadtree except for the way it partitions the space. In the region quadtree, each space is partitioned equally into subspaces. In the LOD-quadtree, each space is partitioned equally in x-y dimensions. In the error dimension, the space is partitioned according to the distribution of data points such that either side of the error plane contains the same number of points. The construction algorithm partitions the data space recursively until the number of points in a subspace is no more than the number of points that can be stored in one database block.

6 Experimental results

For performance testing, we benchmarked the retrieval time needed to display a part of the terrain. Four different indexing methods are compared in the tests:

- Three-dimensional region quadtree
- Three-dimensional R*-tree
- Three-dimensional K-D-B tree
- The LOD-quadtree

Z-ordering is used for both the region quadtree and the LOD-quadtree. The implementation of the Z-ordering for the region quadtree is straightforward: each point in MTM has a three-dimensional Z-value associated with it, without actually maintaining an index structure. The implementation of Z-ordering for the LOD-quadtree is the same as described in Section 4. Eight Z-values, the total space represented by which are minimal but still enclose the query cube, are used to approximate the query cube in both implementations. The implementation of K-D-B tree and R*-tree are both standard. As there is little update of the index after it is created, there is no storage space reserved at each node for data insertion. This results in a more compact structure and hence better performance.

The hardware used comprises of a Pentium III 700 with 256MB memory. The software packages used are Oracle Enterprise Edition Release 9.0.1, Java SDK 1.3, and Java3D SDK (openGL) 1.2. The terrain data used (shown in Figure 5) is a real dataset from Mincom Corp. The dataset covers an area of 6.4 km × 3.2 km and has 109,664 points in the MTM (constructed using quadric error metrics (Garland and Heckbert 1997)). Although this is not a very large dataset, test results show significant improvement. Greater improvement can be expected as the volume of the dataset increases.

Figure 5: Test data (from Mincom Corp.)

Figure 6 shows the "Retrieval time" of different indexing methods when displaying the same terrain part at various LOD. For this set of tests, the size of data (ROI) used is 400m×400m.

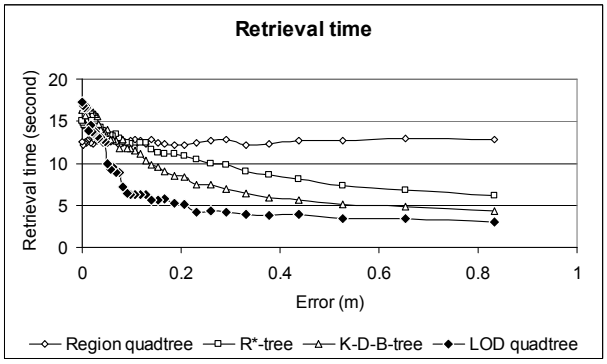


Figure 6: Retrieval time – constant ROI

The "Retrieval time" here includes the time of both pre-retrieval and retrievals occurring during selective refinement. The time of pre-retrieval has two parts: the time for searching the index and the time for retrieving data from secondary storage. However, pre-retrieval can not guarantee that all the data necessary for selective refinement is retrieved. A point can be outside the ROI, but belongs to a triangle that intersects with ROI. In this case, this point is necessary for selective refinement but outside the query cube. These points may not be pre-retrieved. If during selective refinement, the algorithm finds some data needed is not available in main memory, i.e. is not retrieved in the pre-retrieval, it then retrieves that data from the terrain database. The time taken by these retrievals is also included in the retrieval time.

The x-axis in Figure 6 is the maximum approximation error allowed in the mesh reconstructed by the selective refinement algorithm. As the error decreases, the LOD of the approximation mesh increases, which means a more detailed mesh with a larger number of points. As shown in Table 1, the data is skewed with error from 0 to 1. There is little data outside this error range (about 2% of total data). So a mesh with maximum approximation error greater than one would have only a few points. It is difficult to compare the performance of various indexing methods with such a small quantity of data. So only a part of the test results, where maximum error is between 0 and 1, is shown here.

