2

Out-of-core Multi-resolution Terrain Modeling

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

[1] Department of Computer and Information Science,
University of Genoa - Via Dodecaneso, 35, 16146 Genoa, Italy
{danovaro,deflo,puppo}@disi.unige.it
[2] Computer Science Dept., Center for Automation Research, Institute for Advanced Computer Studies
University of Maryland - College Park, Maryland 20742
{danovaro,deflo,hjs}@umiacs.umd.edu

2.1 Introduction

In this chapter we discuss issues about Level Of Detail (LOD) representations for digital terrain models and, especially, we describe how to deal with very large terrain datasets through out-of-core techniques that explicitly manage I/O operations between levels of memory. LOD modeling in the related context of geographical maps is discussed in Chapters 3 and 4.

A dataset describing a terrain consists of a set of elevation measurements taken at a finite number of locations over a planar or a spherical domain. In a digital terrain model, elevation is extended to the whole domain of interest by averaging or interpolating the available measurements. Of course, the resulting model is affected by some approximation error and, in general, the higher the density of the samples, the smaller the error. The same arguments can be used for more general two-dimensional scalar, or vector fields (e.g., generated by simulation), defined over a manifold domain, and measured through some sampling process.

Available terrain datasets are becoming larger and larger, and processing them at their full resolution often exhibits prohibitive computational costs, even for high-end workstations. Simplification algorithms and multi-resolution models proposed in the literature may improve efficiency, by adapting resolution on-the-fly, according to the needs of a specific application [32]. Data at high resolution are preprocessed once in order to build a multi-resolution model, which can be queried on-line by the application. The multi-resolution model acts as a black box that provides simplified representations, where resolution is focused on the region of interest, and at the level of detail required by the application. A simplified representation is generally affected by some approximation error, which is usually associated with either the vertices, or the cells of the simplified mesh.

Since current datasets often exceed the size of main memory, I/O operations between levels of memory are often the bottleneck in computation. A disk access is

about one million times slower than an access to main memory. A naive management of external memory, e.g., with standard caching and virtual memory policies, may thus highly degrades the. Indeed, some computations are inherently non-local and require large amounts of I/O operations. Out-of-core algorithms and data structures explicitly control how data are loaded and how they are stored. Here, we review methods and models proposed in the literature for simplification and multi-resolution management of huge datasets that cannot be handled in main memory. We consider methods that are suitable to manage terrain data, some of which have been developed for more general kinds of data (e.g., triangle meshes describing the boundary of 3D objects).

The rest of this chapter is organized as follows. In Section 2.2, we introduce the necessary background about digital terrain models, focusing our attention on Triangulated Irregular Networks (TINs). In Section 2.3, we review out-of-core techniques for simplification of triangle meshes and discuss their application to terrain data in order to produce approximated representations. In Section 2.4, we review out-of-core multi-resolution models specific for regularly distributed data, while, in Section 2.5, we describe more general out-of-core multi-resolution models that can manage irregularly distributed data. In Section 2.6, we draw some concluding remarks and we discuss open research issues including extensions to out-of-core simplification and multi-resolution modeling of scalar fields in three and higher dimensions, to deal, for instance, with geological data.

2.2 Digital Terrain Models

There exist two major classes of digital terrain models, namely Digital Elevation Models (DEMs) and Triangulated Irregular Networks (TINs).

A DEM consists of a regular tiling of a planar domain into rectangular cells, called *pixels* with elevation values attached to pixels, which provide a piecewise-constant approximation of terrain surface. DEMs are most often defined over rectangular domains encoded just as two-dimensional arrays of elevation values, which are easily geo-referenced by defining an origin, an orientation and the extent of the cells in the tiling.

A TIN consists of a subdivision of the domain into triangles (i.e., a triangle mesh). Triangles do not overlap, and any two triangles are either disjoint or may share exactly either one vertex or one edge. Elevation samples are attached to vertices of triangles, thus each triangle in the mesh corresponds to a triangular terrain patch interpolating measured elevation at its vertices. Linear interpolation is usually adopted at each triangle, hence the resulting surface is piecewise-linear. More sophisticated interpolation models can also be used, without changing the underlying domain subdivision. Data structures used to encode TINs are more sophisticated and expensive than those used for DEMs, but TINs have the advantage of being adaptive. High density of samples may be adopted over more irregular portions of a terrain, while other portions, that are relatively flat, may be represented at the same accuracy with a small number of samples.

Multi-resolution DEMs usually consist of a collection of grids at different resolutions. Well-known multi-resolution representations for DEMs are provided by region quadtrees or pyramids [41]. One recent example in computer graphics is provided by the work by Losasso and Hoppe [31], where fast rendering of a large terrain is supported through a compressed pyramid of images, which can be handled in main memory. Fast decompression algorithms and an effective use of a GPU (Graphics Processing Unit) allow traversing the pyramid, by extracting data at the appropriate resolution, and rendering them on the fly.

In most cases, however, multi-resolution models require using adaptive representations in the form of TINs, even when full resolution data come in the form of a DEM. For this reason, we consider here techniques that process and produce TINs at all intermediate levels of resolution. If a DEM at high resolution is given as input, most of the techniques developed in the literature consider a TIN built as follows. The DEM is interpreted as a regular grid, where elevation values are attached to the vertices (usually placed at the center of pixels) and each rectangular cell is subdivided into two triangles through one diagonal. This model carries the same information as the DEM but is fully compatible with the structure of a TIN. Lower resolution models are TINs obtained adaptively from the full resolution model through some terrain simplification procedure, as described in the following section.

2.3 Out-of-core Terrain Simplification

Dealing with huge data is often a difficult challenge. An obvious workaround is to reduce the dataset to a more manageable size. Simplification algorithms take as input a terrain model and produce a simplified version of it, which provides a coarser (approximated) representation of the same terrain, based on a smaller dataset.

Many simplification algorithms have been proposed in the literature (see, e.g., [16] for a survey of specific methods for terrain, and [32] for a survey of more general methods for polygonal 3D models). Most methods are based on the iterated, or simultaneous application of local operators that simplify small portions of the mesh by reducing the number of vertices. Most popular techniques are based on:

- Clustering: a group of vertices that lie close in space, and the sub-mesh that they define, are collapsed to a single vertex, and the portion of mesh surrounding it is warped accordingly.
- Vertex decimation: a vertex is eliminated, together with all its incident triangles, and the hole is filled by a new set of triangles.
- Edge collapse: an edge is contracted so that its two vertices collapse to a single point (which may be either one of them or a point at a new position computed to reduce approximation error), and its two incident triangles collapse to edges incident at that point; the portion of mesh surrounding such two triangles is warped accordingly.

Classical techniques require that the model is completely loaded in main memory. For this reason, such techniques cannot be applied to huge datasets directly.

Simplification of huge meshes requires a technique that is able to load and process at each step a subset of the data, which can fit in main memory. Existing out-of-core techniques have been developed mainly for simplification of triangle meshes bounding 3D objects, and can be easily adapted to simplifying TINs, which are usually simpler to handle. Out-of-core simplification algorithms can be roughly subdivided into three major classes:

- Methods based on vertex clustering.
- Methods based on space partition.
- Streaming techniques.

