Direct Mesh: a Multiresolution Approach to Terrain Visualization

Kai Xu Xiaofang Zhou
School of Information Technology and Electrical Engineering
University of Queensland
{*kaixu, zxf*}*@itee.uq.edu.au*

Xuemin Lin
School of Computer Science and Engineering
University of New South Wales
lxue@cse.unsw.edu.au

Abstract

Terrain can be approximated by a triangular mesh consisting millions of 3D points. Multiresolution triangular mesh (MTM) structures are designed to support applications that use terrain data at variable levels of detail (LOD). Typically, an MTM adopts a tree structure where a parent node represents a lower-resolution approximation of its descendants. Given a region of interest (ROI) and a LOD, the process of retrieving the required terrain data from the database is to traverse the MTM tree from the root to reach all the nodes satisfying the ROI and LOD conditions. This process, while being commonly used for multiresolution terrain visualization, is inefficient as either a large number of sequential I/O operations or fetching a large amount of extraneous data is incurred. Various spatial indexes have been proposed in the past to address this problem, however level-by-level tree traversal remains a common practice in order to obtain topological information among the retrieved terrain data. In this paper, a new MTM data structure called direct mesh is proposed. We demonstrate that with direct mesh the amount of data retrieval can be substantially reduced. Comparing with existing MTM indexing methods, a significant performance improvement has been observed for real-life terrain data.

1 Introduction

Interactive terrain visualization is a critical component of a wide range of applications, such as 3D environmental analysis, gaming, virtual walkthrough and many other GIS applications. A surface can be approximated using a regular or irregular mesh of millions of 3D points, which are natural environment data measured or remotely sensed, or synthetic data generated from surface simulation software. The current trend to use commercial DBMS to manage terrain data has been accelerated by several major factors. First, the amount of highly detailed terrain data has been increased dramatically, and it is not uncommon nowadays to have terabytes of terrain, ocean or space surface data. Second, many applications need to share terrain data and use them together with other types of data (such as buildings or moving objects on or near the surface). Third, terrain data is captured over a period of time thus multiple versions may be used together for spatiotemporal analysis. Finally, terrain data is typically used at different resolutions at the same time. For example, the part closer to the viewer should be rendered at a higher resolution than those parts further away in the same image scene.

Data retrieval and rendering performance is a key challenge to allow high quality terrain data to be used by not only the applications designed for high-end computers with specialized graphics hardware, but also by 'light-weight' applications for use on ordinary desktops or wireless devices and Internet applications. Storing terrain data at multiple resolutions is a common approach to improve terrain visualization performance. The fact that terrain data can be used at any LOD and that data of different LODs may be used together to construct a single terrain scene means that it is not feasible to pre-generate terrain data at a fixed number of resolutions. Instead, multiresolution triangular mesh (MTM) structures are commonly used to store terrain data [4, 6]. Progressive mesh (PM) is one of the most widely used MTM data structures, where terrain data is organized into a binary tree [8] (a detailed example is provided in Section 2). Each non-leaf node represents a lower-resolution approximation of its descendants, and all the leaf nodes form the terrain approximation with the highest LOD. Given a ROI and a LOD, a common process to retrieve the required terrain data from the database is to traverse the PM tree from the root to reach all the nodes satisfying the ROI and LOD conditions [2, 3]. This process returns a sub-tree of the PM tree, and all the leaf nodes of the sub-tree form an approximation of the terrain in the ROI with the desired LOD. Note that the leaf nodes in the sub-tree can have the same LOD (for *viewpoint- independent queries* where a uniform LOD is applied to all data points in the ROI), or dif-

ferent LOD (for *viewpoint-dependent queries* where LOD of a data point may vary depending on its distance to the viewer). A point in a PM tree can connect to not only its neighboring points at the same LOD, but the points at any LOD. In order to avoid the overhead for recording complete point connectivity information (i.e., how points are connected to form a triangular mesh), most MTM structures have each node in the tree to record only the changes of connectivity between a parent and its direct children. Therefore, while only the leaf nodes of the sub-tree form the final terrain approximation, the internal nodes of the sub-tree are needed to obtain point connectivity information. The process to select data from an MTM tree according to a given ROI and LOD is called *selective refinement*.

While selective refinement as a way to answer MTM queries is easy to understand and is widely used by terrain visualization applications, it is intrinsically inefficient in terms of the I/O cost when the MTM tree nodes are stored on disk, because the process of fetching data following the tree structure means either a large number of sequential I/O operations, i.e., one for each node split to avoid retrieving any data that is not required; or fetching a large amount of extra data (pre-fetch a lot of data to reduce the number of I/O operations). Various spatial indexes have been proposed in the past to address this problem in order to identify and fetch the required data efficiently from the database [9, 12, 18]. Nonetheless, level-by-level tree traversal remains common because of the hierarchical nature of MTM data. A recently proposed method [20] aim to reduce the number of retrievals by using heuristics. However, in the worst case the number of retrievals is still quite large.

In this paper, we propose a new MTM structure called *direct mesh* (DM) to improve the performance of processing multiresolution terrain queries. Instead of designing yet another spatial index to support DM data, we design DM in such a way that existing spatial indexes can be used (we use R*-tree [1] in this paper). Different from the existing MTM structures which, by and large, treat terrain visualization as a memory-resident procedure, DM is the first MTM structure that supports identifying and fetching query results directly from the database using general purpose spatial indexes. A novel topology encoding scheme is used in DM to allow reconstruction of a terrain approximation from a set of points without the need to use all their ancestors as in the PM tree, with a very small overhead. A cost model for MTM query processing based on DM is presented, together with an optimization algorithm for viewpoint-dependent queries. When compared with existing MTM indexing methods, a significant performance improvement has been observed for real-life terrain data.

