A Multi-Resolution Terrain Model for Efficient Visualization Query Processing

Kai Xu, Xiaofang Zhou, *Member, IEEE*, Xuemin Lin, Heng Tao Shen, and Ke Deng

Abstract

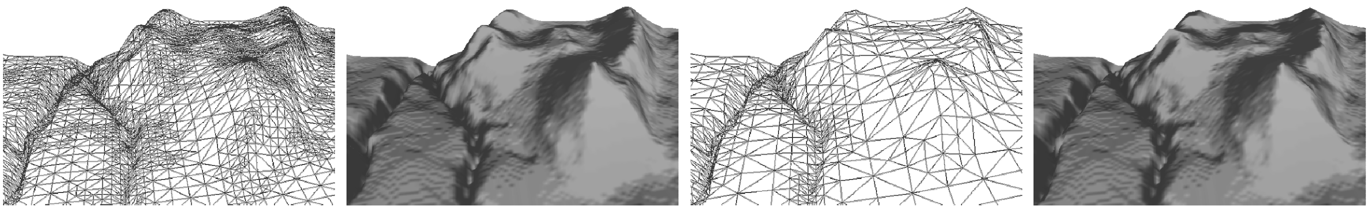
Multi-resolution Triangular Mesh (MTM) models are widely used to improve the performance of large terrain visualization by replacing the original model with simplified one. MTM models, which consist of both original and simplified data, are commonly stored in spatial database systems due to their size. The relatively slow access speed of disk makes data retrieval the bottle-neck of such terrain visualization systems. Existing spatial access methods proposed to address this problem rely on main-memory MTM models, which leads to significant overhead during query processing. In this paper, we approach the problem from a new perspective and propose a novel MTM called Direct Meshes that is designed specifically for secondary storage. It supports available indexing methods natively and requires no modification to MTM structure. Experiment results, which are based on two real-world datasets, show an average of 5-10 times performance improvement over the existing methods.

Index Terms

Multi-resolution visualization, spatial database systems

I. Introduction

TErrain visualization plays an essential role in a wide range of applications such as gaming [1], [2], virtual reality [3], [4], 3D environmental analysis [5], [6], and many GIS applications [7], [8]. Terrain data obtained from the natural environment is usually very large. For instance, the US Geology Survey (www.usgs.gov) provides digital elevation data covering most parts of the US, sampled at 10- or 30-meter resolution. The volume of entire dataset is measured in terabytes. The visualization of such datasets usually requires excessive resources and the resolution can be unnecessarily high in many cases. A typical scenario is that when a terrain with millions of polygons is displayed on a device with limited resolution, it appears to be overly dense and illegible. Fig. 1(a) shows an example with a relatively small number of triangles (10,000), but the top-right part of the terrain already appears to be unnecessarily dense. *Multi-resolution* techniques are introduced to address this problem by replacing the original terrain



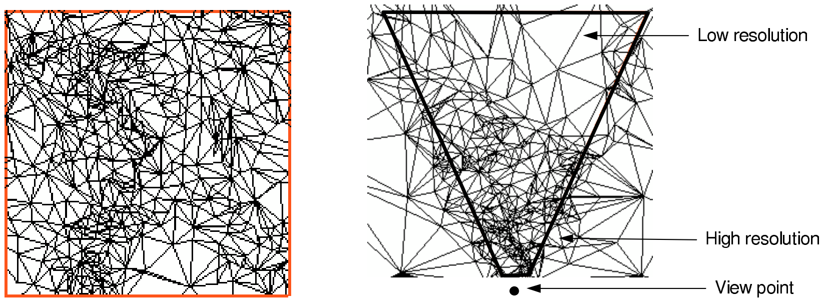
(a) 10,000 triangles (b) 10,000 triangles with texture (c) 1,000 triangles (d) 1,000 triangles with texture

Fig. 1. Terrain at multiple resolutions

model with a simplified approximation (*mesh*) constructed according to application requirements [9]. It reduces the resource requirements significantly while introducing an acceptable sacrifice of visual quality. Following the previous example, a simplified mesh is shown in Fig. 1(c) with only 1000 triangles. The difference is hardly noticeable after texture is applied (Fig. 1(b) and 1(d)).

Mesh construction, which is essentially the simplification of the original model, is expensive due to the usually large size of the terrain dataset. The possible benefit of using a mesh can be overcome by its expensive construction. The fact that terrain data can be used at any resolution and one mesh can have varying resolutions means that it is not feasible to pre-generate meshes at a fixed number of resolutions. An example that requires mesh with varying resolution is shown in Figure 2; to have a uniform-resolution mesh on display (Fig. 2(a)), the actual resolution needs to decrease with the distance from the viewpoint (Fig. 2(b)).

Multi-resolution Triangular Mesh (MTM) models are proposed to address this problem. Most MTM models adopt a tree-like data structure, and each node stores an approximation of the union of its child nodes' data. Mesh construction starts from the root, which contains the least detailed data, and this coarse



(a) Display (b) Mesh

Fig. 2. Mesh with variable resolution

mesh is refined progressively by recursively replacing part of it with the more detailed data stored at child nodes until the desired resolution is achieved (the details are provided in Section II). Since it starts with a very simple mesh rather that the original model, this procedure can significantly reduce construction costs. The initial coarse mesh can be refined to any resolution available and every part can be refined to a different level of detail, which results in a mesh with varying resolution.

MTMs are usually very large because they include both the original model and simplified data. This makes spatial database systems a common choice for their storage. Since the disk access speed is several-order slower than that of main memory, data retrieval becomes the bottle-neck for terrain visualization systems relying on spatial database systems [10], [11]. This problem has attracted significant research attention from both the computer graphics and database research communities. While the graphics community mainly focuses on designing new MTMs suitable for secondary storage [12], [13], [14], database researchers have been proposing spatial access methods to improve data retrieval efficiency [15], [16], [17], [10], [11]. The former approach usually requires the rebuilding of MTM, which is very time consuming; the latter can avoid this problem, but they rely on the tree-like MTM structure, which commonly leads to significant retrieval overhead during query processing. In this paper, we follow the spatial-access-method approach but address the problem from a new perspective. We propose a new MTM called *Direct Meshes*. Different from both existing approaches, the Direct Meshes is a secondary-storage MTM that requires no change to the original MTM structure (no rebuilding) and is designed to seamlessly integrate with well-studied spatial access methods (such as the R*-tree [18] used in this paper). An average improvement of 5-10 times is observed from the evaluation results when comparing against the best existing methods.

A shorter version of this work appeared in [19]. The previous work is extended from several aspects. First, formal definitions of the problem and relevant concepts are included; a general multi-resolution visualization query is introduced to unify the two types of queries in the previous work. Secondly, the Direct Meshes construction algorithm is presented, together with its correctness proof and running time / storage requirement analysis; the design issues such as the topology encoding scheme and the choice of spatial access method are discussed in detail. Thirdly, the query processing algorithm based on the new multi-resolution visualization query replaces the old one; its cost analysis and comparison against existing methods are also added. Fourth, the core of the Direct Meshes, the topology encoding scheme, is enhanced to eliminate the inefficient data retrieval discovered in the previous performance study. Last but not least, extensive test results, including the comparison between the Direct Meshes with different encoding schemes, are added to make the performance evaluation more comprehensive.

The remainder of this paper is organized as follows. The problem is defined in Section II after the introduction to MTM visualization. Related work is reviewed in Section III. The topological information required for multi-resolution visualization and its encoding scheme in Direct Meshes are discussed in Section IV. The Direct Meshes is introduced in Section V, together with its construction algorithm, correctness proof, and complexity analysis. Section VI discusses the query processing using the Direct Meshes, which includes the choice of spatial access methods, cost analysis, and two optimization techniques. A performance study using two sets of real-life terrain data is reported in Section VII. Finally, the paper is concluded in Section VIII.

II. Multi-Resolution Visualization

In this section, we start with an overview of multi-resolution visualization, followed by the definition of *selective refinement query*, which is the main focus of this paper. After that, *Progressive meshes* — one of the most popular MTMs — is used as an example to illustrate selective refinement query processing.

A. Subdivision and Mesh

Let $V = \{v_1, ..., v_n\}$ be a finite set of vertices in a domain $D \subseteq \mathbb{R}^2$. A *subdivision* defined by V is a plane connected straight-line graph having V as set of vertices [20]. Such subdivision can be expressed as a triple $S = (V, E, F)$, where the pair (V, E) is the above mentioned graph, and F is the set of polygonal regions (or faces) induced by this graph. Common choices for S are: *triangular subdivision*, where every region $f_i \in F$ is a triangle; *quadrilateral subdivision*, where every region f_i a quadrilateral. In this paper we only consider the former case, since it is always possible to convert a quadrilateral quadrilateral subdivision into a triangular one.

Given a triangular subdivision $S = (V, E, F)$ and a set of functions $\Phi : V \rightarrow \mathbb{R}$, a *mesh* $M(S, \Phi)$ is a three-dimensional connected straight-line graph that can be expressed as a triple (V', E', F') so that:

1) $\exists\, v_i = (v_i.x, v_i.y) \in V$ iff $\exists\ v'_i \in V'$ so that $v'_i = \psi(v_i) = (v_i.x, v_i.y, \phi_i(v_i.x, v_i.y))$;
2) $\exists\, e_i = (e_i.v_{i1}, e_i.v_{i2}) \in E$ iff $\exists\ e'_i \in E'$ so that $e'_i = (\psi(e_i.v_{i1}), \psi(e_i.v_{i2}))$;
3) $\exists\, f_i = (f_i.v_{i1}, f_i.v_{i2}, f_i.v_{i3}) \in F$ iff $\exists\ f'_i \in F'$ so that $f'_i = (\psi(f_i.v_{i1}), \psi(f_i.v_{i2}), \psi(f_i.v_{i3}))$.

where

- $\Phi = \{\phi_1, \phi_2, \ldots, \phi_n\}$;
- $\psi : V \rightarrow V'$ is a function that maps v_i to v'_i according to ϕ_i;
- $v_i.x$ and $v_i.y$ are the x and y coordinates of v_i respectively;
- $e_i.v_{i1}$ and $e_i.v_{i2}$ are the two end vertices of edge e_i;
- $f_i.v_{i1}$, $f_i.v_{i2}$, and $f_i.v_{i3}$ are the three end vertices of triangle f_i.