In the following subsections, we review the main contributions in each class.

2.3.1 Algorithms based on clustering

Rossignac and Borrel [39] proposed an in-core method for the simplification of triangle meshes embedded in the three-dimensional Euclidean space. Their algorithm subdivides the portion of 3D space on which the mesh is embedded into buckets by using a uniform grid, and collapses all vertices inside each bucket to a new vertex, thus modifying the mesh accordingly. In [27], Lindstrom proposes an out-of-core version of the above method that has a lower time and space complexity, and improves mesh quality. The input mesh is kept in secondary memory, while only the portion of the mesh within a single bucket is loaded in main memory. However, the whole output mesh is assumed to fit in main memory. In [28], Lindstrom and Silva propose further improvements over the method in [27] that increase the quality of approximation further and reduce memory requirements. The new algorithm removes the constraint of having enough memory to hold the simplified mesh, thus supporting simplification of really huge models. The work in [28] also improves the quality of the mesh, preserving surface boundaries and optimizing the position of the representative vertex of a grid cell.

Another extension of the Rossignac and Borrel's approach is the algorithm by Shaffer and Garland [40]. This algorithm makes two passes over the input mesh. During the first pass, the mesh is analyzed and an adaptive space partitioning, based on a BSP tree, is performed. Using this approach, a larger number of samples can be allocated to more detailed portions of the surface. However, their algorithm requires more RAM than the algorithm by Lindstrom in order to maintain a BSP tree and additional information in-core.

Garland and Shaffer in [17] present a technique that combines vertex clustering and iterative edge collapse. This approach works in two steps: the first step performs a uniform vertex clustering, as in the methods described above. During the second step, edge collapse operations are performed iteratively to simplify the mesh further, according to an error-driven criterion. The assumption is that the mesh obtained after the first step is small enough to perform the second step in main memory.

Note that all these methods have been developed for simplifying triangle meshes representing objects in 3D space. Adaptation to terrain data is straightforward: it is sufficient to simplify the triangle mesh subdividing the domain which describes the

structure of the TIN, while elevation values are used just to perform error computation. Therefore, for terrains, it is sufficient to partition the domain of the TIN with a 2D grid subdividing its domain into buckets.

Methods based on clustering are fast, but they are not progressive. The resolution of the simplified mesh is a priori determined by the resolution of the regular grid of buckets and no intermediate representations are produced during simplification. For this reason, the algorithms in this class are not suitable to support the construction of multi-resolution models, that will be discussed in Section 2.5.

2.3.2 Algorithms based on space partitioning

The approach to simplification based on space partitioning consists of subdividing the mesh into patches, each of which can fit into main memory, and then simplifying each patch with standard techniques. Attention must be paid to patch boundaries in order to maintain the topological consistency of the simplified mesh.

The method proposed by Hoppe [20] starts from a regular grid and partitions the domain by using a PR quadtree subdivision based on the data points. The quadtree is built in such a way that data points in each leaf node fit in main memory. Simplified TINs are built bottom-up as described below. A full-resolution TIN is built by connecting the vertices of the input grid inside each leaf, and this is simplified iteratively by applying edge collapse. Only internal edges can be collapsed, while edges on the boundary of each patch are left unchanged. Once the meshes corresponding to the four siblings of a node A in the quadtree are reduced to a manageable size, they are merged into a mesh that will be associated with node A, and the simplification process is repeated recursively. The fact that edges on the boundaries of the quadtree blocks are frozen leads to a somehow unbalanced simplification, because data along boundaries between adjacent blocks are maintained at full resolution even in the upper levels of the quadtree.

The previous technique has been later generalized by Prince [37] to arbitrary TINs, whose vertices do not necessarily lie on a regular grid. While conceptually simple, the time and space overhead of partitioning the TIN and of later stitching the various pieces together leads to an expensive in-core simplification process, making such method less suitable for simplifying very large meshes.

El-Sana and Chiang [12] propose an algorithm that works on irregularly distributed data. They partition the mesh at full resolution into patches, where each patch is bounded by chains of edges of the triangle mesh. Patches are sized in such a way that a few of them can be loaded in main memory if necessary. Simplification of a single patch is performed by iteratively collapsing the shortest internal edge of the corresponding triangle mesh. Simplification of a patch is interrupted when its shortest edge lies on its boundary in order to preserve matching between adjacent patches. Once all patches have been simplified independently, the shortest edge of the mesh lies on the common boundary between two patches. Such two patches are merged in main memory and edge collapse is restarted. In this way, the result of simplification is consistent with that obtained in-core by collapsing each time the shortest edge of

the mesh. This method has been used to build a multi-resolution model as described in Section 2.5.

Magillo and Bertocci [33] present a method specific for a TIN. It subdivides the TIN into patches that are small enough to be simplified individually in-core. They take a different approach to preserve inter-patch boundaries. The skeleton of edges that defines the boundary of the patches in the decomposition is simplified first through an algorithm for line simplification based on vertex removal. This maintains the consistency of patch boundaries through different levels of detail. The history of simplification of each chain of edges forming a boundary line is maintained. Then the interior of each patch is also simplified independently through vertex removal. Removal of internal and boundary vertices can be interleaved to obtain a more uniform simplification process.

Cignoni et al. [3] propose a simplification method for triangle meshes in 3D space based on a hierarchical partition of the embedding space through the use of octrees. Octree subdivision stops when the set of triangles associated with a leaf fits in a disk page. The portion of the triangle mesh contained in each octree leaf is independently simplified through iterative edge collapse. Once the leaves are simplified, they can be merged and simplified further. The problem caused by avoiding performing edge collapses on the boundary of the patches, as in [19], is overcome. Vertices and edges of the mesh do not lie on faces of quadtree blocks, while only edges that cross the boundary between adjacent blocks may exist. Such inter-block edges cannot be collapsed while independently simplifying the patches, but they can be either stretched or identified in pairs, because of other edge collapses occurring inside the patches. Thus, the independent simplification of the interior of one patch is interleaved with some simplification on the strip of triangles joining it to its adjacent patches, thus preserving inter-patch consistency. This method can be easily adapted to the special case of terrain data by using quadtrees instead of octrees to partition the domain of the TIN. Note that the input data do not need to be regularly distributed. In general, the vertices of the mesh will not lie on the boundary of quadrants of the quadtree.

2.3.3 Streaming algorithms

The philosophy underlying streaming techniques is that a strictly sequential processing order is followed, where each datum is loaded only once to main memory and such that the result is written to secondary memory as soon as possible.

Isenburg et al. [23] present a simplification technique for triangle meshes that takes as input a sequential indexed representation of the mesh in which vertices and triangles are suitably interleaved. This representation may be also built from more common mesh data structures, such as triangle soups or indexed data structures, through pre-processing techniques also based on streaming [22]. The algorithm streams very large meshes through main memory, but, at each step, only a small portion of the mesh is kept in-core. Mesh access is restricted to a fixed traversal order, but full connectivity and geometry information is available for the active elements of the traversal. The simplification step is performed only on the portion of the mesh loaded in main memory.

