Multi-resolution Out-Of-Core Modeling of Terrain and Geological Data

Emanuele Danovaro[1,2], Leila De Floriani[1,2], Enrico Puppo[1], Hanan Samet[2]

[1] Dipartimento di Informatica e Scienze dell'Informazione
Università di Genova - Via Dodecaneso, 35, 16146 Genova, Italy

[2] Computer Science Dept., Center for Automation Research, Institute for Advanced Computer Studies
University of Maryland - College Park, Maryland 20742

ABSTRACT

Multi-resolution is a useful tool for managing the complexity of huge terrain and geological data sets. Since encoding large data sets may easily exceed main memory capabilities, data structures and algorithms capable of efficiently working in external memory are needed. In our work, we aim at developing an out-of-core multi-resolution model dimension-independent, that can be used for both terrains, represented by Triangulated Irregular Networks(TINs), and 3D data, such as geological data, represented by tetrahedral meshes. We have based our approach on a general multi-resolution model, that we have proposed in our previous work, which supports the extraction of variable-resolution representations. As first step, we have developed, in a prototype simulation system, a large number of clustering techniques for the modifications in a multi-resolution model. Here, we describe such techniques, and analyze and evaluate them experimentally. The result of this investigation has led us to select a specific clustering approach as the basis for an efficient out-of-core data structure.

Categories and Subject Descriptors

I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling

General Terms

Algorithms

Keywords

Terrain models, multi-resolution, out-of-core

1. INTRODUCTION

Datasets describing terrains, geophysical phenomena, or generated by simulation, usually consist of a mesh of polygonal/polyhedral cells with one or more fields defined either at the vertices, or on the cells of the mesh. A terrain model is described by a mesh subdividing an either flat or spherical domain, with a field representing elevation defined at its vertices. In geological data, a volume is tessellated with polyhedral cells, and a field can represent, e.g., concentration of water, minerals or pollution, pressure, mechanical stress, etc.

Available datasets are becoming larger and larger, and processing them at their full resolution has often prohibitive computational costs, even for high-end workstations. Multi-resolution models proposed in the literature may improve efficiency of processing large datasets, by adapting resolution on-the-fly, according to the needs of an application [19]. Data at high resolution are pre-processed once to build a multi-resolution data structure, which can be queried on-line by the application. The multi-resolution model acts as a black box that provides simplified datasets, where resolution is focused on the region of interest, and at the Level Of Detail (LOD) required by the application. The simplified dataset is an approximated representation of the data set at full resolution. Thus, it is affected by some approximation error, which is usually associated with either vertices or cells of the simplified mesh. For instance, a triangle of a TIN will represent the elevation of a portion of the terrain surface within a certain error from ground truth.

In this work, we consider datasets represented by irregular simplicial meshes. Examples are TINs for describing terrains and tetrahedral meshes for describing geological data. In our previous work, we have developed a dimension-independent multi-resolution model based on simplicial decompositions, called a *Multi-Tessellation (MT)*, which provides a general framework for multi-resolution models proposed in the literature [9, 25]. On the basis of this model, we have already developed systems for terrain modeling [8] and volume visualization [2], as well as a library for the fast prototyping of applications using multi-resolution, that we distribute in the public domain [20].

However, the data structures we have developed so far for the MT were designed to work in main memory. Since current datasets often exceed the size of main memory, I/O between levels of memory is often the bottleneck in computation (consider that a disk access is about one million times slower than an access to primary memory). A naive management of external memory, e.g., with standard caching and virtual memory policies, may thus highly degrade the performance of algorithms Indeed, some computations are inherently non-local and require large amounts of I/O.

Out-of-core algorithms and data structures explicitly control the loading of data and their storage. Our goal here is to define an out-of-core data structure for a multi-resolution model that can store a large dataset and perform queries on it on a consumer-type hardware platform.

To this aim, we have first analyzed the intrinsic structure of multi-resolution models, and the nature of algorithms operating on them to define queries. Generally speaking, data in a multi-resolution model are spatially distributed and organized hierarchically, according to different levels of resolution. Algorithms to perform queries on a multi-resolution model are driven by a combination of spatial and resolution filters, and they are all based on a traversal of the different levels of resolution.

On the basis of our analysis, we have devoted a considerable amount of effort to define, implement, and experiment with a wide range of techniques to cluster the components of a multi-resolution model on disk pages, which follow the resolution hierarchy and/or apply a spatial partitioning and grouping. Experiments have been performed on terrains represented as TINs as well as on volumetric data.

The reminder of this paper is organized as follows. In Section 2, we discuss related work. In Section 3, we briefly describe the Multi-Tessellation. In Section 4, we analyze the operations performed by algorithms to answer queries on the out-of-core data structure for the MT. In Section 5, we analyze a set of clustering techniques and caching policies. In Section 6, we present an experimental comparison among the best clustering techniques. Concluding remarks are drawn in Section 7.

2. RELATED WORK

Several techniques have been proposed in the literature for out-of-core simplification of large-size triangle meshes. In [15], Lindstrom presents an out-of-core simplification algorithm, which is based on the vertex clustering technique first introduced in [26], combined with a quadric error metric to improve mesh quality. In [18], Lindstrom and Silva improve over the above result by proposing an out-of-core simplification algorithm, with reduced memory requirements and which improves on the selection of the representative vertex of the cluster. An extension of [15] has been proposed by Shaffer and Garland [28], who use a BSP tree as an adaptive domain partition of the input mesh to improve vertex clustering. Garland and Shaffer in [12] describe a two-step technique that performs vertex clustering as the first step, and iterative edge collapse as second one. El- Sana and Chiang [11] present an external-memory algorithm to support view-dependent simplification, based on segmenting a triangle mesh into sub-meshes that can be simplified independently, and then merged in a post- processing phase. In [5], Cignoni et al., propose an out-of-core incremental simplification algorithm, which subdivides the embedding space of a mesh through an octree and iteratively simplifies each leaf of the octree. Once leaves are simplified, they are merged and the process is iteratively repeated. Isenburg et al [14] present a simplification technique based on streaming a very large triangle mesh through main memory. A mesh is represented as an interleaved sequence of triangles and vertices, where each triangle is described by its three vertex indexes.