The remainder of this paper is organized as follows. In Section 2 we provide an introduction to multiresolution terrain approximation and discuss different types of MTM queries. Related work is presented in Section 3. Direct mesh is introduced in Section 4, and DM-based query processing and optimization are discussed in Section 5. A performance study using two sets of real-life terrain data is reported in Section 6. We conclude this paper in Section 7.

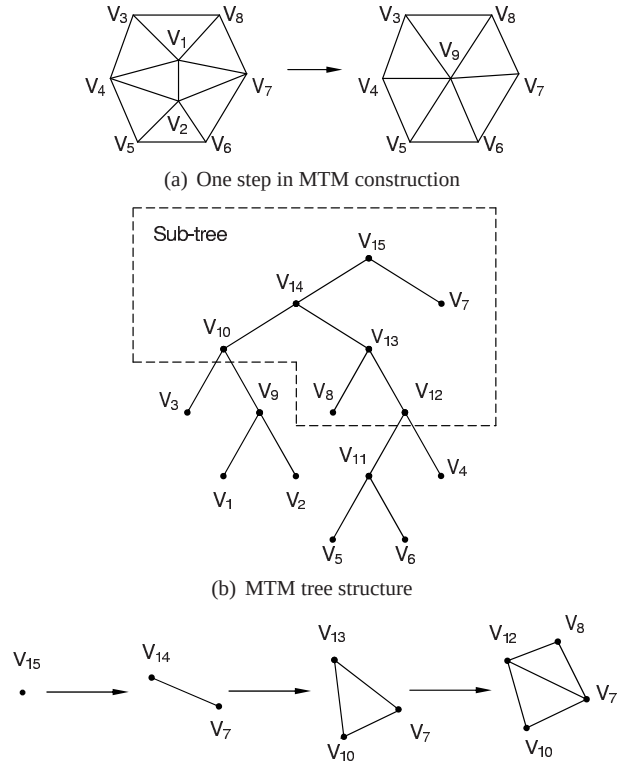
2 Multiresolution Terrain Approximation

In this section, we use PM as an example to explain the construction of an MTM tree from a triangular mesh. Then, we describe the reconstruction of a terrain approximation from MTM for given ROI and LOD.

Constructing an MTM (PM) tree is a bottom-up process. The original terrain data points are stored in the leaf nodes of tree, one point in each node. Two nodes are selected to collapse into their parent node if the resultant terrain (with one less point) cause minimum *approximation error* comparing with the terrain they replace, according to some error measures (e.g., the vertical distance from that point to the terrain surface before collapsing its two children nodes)[7, 13]. The parent node is a newly generated data point associated with an approximation error. An example of such collapse is shown in Figure 1(a), i.e., point v_1 and v_2 collapse into v_9. This process is repeated on the resultant terrain approximation until a tree is formed (so the entire terrain is approximated by one point). Such a tree is an unbalanced binary tree. Following the MTM tree in Figure 1(b), we can track the steps of node collapsing during the construction: point v_1 and v_2 collapse into v_9 (because v_9 is the parent node of v_1 and v_2), then v_9 and v_3 collapse into v_{10}, and so on. Each node in the tree records the following information:

$$(ID, x, y, z, e, parent, child1, child2, wing1, wing2)$$

where ID is the unique ID of the point, (x, y, z) is the coordinates of the point in the 3D space, e is the LOD of the point (as an approximation error value), $parent$, $child1$, $child2$ are the IDs of its parent, left and right child node ($null$ for non-existence), and $wing1$ and $wing2$ are the IDs of the left and the right point connecting to both children ($null$ for non-existence). In the example of Figure 1(a), the wing points of v_9 is v_4 and v_7 (because they connect to both v_1 and v_2, which are the child nodes of v_9). This information is required during the reconstruction of an approximation from MTM, which is essentially the reverse process of MTM construction. Knowing that v_4 and v_7 are the wing points of v_9 makes it possible to reverse the collapse in Figure 1(a). Note that the connectivity information is stored implicitly in the MTM tree structure. For instance, in Figure 1(a) the mesh after collapse is an approximation, but the connectivity information among points in this approximation (i.e. v_8 connects to v_9) can not be derived from the



(a) One step in MTM construction

(b) MTM tree structure

(c) Reconstructed mesh

Figure 1. MTM example.

information stored at these two nodes. The retrieval of connectivity information is discussed in more detail later in this section.

For a given MTM tree M, a *viewpoint-independent query* takes two parameters: a ROI represented as a rectangle r, and a LOD represented as a value e (in the unit used for approximation error measurement when building the MTM tree). The LOD value for a query can either be given by the user, or estimated according to the ROI and the resolution of the target display device [21]. This query, denoted as $Q(M, r, e)$, returns a sub-tree M' of M such that

1. M' contains the root of M;
2. for any leaf node $m \in M'$, point $(m.x, m.y)$ is within r; and
3. for any node $m \in M'$, $m.e \leq e$ if and only if m is a leaf node.

A straightforward method to answer a viewpoint-independent query is to reverse the process of MTM tree construction. M' starts as a sub-tree with only the root and expands following the tree structure (downwards) when a node (or any of its descendants) is within the ROI and whose LOD value is larger than the given LOD. The construction of an approximation of the sub-tree in Figure 1(b) is depicted in Figure 1(c). Taking the split of v_{13} into v_8 and v_{12} as an example (the last step in Figure 1(c)), the connectivity information between the child nodes of v_{13} (v_8 and v_{12}) and other nodes (v_7 and v_{10}) depends on the $wing1$ and $wing2$ of v_{13}. As a boundary point, v_{13} has one wing point v_7 (the other wing point is $null$). Thus, both v_8 and v_{12} connect to v_7.

Note that for any non-leaf node m' of M', $(m'.x, m'.y)$ may not necessarily be inside r (even if some of its descendent nodes are). Thus, all internal nodes of the MTM tree must record its point coordinates, as well as its 'footprint' (as a minimum bounding rectangle, or *MBR*, of its descendant points). In that sense, an MTM tree forms a spatial containment hierarchy.