Note that streaming algorithms, as those based on clustering, are not progressive and cannot be used to build multi-resolution models.

2.3.4 Comparison

Table 2.1 summarizes the main features of out-of-core simplification techniques. For each method, we list: the type of input (*Dataset*); the type of space partition adopted (*Partitioning*); the type of approach (*Simplification*); the possibility to build a multi-resolution model based on the specific simplification technique (*Multi-res*); and the space requirements in main memory (*Space*). The algorithms by Hoppe [20], Prince [37] and Magillo and Bertocci [33] are designed for either regular or irregular terrain data, while all the other technique can handle arbitrary triangle meshes describing the boundary of 3D objects. When applied to terrain simplification, algorithms based on space partitioning should be modified by replacing the octree which partitions 3D space with a simpler quadtree partition of the 2D domain of the TIN. Note that only methods based on space partitioning are suitable to build multi-resolution models.

	Dataset	Partitioning	Simplification	Multi-res	Space
Lindstrom [27]	tri mesh	regular grid	vertex clustering	NO	$O(out)$
Lindstrom et al. [28]	tri mesh	regular grid	vertex clustering	NO	$O(1)$
Shaffer et al. [40]	tri mesh	BSP tree	vertex clustering	NO	$O(out)$
Garland et al. [17]	tri mesh	regular grid	clust.+collapse	YES	$O(out)$
Hoppe [20]	DEM	space part.	edge collapse	YES	$O(1)$
Prince [37]	TIN	space part.	edge collapse	NO	$O(1)$
El-Sana et al. [12]	tri mesh	greedy dec.	edge collapse	YES	$O(1)$
Magillo et al. [33]	tri mesh	user defined	vertex removal	YES	$O(1)$
Cignoni et al. [3]	tri mesh	space part.	edge collapse	YES	$O(1)$
Isenburg et al. [23]	tri mesh	streaming	various	NO	

Table 2.1. Comparison among simplification algorithms for triangle meshes

2.4 Out-of-core Representation of Regular Multi-resolution Models

Multi-resolution terrain models have been proposed in the literature that work on data regularly distributed on a grid [10, 13, 18, 25, 26, 35]. Such models are based on a nested subdivision that starts from a simple regular tiling of the terrain domain into regular triangles and is generated through a refinement process defined by the uniform subdivision of a triangle into scaled copies of it. The two most common refinement operators used for generating regular multi-resolution models are *triangle quadrisection* and *triangle bisection*.

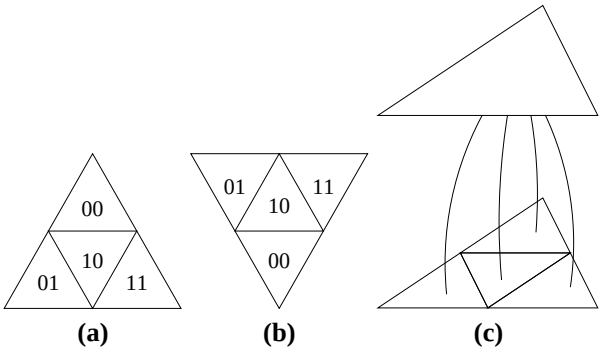

Fig. 2.1. The quadrisection of a triangle oriented (a) tip-up, and (b) tip-down. (c) An example of a triangle quadtree

The quadrisection of a triangle t consists of inserting a new vertex on each edge of t in such a way that the original triangle t is split into four sub-triangles (see Figure 2.1 (a,b)). The resulting (nested) hierarchy of triangles is encoded as a quadtree, called a *triangle quadtree* (see Figure 2.1 (c)). A triangle quadtree is used for both terrain data distributed on the plane, or on the sphere. In this latter case, the idea is to model (i.e., to approximate) the sphere with a regular polyhedron, namely one of the five Platonic solids (i.e., tetrahedron, hexahedron, octahedron, dodecahedron, and icosahedron, which have 4, 6, 8, 12, and 20 faces, respectively), and then to subdivide the surface of the polyhedron using regular decomposition. In the case of the tetrahedron, octahedron, and icosahedron, the individual faces of the solid are triangles and they are in turn represented by a triangle quadtree which provides a representation that has both a variable and multiple resolution variant. Clearly, the fact that the icosahedron has the most faces of the Platonic solids means that it provides the best approximation to a sphere and consequently has been studied the most (e.g., [14, 15, 25]). The goal of these studies has been primarily to enable a way to rapidly navigate between adjacent elements of the surface (termed *neighbor finding*). However, the methods by Lee and Samet [25] are not limited to the icosahedron and, in fact, are also applicable to the tetrahedron and octahedron. In particular, neighbor finding can be performed in worst-case constant time on triangle quadtrees.

Most of regular multi-resolution terrain models are based on *triangle bisection*. The square domain is initially subdivided in two right triangles. The bisection rule subdivides a triangle t into two similar triangles by splitting t at the midpoint of its longest edge (see Figure 2.2 (a)). The recursive application of this splitting rule to the subdivided square domain defines a binary tree of right triangles, in which the children of a triangle t are the two triangles obtained by splitting t. The multi-resolution model generated by this subdivision rule is described by a forest of triangles, called a *triangle bintree*. Each node in a triangle bintree represents a triangle t generated in the recursive subdivision, while the children of node t describe the two triangles arising from the subdivision of t (see Figure 2.2 (b)).

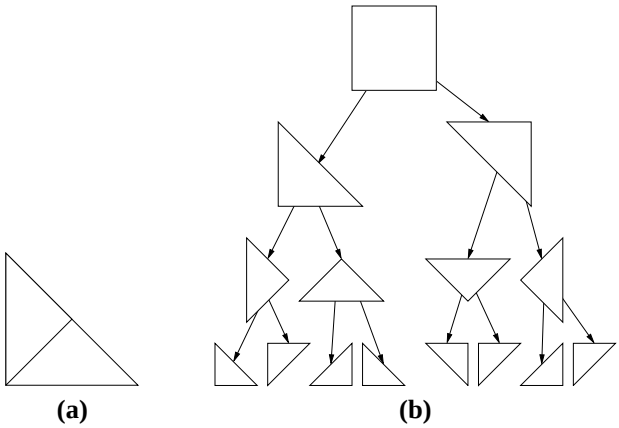

Fig. 2.2. (a) The bisection of a triangle. (b) An example of a triangle bintree

If vertices are available at all nodes of the supporting regular grid, then a triangle bintree consists of two full trees, which can be represented implicitly as two arrays [10, 13]. On the contrary, if data are available at different resolutions over different parts of the domain, then the binary forest of triangles is not complete, and dependencies must be represented explicitly, thus resulting in a more verbose data structure [18].

