Visual Comput (2007) 23: 583–605
DOI 10.1007/s00371-007-0163-2

ORIGINAL ARTICLE

Renato Pajarola
Enrico Gobbetti

Survey of semi-regular multiresolution models for interactive terrain rendering

Published online: 13 June 2007

R. Pajarola (✉)
Visualization and MultiMedia Lab,
Department of Informatics,
University of Zürich, Switzerland
pajarola@acm.org

E. Gobbetti
Visual Computing Group,
Center for Advanced Studies, Research,
and Development in Sardinia (CRS4),
09010 Pula (CA), Italy
gobbetti@crs4.it

Abstract Rendering high quality digital terrains at interactive rates requires carefully crafted algorithms and data structures able to balance the competing requirements of realism and frame rates, while taking into account the memory and speed limitations of the underlying graphics platform. In this survey, we analyze multiresolution approaches that exploit a certain semi-regularity of the data. These approaches have produced some of the most efficient systems to date. After providing a short background and motivation for the methods, we focus on illustrating models based on tiled blocks and nested regular grids, quadtrees and triangle bin-trees triangulations, as well as cluster-based approaches. We then discuss LOD error metrics and system-level data management aspects of interactive terrain visualization, including dynamic scene management, out-of-core data organization and compression, as well as numerical accuracy.

Keywords Terrain rendering · Multiresolution triangulation · Semi-regular meshes

1 Introduction

Efficient interactive visualization of very large digital elevation models (DEMs) is important in a number of application domains, such as scientific visualization, GIS, mapping applications, virtual reality, flight simulation, military command and control, or interactive 3D games. Due to the ever increasing complexity of DEMs, real-time display imposes strict efficiency constraints on the visualization system, which is forced to dynamically trade rendering quality with usage of limited system resources. The investigation of multiresolution methods to dynamically adapt rendered model complexity has thus been, and still is, a very active computer graphics research area. The concept has extensively been studied for general 3D triangle meshes and has been surveyed, for instance, in [10, 18, 25, 38, 39], and more recently in [11]. While general data structure and algorithms are also applicable to digital terrain models, the most efficient systems to date rely on a variety of methods specifically tailored to terrain models, i.e., 2.5-dimensional surfaces.

In this survey, we present and analyze the most common multiresolution approaches for terrain rendering that exploit a certain semi-regularity of the data to gain maximum efficiency. De Floriani et al. [13] provide a classic survey focusing more on multiresolution terrain models over irregular meshes.

After providing a short background and motivation for the methods (Sect. 2), we provide an overview of the most common approaches. Section 3 provides examples of models based on tiled blocks and nested regular grids, Sect. 4 surveys quadtree and triangle bin-trees triangulations, while Sect. 5 is devoted to recent GPU-friendly cluster-based approaches. We then discuss error metrics (Sect. 6) and system-level aspects of interactive terrain visualization (Sect. 7), including dynamic scene management, out-of-core data organization and compression, as well as numerical accuracy. The paper concludes with a short summary in Sect. 8.

2 Background and motivation

A multiresolution terrain model supporting view-dependent rendering must efficiently encode the steps performed by a mesh refinement or coarsening process in a compact data structure, from which a virtually continuous set of variable resolution meshes can be extracted, loaded on demand, and efficiently rendered at run-time. The basic ingredients of such a model are a *base mesh* that defines the coarsest approximation to the terrain surface, a set of *updates* that, when applied to the base mesh, provide variable resolution mesh-based representations, and a *dependency relation* among updates, which allows combining them to extract consistent intermediate representations [11]. Interactive rendering of large data sets consists of extracting at run-time, through a *view-dependent query* a consistent minimum complexity representation that minimizes a *view-dependent error* measure, eventually loading it on demand from external memory. Different specialized multiresolution models, of various efficiency and generality, are obtained by mixing and matching different instances of all these ingredients.