Fewer proposals exist for encoding, generating and querying a multi-resolution model in external memory. Lindstrom and Pascucci propose an out-of-core multi-resolution representation for large terrain models described by a regular nested partition of the domain into a hierarchy of right triangles [17]. Out-of-core multi-resolution data structures for irregular meshes have been proposed in [3, 4, 10, 11, 13, 16, 30]. These latter can be roughly subdivided into two classes depending on whether they subdivide the space where the mesh is embedded, or they cluster the nodes of the multi-resolution data structure. In all the available techniques, the underlying multi-resolution data structure is simply a binary tree, in which a children-parent pair describes an edge collapse operation.

Space-based techniques for terrain models have been proposed in [3, 13, 21]. In [13], the domain is subdivided into a uniform grid and separate vertex trees, one for each patch, are built and maintained. In [3], a hierarchy of nested right triangles is used to subdivide the domain, and an irregular triangle mesh, which may consist of a few thousands of triangles, is associated with each patch. In [21], a technique has been designed for modular multi-resolution modeling based on a subdivision of the domain into irregular cells. A multi-resolution model of the lines bounding the subdivision is built first, and then matching multi-resolution models, one for each cell, are generated in a second step.

In [4], the approach proposed by the same authors in [3] is extended to triangulated surfaces by using a decomposition of the embedding space into a tetrahedral hierarchy, and associating an irregular triangle mesh with each tetrahedral cell. In [30], a triangle mesh is partitioned into a hierarchy of mesh clusters and a progressive mesh is encoded inside each cluster. The method in [16] uses an octree as a multi-resolution out- of-core data structure, which is built through iterative vertex clustering in external memory [18]. In [11], the view-dependent tree built during out- of-core mesh simplification is organized into meta-nodes, thus giving rise to a B-tree-like out-of-core data structure. A similar clustering strategy is applied in [10] by organizing a hierarchy of half-edge collapses into blocks. The construction of the data structure has to be performed in main memory.

All these data structures (with exception of [21]) have been specifically targeted to view-depending rendering. The out-of-core representations described in [10, 11, 13, 16, 30] can be used only for multi-resolution models built through a specific simplification technique (edge collapse or vertex clustering). The representations in [3, 4], although efficient in rendering, keep the multi-resolution data structure in core, and store the triangulation of the patches in external memory. In this way, the granularity of the multi- resolution representation is quite coarse, and not sufficient for a fine spatial analysis in specific regions of interest.

3. THE MULTI-TESSELLATION

The Multi-Tessellation (MT) is a model to represent multi-resolution simplicial meshes in arbitrary dimensions [9, 25]. The MT provides a general framework for multi-resolution models. So, the following concepts apply to most other models proposed in the literature.

An MT is composed of a *base* mesh at a coarse resolution plus a partially ordered set of *modifications* that can be applied to locally refine the base mesh. In general, a *modification u* consists of two sets of cells u^- and u^+, representing an object portion at a lower and a higher LOD, respectively

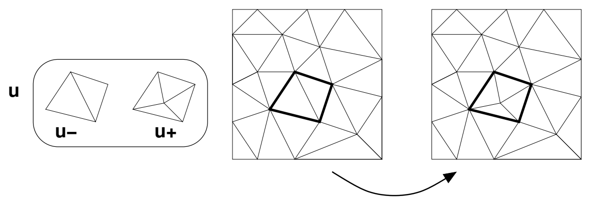


Figure 1: A modification u and its application to locally refine a mesh.

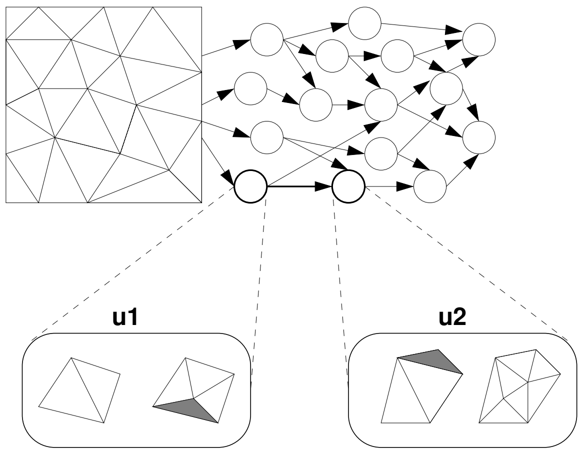


Figure 2: The DAG representing an MT: two nodes connected by an arc are highlighted, and the corresponding modifications are shown. The dark triangle belongs to $u_2^- \cap u_1^+$.

(see Figure 1). A modification can be applied in either way to locally refine or coarsen a mesh.

A *direct dependency* relation is defined among the modifications. A modification u_2 depends on another modification u_1 if u_2 removes some cells inserted by u_1, i.e., if the intersection $u_2^- \cap u_1^+$ is not empty. The transitive closure of the dependency relation is a partial order, which can be represented as a Directed Acyclic Graph (DAG) (see Figure 2).

Any subset of modifications, which is closed with respect to the partial order, can be applied to the base mesh in any total order extending the partial one, and gives a mesh at variable LOD. Any closed set can be seen as a cut in the DAG, and a mesh is extracted by adjusting the position of the cut in the DAG according to query parameters: the cut is moved upwards to coarsen the mesh and downwards to refine it. Spatial queries that we need to perform on a multi-resolution model are essentially aimed at extracting simplified meshes which satisfy some application-dependent requirement. Such queries can be classified into queries at a *uniform* LOD and queries at a *variable* LOD. Uniform LOD queries require a uniform resolution on the whole domain, while variable LOD ones require high resolution inside a Region Of Interest (ROI) and arbitrary resolution outside. Examples of ROI are: a single point, a box, or the set of triangles/tetrahedra intersected by a specific iso-contour. View-dependent queries are also examples of variable-resolution queries, in which the ROI is a view frustum, and resolution smoothly decreases according to the distance from the observer.