When extracting adaptive TINs from a triangle bintree, we need to guarantee that whenever a triangle t is split, the triangle adjacent to t along its longest edge is also split at the same time. In this way, the extracted mesh will be conforming, i.e., it will contain no cracks. This can be achieved through the application of neighbor finding techniques to the hierarchy. In [13], an algorithm for neighbor finding is proposed, which works in worst-case constant time. An alternative approach consists of using an error saturation technique, which through manipulation of the errors associated with the triangles in the hierarchy, allows for extracting conforming meshes (but not with a minimal number of triangles) without the need for neighbor finding [34]. Other representations for multi-resolution models generated through triangle bisection have been proposed which encode the model as a Directed Acyclic Graph (DAG) of atomic updates, each of which is called a *diamond*, formed by pairs of triangles which need to be split at the same time [26, 35, 38].

The space requirements of a hierarchical representation can be reduced through the use of pointerless tree representations, also known as *compressed representations*. There are a number of alternative compressed representations for a binary tree, and, in general, for a tree with fanout equal to 2^d (i.e., a quadtree for $d = 2$ and an octree for $d = 3$). One simple method, known as a DF-expression [24], makes use of a list consisting of the traversal of the tree's constituent nodes, where a one bit code is used to denote if the corresponding node is a non-leaf or a leaf node. This method is very space-efficient but does not provide for random access to nodes without traversing the list from the start for each query. Thus, most implementations

make use of structures that are based on finding a mapping from the cells of the domain decomposition to a subset of the integers (i.e., to one dimension)

In the case of a triangle bintree or triangle quadtree, this mapping is constructed by associating a *location code* with each node that corresponds to a triangle element (i.e., noblock) in the tree. A location code for a node implicitly encodes the node as a bit string consisting of the path from the root of the tree to that node, where the path is really the binary (1 bit) or quaternary (2 bits) representation of the transitions that have been made along the path. If the node is at level (i.e., depth) i in the tree, a string of length d_i binary digits is associated with the node where each step in the descent from the root is represented by d bits. In a triangle bintree, each step is encoded as 0(1) depending on whether the corresponding arc in the tree leads to the left (right) child of its parent, while in a triangle quadtree, a labeling scheme is applied which extends the one used for region quadtrees. In particular, in a region quadtree where the blocks are square, the 2-bit string patterns 00, 01, 10, and 11 are associated with the NW, NE, SW, and SE transitions, respectively. In the case of a triangle quadtree, the same 2-bit string patterns are used with the difference that they correspond to different triangles in the hierarchy depending on the triangle orientation (i.e., whether it is tip-up as in Figure 2.1 (a) or tip-down as in Figure 2.1 (b)).

Note that the location codes in a square quadtree are equivalent to the Z order (or Morton order) (e.g., see [42]) which is an ordering of the underlying space in which the result is a mapping from the coordinate values of the upper-left corner u of each square quadtree block to the integers. The Morton order mapping consists of concatenating the result of interleaving the binary representations of the coordinate values of the upper-left corner (e.g., (a, b) in two dimensions) and i of each block of size 2^i so that i is at the right. In the case of a triangle quadtree, the analog of a Z or Morton order can still be constructed but there is no interpretation in terms of bit interleaving as can be seen by examining the three level labeling of the triangle quadtree in Figure 2.3, that is three levels deep. If we record the depth of the tree at which the node is found and append it to the right of the number corresponding to the path from the root to the node thereby forming a more complex location code, then the result of sorting the resulting location codes of all the nodes in increasing order yields the equivalent of a depth-first traversal of the tree. If we vary the format of the resulting location codes so that we record the depth of the tree at which the node is found on the left (instead of on the right), and the number corresponding to the path from the root to the node is on the right, then the result of sorting the resulting location codes of all the nodes in increasing order yields the equivalent of a breadth-first traversal of the tree access structure [2]. These depth-first and breadth-first traversal characterizations are also applicable to triangle quadtrees.

Location codes are used for performing neighbor finding efficiently as well as to retrieve the vertices of a triangle, the value of the field associated with a vertex, etc. An efficient implementation involving arithmetic manipulation and a few bit operations allows performing such computations in constant time [13, 25].

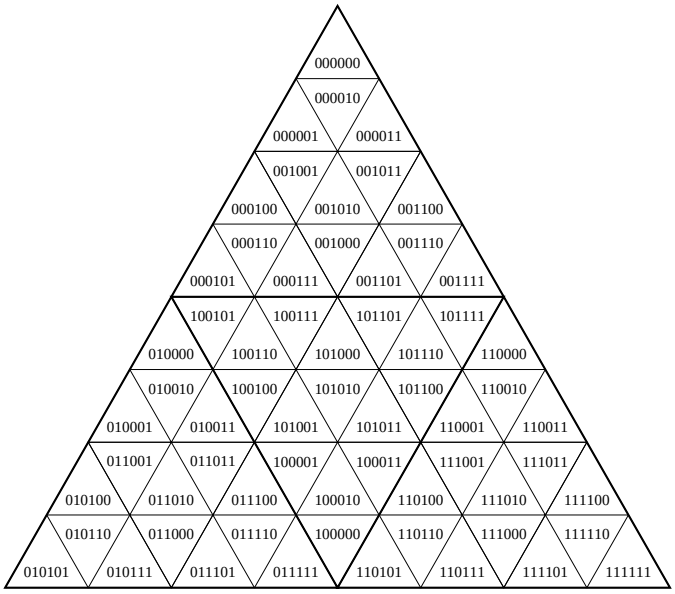

Fig. 2.3. Labeling of a triangle quadtree which is three levels deep

Out-of-core representations of triangle quadtrees and bintrees are based on encoding the location codes of both internal and leaf nodes in an external memory index such as a B-tree, as done for encoding a quadtree or an octree in external memory.

In [18], Gerstner presents a compressed representation of a triangle bintree, that works in main memory, but it could be implemented to provide an effective out-of-core representation. One of the most interesting features of this approach is that the triangle bintree is not necessarily a forest of two full trees as in the implicit in-core representations commonly used for triangle bintrees [10, 13].

In [29], Lindstrom and Pascucci present an implementation of a triangle bintree in external memory with the purpose of visualizing huge sets of terrain data at uniform and variable resolutions. The triangle bintree is represented by using two interleaved quadtrees, which appear at alternating levels. The first quadtree is aligned with the domain boundary, while the other one is rotated of 45 degrees. The two quadtrees are encoded level-by-level, and the resolution increases with the level. Then, the out-of-core handling of data is left to the file paging system of the operating system (data are simply loaded through a `mmap` system call). Two different data layouts are proposed: the first one corresponds to a row by row traversal, while the second one is based on the Z-order. The latter appears to be more efficient even though it is more complex. In this approach, a version of the triangle bintree with error saturation is encoded to avoid neighbor finding in order to extract topologically consistent TINs.