A *viewpoint-dependent query* is similar to a viewpoint-independent query except that the query does not have a fixed LOD value. This permits different LODs for different sub-regions of r such that the region closer to the viewer can have a higher LOD (i.e., a smaller approximation error value) than those sub-regions that are further away. The required LOD for a point in a viewpoint-dependent query can be estimated, for example, using the formula $f(m.e, d) \leq E$ for node m whose distance to the viewer is d, f is a simple rule-of-thumb function, and E is a constant [9]. Conceptually, a viewpoint-dependent query can be considered as a number of viewpoint-independent queries, each with a sub-region and a uniform LOD.

Although only the leaf nodes of M' will appear in the final terrain visualization, those internal nodes of M' need to be traversed in order to determine the triangulation of the leaf points (i.e., the connectivity information among the leaf points). One may suggest not relying on ancestor nodes to record connectivity information by letting each point record all its neighboring points directly. The purpose of using an MTM data structure, however, is defeated by doing so. The number of neighbors of a point at a single LOD level is typically not very large (less than 10 in general). However, as mentioned points at different LODs may connect to each other (for answering viewpoint-dependent queries, for example). This makes the total number of possible neighbors prohibitively large, as we shall show later. An MTM tree as described above records cross LOD level connectivity information implicitly with each node recording connectivity information to only two nodes ($wing1$ and $wing2$ as given before).

3 Related Work

Multiresolution terrain data form a natural spatial containment hierarchy. This property has been exploited for fast multiresolution terrain data retrieval. The LOD-R-tree is one of the first attempts to use spatial indexes to speed

up multiresolution terrain data retrieval [12]. It extends the traditional R-tree index in the following way. A normal R-tree is created on the points of the original mesh (i.e., with the highest LOD), by grouping points into data objects first. A leaf node of the R-tree created is a subset of the original mesh. For each internal node of the R-tree, an approximation mesh at a lower LOD is created by combining and generalizing the meshes of all its children nodes. Therefore, each R-tree node has not only an MBR but also a LOD value. Selective refinement on the LOD-R-tree is converted to a range query whose query window is the ROI. The traverse of the LOD-R-tree stops when the LOD of node is sufficient for the query. Meshes at different R-tree nodes may need to be mosaiced after the data is retrieved. This approach builds its own multiresolution hierarchy, thus does not support MTM structures such as PM. For a viewpoint-dependent query, the query ROI needs to be decomposed into several sub-queries, each with a sub-region and a different LOD. Another problem with this approach is that the MBR of internal R-tree nodes is determined for optimizing R-tree performance. It does not support flexible granularity control thus entire node needs to be retrieved even if only a small part of the area covered by the node is needed.

Hoppe [9] suggests an approach similar to the LOD-R-tree, but using a 2D quadtree [16] instead. A quadtree index is created on the original terrain data. An approximation (with LOD lower than it child nodes) covering its descendant nodes is stored at each internal nodes of quadtree. The data fetching for $Q(M, r, e)$ is similar to the LOD-R-tree, and it shares the same problems of LOD-R-tree.

Shou *et al* improve the LOD-R-tree by considering data visibility, and propose a new indexing method called the HDoV-tree [17, 18]. Visibility information is stored at every node of the LOD-R-tree, so occluded parts of terrain within the ROI can be excluded and approximation with lower LOD can be used for the areas in distance or with a low degree of visibility. HDoV-tree is an optimization for data reduction by visibility. However, it does not address the problems associated with LOD-R-tree as mentioned above.

Xu [20] proposes to use a 3D quadtree, in which the LOD dimension is added. The LOD-quadtree is an adaptive quadtree that can handle the fact that point data are more uniformly distributed in the (x, y) space but severely skewed in the LOD dimension. $Q(M, r, e)$ is translated into a 3D range query using both r and e and a better performance has been achieved compared to Hoppe's method [9]. The performance of using LOD-quadtree can be degraded by treating the internal nodes as point data (rather than a minimum bounding rectangle for all descendants). This means that more queries needed in order to find those internal nodes which are not inside r but some of its descendants are.

4 Direct Mesh

In PM, the connectivity information of each point is stored at the node for that point and all its ancestor nodes in the PM tree. Thus, the cost for processing $Q(M, r, e)$ consists two components: one to retrieve the point data that are used to form the requested terrain, and the other to retrieve their ancestors to obtain connectivity information. As we have shown in the previous section, the cost of the first component can be reduced by using various spatial indexes such as the LOD-quadtree. However, there is no available technique to reduce the cost of the second component. It is well known that the relational model is not efficient to handle such parent-children relationship. The fact that some ancestor points may be outside region r further complicates this issue, as every parent node must include an MBR of its descendants in order for it to be retrieved with any of its descendants. Clearly, the nodes higher in the PM tree will have very large MBRs and the MBRs among sibling nodes will significantly overlap with each other. Therefore, we can conclude that all existing spatial indexing mechanisms, which, in one way or another, depend on non-overlapping data clustering to function efficiently, are not ideal approaches for managing MTM data.

One obvious solution to this problem is to store the complete connectivity information at each node. The points that one point can connect to are called its *connection points*. As viewpoint-dependent query may return a mesh with points at varying LOD, a point may have connection points with different LODs. If point m' is a connection point of point m, then the parent of m' is also a connection point of m, because the parent node of m' connects to all the connection points of m'. Similarly, the child nodes of m' may also be the connection points of m, because at least one child node of m' connects to m. To be precise, the following rules hold:

1. when point m'' is the first common ancestor of m and m', any ancestor of m' up to m'' (excluding m'') is a connection point of m;
2. at least one of the child nodes of m' (for example m'') is a connection point of m; and this also applies to m'', i.e., one of the child nodes of m'' is a connection point of m. This applies recursively until reaches the leaf level.