LOD queries are all based on a basic algorithm, called *selective refinement*, which performs a suitable DAG traversal of a subset of the modifications of an MT. The traversal is not a standard graph visit, but the order used is peculiar to the query, being guided by the dependency relation and by the query parameters.

4. I/O OPERATIONS IN SELECTIVE REFINEMENT

We consider here an implementation of a selective refinement algorithm which is based on a top-down graph traversal and on a data structure that explicitly encodes the geometry of cells and the DAG, such as that used in [20], and we analyze its cost in terms of I/O operations. The data structure basically contains one record per node of the DAG. In this record, we store the geometry of the update associated with the node, its attributes, and the links to its parents and children. Such links will thus refer to other records that, depending on how the DAG is clustered, either will be found in primary memory, or shall be loaded from disk. One I/O operation consists of fetching/loading the record of a given node. We also assume that the currently extracted mesh is stored in main memory. This assumption is reasonable since this mesh usually forms only a small fraction of the mesh at full resolution.

The traversal starts from the base mesh and is guided by the dependency relation and by a selection criterion τ which is application-dependent. Criterion τ takes into account both the desired LOD and the ROI.

Procedure SELECTIVE_REFINEMENT is invoked for the first time on the modification corresponding to the root of the DAG describing the MT, which encodes the base mesh. Next, it traverses the DAG and recursively expands children of the current node, if this is required by the selection criterion τ. Also, all nodes that are ancestors of u are recursively expanded before expanding u. Recursive procedure FORCE_REFINE performs this latter step.

```
Procedure SELECTIVE_REFINEMENT($\tau$, $\Sigma$, $u$)
begin
/* add the modification and its missing ancestors */
    FORCE_REFINE($\Sigma$, $u$)
    for each $u'$ children of $u$ do
        if not IS_ACCEPTABLE($u'$, $\tau$) then
/* start recursion on required children */
            SELECTIVE_REFINEMENT($\tau$, $\Sigma$, $u'$)
end SELECTIVE_REFINEMENT

Procedure FORCE_REFINE($\Sigma$, $u$)
/* Expand $u$ and all its ancestors */
begin
    if not IS_REFINED($u$) then
        $P$ = parents of $u$
        for each $u'$ in $P$ do
            FORCE_REFINE ($\Sigma$, $u'$);
        endfor
        REFINE($\Sigma$, $u$)
    endif
end FORCE_REFINE
```

We first need an I/O operation to load the base mesh, after which the algorithm starts the traversal. Another I/O operation occurs each time primitive IS_ACCEPTABLE is called to check whether or not a node must be expanded. In some cases, it is possible to avoid such I/O operation by storing in the record of a node also some attributes of its parents and children (e.g., error threshold, bounding box, minimum and maximum field value). In this way, only nodes that need to be expanded will be loaded. This comes, of course, at the expense of some additional storage.

Procedure FORCE_REFINE expands a node. Before performing expansion, it checks if all modifications the node depends on have been expanded and, if not, it expands them recursively. This implies one I/O operation per recursive call to FORCE_REFINE.

In a data structure that is designed just for primary memory, access to secondary memory is handled automatically by virtual memory mechanisms. In this case, each read operation loads only a few bytes (usually less than one KByte), which results in a tremendous waste of time due to disk seek time and latency. In order to reduce the number of I/O operations from disk, we need to group modifications into clusters, and store each cluster in a disk block. For a given block size, I/O operations are minimized if all nodes in a given block are visited before another block is loaded. However, since selective refinement algorithms visit the DAG and perform modifications in a dynamically selected order, it is impossible to forecast a storing order which will be the best for all queries. We will thus look for a clustering technique that gives a good performance on average.

An improvement can be achieved by introducing a cluster cache, which is a portion of main memory storing a subset of the clusters already loaded. When the selective refinement algorithm tries to load a modification, the out-of-core engine looks for it in the cache. If it is missing, then the corresponding cluster is loaded and stored in the cache for further use. If the cache is full, then we need to adopt a cache replacement policy.

5. CLUSTERING TECHNIQUES

We have developed a simulation system on top of the MT library [20]. Such system organizes modifications into clusters, according to different policies. By using the system, we have estimated the performance of selective refinement algorithms, by counting the number of I/O operations from secondary memory. In this way, we have compared several clustering and caching techniques in both 2D and 3D settings.

An analysis of LOD queries, of the parameters which determine the shape of the MT, and of the DAG traversal strategies used by the selective refinement algorithms has suggested us two classes of clustering strategies, one based on sorting the modifications in the MT (some derived from strategies for DAG traversal), and the other based on spatial grouping. Uniform LOD queries have suggested partitioning the MT into layers, where each successive layer should guarantee a reduction in the approximation error, defined as the error with which a modification approximates the full-resolution model. Variable-LOD queries focus on a subset of the domain at high resolution, and thus they have suggested grouping the modifications according to a space clustering technique, which takes into account their position in space. We need to take three important elements into consideration, when designing a clustering strategy: a spatial criterion, a coherence in the dependency relation, and the approximation error.

In Subsection 5.1, we discuss clustering techniques based on grouping a totally ordered sequence of the modifications in an MT, while in Subsection 5.2, we present clustering techniques based on spatial grouping. In Subsection 5.3, we discuss combinations of the previous ones. Finally, in Subsection 5.4, we discuss a caching mechanism for clusters of modifications.

5.1 Clustering based on a sorted sequence

In this subsection, we describe clustering techniques applied to a sequence of the modifications in the MT, obtained by either sorting them according to some parameter (approximation error, maximum, minimum or average distance from the root), or by performing a suitable DAG traversal.

Computation of clusters based on a sequence of modifications requires two steps:

- The sequence is computed either by sorting or by DAG traversal.
- Starting from the beginning of the resulting sequence, modifications are stored in disk pages until each disk page is completely full.