2.5 Out-of-core Multi-resolution Models based on Irregular Meshes

Many multi-resolution models based on irregular triangle meshes have been proposed for both terrain and arbitrary 3D shapes (see [9, 32] for surveys). The basic elements of a multi-resolution model are: a *base mesh*, that defines the coarsest available representation; a set of local *updates* U refining it; and a *dependency relation* among updates [38]. In general, an *update* u defines two sets of triangles u^- and u^+, representing the same portion of a surface at a lower and a higher level of detail, respectively. An update can be applied locally either to refine or to coarsen a mesh. The *direct dependency* relation is defined as follows. An update u_2 depends on an update u_1 if it 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, and the direct dependency relation can be represented as a Directed Acyclic Graph (DAG). Any subset of updates, which is closed with respect to the partial order, i.e., which defines a cut of the DAG, can be applied to the base mesh in any total order extending the partial one, and gives a mesh at an intermediate (uniform or variable) resolution. In Figure 2.4 we show a simple multi-resolution model composed by a base mesh and three updates, red line represents a cut of the DAG encoding the dependecy relation and the mesh on the right represents the extracted mesh, associated to the depicted cut. In [8], we have shown that all existing multi-resolution models are captured by this framework, which can be extended to higher-dimensions and to cell complexes.

In some cases, when a multi-resolution model is built through some specific local modification operators (such as vertex removal, edge collapse, or vertex-pair contraction), an implicit, procedural, encoding of the updates can be used and the dependency relation can be encoded in a more compact form than a DAG, such as the view-dependent tree proposed in [11].

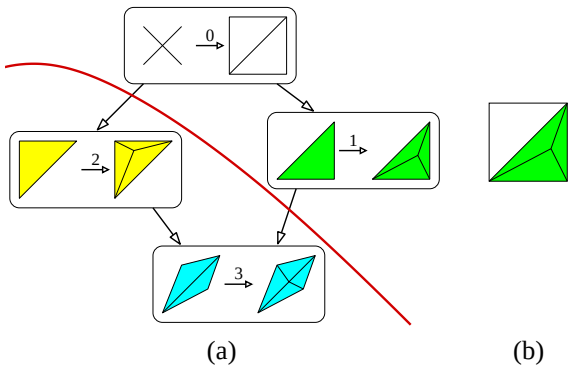

Fig. 2.4. (a) A simple multi-resolution model composed by a base mesh and three updates, red line represents a cut. (b) The mesh corresponding to the front represented by the bold red line

Various techniques have been recently proposed in the literature for out-of-core multi-resolution modeling of large irregular datasets. The general strategy in the design of out-of-core data structure consists of organizing information in disk pages in such a way that the number of page loads/swaps is minimized when traversing the model. In this respect, the basic queries on a multi-resolution model are instances of *selective refinement*, which consists of extracting adaptive meshes of minimal size according to application-dependent requirements. The idea is to select and apply to the base mesh all and only those updates that are necessary to achieve a given, user-defined, level of detail in a user-defined region of interest. Figure 2.5 shows the beaviour of a selective refinement with a top-down depth-first approach. The update without label denotes the dummy modification corresponding to creating the base mesh. White updates satisfy the user criterion, gray updates do not. The dashed line encloses the current set of updates.

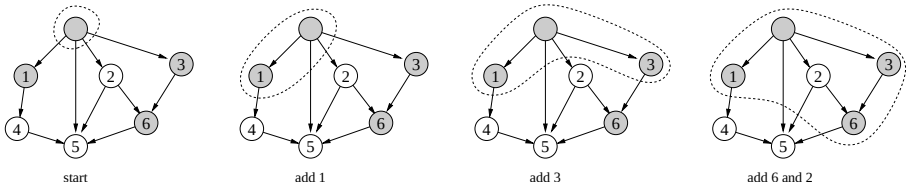

Fig. 2.5. Selective refinement with a top-down depth-first approach

There are two main strategies to organize data into clusters that fit disk pages. The first approach consists of clustering nodes of the hierarchy, corresponding to atomic updates. The second approach consists of clustering data by spatial proximity. We will follow this classification to describe the various methods in the following subsections.

2.5.1 Methods based on clustering of atomic updates

In [12], El-Sana and Chiang build an out-of-core multi-resolution model based on edge collapse, which is generated through the simplification algorithm described in Section 2.3, and targeted to support view-dependent rendering. The dependency relation among updates is represented as a binary forest of vertices, called a *view-dependent tree*. If an edge $e = (v_1, v_2)$ is collapsed into a vertex v, then the node corresponding to v will be the parent of the two nodes corresponding to vertices v_1 and v_2, respectively. There is a one-to-one relation between vertices in the forest and nodes in the DAG of the general model. Arcs of the binary tree are not sufficient to encode all dependencies in the DAG. However, a vertex enumeration mechanism is associated with the vertices in the binary forest, in order to correctly represent the dependency relation among updates, as described in [11].

The binary vertex forest is clustered in subtrees of height h, and, thus, each subtree contains at most $2h - 1$ nodes of the tree. The value h is selected in order to

maximize disk page filling. The selective refinement algorithm keeps in memory the set of subtrees that store the vertices belonging to the extracted mesh, and keeps in cache also their parents and children. If pre-fetched subtrees exceed cache size, the first pre-fetched page, not used by the extracted mesh, is removed from the cache. Figure 2.6 shows a simple binary vertex forest clustered in subtrees of height 2.

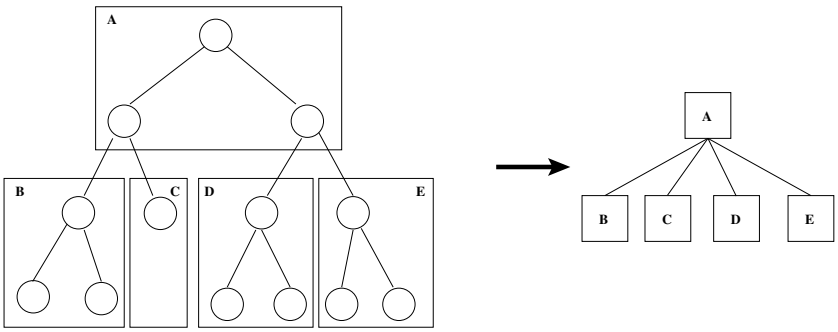

Fig. 2.6. Clustering of a binary vertex forest in subtrees of height 2

Pajarola in [36] proposes a compact representation for multi-resolution models, built through half-edge collapse, i.e., by contracting an edge $e = (v_1, v_2)$ to one of its extreme vertices, let us say v_1. The multi-resolution model encodes a binary forest of half-edge collapses. This forest is obtained from the forest of binary vertices by replacing the subtree formed by vertices v_1, v_2 and by their parent, which will be again v_1 for a half-edge collapse with a pointer to the corresponding half-edge e in the data structure describing the currently extracted mesh. In [7], DeCoro and Pajarola propose an out-of-core representation of the same model, to support view-dependent rendering of large 3D meshes. The proposed technique starts from the binary forest and computes almost balanced subtrees, that can fit in a disk page. Pointers to empty children are removed, thus reducing the storage cost by a 25%. This, together with a compact encoding of the information associated with the nodes of the binary forest of edges, results in a compact data structure that requires less disk accesses. On the other hand, computation of the disk blocks requires two depth-first traversals of the binary forest. The objective is to perform selective refinement out-of-core efficiently, while both the simplification step and the construction of the multi-resolution model are assumed to be performed in-core.