As shown in Figure 6, the LOD-quadtree has a clear performance advantage. There is little difference in retrieval time during the selective refinement because every indexing method retrieves all data points within the query cube. It is the time of pre-retrieval that differs. There are two main factors that affect the performance of pre-retrieval. One is reading the index table and searching for the nodes that should be pre-retrieved. The LOD-quadtree only needs to read the index table storing the position of error plane. This index table is usually very small (the LOD-quadtree index table used for these tests only has 127 records). Then, the pre-retrieval algorithm needs to find eight Z-values to approximate the query cube, which is also very fast. However, the index tables of the R*-tree and the K-D-B tree are much larger (both have more than 10,000 records).

The other factor to be considered is the size of data that needs to be pre-retrieved. Figure 7 shows the pre-retrieval size of different indexing methods in the tests shown in Figure 6.

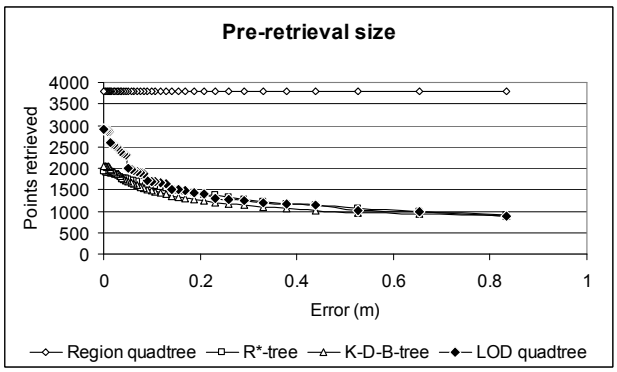


Figure 7: Pre-retrieval size

The y-axis in Figure 7 is the number of data points retrieved in the pre-retrieval. The pre-retrieval size of the LOD-quadtree is very similar to that of the R*-tree and the K-D-B tree, which means that the LOD-quadtree can use only eight Z-values to form a good approximation of the query cube. However, the pre-retrieval size of the region quadtree is much larger, which is the result of inaccurate approximation of the query cube. Eight Z-values are not enough for an accurate approximation using region quadtree. This is the main reason for the poor performance of the region quadtree. As the region quadtree decomposes the space regularly, it is difficult for it to approximate the small but critical changes in the error dimension. As shown in Figure 7, the region quadtree uses the same approximation as the error changes from 1 to 0 because the numbers of points pre-retrieved are always the same. However, the size of actual data necessary for selective refinement changes significantly within this error range (this can be seen from the pre-retrieval size of R*-tree and K-D-B tree). One possible solution to this problem is to increase the number of z-regions used for approximation. As a side effect, it will increase the time for searching and pre-retrieving. Besides, the appropriate number of Z-values could vary as the ROI and/or the LOD changes.

The set of tests whose results are shown in Figure 6 have a constant ROI size and a changing LOD. Figure 8 shows the results of another set of tests with a changing ROI and a constant LOD. The maximum approximation error is set to 0.1 to allow for a reasonable number of points in the approximation mesh. Figure 8 shows that the LOD-quadtree still has clear advantages in this comparison.

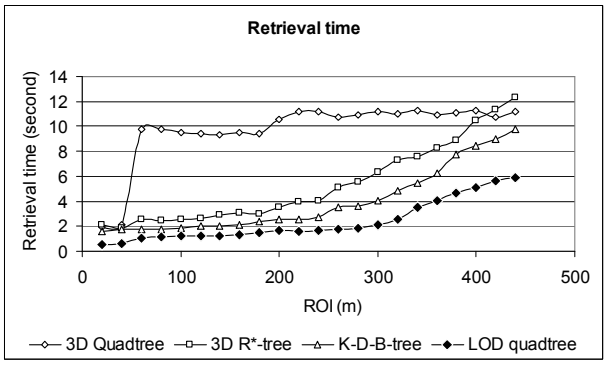


Figure 8: Retrieval time – constant LOD

7 Conclusions

In this paper, X-Y-Error space is introduced to incorporation spatial and level of detail information, both of which are critical to terrain visualization. Based on that, a new indexing method, the LOD-quadtree, is proposed to support selective refinement using tree-structured MTM stored in terrain database. Combining the advantage of region quadtree and adaptive K-D-tree, the LOD-quadtree adapts to the skewed data distribution in MTM and still has a compact structure by using Z-ordering. A Global error plane is introduced to further reduce the size of the index structure. The LOD-quadtree improves the performance considerably by its ability to support both ROI and LOD. This is demonstrated by the test results, which show its clear performance advantage when compared with other available indexing methods. A further benefit is that the LOD-quadtree can support existing MTMs created by various construction methods without modification.