B. Simplification and Refinement

A mesh $M(S, \Phi)$ provides an approximation for the original terrain surface and it can be further simplified or refined by increasing or reducing the number of vertices respectively. To define mesh simplification and refinement, we introduce some preliminary notations. The boundary of a mesh M, a circular path of connected vertices and edges, is denoted as $\theta(M)$. Two meshes M_1 and M_2 are *compatible* if they have the same boundary, i.e., $\theta(M_1) \equiv \theta(M_2)$, and this is denoted as $M_1 \leftrightarrow M_2$. A *mesh modification* constructs a new mesh by replacing a subset of the original mesh, which is a mesh itself, with a compatible mesh. A mesh modification γ that changes mesh M_1 into mesh M_2 is denoted as $M_1 \xrightarrow{\gamma} M_2$. A mesh modification is a:

1) *Simplification*, if $|M_1.V| > |M_2.V|$;
2) *Refinement*, if $|M_1.V| < |M_2.V|$.

where $|M_1.V|$ and $|M_2.V|$ are the vertex number of M_1 and M_2 respectively. Note that a simplification ($M_1 \xrightarrow{\gamma} M_2$) is invertible to a refinement ($M_2 \xrightarrow{\gamma'} M_1$), and vice versa.

C. Approximation Error and Selective Refinement Condition

A simplified mesh $M^i = (V^i, E^i, F^i)$ is an approximation of the original mesh $M^n = (V^n, E^n, F^n)$. The similarity between M^i and M^n can be measured by the approximation error of vertices in M^i. The *approximation error of a vertex* $v^i \in V^i$ relating to the original mesh M^n is:

$$\varepsilon(v^i) = |\phi(v^n.x, v^n.y) - \phi_i(v^i.x, v^i.y)|$$

where v^n is on M^n and $(v^n.x,\ v^n.y) = (v^i.x,\ v^i.y)$. In the case that v^n is not in V^n, $\phi(v^n.x, v^n.y)$ can be obtained from the face $f^n \in F^n$ that contains v^n.

With approximation error, required mesh can be specified by *selective refinement condition*, which defines the maximum acceptable error for every part of the mesh. Formally, given a domain $D \subseteq \mathbb{R}^2$, a selective refinement condition is a function $\delta : D \rightarrow \mathbb{R}$ that determines the maximum acceptable error $\delta(v.x, v.y)$ for every $v \in D$. A mesh M *satisfies* a selective refinement condition δ if all its vertices have approximation error no greater than the value specified by the condition, i.e. $\forall v \in M.V$,

$$\varepsilon(v) \begin{cases} \leq \delta(v.x, v.y), & v \in D \quad (1) \\ arbitrary\ value, & otherwise \quad (2) \end{cases}$$

D. Multi-Resolution Triangular Mesh

A multi-resolution triangular mesh (MTM) representation of a mesh M^n can be built by recursively applying a sequence of simplifications $(\alpha_{n-1}, \alpha_{n-2}, ..., \alpha_0)$ to M^n. Each simplification step produces a simpler mesh, i.e., M^{i-1} (results from $M^i \xrightarrow{\alpha_{i-1}} M^{i-1}$) has fewer vertices than M^i. The sequence of meshes

$$M^n \xrightarrow{\alpha_{n-1}} M^{n-1} \xrightarrow{\alpha_{n-2}}, \cdots, \xrightarrow{\alpha_1} M^1 \xrightarrow{\alpha_0} M^0$$

have a monotonically reducing number of vertices. Due to the large size of the original mesh M^n and the fact that mesh simplification is invertible, the common approach stores the simplest mesh together with a sequence of refinement, i.e.,

$$M^0 \xrightarrow{\beta_0} M^1 \xrightarrow{\beta_1}, \cdots, \xrightarrow{\beta_{n-2}} M^{n-1} \xrightarrow{\beta_{n-1}} M^n$$

where β_i is the inverse of the $\alpha_i, 0 \leq i \leq n-1$. If we regard the initial mesh M^0 as a refinement from a *null* mesh, the set of refinements $R = \{M^0, \beta_0, \beta_1, \ldots, \beta_{n-1}\}$ is the *multi-resolution triangular mesh* (MTM) representation of mesh M^n.

Early MTM models only allow construction of meshes that appear during simplification. This is achieved by only allowing a refinement sequence that follows the exact reverse order of simplification. Such an approach guarantees a proper triangulated mesh given it is checked during MTM construction. However, this total ordering can be quite restrictive for selective refinement query. Later approaches adopt a less restrictive partial ordering, which can be described as a directed acyclic graph (DAG) whose nodes represent refinement and edges are the dependency among refinements. An MTM based on such partial ordering can be defined as a DAG $G = (R, E_R)$, where $R = \{M^0,\ \beta_0, \beta_1, \ldots, \beta_{n-1}\}$ is the set of mesh refinements and E_R is a set of directed edges defined on $R \times R$ such that there is a directed edge $(\beta_i, \beta_j),\ i \neq j$ if β_j refines part of the mesh results from β_i. To avoid confusion, we refer to the point in a mesh as "vertex" and the point in an MTM as "node" from here on.

E. Selective Refinement Query

Selective refinement constructs a mesh according to a given condition δ. Usually there are multiple meshes that meet the selective refinement condition because only the upper bound of mesh error is given. Among them, the one with the least number of vertices is commonly preferred as the result of a selective refinement. Given an MTM G and selective refinement condition δ, a *selective refinement query* $Q(G, \delta)$ returns a mesh M where:

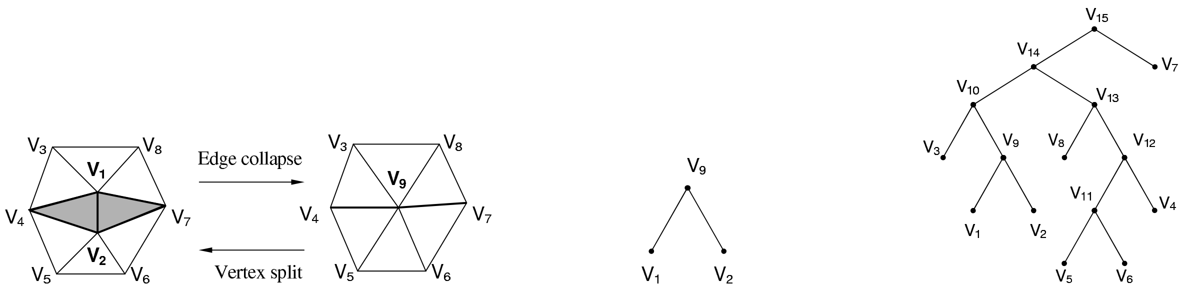
1) M satisfies δ;
2) $|M.V| \leq |M^i.V|$ for all δ-satisfying meshes M^i that can be constructed from G.

We say mesh M is *feasible* relating to δ.

In this paper we focus on selective refinement query processing in multi-resolution terrain databases. Our goal is to reduce the data retrieval for a selective refinement query. One of the main constraints is to support selective refinement with varying resolution as described in Section I. Formally, given an MTM G and a selective refinement condition δ, we try to minimize the overall I/O cost $DA(Q(G, \delta))$, where δ is a function of mesh vertex location.

F. Progressive Meshes

For this paper we use the *Progressive Meshes* [21], one of the most popular MTMs, as an example to illustrate our work. The Progressive Meshes is built upon a simplification process called *edge collapse*, in which one edge collapses into a new vertex (Fig. 3(a)). The new vertex (v_9) is recorded as the parent node of the two end nodes (v_1 and v_2) of the collapsed edge (Fig. 3(b)). Edge collapse is repeated recursively on the resulting mesh until only one vertex is left, and the resulting structure is an unbalanced binary tree (Fig. 3(c)). Progressive Meshes stores the final mesh (one vertex) and a set of *vertex splits* (the inverse of edge collapses, Fig. 3(a)). Progressive Meshes can be defined as a directed binary tree $G_p = (N_p, E_p)$ where every node n in N_p represents a vertex and every edge e in E_p pointing from the parent to its children.



(a) Edge collapse and vertex split

(b) Edge collapse structure

(c) Tree structure of Progressive Meshes

Fig. 3. Progressive Meshes

A linear-time incremental algorithm is proposed in [22] for selective refinement query. The algorithm starts with a mesh with root only and iterates through its vertex list, during which vertex, whose approximation error is greater than the maximum acceptable error, is replaced with its children recursively. The algorithm stops when the mesh satisfies the selective refinement condition. The pseudo code in Algorithm 1 outlines the majors steps.

Algorithm 1: Selective Refinement Algorithm for Progressive Meshes

Input: Progressive Meshes $G_p = (N_p, E_p)$, selective refinement condition δ

Output: mesh M

$M \leftarrow M^0$;
for *every vertex* $v \in M.V$ **do**
    if $\varepsilon(v) > \delta(v)$ **then**
        remove v from $M.V$;
        append v_1, v_2 (the children of v) to $M.V$;
    endif
endfor
return M

III. Related Work

In this section, we survey previous work that employs spatial access methods to improve the I/O performance of selective refinement query processing. These methods require little or no modification to the existing MTM. Not included are methods that require MTM rebuilding and those that can only support grid or right-triangle mesh.

Algorithm 1 discussed in the last section was designed for main-memory Progressive Meshes. Its progressive refinement nature makes it inefficient for secondary-storage data access. The fact that the necessity of every node (except the root) depends on its parent, i.e., a node is required only if its parent's approximation error is greater than the value specified by the selective refinement condition, makes it difficult to retrieve all necessary data together. In fact, the data for every vertex split (two child nodes) has to be fetched individually. Therefore, the data fetch of a selective refinement query is composed of many retrievals with every small amounts, which is very inefficient in terms of query processing. Similar problem exists for any MTM that is based on a DAG structure.