The space utilization of these techniques is optimal. If the MT model consists of N modifications and B is the size of a disk page, then $\lceil \frac{N}{B} \rceil$ disk blocks are used if the number of disk $D = 1$, or $\lceil \frac{N}{B \cdot D} \rceil$ stripes in a RAID system.

We have considered the following parameters to sort the modifications of an MT:

- *Approximation error* (`Err`): error with which a modification locally approximates the full- resolution mesh.
- *Layer* (`Lyr`): length of the shortest path from the root.
- *Level* (`Lev`): length of the longest path from the root.
- *Distance* (`Ly2`): average length of a path from the root.

We have generated totally ordered sequence of modifications according to the following DAG traversals:

- *Depth-first traversal* (`DFS`).
- *Breadth-first traversal* (`BFS`).
- *Graph visit - depth-first* (`GrD`): similar to a depth-first traversal, but before adding a modification u, the traversal algorithm checks if the ancestors of u have been already applied to the current mesh. If not, then they are added in the same order in which they are visited by procedure FORCE_REFINE. A depth-first selective refinement algorithm that visits the whole DAG is simulated.
- *Graph visit - Breadth-first* (`GrB`): a variation of `GrD`, but based on a breadth-first traversal.

Note that `Lyr` and `BFS` give similar results. The only difference is that modifications at the same minimal distance from the root are considered in arbitrary order when sorting according to `Lyr`, while, in `BFS` traversal, they are considered in the order in which they appear in the DAG encoding structure.

For the sake of comparison, we have also implemented two additional criteria:

- *Random* (`Rnd`): a sequence of modifications is generated randomly.

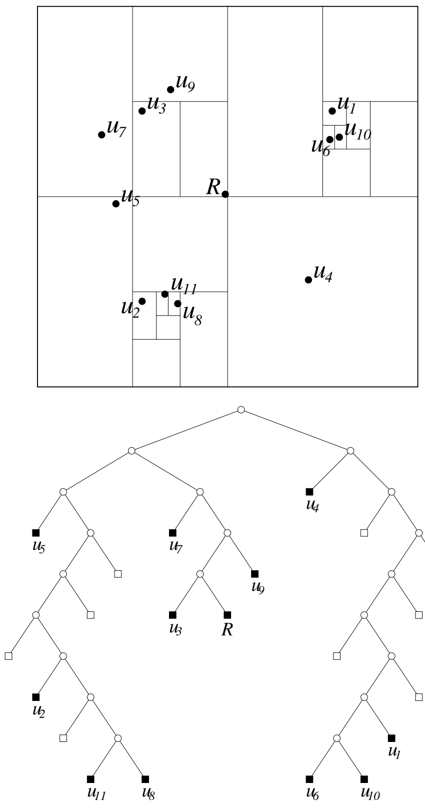


Figure 3: A PR k-d tree with the resulting partition of space and its tree representation.

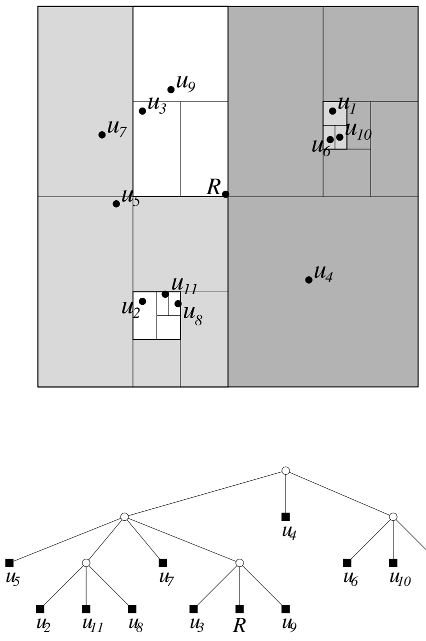


Figure 4: A PK PR k-d tree with $k = 3$ corresponding to the PR k-d tree shown in Figure 3: the resulting partition of space and its tree representation. Bold lines indicate the boundaries of blocks corresponding to internal nodes. Gray levels represent the layer in the tree.

- *Sequential* (`Seq`): we use the same sequence generated by a top-down simplification procedure, or, in we reverse the sequence in the case of bottom-up simplification strategies. Usually the simplification process is error-driven, but there is no guarantee that the approximation error associated with modifications decreases monotonically

A clustering technique, which simply sorts modifications and fills each disk block with a contiguous set of sorted modifications, does not introduce any overhead in storage space. A disk block is usually at least 4 KBytes. There is no upper bound on the block size, but a block that spans an entire disk track best amortizes latency, and, thus, there is no need to create larger blocks. This suggests an upper bound which varies between 50 to 200 Kbytes.

If we consider an MT for terrain data (2D MT), each modification requires on average 112 bytes, while for an MT for volume data (3D MT), the cost increases to 325 bytes. In the case of a 2D (3D) MT, this results in a block transfer size B ranging between 36 and 457-1828 (12 and 157-630) modifications for a single disk architecture. The number of modifications per block can decrease if we decide to store additional information, such as bounding box, or approximation error, in order to speed up the execution of primitive IS_ACCEPTABLE, as discussed in Section 4.

5.2 Spatial clustering of an MT

We have considered two classes of spatial grouping strategies. The methods in the first class are based on R*-trees. We have used the implementation of R*-trees in [1]. We associate a minimum axes-aligned bounding box with each modification (e.g., a rectangle for 2D MTs and a parallelepiped in the 3D case). If the query considers an additional parameter, such as the approximation error, then we have a higher-dimensional bounding box, in which one of the dimension is the additional parameter.

In our prototype, we have implemented three and four dimensional R*-trees, based on Cartesian coordinates and, optionally, on another parameter. Specifically, we have implemented the following techniques, based on R*- tree: Cartesian coordinates (it will be labeled `RTree`); coordinate values plus approximation error `RTreeErr`; coordinate values plus minimum distance `RTreeLyr`; coordinate values plus average distance `RTreeLy2` and coordinate values plus maximum distance `RTreeLev`.