In the most general case, the multiresolution model is based on a fully irregular approach in which the base mesh is an irregular triangulation with unrestricted connectivity, and updates are encoded either explicitly in terms of sets of removed and inserted triangles [12] or implicitly by the operations through which the model is refined or coarsened, i.e., edge collapse/split or vertex insertion/removal [27]. A dimension-independent framework fully covering this kind of model is multitessellation [14, 48]. Because of their flexibility, fully irregular approaches are theoretically capable of producing the minimum complexity representation for a given error measurement. However, this flexibility comes at a price. In particular, mesh connectivity, hierarchy, and dependencies must explicitly be encoded, and simplification and coarsening operations must handle arbitrary neighborhoods. By imposing constraints on mesh connectivity and update operations it is possible to devise classes of more restricted models that are less costly to store, transmit, render, and simpler to modify. This is because much of the information required for all these tasks becomes implicit, and often, because stricter bounds on the region of influence of each local modification can be defined.

Using meshes with semi-regular or regular connectivity, together with fixed subdivision rules, is particularly well adapted to terrains, since input data from remote sensing most often comes in gridded form. Moreover, as the cost of 3D transformations is becoming negligible on current hardware, controlling the shape of each rendered triangle starts to become negligible, favoring methods with the most compact and efficient host-to-graphics interface. For all these reasons, regular or semi-regular approaches have produced some of the most efficient systems to date.

3 Non-conforming and limited adaptivity techniques: tiled blocks and nested regular grids

A number of successful large scale terrain visualization systems are based on data structures that do not support fully adaptive surfaces but are simple to implement and efficient in communicating with the I/O subsystem and with the underlying graphics hardware.

3.1 Multiple static level-of-detail rendering based on tiled blocks

Early LOD terrain rendering methods used a fixed representation approach. With these methods, multiple representations of parts of the terrain, typically square blocks, are precomputed and stored off-line. At run-time, the appropriate approximation mesh is assembled from precomputed blocks based on the current view-parameters. Because different parts of the terrain may be using different representations in the current approximation, cracks can occur at the boundaries between different-resolution representations.

The NPSNET military simulation system [17], for instance, decomposes the terrain into adjacent blocks represented at four different levels of detail. The representations are precomputed and stored on disk. A 16×16 grid of blocks is kept in memory, and a simple distance metric is used to determine the resolution at which each block will be rendered (Fig. 1). No effort is made to stitch blocks. As the viewer moves, an entire row or column is paged out while the opposite one is paged in. This technique is also used in the most recent and general work on geometry clipmaps [37]. As the number of LODs is fixed, the model provides very limited adaptivity and is tuned to particular applications with narrow fields of view.

In [32, 50], rather than dividing the terrain into a grid, the authors represent it using a quadtree. Each level of the quadtree has a single LOD representation that consists of a uniform grid over a fixed number of sample points. The root-level mesh represents the entire terrain, while deeper levels represent one quarter of the previous level's area. At run-time, the quadtree is traversed and a decision is made about which blocks of terrain should be used to represent the terrain. To visually deal with discontinuities at tile boundaries, vertical wall polygons are constructed between the tile edges. This work was then extended in [6] by associating at each block a precomputed fixed representation, which is chosen among uniform meshes, non-uniform grids, and TINs. This allows