The LOD-R-tree [23] is one of the first attempts among the various spatial access methods proposed to address this problem. An LOD-R-tree is constructed by building a traditional R-tree into an MTM in the following way. First, a two-dimensional R-tree is created on the vertices of the original model. Then approximation mesh is added to the internal node in a bottom-up fashion by merging and simplifying the meshes stored at its child nodes. Selective refinement query is converted to a range query whose query window is the domain of the selective refinement condition. The query processing algorithm traverses down the LOD-R-tree and stops at the level where the mesh resolution is sufficient for the LOD condition. The main problem of the LOD-R-tree is that it relies on the hierarchical structure of the MTM and uses a progressive refinement approach similar to that of the main-memory algorithm. This introduces significant overhead caused by multiple retrievals. Also, the LOD-R-tree does not support mesh with continuously changing resolution because data stored at the internal nodes always have uniform LOD. Another problem is that the mesh size at internal nodes is pre-defined, and the entire mesh needs to be retrieved even if only a fraction is needed, which can cause considerable redundant data retrieval.

Hoppe [15] suggests an approach similar to the LOD-R-tree, but using a two-dimensional Region Quadtree [24] instead. Its construction and query processing are analogous to those of the LOD-R-tree; therefore they share the same problems.

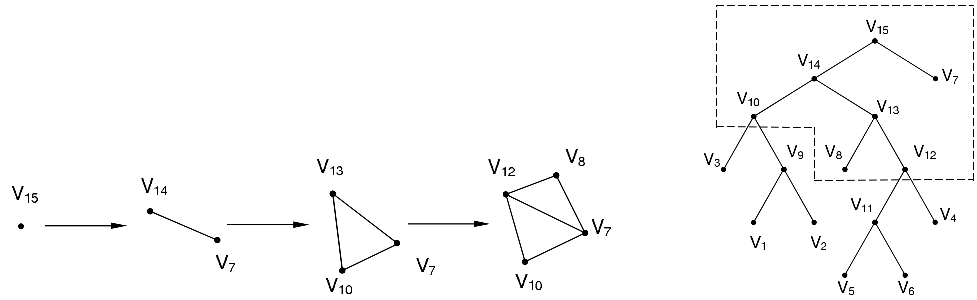
Shou et al. improved the LOD-R-tree by including visibility data and proposed a new indexing structure called the HDoV-tree [10]. Visibility information is stored at every node of the LOD-R-tree, so occluded parts can be excluded during query processing and low resolution is used for areas that are partially visible. However, the HDoV-tree does not address the problems associated with the LOD-R-tree.

The LOD-quadtree proposed by Xu [11] treats every node in an MTM as a point in the $x - y - e$ ("e" stands for "approximation error") three-dimensional space and ignores the DAG structure. The selective refinement query is translated into a three-dimensional range query defined by the selective refinement condition δ and its domain D. Therefore, one retrieval fetches the majority of data. However, not all the necessary nodes are included, and the missing ones still need to be retrieved individually.

In summary, all these methods try to improve the retrieval efficiency of selective refinement query by avoiding fetching data separately. However, this can be totally avoided and it is usually achieved at the cost of being less flexible and retrieving extra data. More importantly, they all keep the progressive refinement routine for mesh construction, which is inherited from the main-memory MTM. We believe avoiding progressive refinement is the fundamental approach that secondary-storage MTM should follow, which is the main motivation of our work on the Direct Meshes.

IV. Mesh Topology

Before introducing the Direct Meshes, we discuss the selective refinement query processing in more detail. Essentially, the feasible mesh M of a selective refinement condition δ defines a subtree T of the Progressive Meshes. Fig. 4 shows a sequence of refinements during a selective refinements query processing and the subtree of Progressive Meshes defined by this query. The leaf nodes of this subtree $L(T)$ are the vertices in the resulting mesh M, i.e., $L(T) \equiv M.V$. The topology of M, i.e., edges among the vertices, are obtained implicitly from the structure of T. For instance, there is always an edge between two children by the definition of vertex split. We believe that storing the topology information using MTM tree structure — which is the key characteristic of main-memory MTM — is the major cause of the poor performance of selective refinement query processing, because it is well known that the relational model used in many spatial database systems cannot efficiently handle deductive reasoning, such as processing the parent-children relationship in the Progressive Meshes.



(a) Selective refinements

(b) Subtree in Progressive Meshes

Fig. 4. Selective refinement and its subtree in the Progressive Meshes

A naive solution is to pre-compute and store the mesh topology information, which consists of every possible edge between vertices represented by MTM nodes. The fact that a vertex can connect to other vertices with a wide range of approximation error makes the total number of possible edges prohibitively large. Specifically, if vertex v can connect to vertex v', it can also connect to the parent of v', because a parent vertex connects to all the neighboring vertices of its children. Similarly, v can connect to the child vertices of v', because at least one child vertex of v' connects to v. To be precise:

1) If v can connect to v', v can connect to any ancestor of v' up to v'' (excluding v'') where v'' is the common ancestor of v and v' of the lowest height in the Progressive Meshes;
2) If v can connect to v', v can connect to at least one descendant of v' recursively till leaf level.

Theorem 4.1: Given a Progressive Meshes $G_p = (N_p, E_p)$, the total number of possible edges is $O(|N_p|^2)$, where $|N_p|$ is the cardinality of N_p.

Proof: The previous observation implies that the number of possible edges for a given vertex is proportional to the level of vertices it can connect to, which is proportional to the height of the Progressive Meshes. Since a Progressive Meshes is an unbalanced binary tree, its height is between $O(\log |N_p|)$ and $O(|N_p|)$. Therefore, the total number of possible edges is $O(|N_p|^2)$ in the worst case. ■

A naive method to store all possible edges is not practical due to the substantial storage overhead and resulting increased I/O costs.

Another option is to encode each node as an MBR (minimum bounding rectangle) of its descendants. As a result, the MBRs of required nodes intersect with the domain of selective refinement condition δ. This, combined with the approximation error range defined by δ, can be used to identify the needed nodes in MTM. However, the MBRs of the nodes in the upper part of MTM cover a large area and tend to overlap with each other. Such overlapping can significantly degrade the search performance of most available spatial access methods, which are based on data partition.

The Direct Meshes only encodes selected topology information, which avoids storing the large amount of possible edges and can answer a selective refinement query without relying on MTM structure. The details are elaborated in the next section.

V. Direct Meshes

To introduce Direct Meshes, we need to define *node approximation error*. Note that this is different from the vertex approximation error (defined in Section II), which is the absolute difference between the height of a vertex ($\phi_i(v^i)$) and that of its corresponding vertex in the original model ($\phi(v^n)$). In an MTM, the error introduced by a node should reflect the error of all its descendants, i.e., this vertex accumulates the distortion that all its descendant nodes bring to the mesh. Therefore, given a Progressive Meshes $G_p = (N_p, E_p)$, the *approximation error of node* $n \in N_p$ that represents vertex v is:

$$e(n) = \begin{cases} 0, & \text{if } n \text{ is a leaf node} \quad (3) \\ \max(\varepsilon(v), e(n.child1), e(n.child2)), & \text{otherwise} \quad (4) \end{cases}$$

where $n.child1$ and $n.child2$ are the two children of n. This definition guarantees that node approximation error value increases monotonically along any path from a leaf to the root.

Another important concept of the Direct Meshes is *LOD interval* ("LOD" stands for "Level Of Detail"). For a node $n_i \in N_p$, its *LOD interval* $\sigma(n_i) \subseteq \mathbb{R}$ is an interval that

$$\sigma(n_i) = \begin{cases} [\ e(n_i), +\infty\ ), & \text{if } n_i \text{ is the root} \quad (5) \\ [\ e(n_i), e(n_i.parent)\ ), & \text{otherwise} \quad (6) \end{cases}$$

where $n_i.parent$ is the parent node of n_i.

Given a Progressive Meshes $G_p = (N_p, E_p)$, a *Direct Meshes* $G_d = (N_p, E_p, E_d)$ is a multi-relational graph with an additional relation E_d, which is a subset of the cartesian product $N_p \times N_p$. Specifically:

$$E_d = \{(n_i, n_j) \mid n_i, n_j \in N_p,\ i \neq j,\ n_i \sim n_j,\ \sigma(n_i) \cap \sigma(n_j) \neq \emptyset\}$$

where $n_i \sim n_j$ denotes that n_i can connect to n_j. E_d is the set of possible edges whose end nodes have overlapping LOD intervals. We name them *candidate edges*. The candidate edge set is a subset of all possible edges, and its size is much less than that of the latter because the LOD-interval-overlapping constraint limits the number of different resolution levels a candidate edge can cross. In fact, the average number of candidate edges for each node is a constant (proof is given later in this section). Thus, the size of E_d is linear to the number of nodes $O(|N_p|)$.