As these rules apply to connection points recursively, the total number of the points a point can connect to is potentially very large. A naive way to store complete connectivity information at each node could incur substantial overhead and cause redundancy. This is the reason that most MTM structures adopt a tree-like structure.

Before we introduce a new connectivity encoding scheme, we need to normalize the LOD definition such that

for any two nodes $m, m' \in M$, $m.e \geq m'.e$ if m is the parent of m' (note that many approximation error measures, such as the vertical distance method mentioned before, does not guarantee this property) [4]. For a given MTM tree M where each node has already been assigned with a LOD value, we define its new LOD value using the following formula:

$$m.e = \begin{cases} 0, & \text{if } m \text{ is a leaf node} \\ \max(m.e, m.child1.e, m.child2.e), & \text{otherwise.} \end{cases}$$

Now we assign each node an *LOD interval* $[e_{low}, e_{high})$. If m is not the root, $m.e_{low} = m.e$ and $m.e_{high} = m.parent.e$; and for the root node, its LOD interval is defined as $[m.e, \infty)$. Using LOD intervals, selective refinement to find a mesh of LOD e is equivalent to finding all point nodes whose LOD intervals encloses e.

Finally, we define that point m and m' have a *similar LOD* if their LOD intervals overlap. Two points with a similar LOD implies there exists a LOD value e such that a viewpoint-independent query $Q(M, r, e)$, where r is an area that covers both m and m', will retrieve both points. Therefore, we propose to store at each node m a list L of connection point IDs, such that for any $m' \in L$, m and m' have a similar LOD. The number of connection points with similar LOD is substantially smaller than the number of all possible connection points. Tests show that for each point the average number of connection points with a similar LOD is 12 in both test datasets we used (one with 2 million points, the other with 17 million points). Whereas the average number of total connection points is 180 for the 2-million-point dataset and 840 for the 17-million-point dataset.

A direct mesh (DM) is constructed from a PM by adding a list of IDs for the connection points of similar LOD to each node. An example is shown in Figure 2. Figure 2(a) presents the LOD interval of v_4, v_{10}, v_{11} and v_{12}. Note that point v_4 is a leaf node (Figure 2(b)), so its LOD interval starts from 0. Point v_{12} is the parent of v_4 and v_{11} (Figure 2(b)), so their LOD intervals do not overlap and they do not have a similar LOD with each other. They are not connection points to each other, either, because parent-child points can not exist together in any approximation. Other points are points with a similar LOD to each other since their LODs interval overlap. We assume that they are connection points to each other. The lists stored at these four points are shown in Figure 2(b). For example, the list of v_4 contains v_{10} and v_{11} (and possibly other points that are not in Figure 2(a)). DM inherits PM's property to be able to derive viewpoint-dependent or viewpoint-independent terrain of any size and any LOD. More importantly, it eliminates the need to fetch all internal nodes to the root in order to obtain connectivity information. As explained in the next section, for a viewpoint-independent query, a single SQL query is sufficient to retrieve both terrain data and their connectivity information; and for a viewpoint-dependent query, selective refinement can be performed starting from the lowest LOD required by the query instead of the root node.

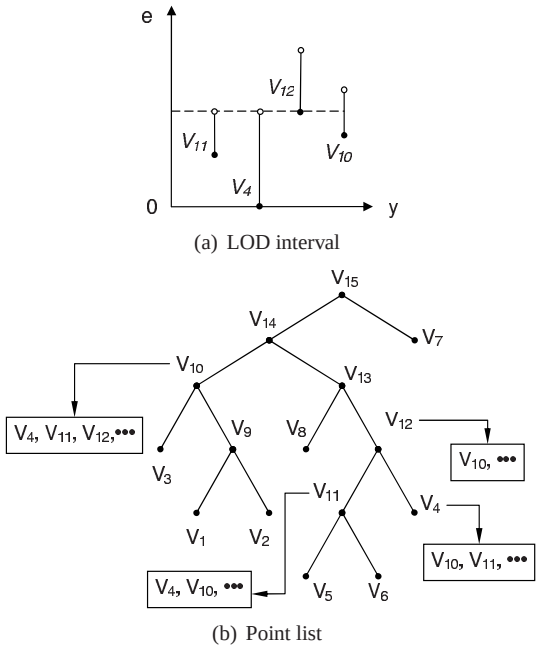


Figure 2. Direct mesh.

To store the DM data in a database we can build a 3D R-tree on the DM points (for leaf and internal nodes) in the (x, y, e) space. A point m becomes a vertical line segment in the 3D space (i.e., $< (m.x, m.y, m.e_{low}), (m.x, m.y, m.e_{high}) >$). Next we demonstrate efficient processing of different types of MTM queries on DM with a 3D R-tree.

5 DM-Based Query Processing

In this section, we discuss the processing of viewpoint-independent and viewpoint-dependent queries using the DM structure and the R-tree index.

5.1 Viewpoint-Independent Query

Viewpoint-independent query processing based on DM is simple. For $Q(M, r, e)$, a *query plane* parallel to the (x, y) plane, shown as the shaded part in Figure 3, is used to retrieve the data using the 3D R-tree index. Compared with processing the same query on PM data using LOD-quadtree [20], DM retrieves much less data because only the data having LOD interval intersects with the query plane needs to be fetched. Under the LOD-quadtree, the query needs to be converted into a 3D range query using a query cube defined by the r, e and the maximum LOD of the dataset

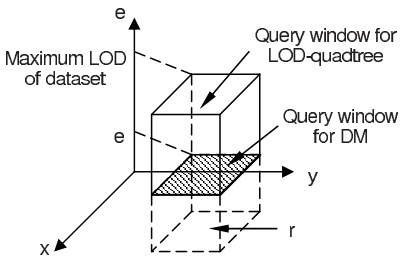


Figure 3. DM for viewpoint-independent queries.

(shown as the solid line cube in Figure 3). Data retrieved by DM all have similar LOD as their LOD interval at least overlap at e, so their connectivity information can be found at these nodes.