References

BENTLEY, J.L. and FRIEDMAN, J.H. (1979): Data structures for range searching. *ACM Computing Surveys* **11**(4):397-409.

DE FLORIANI, L., MAGILLO, P. and PUPPO, E. (1998): Efficient implementation of multi-triangulations. *Proc. IEEE Visualization '98*, Research Triangle Park, NC, USA, 43-50, Genoa Univ. Italy.

DE FLORIANI, L., MARZANO, P. and PUPPO, E. (1996): Multiresolution models for topographic surface description. *Visual Computer* **12**(7):317-345.

DE FLORIANI, L., PUPPO, E. and MAGILLO, P. (1999): Geometric Structures and Algorithms for Geographic Information System. In *Handbook of Computational Geometry*. 333-388. SACK, J.R. and URRUTIA, J. (eds). Elsevier Science Publishers B.V.

GAEDE, V. and GUNTHER, O. (1998): Multidimensional access methods. *ACM Computing Surveys* **30**(2):170-231.

GARLAND, M. (1999): Multiresolution Modeling: Survey & Future Opportunities. *Proc. Eurographics '99 -- State of the Art Reports*, 111--131, Aire-la-Ville (CH).

GARLAND, M. and HECKBERT, P.S. (1997): Surface simplification using quadric error metrics. *Proc. 24th International Conference on Computer Graphics and Interactive Techniques*, Los Angeles, CA, USA, 209--216, Carnegie Mellon Univ. Pittsburgh PA USA.

HOPPE, H. (1996): Progressive meshes. *Proc. 23rd International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH'96)*, New Orleans, LA, USA, 99--108, Microsoft Corp. Redmond WA USA.

HOPPE, H. (1998): Smooth view-dependent level-of-detail control and its application to terrain rendering. *Proc. IEEE Visualization '98*, Research Triangle Park, NC, USA, 35--42, IEEE Piscataway NJ USA.

KIDNER, D.B., WARE, J.M., SPARKES, A.J. and JONES, C.B. (2000): Multiscale terrain and Topographic Modelling with the Implicit TIN. *Transactions in GIS* **4**(4):379 - 408.

LINDSTROM, P. and PASCUCCI, V. (2001): Visualization of Large Terrains Made Easy. *Proc. IEEE Visualization*, San Diego, California, 363--370.

MAGILLO, P. and BERTOCCI, V. (2000): Managing large terrain data sets with a multiresolution structure. *Proc. 11th International Workshop on Database and Expert Systems Applications.*, xxvii+1164, Dept. of Comput. Sci. Genova Univ. Italy.

NELSON, R.C. and SAMET, H. (1986): A consistent hierarchical representation for vector data. *Computer Graphics* **20**(4):197-206.

ORENSTEIN, J. and MERRETT, T.H. (1984): A class of data structures for associative searching. *Proc. 3rd ACM SIGACT-SIGMOD Symp. on Principles of Database Systems*, 181-190, ACM Press.

ROBINSON, J.T. (1981): The K-D-B tree: A search structure for large multidimensional dynamic indexes. *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 10-18, ACM Press.

SAMET, H. (1984): The quadtree and related hierarchical data structure. *ACM Computing Surveys* **16**(2):187-260.

STONEBRAKER, M., SELLIS, T. and HANSON, E. (1986): An analysis of rule indexing implementations in data base system. *Proc. 1st Int. Conf. on Expert Data Base Systems*, Charleston, South Carolina, 465-476, Benjamin Cummings 1987.

XU, K. and ZHOU, X. (2002): Secondary Storage Terrain Visualization in a client-server environment: A Survey. *Proc. Networks, Parallel and Distributed Processing, and Applications (NPDPA 2002)*, Tsukuba Japan, 206-210.