The other class of strategies uses space subdivision combined with a grouping mechanism. Since the modifications in the MT contain a small number of cells (4 triangles on average in a 2D MT, and 13 tetrahedra on average in a 3D MT), we associate each modification to its centroid. Moreover, since an MT is a dimension-independent model, these centroids can be points in d-dimensional space. Thus, we have selected a space partitioning technique that does not depend on the dimension of the underlying space, i.e., a *Point Region k-d tree* (PR k-d tree) [23].

A PR k-d tree is a spatial index based on the recursive subdivision of the domain containing the data points (in our case, the centroids). At each step the current block is halved. The halving process cycles through different dimensions in a predefined and constant order. In the construction of the PR k-d tree, we have considered the dimension of the space in which the MT is embedded (2D or 3D in most cases). For a two-dimensional space, we split at the first level along x, at the second level along y, and then the process restarts from x. We have performed space partitioning not only by using the centroids, but also by using these latter combined with other parameters, like the approximation error, the maximum, minimum or average distance from the root. For instance, if we consider a 2D MT and we want to use an additional parameter, then we construct a three-dimensional PR k-d tree by considering the additional parameter as the first halving criterion and then testing all the coordinates in the standard order.

Figure 3 shows the space partition generated by a PR k-d tree in two dimensions together with its corresponding tree. White squares represent empty leaves. It is easy to notice that a PR k-d tree can result in an unbalanced tree if data are not uniformly distributed. This fact is well known for all point-region trees [27]. However, the problem can be partially overcome by the subsequent grouping step.

One way to group the full leaves of a PR k-d tree on disk is by associating location codes with them, and storing these codes in a B-tree [27]. This technique has the disadvantage that the boundary of the domain covered by leaves stored in the same node of the B-tree can be arbitrary complex. In general, the domain might be not connected and, thus, we could loose spatial coherence.

An interesting alternative consists of using a PK-tree as a grouping mechanism. Originally proposed in [29], a PK-tree is characterized by a parameter i, called the *instantiation value*, which is the minimum number of nodes in the tree grouped in a cluster. A PK tree is built recursivley by applying a bottom-up grouping process to the nodes of a tree T. Nodes belonging to T are grouped into clusters until the minimum occupancy i has been reached. During the grouping process empty leaves in T are removed. Since the PR k-d tree is a binary tree, an interesting property of the PR k-d tree grouped according to the PK-tree node aggregation policy, that we call a *PK PR k-d tree*, is that each node has a minimum of i children and a maximum of $2 \cdot (i-1)$, regardless of the dimensionality of the space [29]. This guarantees that disk blocks are at least half-full.

Figure 4 shows the PK PR k-d tree corresponding to the PR k-d tree presented in Figure 3 with instantiation value $i = 3$.

We have tested several clustering policies based on a PK PR k-d tree. Specifically, we have implemented techniques based on the values of the Cartesian coordinates of the centroid (it will be labeled `PKTree`); on the coordinate values plus approximation error `PKTreeErr`; on the coordinate values plus minimum distance from the root `PKTreeLyr`;on the coordinate values plus average distance `PKTreeLy2`; and on the coordinate values plus maximum distance `PKTreeLev`.

5.3 Combining sorting with spatial criteria

We have also combined the strategies described in the previous two subsections together. We have developed a technique that interleaves the effect of a sorting rule with that of a space partitioning rule similar to that applied in a PR k- d tree, resulting in a technique which resembles priority search trees [22]. We consider a sequence of modifications in an MT generated by any sorting criterion, or any DAG traversal, as described in Subsection 5.1. We store in a disk page the first k modifications in the sequence, and then we subdivide the remaining modifications according to the selected space partitioning criterion. This process is applied recursively to each subset of modifications.

We perform a recursive subdivision of the domain containing the centroids associated with the modifications, or of the domain defined by the centroids plus an additional parameter, as in the PR k-d tree, since we subdivide the domain in half. The halving process cycles through different dimensions in a predefined and constant order. The difference, compared witg a PR k-d tree, is that each node of the tree can store up to k modifications. The resulting data structure is a binary tree in which each node corresponds to a disk block.

Our algorithm can be summarized as follows:

1. Compute a sequence of modifications. Let L_S be such a sequence.
2. Let B be the first block.
3. Remove the first k modifications from L_S and store them with the current disk block B.
4. If L_S is not empty, then split L_S into two sublists L'_S and L''_S according to the halving criterion adopted by the PR k-d tree;
5. Apply 3 recursively on L'_S and L''_S.

This technique, that combines sorting with space partitioning, has been applied by using all the sorting criteria that we have developed which are described in Subsection 5.1, and the five halving rules adopted for the PR k-d tree, described in Subsection 5.2.

5.4 Caching policy

During selective refinement, we need to load clusters of modifications in main memory. Sometimes a cluster can be referenced more than once, especially if we are building large clusters. This suggests using a buffer to store the clusters just loaded from disk. A common graphics workstation has between 1 and 4 GBytes of main memory (this means it can hold between 3 and 12 million of modifications). Even if a huge multi-resolution model can be several times larger, it is useful to use part of the main memory as a cache for clusters of modifications. In our prototype it is possible to have a cluster cache, whose size can be defined by a user, and which can be dynamically redefined.

We have implemented two different cache replacement policies: oldest and least-recently-used. According to our tests, the least-recently-used policy seems to offer better results. Every test reported in Section 6 is based on this latter cache replacement policy.