5.2 Viewpoint-Dependent Query - Single Base

Let the ROI of a viewpoint-dependent query be r and the LOD range be (e_{min}, e_{max}). The query plane for a viewpoint-dependent query can be arbitrarily positioned in the (x, y, e) space. We start with a straightforward algorithm using a cube to approximate the query plane. The *top plane* and *bottom plane* are defined as the planes specified by (r, e_{max}) and (r, e_{min}) respectively (as shown in Figure 4). Both planes are parallel to the (x, y) plane and when combined define a query cube (shown as the solid line cube in Figure 4). Then, the following algorithm can be used to process a viewpoint-dependent query.

Algorithm 1 $SingleBase(r, e_{min}, e_{max})$

1: let C be the query cube defined using r, e_{min}, e_{max}
2: fetch all the points in the cube
3: construct a mesh M' on the top plane
4: refine M' using the retrieved data to create the mesh on the query plane
5: return the final mesh

Data retrieval is done in step 2, where the 3D R-tree index on the dataset can be effectively used. The query results are returned in the descending order of approximation error, such that the mesh on the top plane can be created in step 3 (similar as in processing the viewpoint-independent query). The refinement process in step 4 is essentially the same to selective refinement using PM.

Comparing to PM-based query processing, the query cube used here is smaller as the top plane is no longer the maximum LOD of the data set (i.e., that of the root node).

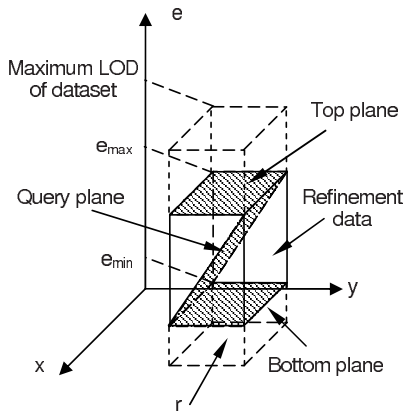


Figure 4. Single base method for viewpoint-dependent queries.

5.3 Viewpoint-Dependent Query - Multiple Base

The total amount of data retrieved for a viewpoint-dependent query, represented as volume of the query cube in Figure 4, can be further reduced. The ideal approach is to retrieve only points whose LOD interval intersects with the query plane in Figure 4, because these are the points that form the approximation. However, these points do not necessarily have a similar LOD since the LOD of the query plane varies. As discussed in Section 4, this requires storing the connectivity information of all possible LODs, which will introduce excessive overhead. Here we propose a different approach to reduce the data amount of viewpoint-dependent queries. For simplicity of presentation, we assume the query plane is parallel to the x-axis, so we can use the projection on the (y, e) plane for discussion hereafter. In practice, the method is applicable when the query plane is arbitrarily positioned. The rectangle in Figure 5(a) shows the total amount of data to be fetched by the single base algorithm, whereas Figure 5(b) shows a possible optimization using two top planes (where the amount of data to be fetched is the sum of the two rectangles).

Each rectangle in Figure 5 implies a range query. Intuitively, the more range queries used, the less the total amount of data retrieved. At the same time, the cost related to the number of queries executed increases (mainly related by repeated index search). The key to this optimization problem is to find the optimal number of queries for a viewpoint-dependent query, which requires estimation of the I/O cost for executing a range query using the R-tree index. The problem of analyzing the I/O cost for range queries using R-tree and its variants has been studied extensively in the past [5, 10, 11, 14, 15, 19]. The number of disk accesses (DA) using a 3-dimensional R-tree index R with N nodes to process a range query q can be estimated

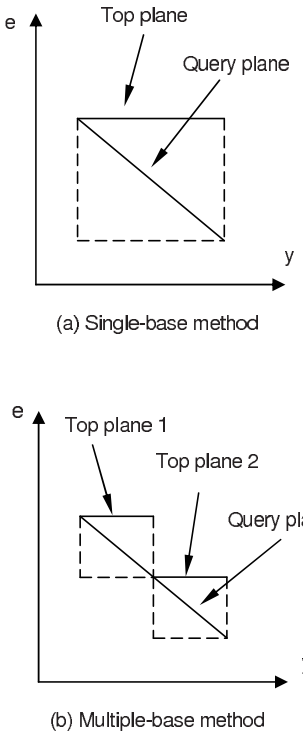


Figure 5. Multiple base optimization.

using the following formula [11, 14]:

$$DA(R,q) = \sum_{i=1}^{N}(q_x + w_i) \cdot (q_y + h_i) \cdot (q_z + d_i) \quad (1)$$

where q_x, q_y and q_z are the width, height and depth of the query cube, and w_i, h_i and d_i are the width, height and depth of node i of R. All the values are normalized according to the data space.

As there is only one range query in the single-base method, formula (1) provides the estimation of its I/O cost. When two top planes are used by the multi-base algorithm (Figure 5(b)), the total cost $DA'(R,q)$ is:

$$DA'(R,q) = \sum_{i=1}^{N}(q_{x1} + w_i) \cdot (q_{y1} + h_i) \cdot (q_{z1} + d_i) + \sum_{i=1}^{N}(q_{x2} + w_i) \cdot (q_{y2} + h_i) \cdot (q_{z2} + d_i) \quad (2)$$

where q_{x1}, q_{y1}, q_{z1} and q_{x2}, q_{y2}, q_{z2} are the sides of the two query cubes used. It is beneficial to use two query cubes instead of one if the following condition holds:

$$DA(R,q) - DA'(R,q) > 0 \quad (3)$$

From Figure 5, we have:

$$q_x = q_{x1} = q_{x2} \quad (4)$$

$$q_y = q_{y1} + q_{y2} \quad (5)$$

$$q_z = q_{z1} + q_{z2} \quad (6)$$

Note that formula 4 to 6 still hold if the query plane is not parallel to the x-axis, because this will only change the position of query plane, not its size. Combining(1)-(6), we have

$$\sum_{i=1}^{N}(q_{x1} + w_i)(q_y q_z - (q_{y1} q_{z1} + q_{y2} q_{z2}) - h_i d_i) > 0 \quad (7)$$