The strategies described above apply only to models based on edge collapse, in which the dependency relation is represented as a binary forest. In [6] we propose and analyze clustering techniques which work on the full DAG of dependency relations. Such techniques are general and can be applied to any multi-resolution model, regardless of the way it is generated. An important assumption is that the updates in the multi-resolution model are atomic, i.e., each update involves a relatively small number of triangles. Conversely, the DAG may have a very large number of nodes.

On the basis of an analysis of selective refinement queries and of the shape of the DAG describing the MT, we have defined and implemented the following techniques for grouping the updates in an MT according to sorting criteria:

- Approximation error (`Err`).
- Layer: shortest path from the root (`Lyr`).
- Level: longest path from root (`Lev`).
- Distance: average path length from the root (`Ly2`).
- Depth-first (`DFS`) and Breadth-first (`BFS`) DAG traversal.
- Multi-resolution depth-first traversal (`GrD`) and Multi-resolution breadth-first traversal (`GrB`): similar to depth-first or breadth-first DAG traversal, respectively, but before performing an update u on the currently extracted mesh, all ancestors of u are visited recursively if they have not been visited before. These criteria simulate the strategies used in a selective refinement algorithm.

We have also defined and implemented two spatial grouping strategies. The first strategy is based on the *R*-tree* [1], while the second strategy is based on a *Point Region k-d (PR k-d) tree* for partitioning in space combined with a *PK-tree* [44] as a grouping mechanism.

Finally, we have designed and implemented a class of strategies which combine space grouping with DAG traverals (according to depth). At a lower resolution, we are interested to have clusters that span the whole domain, while, as the resolution increases, we are looking for clusters associated with finer space subdivisions. In order to achieve this goal, we have developed techniques that interleave the effect of a sorting rule and the effect of a space partitioning rule similar to a PR k-d tree. A description of such techniques can be found in [6].

In all our experiments, the `GrB` outperforms the other clustering techniques. It is interesting to note that, even with a small cache (about 1% the size of the whole model), a clustering technique based on `GrB` exhibits a very limited overhead, compared to loading the whole model just once in main memory.

2.5.2 Methods based on domain partitioning

In [20], Hoppe describes an out-of-core multi-resolution model generated through edge collapse, called a PM (Progressive Mesh) quadtree, which is built through the simplification method described in Section 2.3. The model works on gridded data (DEM). The input grid is triangulated and then partitioned into square blocks in such a way that the content of each block fits in main memory. Simplification is performed bottom-up, and a quadtree is built, where every quadrant contains a simplified representation of the terrain represented by its four children. The resulting data structure is a sort of pyramid, where each block contains a sequence of vertex splits, which inverts the sequence of edge collapses that produced the mesh associated with that block. When selective refinement is performed, only the blocks of the quadtree that are interested in the query are transferred to main memory. Also this model was designed specifically for visualization.

In [4], Cignoni et al. propose an out-of-core multi-resolution model based on the decomposition of the domain into a nested triangle mesh described as a triangle bintree. Each triangle in the bintree, which we call a *macro-triangle*, contains an irregular mesh, formed by a relatively large number of triangles (typically between 256 and 8k triangles). Leaves of the triangle bintree are associated with portions of the mesh at full resolution, while internal nodes are associated with simplified meshes. This multi-resolution representation is created during a fine-to-coarse simplification process, based on Hoppe's method [21] and adapted in order to keep boundary coherence among adjacent macro-triangles. The meshes associated with the macro-triangles are carefully created during the simplification process, so that, when assembled together, they form a conforming triangle mesh. As in Hoppe's approach, this multi-resolution model is targeted at out-of-core terrain visualization. In order to enhance rendering performances, each mesh associated with a macro-triangle is organized into triangle strips. The out-of-core organization in this case is very simple, thanks to the approach that acts on the different levels of granularity of nodes in the hierarchy. The triangle bintree is relatively small, even for huge models, and can be easily kept in main memory. The mesh associated with a macro-triangle is stored in a disk page through a compact data structure.

In [5], the same approach is extended to work on Multi-Tessellations, i.e. the general model, in which the hierarchy is represented by a DAG. Also in this case, every node of the DAG contains a patch of a few thousands of triangles, and is stored in secondary memory, while the DAG describing the dependency relation among updates is small enough to be kept in core. As for the standard Multi-Tessellation, this model can be used for both terrains and 3D shapes.

The model is constructed by computing a partition of the input mesh into patches of almost equal size. Such patches are computed by uniformly distributing on the mesh a set of seed points. The number of such points must be proportional to the mesh size. Then, an approximated Voronoi diagram of the seed points is computed, so that the Voronoi cells define the patches of the subdivision. Each patch is then independently simplified, without modifying its boundary. Simplification of a patch usually reduces the number of triangles by a factor of two. The same process described above is applied again to the simplified mesh by starting with a new, smaller, set of seed points. The process is repeated until the resulting mesh satisfies some size constraint. It has been shown that the number of steps is usually logarithmic in the size of the mesh. The DAG describing the dependency relation is built during this process.

During selective refinement, the DAG is traversed, and only those patches that are needed in order to achieve the resolution required by the user are loaded into memory. Each patch is stored as a compressed triangle strip, in order to improve rendering performances.

Lindstrom [30] and Shaffer and Garland [43] have proposed two similar out-of-core multi-resolution models for massive triangle meshes describing 3D scenes generated through vertex clustering. The purpose is in both cases view-dependent rendering of very large 3D meshes. The multi-resolution model is an octree in which the leaves store the vertices of the mesh at full resolution, or of an already simplified

mesh (in the case of Lindstrom's approach), while the internal octree cells represent vertices obtained as result of the vertex clustering process and triangles are associated with cells containing their vertices.

Lindstrom's approach is based on the out-of-core simplification technique reported in Section 2.3. The construction of the multi-resolution model is performed in two steps. During the first step, the mesh is regularly sampled. Each side of the sampling grid is composed by 2^n cells, where n is a user defined quantity. After that, vertices are sorted in external memory according to their position in the grid. This guarantees local access during the construction of the hierarchy. The second step considers the simplified mesh, composed of the list of vertices, error information, and the list of triangles, and produces an octree having the simplified mesh stored in its leaves. Starting from a group of sibling vertices, position, normal and error of the parent are computed. Note that a leaf cell stores just the vertex and its normal, while an internal cell stores a vertex v and the triangles removed from the representation when v is collapsed.

The mesh indexed in the leaves of the resulting multi-resolution model is the mesh generated by the first simplification step. Thus, the original full-resolution mesh cannot be reconstructed. The multi-resolution model is completely built on disk. View-dependent refinement is performed by two threads: one extracts the variable-resolution mesh, according to an approximate breadth-first octree traversal of the tree, while the other thread renders the mesh. Disk paging is left to the operating system. This is a reasonable choice, since data are sorted in a cache coherent way, but the technique could be further improved by an explicit paging scheme.