Each node of a Direct Meshes has the following data structure:

Class *DirectMeshNode* {

Integer NodeID;

Double x, y, z;

Double e; // node approximation error

DirectMeshNode parent, child1, child2;

DirectMeshNodeArray N_b; // the other end node of candidate edges

}

Algorithm 2: Direct Meshes Construction

Input: Progressive Meshes $G_p = (N_p, E_p)$

Output: Direct Meshes $G_d = (N_p, E_p, E_d)$, every node $n \in N_p$ has a list of candidate edges N_b

$N_e \leftarrow$ the ordered set of all nodes in N_p with a descending approximation error (ancestor is ahead of descendant if they have the same error) ;

while N_e *is not empty* **do**

  $n_0 \leftarrow$ the first node in N_e;

  remove n_0 from N_e;

  if $n_0.parent \neq null$ **then**

    $n_1 \leftarrow$ the other child node of $n_0.parent$;

    $n_0.N_b \leftarrow n_0.N_b \cup \{n_1\}$;

    for *every node* $n_i \in n_0.parent.N_b$ **do**

      if $n_0.N_b \cap \{n_i\} = \emptyset$ *AND* $\sigma(n_i) \cap \sigma(n_0) \neq \emptyset$ *AND* $n_i \sim n_0$ **then**

        $n_0.N_b \leftarrow n_0.N_b \cup \{n_i\}$;

        $N_i^d \leftarrow$ the set of descendent nodes of n_i whose LOD interval overlaps with that of n_0;

        for *every node* $n_j^d \in N_i^d$ **do**

          if $n_0.N_b \cap \{n_j^d\} = \emptyset$ *AND* $n_i \sim n_0$ **then**

            $n_0.N_b \leftarrow n_0.N_b \cup \{n_j^d\}$;

          endif

        endfor

      endif

    endfor

  endif

endw

Algorithm 2 outlines the major steps of the Direct Meshes construction. The algorithm scans the node set N_p in a descending-approximation-error order and finds the candidate edge set N_b for every node. It starts with ordering the nodes according to their approximation error (line 1). For every node n, the first candidate edge is the one between itself and the node sharing the same parent (line 5-7). To find other candidate edges, the algorithm checks the nodes in the parent candidate edge set and their descendants (line 8-18). For algorithm correctness proof, we need the following lemma:

Lemma 5.1: Given a node n and its candidate edge list $n.N_b$, n' (the parent node of n) and its candidate edge list $n'.N_b$, for every node $n_i \in n.N_b$ (except the node (n_0) sharing the same parent), $n_i \in n'.N_b$ or there is a node $n_i' \in n'.N_b$ such that n_i' is an ancestor of n_i.

Proof: First, we show that for a node n_i, if $n_i \sim n$, then $n_i \sim n'$. Because $n_i \sim n$, we can construct a mesh that has (n_i, n) as one of its edge and (n, n_0) as another edge. Then, we can simplify the mesh by collapsing edge (n, n_0) into n'. There will be an edge (n_i, n') in the new mesh, therefore $n_i \sim n'$. This can be extended to any ancestor of n, i.e., $n'' \sim n_i$ if $n \sim n_i$ and n'' is an ancestor of n.

For a node $n_i \in n.N_b$, $n_i \sim n$, therefore $n_i \sim n'$. If $\sigma(n_i) \cap \sigma(n) \neq \emptyset$, $n_i \in n.N_b$, proved. Otherwise, $e(n_i.parent) < e(n')$ because $\sigma(n_i) \cap \sigma(n) = \emptyset$ and $e(n) \leq e(n')$. Let n_i' be one of the ancestors of n_i. Because $n_i \sim n'$, $n_i' \sim n'$. Because $e(n_i') \geq e(n_i)$, there is always an ancestor n_i' such that $\sigma(n_i') \cap \sigma(n') \neq \emptyset$, therefore $n_i' \in n'.N_b$, proved. ■

Theorem 5.1: Algorithm 2 finds all the candidate edges in a Progressive Meshes.

Proof: To prove this, it is sufficient to show that the algorithm finds all the candidate edges for every node. The root is the only node that has no parent. It is obvious that the root has no candidate edge, because the only possible mesh that has the root is the one that has the root as the only vertex. Except the root, every node has a parent. From Lemma 5.1 we know all the candidate edges of a node can be found by searching the nodes in its parent candidate edge set and their descendants. The ordering of the nodes guarantees that parent candidate edges are always computed before that of its children. Therefore, the algorithm finds all the candidate edges for every node. ■

Next, we discuss the time complexity and storage requirement of the algorithm, which requires the following lemmas.

Lemma 5.2: Given two intervals $A, B \subseteq \mathbb{R}$ that are both partitioned into a set of sub-intervals, i.e.,

$$A = \cup_{k=1}^{m} a_k,\ a_i \cap a_j = \emptyset,\ i \neq j,\ 1 \leq i, j \leq m$$

$$B = \cup_{k=1}^{n} b_k,\ b_i \cap b_j = \emptyset,\ i \neq j,\ 1 \leq i, j \leq n$$

For $1 \leq i \leq m,\ 1 \leq j \leq n$, there is a ***interval edge*** (a_i, b_j) if $a_i \cap b_j \neq \emptyset$. The maximum total number of interval edges ($\max(|E_i|)$) is $m + n - 1$.

Proof: This can be proved by induction.

- Base case: $m = 1, n = 1$. $|E_i| = 0$ when $A \cap B = \emptyset$; $|E_i| = 1$ otherwise. Therefore, $\max(|E_i|) = 1$
- Inductive step: assume that $\max(|E_i|) = k - 1$ when $m + n = k$.

  Let $m+n = k+1$. Without loss of generality, we assume that the increased partition is introduced by dividing an existing partition $a_i \subset A$ at point p. It is obvious that $|E_i|$ will not change if $\{p\} \cap B = \emptyset$; $|E_i|$ will increase by 1 otherwise. Therefore $\max(|E_i|) = (k - 1) + 1 = k$.

■

Based on Lemma 5.2, we have the following lemma.

Lemma 5.3: Given a Direct Meshes $G_d = (N_p, E_p, E_d)$, the total number of candidate edges $|E_d|$ is $O(|N_p|)$.

Proof: The Progressive Meshes $G_p = (N_p, E_p)$, which is a binary tree, can be divided into a set of paths whose approximation error changes monotonically. For instance, Fig. 5 shows a partition of the Progressive Meshes in Fig. 4(b) into a set of such paths. Every path is an approximation error interval that is the union of the LOD interval of every node. For example the left-most path is an error interval $[0, +\infty)$ and it is the union of the LOD interval of v_{15} $([e(v_{15}), +\infty))$, v_{14} $([e(v_{14}), e(v_{15})))$, etc. According to Lemma 5.2, the total number of candidate edges, which is equivalent to interval edge, is linear to the total number of interval partitions, which in this case equals $|N_p|$.

However, each path can be paired with more than one other path. If we take any mesh that can be constructed from G_p, the average node degree is constant because it is a triangulation. In other words, on average one path can pair with a constant number of paths. Thus, the total number of interval edges is $k \cdot N_p$ where k is a constant, i.e., the total number of candidate edges is $O(|N_p|)$. ■

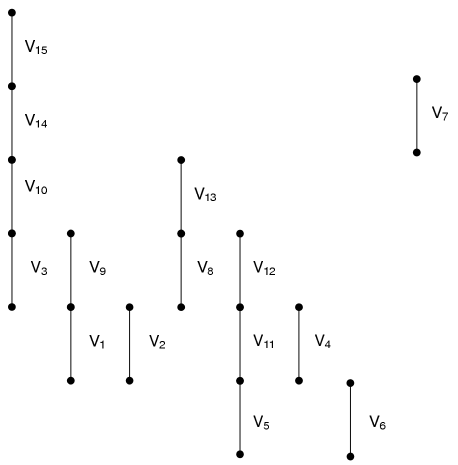


Fig. 5. Partition of a Progressive Meshes into a set of paths

Given the Lemma 5.3, we have the following theorem.

Theorem 5.2: Given a Progressive Meshes $G_p = (N_p, E_p)$, the running time of Algorithm 2 is $O(|N_p|)$ and it requires $O(|N_p|)$ space for the candidate edge set of the resulting Direct Meshes.

Proof: The algorithm starts with sorting all the nodes in N_p according to their approximation error. Because the error decreases monotonically along any path from the root to leaf, this can be done in $O(|N_p|)$ time. Within the main loop (line 2-20), removing the first node n_0 from the sorted node set N_e takes constant time (line 3-4). It is the same for finding the node n_1 sharing the same parent (line 6-7), given that every node stores links to its parent and children. From Lemma 5.3, we know on average every node has a constant number of candidate edges, therefore the next loop (line 8-18) repeats a constant time. Similarly, we can show that the average node number in N_i^d (line 11) is also constant. Thus, the next loop (line 12-15) runs in constant time. Therefore, the running time of Algorithm 2 is $O(|N_p|)$. From Lemma 5.3 we know there are $O(|N_p|)$ candidate edges in total, so it requires $O(|N_p|)$ space. ■

In summary, the Direct Meshes requires no modification to the original MTM structure; it adds an candidate edge set for every node, which can be done in linear time with linear space. Selective refinement query processing using the Direct Meshes is discussed in the next section.

VI. SELECTIVE REFINEMENT QUERY PROCESSING

Once the Direct Meshes is constructed, an important issue is the choice of spatial access method, which needs to support the characteristics of the Direct Meshes database and selective refinement query. As mentioned in [11], the distribution of MTM nodes in the x-y-e three-dimensional space is highly skewed, which makes it unsuitable for spatial access methods based on regular space partition, such as Region Quadtree [24]. As we will see later in this section, selective refinement query can be converted to range query in x-y-e space, which makes it unnecessary for any particular method to support graph traversing (the Direct Meshes is essentially a DAG), such as the connectivity-clustered access method [25]. The results in [26] showed that the R-tree and the R+-tree have much better query performance than the K-D-B-tree [27] when the datasets contain rectangles of varying size, which is very similar to the case of selective refinement query (as we shall see later in this section). We did not use variations of Grid File, such as the Multilevel Grid File [28], as they share a similar indexing structure as the K-D-B-tree. Eventually, we choose the R*-tree [29] because it is reported to have the best performance among the R-tree variations [29]. Every node n in a Direct Meshes is indexed using a R*-tree as a three-dimensional line segment

$$((n.x, n.y, e(n)), (n.x, n.y, e(n.parent))$$

representing its LOD interval in the $x - y - e$ space.

Given a Direct Meshes $G_d = (N_p, E_p, E_d)$ and a selective refinement condition δ (with domain D), Algorithm 3 outlines the main steps of processing the query $Q(G_d, \delta)$. The algorithm retrieves all the nodes within D and having an LOD interval that overlaps with the error interval $(\delta_{min}, \delta_{max})$ defined by the selective refinement condition δ (line 2). The *base mesh* M_0 is then built according to the selective refinement condition $\delta' : D \rightarrow \delta_{max}$ (line 3-11). M_0 is further refined using the Algorithm 1 until the required mesh M is obtained. Note that only one retrieval is performed (line 2) and it fetches all the data used by the algorithm.