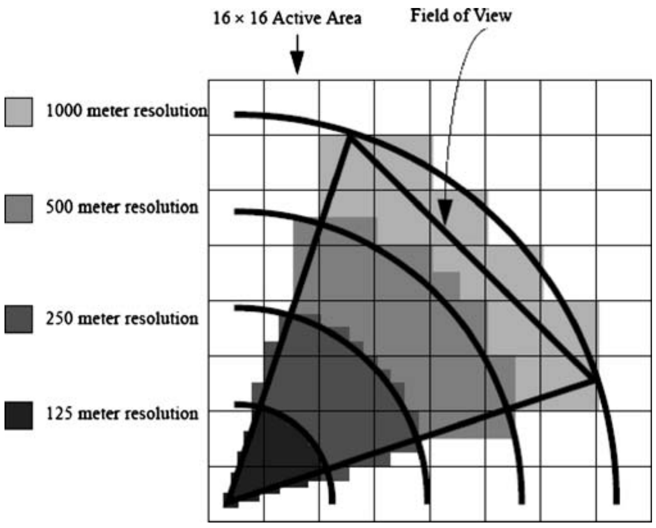

Fig. 1. Multiresolution rendering in NPSNET

nodes that are deep in the tree to represent fine-grained features (such as river beds or roadways) using a TIN representation, while allowing a uniform mesh representation to be at shallower levels in the tree. As in [32], cracks between adjacent blocks of terrain are filled by vertical wall polygons. Other visual crack filling methods include adding flanges around blocks, so that neighboring meshes interpenetrate slightly, as well as joining blocks with special meshes at run-time [59].

Even if these methods may seem overly simplistic, since they do not produce continuous levels-of-detail and require work to fix the cracks at block boundaries, and introduce hard to control visual artifacts, they are still very popular, especially for networked applications, mostly because of scalability, ease of implementation, and simplicity of integration with an efficient tile-based I/O system.

3.2 Nested regular grids

Losasso and Hoppe [37] have recently proposed the geometry clipmap, a simple and efficient approach that parallels with the LOD treatment of images. A prefiltered mipmap pyramid is a natural representation of terrain data. The pyramid represents nested extents at successive power-of-two resolutions. Geometry clipmaps cache in video memory nested rectangular extents of the pyramid to create view-dependent approximations (see Fig. 2). As the viewpoint moves, the clipmap levels shift and are incrementally refilled with data. To permit efficient incremental updates, the array is accessed toroidally, i.e., with 2D wraparound addressing using mod operations on x and y. Transition regions are created to smoothly blend between levels, and T-junctions are avoided by stitching the level boundaries using zero-area triangles. The LOD transition scheme allows independent translation of the clipmap levels, and lets levels be cropped rather than invalidated atomically. Since LODs are purely based on 2D distances from the clipmap center, the terrain data does not require precomputation of refinement criteria. Together with the simple grid structure, this allows the terrain to be synthesized on-the-fly, or to be stored in a highly compressed format. For compression, the residuals between levels are compressed using an advanced image coder that supports fast access to image regions.

Storing in a compressed form only the heights and reconstructing at run-time both normal and color data (using a simple height color mapping) provides a very compact representation that can be maintained in main memory even for large data sets. The pyramidal scheme however limits adaptivity. In particular, as with texture clipmap-based methods, the technique works best for wide field of views and nearly planar geometry, and would not apply to planetary reconstructions that would require more than one nesting neighborhood for a given perspective.

3.3 Discussion

The methods surveyed in this section strive to provide a simple and efficient implementation at the cost of imposing limitations in adaptivity and approximation quality. In the next sections we will see methods that rely on more complex, but also more powerful data structures. We will first survey quadtree and bin-tree triangulations, i.e.,

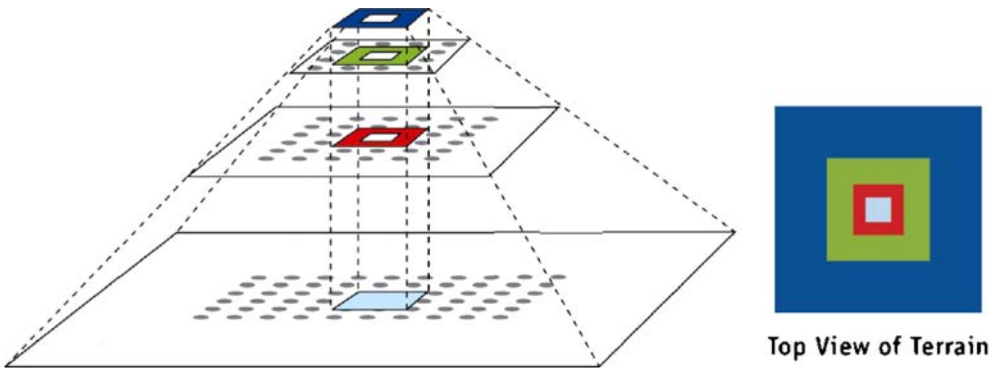

Fig. 2. Geometry clipmap

methods able to construct fully continuous levels of details by imposing consistency rules on local subdivision. We will then show how these methods can be made more efficient in terms of raw triangle throughput by employing a cluster-based approach.