As the size of R-tree nodes (h_i, d_i, w_i) can be found from the R-tree index, all the data required for this optimization is available. Intuitively, whether to use single-base or multi-base algorithm depends on the size of the query window. The bigger the query window, the more likely (7) holds. The maximum benefit can be achieved when the value below is maximized:

$$q_y \cdot q_z - (q_{y1} \cdot q_{z1} + q_{y2} \cdot q_{z2}) \quad (8)$$

This gives the area difference between the rectangle for the single-base case in Figure 5(a) and the sum of the two rectangles for the multi-base case in Figure 5(b). As q_y and q_z are given by the query, $q_y \cdot q_z$ is a constant. Therefore,

$$q_{y1} \cdot q_{z1} + q_{y2} \cdot q_{z2} \quad (9)$$

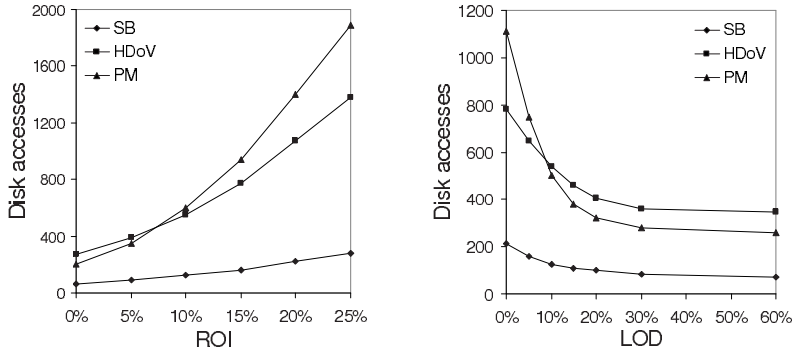
is the only varying part. To maximize the value of (8), (9) should be minimized. This means that dividing the top plane used by the single-base algorithm in the middle will give the maximum reduction in the I/O cost when two range queries are used.

The discussion above can be recursively extended such that more than two query cubes may be used by the multi-base algorithm.

We should mention that all connectivity information can still be found even when multiple top planes are used. Although the meshes created on these different top planes cannot be directly connected with each other, the meshes refined to the query plane can be connected. This is possible because the points on the query plane where two query cubes meet always have similar LOD.

6 Performance Evaluation

In this section, we test the performance of query processing based on our proposed methods and compare them with PM and HDoV tree. DM is created as described in Section 4 and a 3D R*-tree is used. For DM, both the single-base and multi-base algorithms are tested (indicated as SB and MB respectively). The PM approach is implemented following the algorithms in [9]. A LOD-quadtree, which is



(a) Varying ROI - 2M dataset

(b) Varying LOD - 2M dataset

(c) Varying ROI - 17M dataset

(d) Varying LOD - 17M dataset

Figure 6. Uniform mesh.

reported as having better performance than other spatial indexes for MTM data[20], is used for PM data. The HDoV tree is constructed following the algorithms in [18]. The terrain is partitioned into grids, which serve as the objects in the HDoV tree. Visibility data is stored using the "indexed-vertical storage scheme", which is reported to have the best performance among the proposed schemes for the HDoV-tree. No additional spatial index is used with the HDoV tree.

We use Oracle Enterprise Edition Release 9.0.1 in our tests. Its object-relational features and the Oracle Spatial Option are not used in order to have a better control and understanding of query execution performance. All spatial indexes used in this test are implemented by ourself. B^+-tree indexes are created wherever necessary for all the tables used. The database and system buffer is flushed before each test. Other software packages used are Java SDK 1.3 and Java3D SDK (openGL) 1.2. The hardware used is a Pentium III 700 with 512MB memory. We use two terrain datasets. The first one is a real dataset of 2 million points from a local mining software company. The second dataset is the DEM model of "Crater Lake National Park" from U.S. Geological Survey (www.usgs.gov) with 17 million points. Both datasets are pre-processed using the Quadric Error Metrics [7].

The total cost of MTM query processing is composed of two parts: the cost of data retrieval (I/O cost) and the cost of mesh construction (CPU cost). It is found that the cost of mesh construction is very small in comparison to the cost of data retrieval. Therefore, we focus on the I/O cost , which is measured by the number of disk accesses (obtained from Oracle's performance statistics report). However, the CPU cost of DM mesh construction is smaller than the other two methods because it retrieve less and thus a smaller amount of refinement, if there is any. Not measured are those once-off costs, including the initial construction of indexes.

All tests are performed on both datasets and the values in the results are the average value of creating the same mesh (same ROI and LOD) at 20 randomly-selected locations. Terrain data is arranged on the disk in such a way that their (x, y) clustering is preserved as much as possible.

6.1 Viewpoint-Independent Query Performance

There are two main factors that affect the performance of viewpoint-independent queries, ROI size and LOD. Generally, the I/O cost increases as the ROI increase, or as the

LOD value decreases (the decrease of LOD value means a more detailed mesh). To separate the effects of ROI and LOD, we include two sets of tests. The first set tests the performance of queries with varying ROI, whereas the second set tests the performance of queries with varying LOD. Only the performance of the single-base method (SB) is included because the multi-base method (MB) is not applicable to viewpoint-independent queries. Figures 6(a) and 6(c) show the results of tests with varying ROI on the datasets with 2 million points and 17 million points respectively. The x-axis measures ROI size, shown as the percentage of the dataset area, and y-axis measures the number of disk accesses. The LOD of the mesh is set to the average LOD value of the dataset, and the range of ROI in these tests is chosen to allow for a mesh with reasonable data density when displayed, i.e., avoid the case that a mesh is crowded with points and becomes illegible when displayed. Figures 6(b) and 6(d) show the results of tests with varying LOD, again on the dataset with 2 million points and 17 million points respectively. The x-axis measures LOD value, shown as the percentage of maximum LOD value in the dataset, and y-axis measures the number of disk accesses. The ROI is set to 10% for the 2M dataset and 5% for the 17M dataset. We include the results of LOD value range that contains substantial number of points. Performance change can hardly be observed when the LOD value is beyond these ranges. A similar trend is observed in Figures 6(a) and 6(c) (also in Figures 6(b) and 6(d)) where DM clearly outperforms the other two methods.