Figure 5 shows the effect of clustering based on an ordered sequence of modifications, and of a small cache that can fit up to 2% of the modification in a multi-resolution trianglular model. We show tests have been performed on a terrain data set, the Mount Marcy dataset. This data set at full resolution contains 1,442,401 points and 2,880,000 triangles. It has a storage cost of 445 MBs. The same experiments on other 2D and 3D data sets show similar results [6]. The graphs report the number of I/O operations for an MT

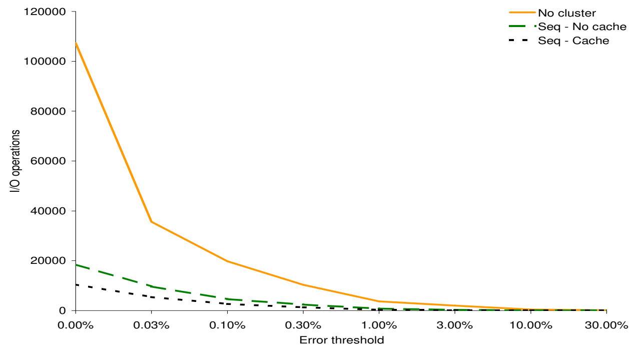


Figure 5: Number of I/O operations for a 2D MT stored on external memory. We used the explicit MT data structure. Disk accesses without clustering modifications (continuous line), with sequential clustering (dashed line), with clustering and caching (dotted line) for a query at variable LOD, based on a region of interest.

stored in external memory as a function of the error threshold. The solid line represents disk accesses without clustering modifications, the dashed one represents disk accesses with clustering and no caching, the dotted line represents disk accesses with clustering and an additional cache.

6. ANALYSIS AND COMPARISONS

In this Section, we present the results of comparisons among the clustering techniques we have described in the previous section. Specifically, here we report results for the ones that performed best in their category, which include: `Err`, `BFS`, `DFS`, `GrB`, `GrD`, space partitioning on an approximation error and three coordinate values, and grouping with a PK tree (`PK tree`) and with an R*-tree (`R* tree`). We also present results obtained with a clustering technique that combines the effects of sorting and space partitioning: space partitioning on average distance and three coordinate values, and `Ly2` sorting (`SpaLy2`). Each graph represents the ratio between the number of disk accesses required to perform a set of selective refinement queries and the number $n = N/B$ of disk blocks required to store the model. Recall that N is the size of the data, i.e. the number of modifications associated with an MT, and B is the size of a disk page, i.e. the number of modifications that can fit in a disk page. This permits us to compare results, independently of model size.

We show results on a 2D data set representing the Grand Canyon in Arizona. This data set consists of 4,194,304 points and the mesh at full resolution contains 8,380,418 triangles. It requires 1389 MBs of storage. We have obtained similar results with other 2D and 3D datasets. A complete set of experiments can be found in [6]. The cache size is set at 1% of the size of the model, in order to enhance differences among clustering techniques.

Figure 6 (a) shows the results for selective refinement at uniform resolution on the Grand Canyon dataset. Sorting based on the error threshold (`Err`) has the worst performance. `DFS` performs quite well, and `GrD`, which performs a depth-first traversal with inclusion of ancestors, presents even better results. In this case, each of the space partitioning methods coupled with sorting rules behaves in nearly the same way. The difference between the different techniques is less than 5%. This behavior is likely to be a result of the regular structure of the Grand Canyon dataset. The best results are obtained by a breadth- first traversal with inclusion of ancestors (`GrB`).

This result is somehow expected in the case of selective refinement queries at a uniform resolution, since `GrB` clusters modifications in the same order of a selective refinement traversal at uniform resolution. It is much more interesting to note that `GrB`, even with queries at variable resolution, offers excellent results. Results on selective refinement at variable resolution with a field value as a focus set are shown in Figure 6 (b). In this case, considerations on the efficiency of clustering techniques are comparable to the puniform case. Results for selective refinement at variable resolution with a box as focus set are shown in Figure 6(c). In this case, `GrD` is really close to `GrB`, and all results obtained with clustering techniques based on space clustering are close to each other.

Note that, even for a really small cache, that is about 1% of the size of the whole model, a clustering technique based on `GrB` exhibits a small overhead, compared to loading of the whole model.

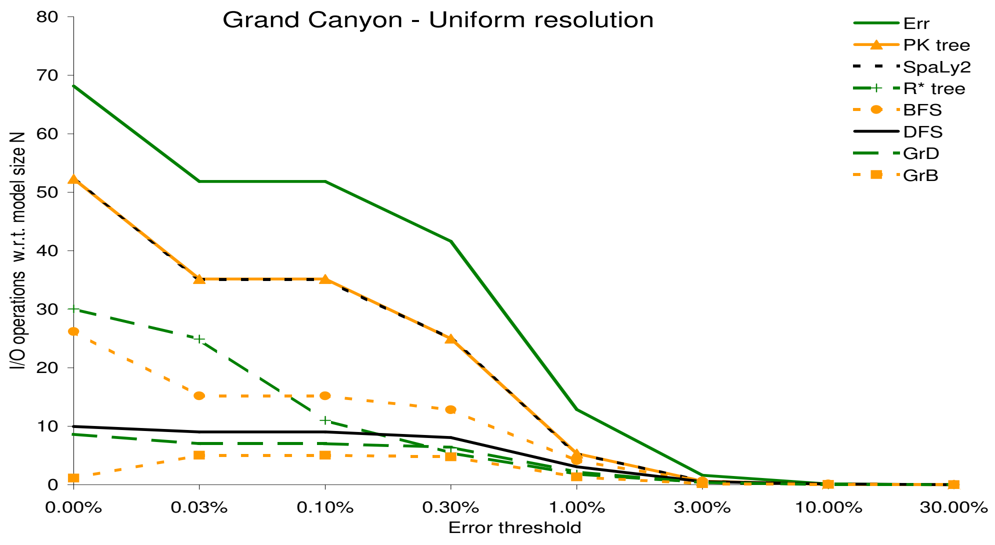
`GrB` is clearly the best performing technique implemented in our prototype. Its overhead is negligible, and it scales well based on the cache size. Its only drawback is that, in order to be built, it requires a traversal of the DAG, which mimics the selective refinement algorithm and must performed out-of-core.