4 Variable resolution triangulation using quadtree and triangle bin-tree subdivision

From the point of view of the rapid adaptive construction and display of continuous terrain surfaces, some of the most successful examples are based on quadtree or triangle bin-tree triangulation. As we will see, the scheme permits the creation of continuous variable resolution surfaces without having to cope with the gaps created by other regular grid schemes, as those in Sect. 3. The main idea shared by all of these approaches is to build a regular multiresolution hierarchy by refinement or by simplification. The refinement approach starts from an isosceles right triangle and proceeds by recursively refining it by bisecting its longest edge and creating two smaller right triangles. In the simplification approach the steps are reversed: given a regular triangulation of a gridded terrain, pairs of right triangles are selectively merged. The regular structure of these operations enables to implicitly encode all the dependencies among the various refinement/simplification operations in a compact and simple way.

Depending on the definition of the triangulation rule, there is potentially a difference in the adaptive triangulation power of quadtree-based triangulations versus triangle bin-trees. Generally, any of the discussed quadtree triangulations can be considered a special case of recursive triangle bisection. Nevertheless, from the refined definition of a restricted quadtree triangulation as presented in [57, 58] and the following works [34, 42], one can arguably consider the restricted quadtree triangulation and triangle bin-tree to produce the same class of adaptive grid triangulations. Hence we use the term restricted quadtree triangulation more in line with [57] rather than with the more strict definition as in [49].

4.1 Quadtree triangulation

In this section we discuss the various algorithms of quadtree-based adaptive triangulation of height-fields (or parametric 2D surfaces). The closely related triangle bisection approaches are discussed in Sect. 4.2. A typical example of a simplified triangulated surface that can be constructed using this class of multiresolution triangulation methods is given in Fig. 3.

Restricted quadtrees. Hierarchical, quadtree-based adaptive triangulation of 2-manifold surfaces was first presented in [62] and applied to adaptively sample and triangulate curved parametric surfaces. In parameter space, the quadtree subdivision is performed recursively until for each sampled region the Lipschitz condition for the parametric curve is met that bounds the accuracy of the resulting polygonal approximation. Furthermore, the quadtree subdivision is restricted such that neighboring regions must be within one level of each other in the quadtree hierarchy as shown in Fig. 4.

The basic approach for triangulation and visualization uses the following steps:

1. Initial sampling of function on a uniform grid
2. Evaluation of each region with respect to some acceptance criteria (approximation error metric)
3. A 4-split of unacceptable regions
4. Repetition of Steps 2 and 3 until adaptive sampling satisfies acceptance criteria over the entire surface
5. Triangulation and rendering of all quadtree regions.

To prevent possible cracks in the polygonal representation of a restricted quadtree as shown in Fig. 5, every quadtree region is triangulated with respect to the resolution of its adjacent regions. Due to the constraint of the restricted quadtree hierarchy that the levels of adjacent