6.2 Viewpoint-Dependent Query Performance

For viewpoint-dependent queries, there are three major factors that affect the performance. Besides the ROI and LOD, the third factor is the changing rate of LOD. To describe the LOD changing rate, we introduce an *angle* parameter, which is the angle between the query plane and the bottom plane (Figure 7). The larger the angle, the more rapid the change of LOD in the mesh is. For viewpoint-dependent queries, there are three sets of tests based on these three factors. The first two sets are similar to the previous tests: the performance of viewpoint-dependent query is tested with varying ROI and LOD respectively. The third set of tests assess the performance of viewpoint-dependent query with different angle. In this set, the query plane has a fixed e_{min}, and the e_{max} changes according to the variation of angle. The angle values shown in the results are shown as the percentage of the maximum possible angle value θ_{max}, which is given by the following formula:

$$\theta_{max} = \arctan\left(\frac{LOD_{dataset\_max}}{ROI}\right)$$

where the $LOD_{dataset\_max}$ is the maximal LOD value of the dataset.

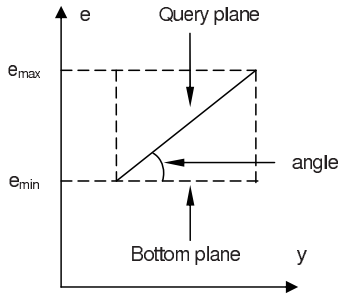


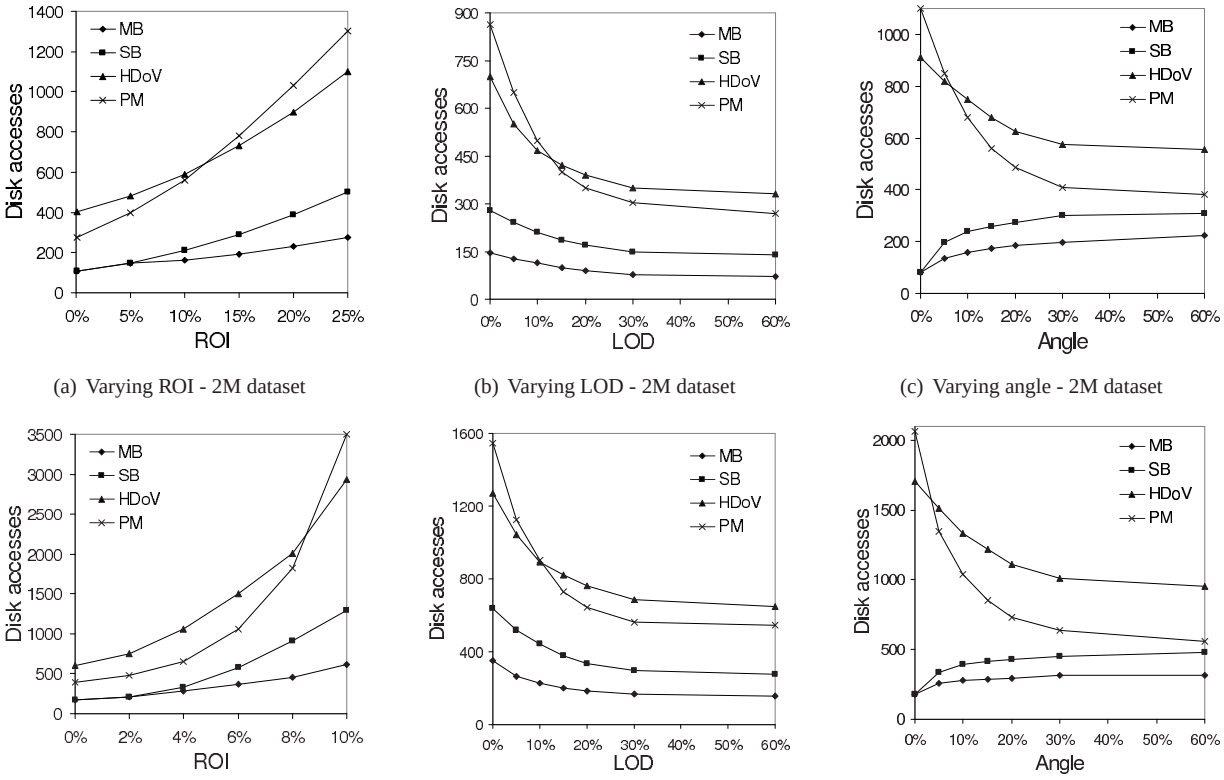
Figure 7. Angle of viewpoint-dependent mesh.

Figures 8(a) and 8(d) show the test results of the first set. The x-axis is the ROI size and y-axis is the number of disk accesses. We set the angle to half the value of θ_{max}. Other parameters are the same as those of analogous tests in the viewpoint-independent query section. Figures 8(b) and 8(e) show the test results of the second set. The x-axis is the e_{min} of the query and y-axis is the number of the disk accesses. The angle is the same as in the previous set and the e_{max} is decided by the e_{min} and angle. All other parameters are the same as those in the uniform mesh section. Figures 8(c) and 8(f) show the test results of the third set. The x-axis is the angle and y-axis is the number of the disk accesses. The ROI setting is the same as the one in the previous set and the e_{min} is set to 1% to allow for a large angle range.