Algorithm 3: Selective refinement using Direct Meshes

Input: Direct Meshes $G_d = (N_p, E_p, E_d)$, selective refinement condition δ

Output: mesh M that is feasible regarding to δ

$\delta_{min}/\delta_{max} \leftarrow$ the minimal / maximal approximation error value defined by δ over domain D;

$N \leftarrow$ set of nodes that are within D and their LOD interval overlaps with $(\delta_{min}, \delta_{max})$;

$N_0 \leftarrow$ set of nodes that are within D and their LOD interval contains δ_{max};

Mesh $M_0 = (N_0, E_0)$, $E_0 \leftarrow \emptyset$;

for *every node* $n_i \in N_0$ **do**

    for *every node* $n_j \in n_i.N_b$ **do**

        if $n_j \in N_0$ *AND* $E_0 \cap \{(n_i, n_j)\} = \emptyset$ **then**

            $E_0 \leftarrow E_0 \cup \{(n_i, n_j)\}$;

        endif

    endfor

endfor

Mesh $M \leftarrow$ Algorithm 1(M_0,N);

return M;

Theorem 6.1: Given a Direct Meshes $G_d = (N_p, E_p, E_d)$, the mesh M returned by Algorithm 3 is feasible for the selective refinement condition δ.

Proof: First, we show that mesh M_0 produced by the algorithm is a proper mesh. For every node $n_i \in N_0$, its LOD interval contains δ_{max}, therefore their LOD intervals all overlap with each other. Because the Direct Meshes contains every possible candidate edge and E_0 is a subset of E_d, then every edge in E_0 is contained in the candidate edge list of nodes in N_0. Since all the nodes have overlapping LOD intervals, mesh M_0 should be checked during the Direct Meshes construction. Any candidate edge that may breach the validity of a mesh (such as causing an improper triangulation) should be excluded from the Direct Meshes. Therefore, every candidate edge in E_0 is necessary for M_0. Thus, M_0 is a proper mesh. Once M_0 is constructed, it is further refined by Algorithm 1, which is proved in the previous work to produce the feasible mesh for the given selective refinement condition δ. ■

A. Cost Analysis

The I/O cost of processing a selective refinement query using the Direct Meshes can be modeled as:

$$DA(Q) = I + t_0 \cdot \lceil |N|/B \rceil \tag{7}$$

where I is the cost of searching the R*-tree of the Direct Meshes to find the required nodes, t_0 is the cost of retrieving one disk page, $|N|$ is the total number of nodes needed, and B is the number of nodes each disk page can hold. Here we assume that the required nodes are stored continuously on the disk. Although this cannot be fully achieved in practice, it can be seen as a close estimation of the real cost. Similarly, the cost of previous methods can be modeled as:

$$DA(Q) = (I' + k \cdot t_0) \cdot \lceil |N'|/(k \cdot B) \rceil \tag{8}$$

where k is a nature number and indicates the node size of the indexing structure, because these methods with customized index can have node size more than one disk page.

From the cost models we can see that the Direct Meshes incurs less I/O cost than the previous methods:

1) Less index scan. While index scan only occurs once using the Direct Meshes, $\lceil |N'|/(k \cdot B) \rceil$ scans are needed previously because one is needed for every retrieval of child nodes during progressive refinement.
2) Less redundant data. A node could contain unnecessary data, especially when the area it covers is on the boundary of the domain D. Though this is unavoidable, the smaller the index node size, the less the redundant data. The Direct Meshes, which have the minimal index node size one, retrieve less redundant data than previous methods (index node size k) when answering the same query.
3) Less MTM node retrieval. Previous methods always start from the MTM root to find the feasible mesh; whereas the Direct Meshes can start from the lowest mesh resolution (δ_{max}) and skip all

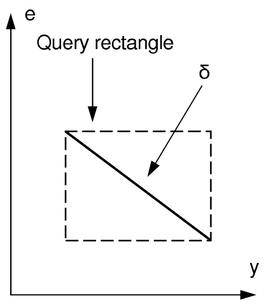
the MTM nodes with lower resolution. The saving can be significant, especially when the required mesh has a uniform resolution.

B. Query Optimization

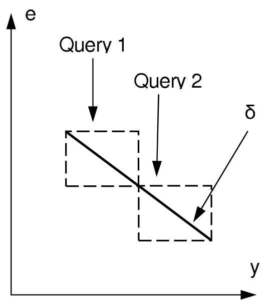
The Direct Meshes works best when the selective refinement condition δ specifies a constant δ_0 over D, in which case the base mesh M_0 is feasible and no redundant data is retrieved. When δ is not a constant, we consider the case that the required resolution reduces linearly to the vertex distance to a reference (such as the viewpoint). Such selective refinement produces a uniform resolution mesh on display (Fig. 2), and is arguably the most common query in multi-resolution visualization. Specifically, for a vertex $v \in D$ its required resolution value $\delta(v)$ is:

$$\delta(v) = \delta_0 - k \cdot |\, v - \tau_0| \tag{9}$$

where k is a constant and $|\, v - \tau_0|$ is the Euclidean distance between v and the reference τ_0. In this case the total amount of data retrieval can be further reduced by using multiple queries. The idea is illustrated in Fig. 6. For simplicity, we assume the reference is the x-axis, so we can use the projection on the



(a) Single-base approach



(b) Multi-base approach

Fig. 6. Possible optimization using multiple queries.

(y, e) plane for discussion hereafter. Note that the results can be easily extended to the case where the mesh is arbitrarily positioned. The area of rectangle in Fig. 6(a) indicates the total amount of data to be fetched by the *single-base* algorithm (Algorithm 3), whereas Fig. 6(b) shows a possible optimization using two queries. In the latter case the total data retrieval amount (indicated as the sum of the area of two query rectangles) is less than that of using one query only (Fig. 6(a)). However, the *multi-query* approach has its own overhead such as the cost of index scan for each query. Therefore, to minimize the overall retrieval, the optimization algorithm should partition the domain D into a set of sub-domains $S(D) = \{d_1, d_2, \ldots, d_n\}$ so that the total retrieval cost $DA(D) = DA(d_1) + DA(d_2) + \ldots + DA(d_n)$ is minimized. In practice, it is difficult to find the optimal partition $S(D)$ that minimized the retrieval cost. Here we propose a heuristic approach that partitions the domain recursively, based on the retrieval cost estimation.

The problem of analyzing the I/O cost of range queries using the R-tree and its variants has been extensively studied in the past ([30], [31], [32], [33], [34], [35]). The number of disk accesses (DA) using a three-dimensional R-tree index T_r with N_r nodes to process a range query q can be estimated using the following formula [32], [33]:

$$DA(T_r, q) = \sum_{i=1}^{N_r} (q_x + w_i) \cdot (q_y + h_i) \cdot (q_z + d_i) \tag{10}$$

where q_x, q_y and q_z are the width, height and depth of the query cube respectively, and w_i, h_i and d_i are the width, height and depth of node n_i of T_r respectively. All the values are normalized according to the data space, which has a unit size ($1 \times 1 \times 1$). The value of $DA(T_r, q)$ includes the I/O cost of both index scan and data retrieval. Formula (10) provides an estimation of I/O cost for single-base approach (Fig. 6(a)); when two queries are used (Fig. 6(b)), the total cost $DA'(T_r, q)$ is:

$$DA'(T_r, q) = \sum_{i=1}^{N_r} (q_{x1} + w_i) \cdot (q_{y1} + h_i) \cdot (q_{z1} + d_i) + \sum_{i=1}^{N_r} (q_{x2} + w_i) \cdot (q_{y2} + h_i) \cdot (q_{z2} + d_i) \tag{11}$$

where q_{x1}, q_{y1}, q_{z1} and q_{x2}, q_{y2}, q_{z2} are the width, height and depth of the two query cubes respectively. More queries should be used only if it incurs less total I/O cost, i.e.,

$$DA(T_r, q) - DA'(T_r, q) > 0 \tag{12}$$

From Fig. 6, we have:

$$q_x = q_{x1} = q_{x2} \tag{13}$$

$$q_y = q_{y1} + q_{y2} \tag{14}$$

$$q_z = q_{z1} + q_{z2} \tag{15}$$

Note that formula (13) to (15) still holds if the query plane is not parallel to the x-axis, because this change only affects the position of the query plane, not its size. Combining (10)-(15), we have:

$$\sum_{i=1}^{N_r}(q_{x1} + w_i)(q_y q_z - (q_{y1}q_{z1} + q_{y2}q_{z2}) - h_i d_i) > 0 \tag{16}$$

i.e., so long as condition (16) holds, more queries should be used. As the size of the R-tree nodes (h_i, d_i, w_i) can be obtained from the R-tree index, all the data required for this optimization is available. From formula (16), we can know that the maximum reduction can be achieved when the value below is maximized:

$$q_y \cdot q_z - (q_{y1} \cdot q_{z1} + q_{y2} \cdot q_{z2}) \tag{17}$$

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

$$q_{y1} \cdot q_{z1} + q_{y2} \cdot q_{z2} \tag{18}$$

is the only variable element. To maximize the value of (17), the value of (18) should be minimized, which means that dividing the original query into two equal-sized sub-queries will give the maximum I/O reduction. Given a Direct Meshes $G_d = (N_p, E_p, E_d)$, its R*-tree index T_r, and a selective refinement condition δ, Procedure "SelectiveRefinement-MultiBase" outlines the main steps of a query processing algorithm that adopts the multi-base approach.