Shaffer and Garland's approach [43] is to develop a design for a data structure that offers explicit access to the original mesh. On the other hand, Lindstrom's method has the benefit of working completely out of core, while Shaffer and Garland's method keeps a hash table that refers only to non-empty cells of the grid in memory. This could be a problem for very dense meshes filling the space. Moreover, hash keys are stored in 32 bits, and each key is composed of the three vertex coordinates. This bounds the size of the uniform grid to 1024^3. For out-of-core terrain modeling, both approaches can be simplified by using a quadtree to describe the vertex clustering. In this scenario, Lindstrom's approach could be definitely more efficient, since its performances are not affected by the percentage of full cells in the domain decomposition.

In [45], Yoon et al. propose an out-of-core multi-resolution model for view-dependent rendering of massive triangle meshes describing 3D scenes. The multi-resolution model is called a Clustered Hierarchy of Progressive Meshes (CHPMs), and consists of a hierarchy of clusters, which are spatially localized mesh regions, and of linear sequences of edge collapses, each associated with a cluster, that simplify the corresponding meshes. Each cluster consists of a mesh formed by a few thousand of triangles. The clusters are used to perform coarse-grained view-dependent refinement of the model, while the linear sequences of edge collapses are used for fine-grained local refinement. The cluster hierarchy is enhanced with dependencies among clusters which act as constraints in order to be able to generate crack-free meshes. A CHPM is computed in three steps. First, the vertices of the mesh at full

resolution are organized into clusters containing almost the same number of vertices. A regular grid is superimposed to the set of vertices and a graph $G = (N, A)$ is computed, in which the nodes correspond to the non-empty cells of the grid, while any arc in A connects a node in N and its k-nearest neighboring cells. Graph G is partitioned into clusters, and a new graph is computed in which nodes are associated with clusters, and two nodes are connected by an arc if the corresponding clusters share vertices or are within a threshold distance of each other. Then, the cluster hierarchy is generated top-down by recursively partitioning the cluster graph into halves, thus producing a binary hierarchy. Finally, the triangles of the full-resolution mesh are associated with the clusters in the hierarchy and a mesh simplification process is applied bottom up on the hierarchy of clusters by performing half-edge collapses. During each pass of simplification only the cluster being simplified and the clusters with which it shares vertices must be resident in memory. When performing view-dependent refinement, the cluster hierarchy is kept in main memory, while the sequences of edge collapses are fetched from disk. Also, vertices and triangles corresponding to the active clusters are stored in GPU memory. This approach has been developed for 3D meshes, but can be easily adapted to TINs by using a 2D grid built on the projection of the data points in the plane.

2.5.3 Comparison

Table 2.2 summarizes the various multi-resolution techniques, that we have presented, by highlighting the approach used to organize the out-of-core data structure, the data that can be represented, and the size of updates. Overall, models based on the clustering of nodes in the hierarchy are more general, but also more complex to manage. They have the advantage that the meshes extracted from them have the same granularity and, thus, the same accuracy of the meshes extracted from the corresponding in-core multi-resolution models. On the other hand, dealing with large sets of atomic updates can become a bottleneck in some visualization tasks.

On the contrary, methods that use large patches highly simplify the management of secondary memory and result more efficient, but they trade-off this advantage by being coarser-grained, hence less smooth in the transition between different levels of detail. In particular, the method by Yoon et al. [45] seems to be more suitable for large scenes than terrains, and also the dependencies among clusters are not easily managed.

2.6 Conclusions

We have analyzed and compared out-of core approaches for simplification of triangle meshes and for out-of-core multi-resolution modeling of TINs, both for regularly and irregularly distributed data sets. Most of the mesh simplification algorithms and of the out-of-core multi-resolution representations have been developed for visualization purposes. Most of the techniques for irregular meshes have been developed

	Approach	Data	Update size	in RAM
El-Sana and Chiang [12]	clustering dep.	free form	atomic	clusters of C.M.
DeCoro and Pajarola [7]	clustering dep.	free form	atomic	binary forest
Danovaro et al. [6]	clustering dep.	nD	atomic	a few clusters
Hoppe [20]	partitioning	scalar field	atomic	cluster hierarchy
Cignoni et al. [4]	partitioning	scalar field	large	binary tree
Yoon et al. [45]	partitioning	free form	large	cluster hierarchy
Lindstrom [30]	partitioning	free form	medium	a few MB
Shaffer and Garland [43]	partitioning	free form	medium	cluster indices
Cignoni et al. [5]	partitioning	free form	large	cluster hierarchy

Table 2.2. Comparison among multi-resolution approaches

for triangle meshes describing the boundary of a 3D object, or a 3D scene. These techniques, however, either can be applied directly or can be easily adapted to TINs.

While there are several techniques able to deal with huge triangle meshes, or regular tetrahedral meshes, there is no technique for simplification and multi-resolution modeling of tetrahedral meshes. We are currently developing an out-of-core multi-resolution modeling system for scalar fields based on the out-of-core Multi-Tessellation, able to handle both triangle and tetrahedral meshes, as well as simplicial meshes in higher dimensions.

Acknowledgments

This work has been partially supported by the European Network of Excellence AIM@SHAPE under contract number 506766, by the SHALOM project under contract number RBIN04HWR8, by the National Science Foundation under Grants CCF-0541032, IIS-00-91474, and CCF-05-15241.

References

1. Beckmann N, Kriegel HP, Schneider R, Seeger B (1990) The R*-tree: an efficient and robust access method for points and rectangles. In: Proc. of ACM SIGMOD Conference, Atlantic City, NJ, ACM Press, 322–331
2. Bogdanovich P, Samet H (1999) The ATree: a data structure to support very large scientific databases. In: Agouris P, Stefanidis A (eds) Integrated Spatial Databases: Digital Images and GIS, Portland, ME, 235–248, also University of Maryland Computer Science Technical Report TR–3435, March 1995
3. Cignoni P, Montani C, Rocchini C, Scopigno R (2003) External memory management and simplification of huge meshes. IEEE Transactions on Visualization and Computer Graphics 9:525–537
4. Cignoni P, Ganovelli F, Gobbetti E, Marton F, Ponchio F, Scopigno R (2003) Bdam: Batched dynamic adaptive meshes for high performance terrain visualization. Computer Graphics Forum 22:505–514