7. CONCLUDING REMARKS

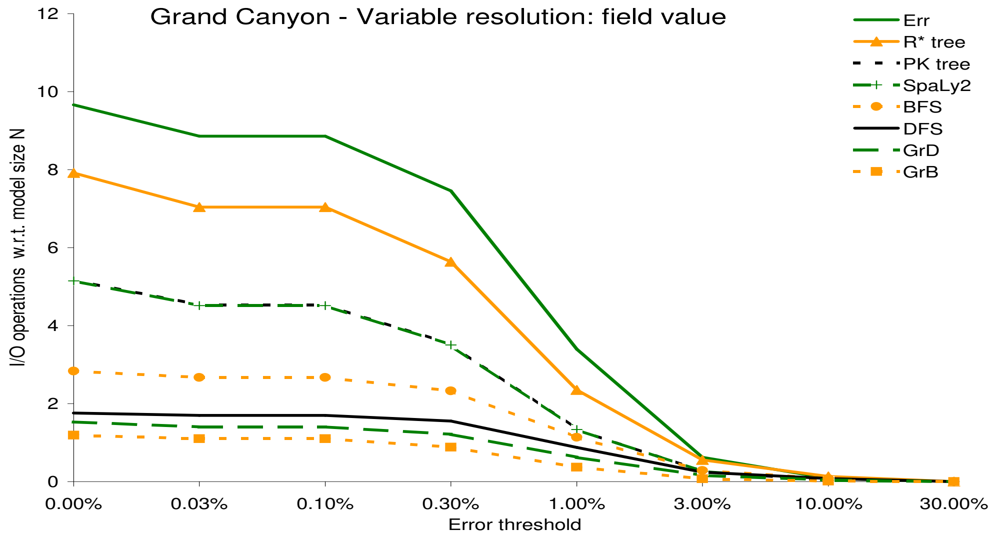
We have investigated out-of core approaches for encoding and manipulating multi-resolution models for two- and three-dimensional scalar fields. The major issue here has been developing and experimenting with effective techniques for clustering the modifications in the multi-resolution model. We have designed and developed a large variety of clustering techniques (actually, 65), and we have experimented with them in connection with the major selective refinement queries, as well as with both terrain and geophysical data sets. The clustering algorithms, that we have developed, can be coupled either with a compact, or with an explicit representation of the multi-resolution model, and form the basis for the development of a system for out-of-core multi-resolution modeling.

We are currently developing an out-of-core multi-resolution modeling system for scalar fields based on the MT. The data structure on which we will perform the selective refinement queries is based on an explicit encoding of the modifications as sets of simplices (triangles or tetrahedra), and on the clustering of the modifications according to the `GrB` traversal technique. In order to build this data structure, we consider the coarse mesh and the sequence of modifications produced by an out-of-core simplification algorithm. We have designed an out-of-core algorithm which extracts the dependency relation and produces an intermediate data structure, an out-of-core algorithm for both traversing such a data structure and for generating the clustered representation, and an out-of-core algorithm for selective refinement (that we have already implemented in our simulation environment) [6].

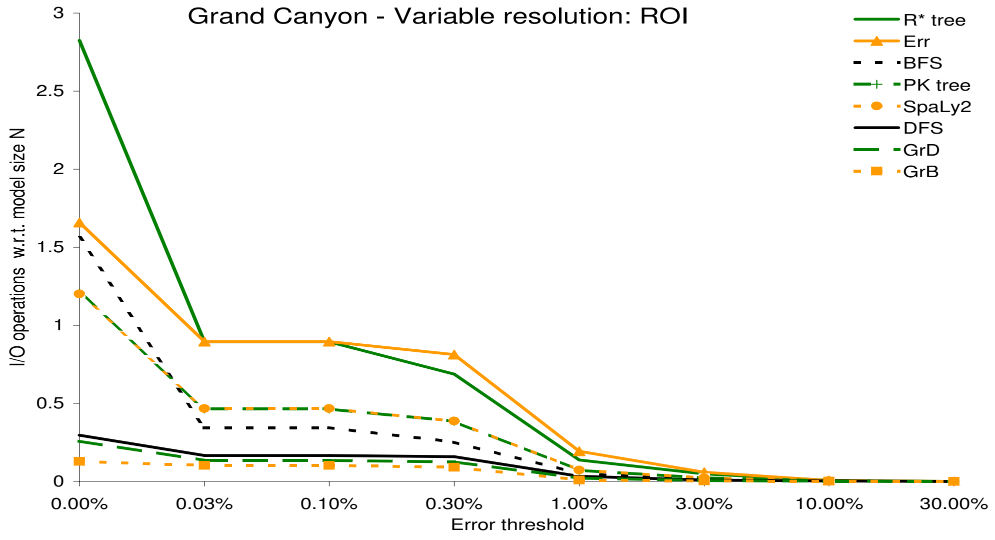
At a later stage, we also plan to develop specific implementations of the out-of-core modeling system based on im-



(a)



(b)



(c)

Figure 6: Number of I/O operations for performing queries at uniform resolution (a) or variable resolution with a field value as focus set (b) or variable resolution in a box (c) on the Grand Canyon dataset.

plicit representations of the modifications, in particular by using implicit procedural encodings of half- edge collapse on triangle [24] and tetrahedral meshes [7].

There still remain important, yet relatively unexplored issues concerning a multi-thread implementation, in which the I/O process is a thread, and the selective refinement is a separate one. This can be used to implement a pre-fetching technique. Clusters that are likely to be required by the selective refinement algorithm can be pre-loaded by the I/O thread, with a reduction in the total execution time. We plan to consider pre-fetching in our future work.

Acknowledgments

This work has been partially supported by the projects funded by Italian Ministry of Education, University and Research (MIUR) on "Representation and Management of Spatial Data in the Web (SPADA-WEB)" and on "Low-cost sensors and geometric techniques for scanning and modeling solid objects (LIMA3D)", by the European Network of Excellence AIM@SHAPE under contract number 506766, by the National Science Foundation under Grant EIA-00- 91474, as well as by Microsoft Research.

8. REFERENCES

[1] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In *Proceedings ACM SIGMOD Conference*, pages 322–331, Atlantic City, NJ, June 1990. ACM Press.