Procedure `SelectiveRefinement-MultiBase(`G_d, T_r, δ`)`

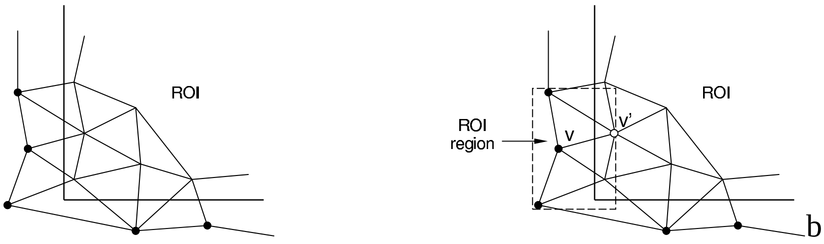
```
Mesh M ← ∅;
D_1, D_2 ← the two partitions result from dividing D equally;
if DA(D_1) + DA(D_2) < DA(D) then
    Mesh M_1 ← SelectiveRefinement-MultiBase(D_1);
    Mesh M_2 ← SelectiveRefinement-MultiBase(D_2);
    M ← M_1 + M_2;
else
    M ← Algorithm3(G_d, δ(D));
endif
return M;
```

C. ROI Region

During performance analysis, we find that some vertices required for a selective refinement query are not retrieved at step 2 in Algorithm 3 and need to be fetched individually during mesh construction, which introduces I/O overhead and can be substantial for large datasets. After close examination, we find that these points are not covered by the selective refinement domain D (indicated as the black points in Fig. 7(a)), and are missed by the algorithm. However, they are essential for a correct mesh boundary. Expanding D so that it can cover these points is not a feasible solution because it is difficult to estimate the amount of expansion, and unnecessary data can also be included.

To address this problem, we propose to associate every node with a *ROI region* ("ROI" stands for "Region Of Interest"). ROI region is the minimal two-dimensional interval that covers the node itself and all its candidate edges. The ROI region of vertex v is shown in Fig. 7(b). Formally, given a node n with its candidate edge list $n.N_b$, its ROI region $n.r$ is a two dimensional interval so that:

$$n.r = \{(x, y) \,|\, x_{min} \le x \le x_{max}, y_{min} \le y \le y_{max}\}$$



(a) ROI feasible points that are outside ROI

(b) ROI region of v

Fig. 7. ROI region

where:

- $x_{min} = \min(n.x, n_i.x \,|\, n_i \in n.N_b)$;
- $x_{max} = \max(n.x, n_i.x \,|\, n_i \in n.N_b)$;
- $y_{min} = \min(n.y, n_i.y \,|\, n_i \in n.N_b)$;
- $y_{max} = \max(n.y, n_i.y \,|\, n_i \in n.N_b)$;

ROI region can be computed in constant time once all the candidate edges are identified; therefore including ROI region will not change the running time of the Direct Meshes construction algorithm. Also, each node requires constant space for an ROI region, thus it will not change the storage requirement of the Direct Meshes.

With ROI region, each node in a Direct Meshes is encoded as a cube whose spatial extent is the ROI region and the resolution extent is the LOD interval (Fig. 8). Accordingly, nodes can be identified by

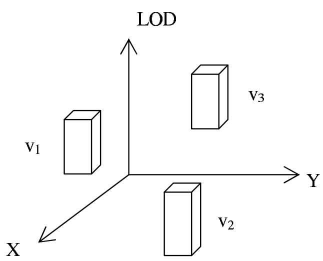


Fig. 8. Direct Meshes nodes in the x-y-LOD space.

their ROI region, i.e., a node is retrieved if its ROI region intersects with the selective refinement domain.

This is the only modification required for the query processing algorithm (Algorithm 3).

VII. Performance Evaluation

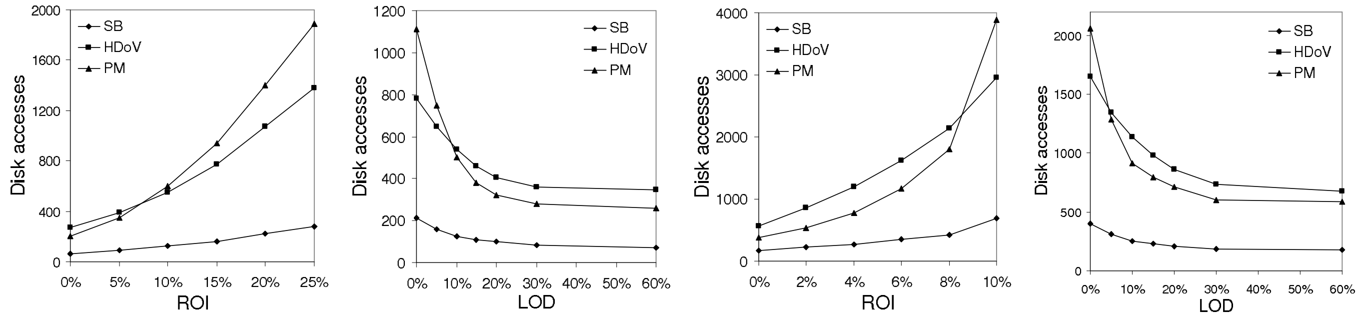
In this section, we evaluate the performance of the selective refinement query processing based on the Direct Meshes. Two different versions of the Direct Meshes are implemented: one does not apply "ROI region" (as described in Section V); the other does (as described in Section VI). Both Direct Meshes have a three-dimensional R*-tree built upon them for query processing. For comparison we also include the LOD-quadtree and the HDoV-tree, because they provide the best performance among the available methods that support MTM with DAG structure. The Progressive Meshes is implemented as described in [21]. The LOD-quadtree is created according to [11]. The HDoV tree is constructed following the algorithms in [10]. For the HDoV tree, 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. Two types of selective refinement queries are tested: one is the *uniform-LOD query* where the selective refinement condition specifies a constant value over its domain, the other is the *variable-LOD query* where the selective refinement condition specifies a value reducing linearly to the vertex distance. For the latter, the test results of both single-base and multi-base methods are included.

The total cost of the MTM query processing is composed of two parts: the cost of data access (I/O cost) and the cost of mesh construction (CPU cost). In terms of running time, it is found that the former dominates the overall cost. Therefore, we focus our performance evaluation on the I/O cost, which is measured by the number of disk accesses (obtained from Oracle's performance statistics report). Note that the CPU cost of the Direct Meshes is generally smaller than that of the other two methods due to the fact that it retrieves less data, and thus requires less computation for refinement, if there is any. All the values in the results are the average value of repeating the same query 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.

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 the query execution performance. All spatial indexes used are implemented by ourselves. B^+-tree indexes are created wherever necessary. The database and system buffer is flushed before each test to minimize the effect of caching. Other software packages used are Java SDK 1.3 (for data retrieval) and Java3D SDK (openGL) 1.2 (for mesh visualization). The hardware used is a Pentium III 700 with 512MB memory. Two real-world terrain datasets are used in the tests. The first one is a 2-million-point terrain model 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.

A. Uniform-LOD Query Performance

The two main parameters that affect the performance of the uniform-LOD queries are the resolution value specified by the selective refinement condition and the size of its domain. Generally, the I/O cost increases as the resolution value decreases (which implies a more detailed mesh) or the domain size increases. To separate their effects we conducted two sets of tests. The first set has varying domain size and fixed resolution value, whereas the second set is the opposite. Figures 9(a) and 9(c) show the results of the first set with 2 and 17 million points respectively. The x-axis measures the domain size (indicated as “ROI”), shown as the percentage of the dataset area, and y-axis measures the number of disk accesses. The mesh resolution is set to the average value of the dataset, and the range of domain size is chosen to allow for a mesh with reasonable data density when displayed, i.e., avoid meshes that are overly crowded when displayed. The LOD-quadtree is indicated as “PM” (stands for “Progressive Meshes” because it does not change its structure), the HDoV tree as “HDoV”, and the Direct Meshes as “SB” (single-base method). Similarly, figures 9(b) and 9(d) show the results of the second set. The x-axis denotes resolution (indicated as “LOD”), shown as the percentage of maximum LOD value in the dataset. The domain size is set to 10% for the 2M dataset and 5% for the 17M dataset. We include the results of a resolution range that contains a substantial number of points. Performance changes are hardly noticeable when the resolution value is beyond this range.

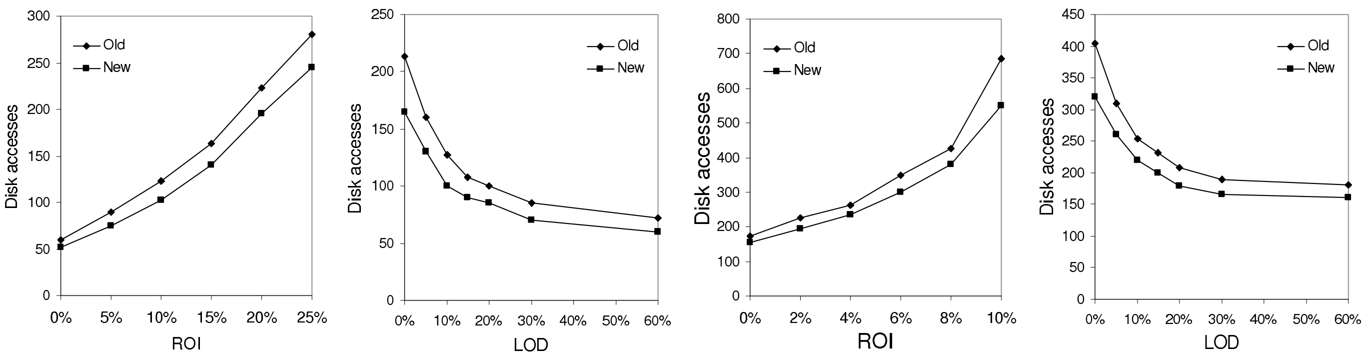


(a) Varying ROI - 2M dataset (b) Varying LOD - 2M dataset (c) Varying ROI - 17M dataset (d) Varying LOD - 17M dataset

Fig. 9. Uniform mesh.