5. Cignoni P, Ganovelli F, Gobbetti E, Marton F, Ponchi F, Scopigno R (2005) Batched multi triangulation. In: Proc. of IEEE Visualization 2005, IEEE Computer Society, 207–214
6. Danovaro E, De Floriani L, Puppo E, Samet H (2005) Multi-resolution out-of-core modeling of terrain and geological data. In: Symposium on Advances in Geographic Information Systems, New York, NY, USA, ACM Press, 143–152
7. De Coro C, Pajarola R (2002) XFastMesh: Fast view-dependent meshing from external memory. In: Proc. of IEEE Visualization 2002, Boston, MA, IEEE Computer Society, 263–270
8. De Floriani L, Magillo P (2002) Multi-resolution mesh representation: models and data structures. In: Floater M, Iske A, Quak E, (eds) Principles of Multi-resolution Geometric Modeling. Lecture Notes in Mathematics, Berlin, Springer Verlag, 364–418
9. De Floriani L, Kobbelt L, Puppo E (2004) A survey on data structures for level-of-detail models. In: Dodgson N, Floater M, Sabin M, (eds) Multi-resolution in Geometric Modeling, Springer Verlag, 49–74
10. Duchaineau M, Wolinsky M, Sigeti DE, Miller MC, Aldrich C (1997) Mineev-Weinstein, M.B.: ROAMing terrain: real-time optimally adapting meshes. In: Yagel R, Hagen H (eds) Proc. of IEEE Visualization 1997, Phoenix, AZ, IEEE Computer Society, 81–88
11. El-Sana J, Varshney A (1999) Generalized view-dependent simplification. Computer Graphics Forum 18:C83–C94
12. El-Sana J, Chiang Y J (2000) External memory view-dependent simplification. Computer Graphics Forum 19:139–150
13. Evans W, Kirkpatrick D, Townsend G (2001) Right-triangulated irregular networks. Algorithmica 30:264–286
14. Fekete G, Davis LS (1984) Property spheres: a new representation for 3-d object recognition. In: Proc. of the Workshop on Computer Vision: Representation and Control, Annapolis, MD (1984), 192–201, also University of Maryland Computer Science Technical Report TR–1355
15. Fekete G (1990) Rendering and managing spherical data with sphere quadtrees. In: Kaufman A (ed) Proc. of IEEE Visualization 1990, San Francisco, 176–186
16. Garland M, Heckbert PS (1997) Surface simplification using quadric error metrics. In: Computer Graphics Proc., Annual Conference Series (SIGGRAPH'97), ACM Press, 209–216
17. Garland M, Shaffer E (2002) A multiphase approach to efficient surface simplification. In: Proc. of Visualization 2002, Boston, Massachusetts, IEEE Computer Society, 117–124
18. Gerstner T (2003) Multi-resolution visualization and compression of global topographic data. GeoInformatica 7:7–32
19. Hoppe H (1998) Smooth view-dependent level-of-detail control and its application to terrain rendering. In: Proc. of IEEE Visualization 1998, Research Triangle Park, NC, IEEE Computer Society, 35–42
20. Hoppe H (1998) Efficient implementation of progressive meshes. Computer & Graphics 22:27–36
21. Hoppe H (1999) New quadric metric for simplifying meshes with appearance attributes. In: Proc. of IEEE Visualization 1999, San Francisco, California, United States, IEEE Computer Society Press, 59–66
22. Isenburg M, Gumhold S (2003) Out-of-core compression for gigantic polygon meshes. ACM Transactions on Graphics 22:935–942
23. Isenburg M, Lindstrom P, Gumhold S, Snoeyink J (2003) Large mesh simplification using processing sequences. In: Proc. of Visualization 2003 Conference, IEEE Computer Society Press, 465–472

24. Kawaguchi E, Endo T (1980) On a method of binary picture representation and its application to data compression. IEEE Transactions on Pattern Analysis and Machine Intelligence 2:27–35
25. Lee M, Samet H (2000) Navigating through triangle meshes implemented as linear quadtrees. ACM Transactions on Graphics 19:79–121
26. Lindstrom P, Koller D, Ribarsky W, Hodges LF, Faust N, Turner GA (1996) Real-time continuous level of detail rendering of height fields. In: Proc. of SIGGRAPH 1996, 109–118
27. Lindstrom P (2000) Out-of-core simplification of large polygonal models. In: ACM Computer Graphics Proc., Annual Conference Series (SIGGRAPH'00), New Orleans, LA, USA, ACM Press, 259–270
28. Lindstrom P, Silva CT (2001) A memory insensitive technique for large model simplification. In: IEEE Visualization 2001, IEEE Computer Society, 121–126
29. Lindstrom P, Pascucci V (2002) Terrain simplification simplified: a general framework for view-dependent out-of-core visualization. IEEE Transactions on Visualization and Computer Graphics 8:239–254
30. Lindstrom P (2003) Out-of-core construction and visualization of multi-resolution surfaces. In: Proc. of ACM SIGGRAPH 2003 Symposium on Interactive 3D Graphics, Monterey, California, ACM Press, 93–102
31. Losasso F, Hoppe H (2004) Geometry clipmaps: terrain rendering using nested regular grids. ACM Transaction on Graphics 23:769–776
32. Luebke D, Reddy M, Cohen J, Varshney A, Watson B, Huebner R (2002) Level of Detail for 3D Graphics. Morgan-Kaufmann, San Francisco
33. Magillo P, Bertocci V (2000) Managing large terrain data sets with a multiresolution structure. In: Tjoa A M, Wagner R R, Al-Zobaidie A (eds) Proc. 11th Int. Conf. on Database and Expert Systems Applications, Greenwich, UK, IEEE, 894–898
34. Ohlberger M, Rumpf M (1997) Hierarchical and adaptive visualization on nested grids. Computing 56:365–385
35. Pajarola R (1998) Large scale terrain visualization using the restricted quadtree triangulation. In: Ebert D, Hagen H, Rushmeier H, (eds) Proc. of IEEE Visualization 1998, Research Triangle Park, NC, IEEE Computer Society, 19–26
36. Pajarola R (2001) FastMesh: efficient view-dependent meshing. In: Proc. of Pacific Graphics 2001, IEEE Computer Society, 22–30
37. Prince C (2000) Progressive Meshes for Large Models of Arbitrary Topology. Master Thesis, Department of Computer Science and Engineering, University of Washington
38. Puppo E (1998) Variable resolution triangulations. Computational Geometry Theory and Applications 11:219–238
39. Rossignac J, Borrel P (1993) Multi-resolution 3D approximations for rendering complex scenes. In: Falcidieno B, Kunii T L, (eds) Modeling in Computer Graphics, Springer-Verlag, 455–465
40. Shaffer E, Garland M (2001) Efficient adaptive simplification of massive meshes. In: Proc. of IEEE Visualization 2001, IEEE Computer Society, 127–134
41. Samet H (1990) The Design and Analysis of Spatial Data Structures. Addison-Wesley, Reading, MA
42. Samet H (2006) Foundations of Multidimensional and Metric Data Structures. Morgan-Kaufmann, San Francisco
43. Shaffer E, Garland M (2005) A multi-resolution representation for massive meshes. IEEE Transactions on Visualization and Computer Graphics 11:139–148

44. Wang W, Yang J, Muntz R (1998) PK-tree: a spatial index structure for high dimensional point data. In: Tanaka K, Ghandeharizadeh S (eds) Proc. 5th International Conference on Foundations of Data Organization and Algorithms (FODO), Kobe, Japan, 27–36
45. Yoon S, Salomon B, Gayle R, Manocha D (2005) Quick-VDR: Out-of-core view-dependent rendering of gigantic models. IEEE Transactions on Visualization and Computer Graphics 11:369–382