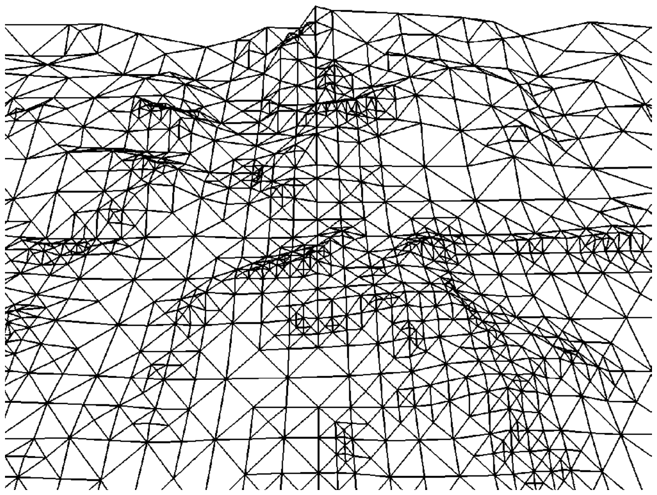
Fig. 3. Adaptive quadtree based terrain triangulation

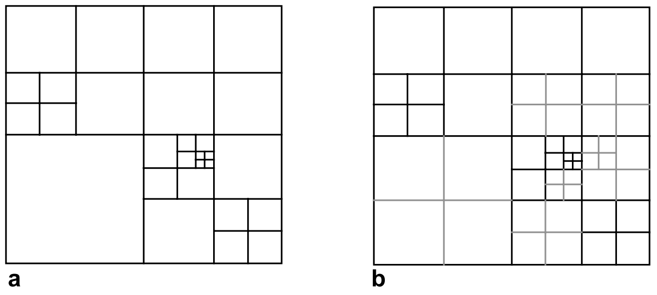
Fig. 4. a Example of an unrestricted quadtree subdivision in parameter space. **b** The restricted subdivision

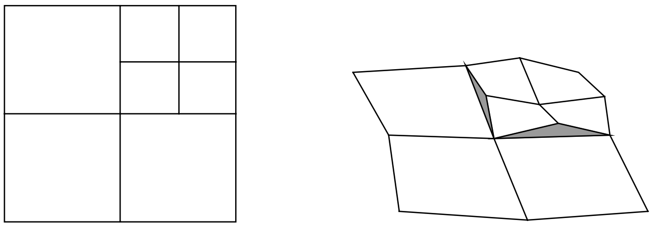
Fig. 5. Cracks (*shaded* in *gray*) resulting from a quadrilateral polygonal representation of a restricted quadtree

regions differ at most by one, the regions can be triangulated such that no cracks appear as outlined below. Such a crack-free triangulation is also called *conforming*.

The triangulation rule as stated in [62] is the following: Each square is subdivided into eight triangles, two triangles per edge, unless the edge borders a larger square in which case a single triangle is formed along that edge. Figure 6 shows a triangulation of a restricted quadtree following this rule.

No detailed algorithms and data structures are given in [62] to construct and triangulate a restricted quadtree. Nevertheless, the presented restricted quadtree subdivision and its triangulation forms the fundamental basis on which a number of the surveyed triangulation approaches are built.

Quadtree surface maps. In [57, 58], the restricted quadtree technique is refined and applied to 2.5-dimensional surface data consisting of points on a regular 2D-grid and each having a *height* value associated with it. This is the common representation of grid-digital terrain elevation models. In addition to the basic method as presented in [62], in [58] two efficient construction algorithms to generate and triangulate a restricted quadtree from a set of points on a regular grid are provided. One method is performed bottom-up and the other top-down to generate the restricted quadtree hierarchy. Furthermore, in [58] it is also observed that edges shared by two quadtree nodes on the same hierarchy level do not have to be split to guarantee a conforming triangulation as shown in Fig. 7 in comparison to Fig. 6.