The Direct Meshes clearly outperforms the other two methods in these tests, and the performance gap grows as the mesh size increases. This complies well with our analysis in Section VI, since the Direct Meshes incurs much less retrieval overhead than the other two methods. The slow increase of the Direct Meshes also indicates that it scales much better with the mesh size than the other two.

To illustrate the effect of “ROI region”, we also compare the performance of the Direct Meshes with and without this encoding scheme. The results are shown in Fig. 10. The Direct Meshes without ROI-region encoding is labeled as “Old”, whereas the one with encoding is labeled as “New”. The improvement



(a) Varying ROI - 2M dataset (b) Varying LOD - 2M dataset (c) Varying ROI - 17M dataset (d) Varying LOD - 17M dataset

Fig. 10. Uniform mesh.

made by the ROI-region encoding is not significant in these tests. We think the reason is that we have little control over the query processing in the database system. We empty the buffer at the beginning of every test to reduce its effect, but we have no control over buffer usage during query processing. As a result, the cost of individually retrieving necessary nodes outside selective refinement condition domain is considerably reduced, because the index scanning associated with such retrieval is very likely to be performed in the buffer with considerably reduced cost. Nevertheless, the new encoding scheme still achieves reasonable reduction on retrieval cost.

B. Variable-LOD Query Performance

For variable-LOD queries, the resolution changing rate k is the third parameter that affects the cost of a variable-LOD query, besides the mesh size and average resolution value. In the tests we use an *angle* parameter θ, which is the angle between the mesh and the horizontal plane (Fig. 11 where δ_{min} and δ_{max} are the minimum and maximum approximation error of the mesh), to describe the resolution changing rate. The larger the angle, the more rapid the change of resolution. Three sets of tests are conducted to

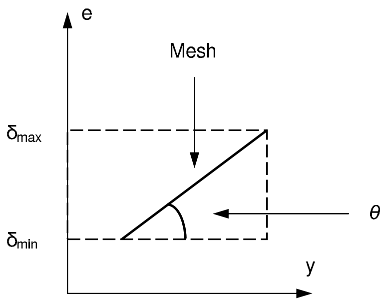
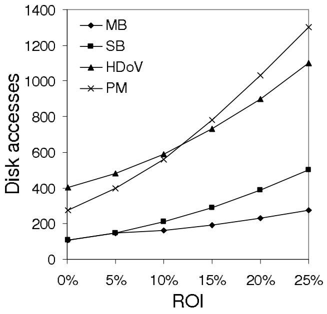


Fig. 11. Angle of variable-LOD mesh.

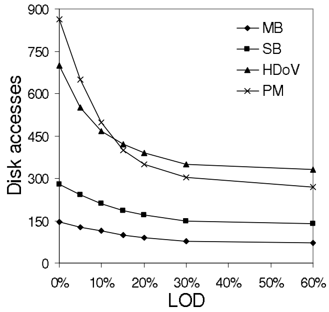
illustrate the effect of each parameter. The first two sets are similar to those of the uniform-LOD queries: the performance of the variable-LOD query is tested with varying domain size and average resolution value respectively. The third set of tests assess the performance with different angles. In this set, the mesh has a fixed δ_{min}, and the δ_{max} changes according to the variation of angle. The angle values in the results are shown as the percentage of the maximum possible angle value θ_{max} given by the following formula:

$$\theta_{max} = \arctan\left(\frac{E_{max}}{|D|}\right)$$

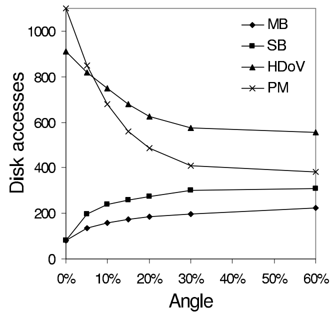
where the E_{max} is the maximum approximation error of the dataset and $|D|$ is the dimension of the D.



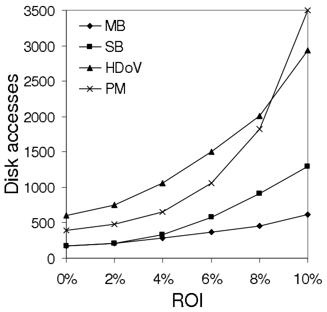
(a) Varying ROI - 2M dataset



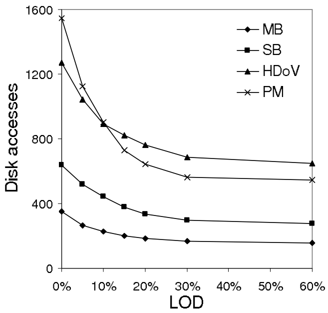
(b) Varying LOD - 2M dataset



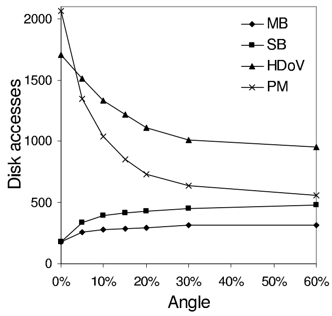
(c) Varying angle - 2M dataset



(d) Varying ROI - 17M dataset



(e) Varying LOD - 17M dataset



(f) Varying angle - 17M dataset

Fig. 12. Variable-LOD mesh.

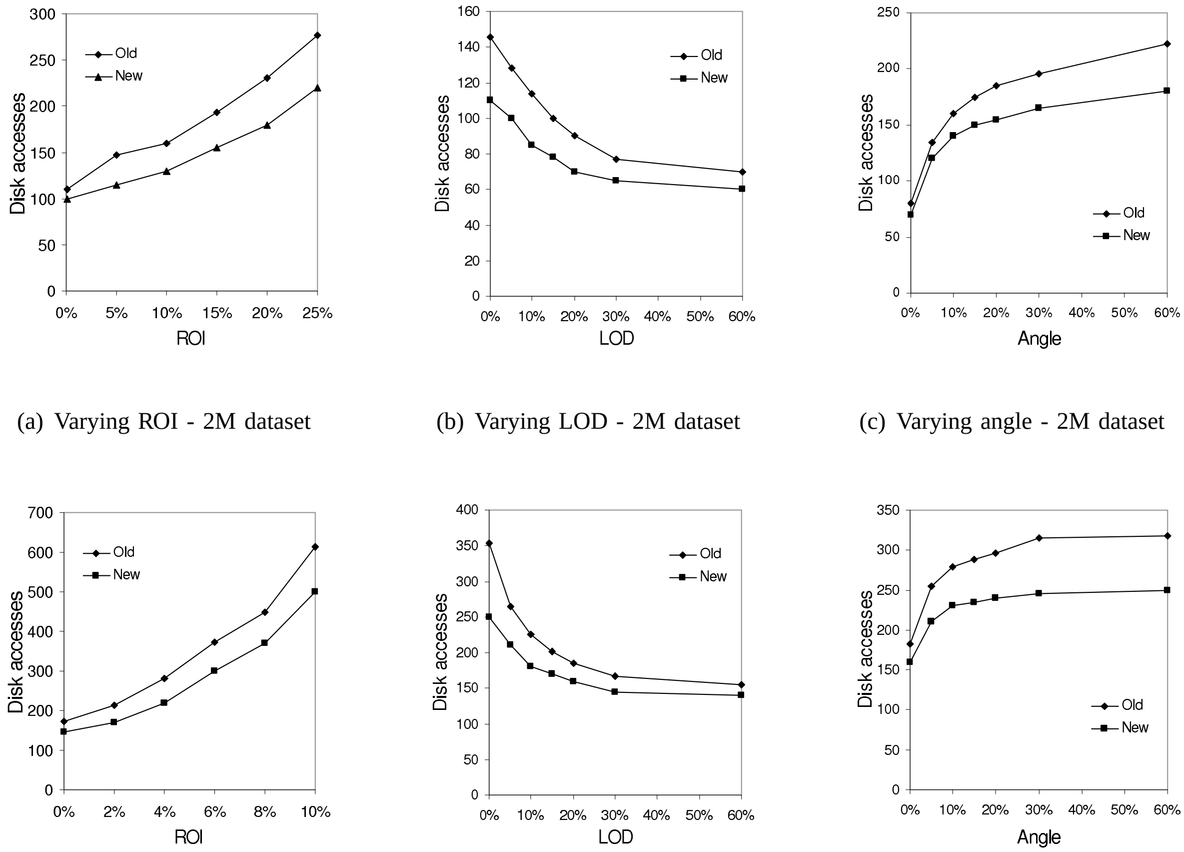
Figures 12(a) and 12(d) show the test results of the first set. The Direct Meshes query processing using single and multiple query are shown as “SB” and “MB” (“Multi-Base”) respectively. The x-axis denotes the mesh size and the y-axis indicates the number of disk accesses. We set the angle to half the value of θ_{max}. Other parameters are the same as those of analogous tests in the uniform-LOD query section. Figures 12(b) and 12(e) show the test results of the second set. The x-axis is the δ_{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 δ_{max} is decided by the δ_{min} and angle. All other parameters are the same as those in the uniform mesh section. Figures 12(c) and 12(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 mesh size setting is the same as the one in the previous set and the δ_{min} is set to 1% to allow for a large angle range.

The tests results are consistent with those in the previous sub-section. The LOD-quadtree and the HDoV-tree have similar costs, which are much larger than that of the Direct Meshes. The LOD-quadtree retrieves substantially more data than the Direct Meshes, which is the main cause of its poor performance. Visibility selection does not help the HDoV-tree much because obstruction among the areas of the tested terrain is not as much as in the synthetic city model described in [10]. Hence, the visibility constraint does not always significantly reduce data amount. The multi-base approach performs best. Its comparison against the single-base method shows that the optimization can significantly reduce retrieval cost. Note that the performance of approaches based on the Direct Meshes decreases as the angle increases (Fig. 12(c) and 12(f)). The reason is that the increase of angle implies a bigger difference between the δ_{min} and δ_{max}; thus, a larger query cube for single-base method (similar to multi-base method) as the δ_{min} is fixed. However, even the single-base method still has a considerable performance advantage against the other two methods.