In these tests, the PM and HDoV-tree have similar costs, which are much larger than the cost of DM. The PM retrieves substantially more data than DM, which is the main cause of its poor performance. The visibility selection does not help the HDoV-tree much because obstruction among the areas of the terrain is not as much as in the synthetic city model described in [18]. Hence the HDoV-tree does not always significantly reduce the amount of data retrieved. DM with multi-base algorithm performances the best. The comparison with the single-base method shows that the optimization significantly reduces the retrieval cost. Note that the performance of the DM decreases as the angle increase (Figure 8(c) and 8(f)). The reason is that the increase of angle implies a bigger difference between the e_{min} and e_{max}, thus a larger query cube for single-based method (similar to multi-based method) as the e_{min} is fixed. However, even single-base method still keeps a margin of performance advantage (as explained in Figure 4)

7 Conclusion

In this paper we have presented a new multiresolution terrain data structure called direct mesh. It is a set-operation-friendly data structure particularly suitable for us-



(a) Varying ROI - 2M dataset (b) Varying LOD - 2M dataset (c) Varying angle - 2M dataset

(d) Varying ROI - 17M dataset (e) Varying LOD - 17M dataset (f) Varying angle - 17M dataset

Figure 8. Viewpoint-dependent mesh.

ing a relational DBMS to manage hierarchical multiresolution terrain data. It achieves a good balance between the need of tree-like traversal in order to obtain connectivity information and the overhead of materializing excessive level of redundant information by allowing each node to record only the connectivity information to the nodes with a similar LOD. It is the first MTM structure purposely built to support efficient database query processing.

Acknowledgment: The work reported in this paper has been support by an Australian Research Council Discovery Project grant (grant number: DP0345710).

References

[1] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In *9th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 322–331, Nashville, TN, 1990. ACM Press.

[2] E. Danovaro, L. De Floriani, P. Magillo, and E. Puppo. Compressing multiresolution triangle meshes. In *Advances in Spatial and Temporal Databases. 7th International Symposium SSTD 2001*, pages 345–364. Springer Verlag, 2001.

[3] L. De Floriani, P. Magillo, and E. Puppo. Data structures for simplicial multi-complexes. In *6th International Symposium on Advance in Spatial Database (SSD '99)*, pages 33–51. Springer Verlag, 1999.

[4] L. De Floriani, P. Marzano, and E. Puppo. Multiresolution models for topographic surface description. *Visual Computer*, 12(7):317–345, 1996.

[5] C. Faloutsos and I. Kamel. Beyond uniformity and independence: Analysis of R-trees using the concept of fractal dimension. In *13th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 4–13, Minneapolis, MN, 1994. ACM Press.

[6] M. Garland. Multiresolution modeling: Survey & future opportunities. In *Eurographics '99 – State of the Art Reports*, pages 111–131. Aire-la-Ville (CH), 1999.

[7] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In *24th International Conference on Computer Graphics and Interactive Techniques*, pages 209–216, Los Angeles, CA, USA, 1997. Carnegie Mellon Univ. Pittsburgh PA USA.

[8] H. Hoppe. Progressive meshes. In *23rd International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH'96)*, pages 99–108, New Orleans, LA, USA, 1996. Microsoft Corp. Redmond WA USA.

[9] H. Hoppe. Smooth view-dependent level-of-detail control and its application to terrain rendering. In *IEEE Visualization '98*, pages 35–42, Research Triangle Park, NC, USA, 1998. IEEE Piscataway NJ USA.

[10] J. Jin, N. An, and A. Sivasubramaniam. Analyzing range queries on spatial data. In *16th International Conference on Data Engineering*, pages 525–534, San Diego, California, 2000.

[11] I. Kamel and C. Faloutsos. On packing R-trees. In *2nd ACM International Conference on Information and Knowledge Management*, pages 490–499, Washington, DC, 1993.

[12] M. Kofler, M. Gervautz, and M. Gruber. R-trees for organizing and visualizing 3D GIS database. *Journal of Visualization and Computer Animation*, (11):129–143, 2000.

[13] P. Lindstrom and V. Pascucci. Visualization of large terrains made easy. In *IEEE Visualization*, pages 363–370, San Diego, California, 2001.

[14] B. Pagel, H. Six, H. Toben, and P. Widmayer. Towards an analysis of range query performances. In *ACM-SIGMOD Symposium on Principles of Database Systems*, pages 214–221, Washington, DC, 1993.

[15] G. Proietti and C. Faloutsos. I/O complexity for range queries on region data stored using an R-tree. In *15th International Conference on Data Engineering*, pages 628–635, Sydney, Australia, 1999. IEEE Computer Society.

[16] H. Samet. The quadtree and related hierarchical data structure. *ACM Computing Surveys*, 16(2):187–260, 1984.

[17] L. Shou, C. Chionh, Y. Ruan, Z. Huang, and K. L. Tan. Walking through a very large virtual environment in real-time. In *27th International Conference on Very Large Data Bases*, pages 401–410, Roma, Italy, 2001.

[18] L. Shou, Z. Huang, and K.-L. Tan. HDoV-tree: The structure, the storage, the speed. In *19th International Conference on Data Engineering (ICDE) 2003*, pages 557–568, Bangalore, India, 2003.

[19] Y. Theodoridis and T. Sellis. A model for the prediction of R-tree performance. In *15th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 161–171, Montreal, Canada, 1996. ACM Press.

[20] K. Xu. Database support for multiresolution terrain visualization. In *The 14th Australian Database Conference, ADC 2003*, pages 153–160, Adelaide, Australia, 2003. Australian Computer Society.

[21] X. Zhou, K. Sham, and M. Kitsuregawa. Database support for spatial generalisation for WWW and mobile applications. In *3rd International Congerence on Web Information Systems Engineering*, pages 239–246. IEEE Computer Press, 2002.