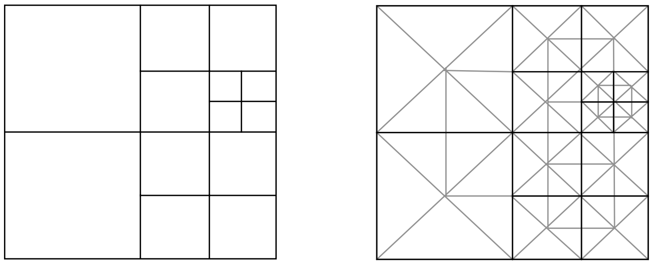
Fig. 6. Conforming triangulation of a restricted quadtree subdivision as in [62]

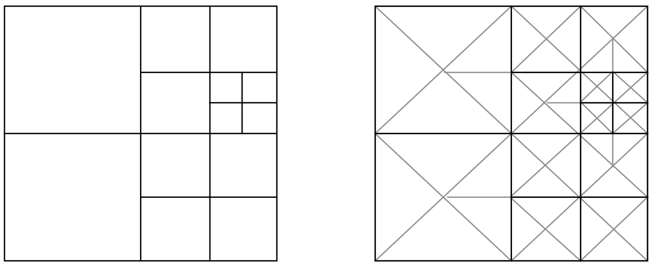
Fig. 7. Improved conforming triangulation of a restricted quadtree subdivision as in [58]

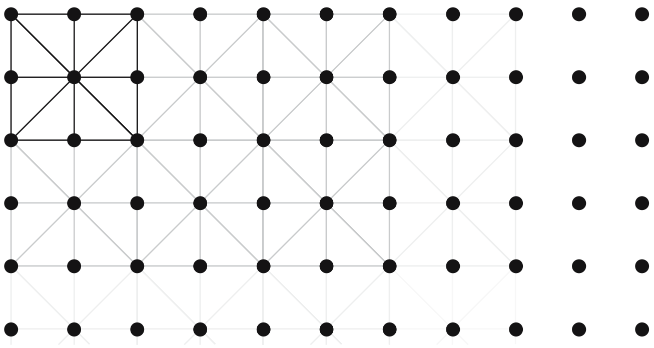
Fig. 8. Atomic leaf node for bottom-up construction of restricted quadtree

In the bottom-up construction method, the (square) input grid is partitioned into atomic nodes of 3×3 elevation points as shown in Fig. 8. These nodes form the leaf nodes of a complete and balanced quadtree over the entire input grid.

The main phase of this method then consists of coalescing all mergible nodes bottom-up to create the restricted quadtree. Nodes must pass two main criteria before they can be merged:

1. *Error measure*. The approximation error introduced by removing the edge midpoint vertices of the nodes being merged must be within the tolerance of a given error threshold.
2. *Quadtree constraints*. The size of the node is equal to the size of its three siblings in the quadtree hierarchy, and neither the node nor its siblings have any smaller sized neighbors.

The approximation error of Criterion 1 used in [58] is further discussed in Sect. 6. The algorithm terminates if no further merges can be performed, and it has a linear space and time cost $O(n)$ in the size n of the input data set.

The second algorithm presented in [58] is a top-down construction of the restricted quadtree hierarchy. This method starts with representing the entire data set simplified by one root node and splits nodes recursively, never merges any, as necessary to approximate the data set. The method maintains at all time the restricted quadtree property that adjacent leaf nodes do not differ by more than one level in the hierarchy.

Vertices that can conceptually be removed by merging four sibling nodes are called *non-persistent*. Starting with the root node as shown in Fig. 9a, for each node of the partially constructed restricted quadtree the non-persistent vertices are identified in the input data set and their error metric compared to the given approximation threshold. If any non-persistent vertex is not within the tolerated threshold it is added to the current quadtree. However, insertion of vertices can lead to complex updates of the quadtree as outlined below.

To permanently maintain a restricted quadtree, the insertion of a vertex can lead to propagated splits in the parent and adjacent quadtree nodes. As shown in Fig. 10, it may happen that a node on level l is not split because no vertices of level $l+1$ are inserted; however, a vertex v_2 on level $l+2$ has to be added. This insertion cannot be performed directly since no parent node covering v_2 has been created yet on level $l+1$. First the parent node of v_2 and its siblings on level $l+1$ have to be inserted by splitting the smallest node on level l enclosing v_2 into four nodes. Such propagated splits can occur over multiple levels.