Test results of similar comparison against the improved Direct Meshes with ROI-region encoding is presented in Fig. 13. Here the improved version (labeled as “New”) is compared against Direct Meshes using multiple-base method (labeled as “Old”), which has the best performance in the tests so far. The new encoding scheme exhibits more performance advantage here. We think the reason is that, in the variable-LOD query, the amount of nodes outside of ROI has a larger proportion than that in the uniform-LOD query. Therefore, its improvement has a larger impact on the total cost. This is clearly shown in Fig. 13(c) and 13(f): the gap between the two methods increases as the angle increases, i.e., the performance improvement grows with the query window volume.

VIII. Conclusions

In this paper we presented a novel multi-resolution terrain data structure called the Direct Meshes. It is a secondary-storage MTM particularly suitable for selective refinement query processing with a relational DBMS. It achieves a good balance between the need of tree-like traversal in order to obtain mesh topology information and the overhead of materializing such information by allowing each node to record selected topology information only. It requires no modification to the original MTM structure and can be constructed in linear time with linear space. Selective refinement query processing using the



(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

Fig. 13. Variable-LOD mesh.

Direct Meshes improves the performance by avoiding multiple retrievals and reducing the total amount of required data. The multi-base approach further reduces the cost of varying-LOD queries. The introduction of ROI region encoding provides a solution for retrieving necessary data outside of the selective refinement condition domain. Significant performance improvement is observed from tests based on the real-world datasets. The scalability of the Direct Meshes is demonstrated when comparing against best available methods with different data sizes.

REFERENCES

[1] W. Piekarski and B. Thomas, "Arquake: the outdoor augmented reality gaming system," *Communications of the ACM*, vol. 45, no. 1, pp. 36–38, 2002.

[2] M. C. Whitton, "Making virtual environments compelling," *Communications of the ACM*, vol. 46, no. 7, pp. 40 – 47, 2003.

[3] D. Green, J. Cosmas, T. Itagaki, M. Waelkens, R. Degeest, and E. Grabczewski, "A real time 3D stratigraphic visual simulation system for archaeological analysis and hypothesis testing," in *Conference on Virtual Reality, Archeology, and Cultural Heritage*. Glyfada, Greece: ACM Press, 2001, pp. 271 – 278.

[4] S. Kiss and A. Nijholt, "Viewpoint adaptation during navigation based on stimuli from the virtual environment," in *Eighth International Conference On 3D Web Technology*. Saint Malo, France: ACM Press, 2003, pp. 19 – 26.

[5] B. Benes and R. Forsbach, "Parallel implementation of terrain erosion applied to the surface of mars," in *1St International Conference on Computer Graphics, Virtual Reality and Visualisation*. Camps Bay, Cape Town, South Africa: ACM Press, 2001, pp. 53 – 57.

[6] T. Gerstner, D. Meetschen, S. Crewell, M. Griebel, and C. Simmer, "A case study on multiresolution visualization of local rainfall from weather radar measurements," in *Conference on Visualization*. Boston, Massachusetts: IEEE Computer Society, 2002, pp. 533 – 536.

[7] J. Randall W. Hill, Y. Kim, and J. Gratch, "Anticipating where to look: predicting the movements of mobile agents in complex terrain," in *International Joint Conference on Autonomous Agents and Multiagent Systems*. Bologna, Italy: ACM Press, 2002, pp. 821 – 827.

[8] B. Ben-Moshe, J. S. B. Mitchell, M. J. Katz, and Y. Nir, "Visibility preserving terrain simplification: an experimental study," in *Eighteenth Annual Symposium on Computational Geometry*. Barcelona, Spain: ACM Press, 2002, pp. 303 – 311.

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

[10] 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*, Bangalore, India, 2003, pp. 557–568.

[11] K. Xu, "Database support for multiresolution terrain visualization," in *The 14th Australian Database Conference, ADC 2003*. Adelaide, Australia: Australian Computer Society, 2003, pp. 153–160.

[12] C. DeCoro and R. Pajarola, "Xfastmesh: fast view-dependent meshing from external memory," in *Conference on Visualization*. Boston, Massachusetts: IEEE Computer Society, 2002, pp. 363 – 370.

[13] P. Lindstrom, "Out-of-core construction and visualization of multiresolution surfaces," in *Symposium on Interactive 3D graphics*. Monterey, California: ACM Press, 2003, pp. 93 – 102.

[14] M. Isenburg and S. Gumhold, "Out-of-core compression for gigantic polygon meshes," *ACM Transactions on Graphics*, vol. 22, no. 3, pp. 935 – 942, 2003.

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

[16] M. Kofler, M. Gervautz, and M. Gruber, "R-trees for organizing and visualizing 3D GIS database," *Journal of Visualization and Computer Animation*, no. 11, pp. 129–143, 2000.

[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*, Roma, Italy, 2001, pp. 401–410.

[18] 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. Nashville, TN: ACM Press, 1990, pp. 322–331.

[19] K. Xu, X. Zhou, and X. Lin, "Direct mesh: a multiresolution approach to terrain visualization," in *20th International Conference on Data Engineering*, Boston, USA, 2004, pp. 766–777.

[20] F. P. Preparata and M. I. Shamos, *Computational Geometry: an Introduction*. Springer-Verlag, 1985.

[21] H. Hoppe, "Progressive meshes," in *23rd International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH'96)*. New Orleans, LA, USA: ACM press, 1996, pp. 99–108.

[22] H.Hoppe, "View-dependent refinement of progressive meshes," in *24th International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '97)*. Los Angeles, CA, USA: ACM press, 1997, pp. 189–198.

[23] M. Kofler, "R-trees for visualizing and organizing large 3D GIS databases," Ph.D. dissertation, Technische Universitat Graz, 1998.

[24] R. A. Finkel and J. L. Bentley, "Quad trees: A data structure for retrieval on composite keys," *Acta Informatica*, vol. 4, pp. 1–9, 1974.

[25] S. Shekhar and D.-R. Liu, "CCAM: A connectivity-clustered access method for aggregate queries on transportation networks: A summary of results," in *Proceedings of the Eleventh International Conference on Data Engineering*, P. S. Yu and A. L. P. Chen, Eds. IEEE Computer Society, 1995, pp. 410–419.

[26] D. Greene, "An implementation and performance analysis of spatial data access methods." in *Proceedings of the Fifth International Conference on Data Engineering*, 1989, pp. 606–615.

[27] J. T. Robinson, "The K-D-B-tree: A search structure for large multidimensional dynamic indexes," in *ACM SIGMOD Int. Conf. on Management of Data*. ACM Press, 1981, pp. 10–18.

[28] K.-Y. Whang and R. Krishnamurthy, "The multilevel grid file - a dynamic hierarchical multidimensional file structure," in *Proceedings of the Second International Symposium on Database Systems for Advanced Applications*. World Scientific Press, 1992, pp. 449–459.

[29] 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-SIGMOD Symposium on Principles of Database Systems*, Nashville, TN, 1990, pp. 322–331.

[30] 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*. Minneapolis, MN: ACM Press, 1994, pp. 4–13.

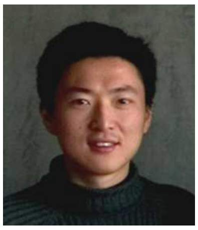
[31] J. Jin, N. An, and A. Sivasubramaniam, "Analyzing range queries on spatial data," in *16th International Conference on Data Engineering*, San Diego, California, 2000, pp. 525–534.

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

[33] 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*, Washington, DC, 1993, pp. 214–221.

[34] 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*. Sydney, Australia: IEEE Computer Society, 1999, pp. 628–635.

[35] 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*. Montreal, Canada: ACM Press, 1996, pp. 161–171.



Kai Xu is currently a researcher at National ICT Australia. He is also an Honorary Associate of School of Information Technologies at University of Sydney. He received his PhD in Computer Science in 2004 from the University of Queensland. Before that, he received his bachelor degrees in Computer Science and Business from Shanghai Jiao Tong University in 1999. His main research interests are information visualization, bioinformatics, and spatial database systems.

Xiaofang Zhou is currently a Professor at the University of Queensland, Australia. He is the Research Director of Australia Research Council (ARC) Research Network in Enterprise Information Infrastructure (EII), a Chief Investigator of ARC Centre in Bioinformatics, and a Senior Researcher of National ICT Australia (NICTA). He received his BSc and MSc degrees in Computer Science from Nanjing University, China, in 1984 and 1987 respectively, and PhD in Computer Science from the University of Queensland in 1994. His research interests include spatial information systems, high performance query processing, Web information systems, multimedia databases, data mining and bioinformatics.

Xuemin Lin is an Associate Professor in the School of Computer Science and Engineering, the University of New South Wales. He has been the head of database research group at UNSW since 2002. Before joining UNSW, Xuemin held various academic positions at the University of Queensland and the University of Western Australia. Dr. Lin got his PhD in Computer Science from the University of Queensland in 1992 and his BSc in Applied Math from Fudan University in 1984. During 1984-1988, he studied for PhD in Applied Math at Fudan University. His current research interests lie in data streams, approximate query processing, spatial data analysis, and graph visualization.

Heng Tao Shen is a Lecturer in School of Information Technology and Electrical Engineering, The University of Queensland. He obtained his BSc (with 1st class Honors) and PhD from Computer Science, National University of Singapore in 2000 and 2004 respectively. His research interests include Database, Web/multimedia search, mobile Peer-to-Peer computing. He has served as a PC member for international conferences including ICDE’06, EDBT’06, SAC’06, DASFFA’05, and publication chair for APWEB06. His papers appeared in ACM SIGMOD, ACM MM, ICDE, VLDB Journal, TKDE, ACM Multimedia Sysmte Journal, and other major venues.

Ke Deng is currently a PhD student in the School of Information Technology and Electrical Engineering, University of Queensland, Australia. He holds a Master degree in compute science from Griffith University and a Bachelor degree in Electrical Engineering. His research area is in the spatial database systems.