[2] P. Cignoni, L. De Floriani, P. Magillo, E. Puppo, and R. Scopigno. Selective refinement queries for volume visualization of unstructured tetrahedral meshes. *IEEE Transactions on Visualization and Computer Graphics*, 10(1):29–45, January-February 2004.

[3] P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio, and R. Scopigno. Bdam: Batched dynamic adaptive meshes for high performance terrain visualization. *Computer Graphics Forum*, 22(3):505–514, September 2003.

[4] P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio, and R. Scopigno. Adaptive TetraPuzzles – efficient out-of-core construction and visualization of gigantic polygonal models. *ACM Transactions on Graphics*, 23(3), August 2004.

[5] P. Cignoni, C. Montani, C. Rocchini, and R. Scopigno. External memory management and simplification of huge meshes. *IEEE Transactions on Visualization and Computer Graphics*, 9(4):525–537, November 2003.

[6] E. Danovaro. *Multi-resolution Modeling of Discrete Scalar Fields.* PhD thesis, Dept. of Computer and Information Sciences, University of Genova (Italy), 2005.

[7] E. Danovaro, L. De Floriani, P. Magillo, E. Puppo, D. Sobrero, and N. Sokolovsky. The half-edge tree: A compact data structure for level-of-detail tetrahedral meshes. In *Proceeding of the International Conference on Shape Modeling*, June, 15-17 2005.

[8] L. De Floriani, P. Magillo, and E. Puppo. VARIANT: a system for terrain modeling at variable resolution. *Geoinformatica*, 4(3):287–315, 2000.

[9] L. De Floriani, E. Puppo, and P. Magillo. A formal approach to multi-resolution modeling. In W. Strasser, R. Klein, and R. Rau, editors, *Geometric Modeling: Theory and Practice*, pages 302–323. Springer-Verlag, 1997.

[10] C. DeCoro and R. Pajarola. XFastMesh: Fast view-dependent meshing from external memory. In *Proceedings IEEE Visualization 2002*, pages 263–270, Boston, MA, October 2002. IEEE Computer Society.

[11] J. El-Sana and Y.-J. Chiang. External memory view-dependent simplification. *Computer Graphics Forum*, 19(3):139–150, August 2000.

[12] M. Garland and E. Shaffer. A multiphase approach to efficient surface simplification. In *Proceedings Visualization '02*, pages 117–124, Boston, Massachusetts, October 2002. IEEE Computer Society.

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

[14] M. Isenburg, P. Lindstrom, S. Gumhold, and J. Snoeyink. Large mesh simplification using processing sequences. In *Proceedings Visualization 2003 Conference*, pages 465–472. IEEE Computer Society Press, October 2003.

[15] P. Lindstrom. Out-of-core simplification of large polygonal models. In *ACM Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH'00)*, pages 259–270, New Orleans, LA, USA, July 2000. ACM Press.

[16] P. Lindstrom. Out-of-core construction and visualization of multi-resolution surfaces. In *ACM SIGGRAPH 2003 Symposium on Interactive 3D Graphics*, pages 93–102, Monterey, California, April 2003. ACM Press.

[17] P. Lindstrom and V. Pascucci. Terrain simplification simplified: a general framework for view-dependent out-of-core visualization. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):239–254, 2002.

[18] P. Lindstrom and C. T. Silva. A memory insensitive technique for large model simplification. In *IEEE Visualization 2001*, pages 121–126, 550. IEEE Computer Society, October 2001.

[19] D. Luebke, M. Reddy, J. Cohen, A. Varshney, B. Watson, and R. Huebner. *Level of Detail for 3D Graphics.* Morgan-Kaufmann, San Francisco, 2002.

[20] P. Magillo. The MT library. http://www.disi.unige.it/person/MagilloP/MT/.

[21] P. Magillo and V. Bertocci. Managing large terrain data sets with a multiresolution structure. In A. M. Tjoa, R. R. Wagner, and Ala Al-Zobaidie, editors, *Proc. 11th Int. Conf. on Database and Expert Systems Applications*, pages 894–898, Greenwich, UK, September 2000. IEEE.

[22] E. M. McCreight. Priority search trees. *SIAM Journal on Computing*, 14(2):257–276, May 1985.

[23] J. A. Orenstein. Multidimensional tries used for associative searching. *Information Processing Letters*, 14(4):150–157, 1982.

[24] R. Pajarola. FastMesh: efficient view-dependent meshing. In *Proceedings Pacific Graphics 2001*, pages 22–30. IEEE Computer Society, 2001.

[25] E. Puppo. Variable resolution terrain surfaces. In *Proceedings Eight Canadian Conference on Computational Geometry, Ottawa, Canada*, pages 202–210, August 12-15 1996. Extended version appeared as Variable Resolution Triangulations, 1998, *Computational Geometry Theory and Applications*, 11(3-4):219–238.

[26] J. Rossignac and P. Borrel. Multi-resolution 3D approximations for rendering complex scenes. In B. Falcidieno and T. L. Kunii, editors, *Modeling in Computer Graphics*, pages 455–465. Springer-Verlag, October 17–18 1993.

[27] H. Samet. *Foundations of Multi-Dimensional Data Structures*. Morgan-Kaufmann, to appear, 2005.

[28] E. Shaffer and M. Garland. Efficient adaptive simplification of massive meshes. In *Proceedings IEEE Visualization '01*. IEEE Computer Society, October 2001.

[29] W. Wang, J. Yang, and R. Muntz. PK-tree: a spatial index structure for high dimensional point data. In K. Tanaka and S. Ghandeharizadeh, editors, *Proceedings 5th International Conference on Foundations of Data Organization and Algorithms (FODO)*, pages 27–36, Kobe, Japan, November 1998.

[30] S.E. Yoon, B. Salomon, R. Gayle, and D. Manocha. Quick-VDR: Out-of-core view-dependent rendering of gigantic models. *IEEE Transactions on Visualization and Computer Graphics*, 11(4):369–382, 2005.