For further details, in particular of the top-down algorithm we refer to the detailed description of surface maps from restricted quadtrees in [57].

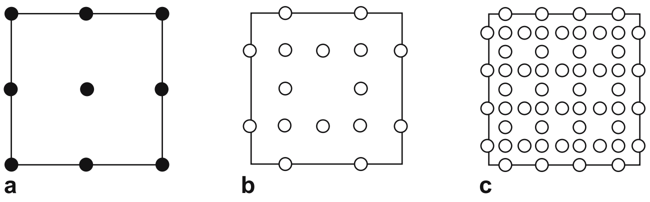

Fig. 9a–c. Vertices of the root node (level 0) (**a**) as well as the non-persistent vertices of level 1 (**b**) and level 2 (**c**)

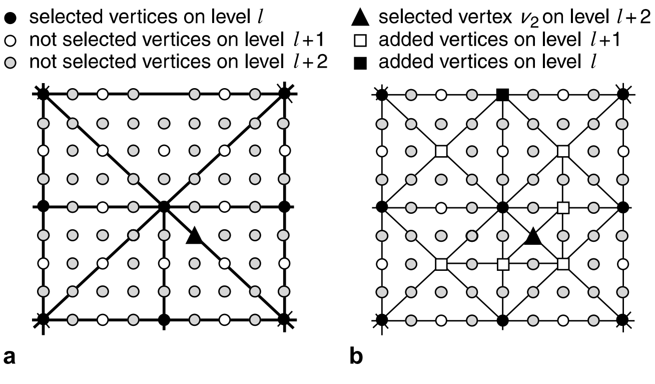
● selected vertices on level l
○ not selected vertices on level $l+1$
○ not selected vertices on level $l+2$
▲ selected vertex v_2 on level $l+2$
□ added vertices on level $l+1$
■ added vertices on level l
a
b

Fig. 10a,b. Starting triangulation of a node on level l (**a**). No vertices are initially selected on level $l+1$. The selection of a vertex on level $l+2$ leads to forced splits and added vertices on previous levels (**b**)

The proposed top-down algorithm to create an adaptive surface mesh processes the entire data set and thus its cost is $O(n)$, linear in the size n of the input. While the number of generated quadtree nodes is indeed output-sensitive, the overall run-time is still directly proportional to the input data set since all vertices have to be visited. However, in contrast to the bottom-up algorithm, this top-down method correctly calculates the approximation error at each vertex as discussed in Sect. 6.

Both methods presented in [57,58] operate on a hierarchical quadtree data structure that must provide functionality for inserting vertices, calculating distance of a vertex to a piece-wise linear surface approximation, neighbor-finding, and for merging and splitting nodes. Furthermore, the restricted quadtree nodes must be post-processed to generate the resulting conforming triangulation. The presented algorithms are capable of creating adaptive and continuous LOD triangulations within the limits of the error metric. However, efficiency is not optimized for real-time rendering of very large terrain data sets due to the input sensitiveness of the basic triangulation algorithms.

Continuous LOD quadtree. A different approach to generate and triangulate a restricted quadtree is presented in [34] based on the notion of *triangle fusion*. Starting with a triangulation of the entire grid-digital terrain data set the triangle mesh is simplified bottom-up by consecutive merging of symmetric triangle pairs. The full resolution grid triangulation as shown in Fig. 11a is equivalent to the atomic leaf nodes of the bottom-up triangulation method in [58]. Triangle merging is performed in two phases as shown in Figs. 11b, c. First, in an atomic node pairs of isosceles triangles (i.e., a_l and a_r in Fig. 11b) sharing a short edge are coalesced and the midpoint on the boundary edge of the quad is removed. In the second phase the center vertex of a quad region is removed by merging isosceles triangle pairs along the diagonal (i.e., e_l and e_r in Fig. 11c). However, to prevent cracks from occurring due to triangle merging, two pairs of isosceles triangles that all share the same removed base vertex must always be coalesced simultaneously (i.e., both pairs e_l, e_r and f_l, f_r in Fig. 11c).

One of the main contributions of [34] is the introduction of vertex dependencies that can be used to prevent cracks

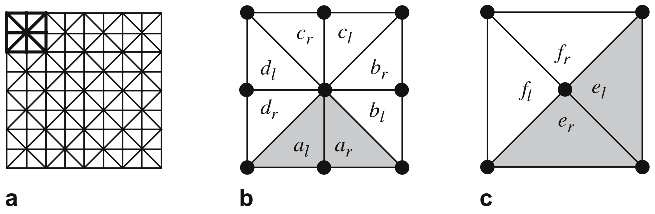

Fig. 11. a Full resolution triangulation. Merging of triangle pairs along **b** the bottom boundary edge and **c** along the diagonal

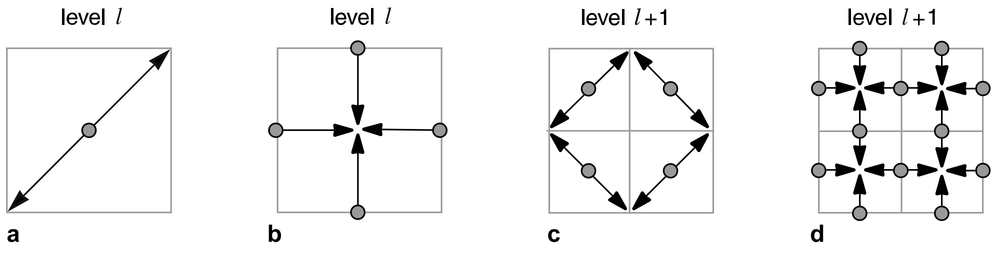

Fig. 12a–d. Dependency relation of a restricted quadtree triangulation. The center vertex in **a** depends on the inclusion of two corners of its quad region. The boundary edge midpoints in **b** depend on the center vertex of the quad region. Dependencies within and between the next resolution levels are shown in **c** and **d**

and create conforming triangulations at variable LOD. For example, considering Fig. 11b, c it is clear that the midpoint of the bottom edge on level l, the base vertex of triangles a_l and a_r, cannot be part of a conforming triangulation if the center vertex of the quad region, the base vertex of e_l, e_r, is missing. Or from the opposite viewpoint, the base vertex of triangles e_l, e_r and f_l, f_r cannot be removed if any of the base vertices of triangle pairs a_l, a_r, b_l, b_r, c_l, c_r or d_l, d_r persist. These constraints of a conforming restricted quadtree triangulation define a binary, hierarchical dependency relation between vertices as shown in Fig. 12. Each vertex to be included in a triangulation depends on two other vertices (on the same or lower resolution level) to be included first. Therefore, a triangulation of a (restricted) quadtree is a conforming triangulation only if no such dependency relation is violated. The triangulation method proposed in [34] recursively resolves the dependency relations of a set S of selected vertices (i.e., all vertices exceeding a given error tolerance) as follows: For each vertex $v \in S$, all its dependent parents according to the dependency rules shown in Fig. 12 are recursively activated and included in the triangulation as well.

Another important feature presented in [34] is the construction of triangle strips, similar to the earlier work in [24], for fast rendering. In fact, a triangulation of a restricted quadtree can be represented by one single *generalized triangle strip*.[1] The triangle strip generation method described in [34] is based on a recursive preorder traversal of the triangular quadrants of quadtree blocks. Starting with a counterclockwise ordering of triangular quadrants of the root node as shown in Fig. 13a, each quadrant is recursively traversed and the traversal is stopped when a triangle is not further subdivided. In alternating order, children of triangular quadrants are visited *left-first* as for quadrant q_0 (and in Fig. 13c), or em right-first as for the triangles in the next level shown in Fig. 13b. Based on this traversal, vertices can be ordered and output to form a generalized triangle strip for efficient rendering (see [34] for code details).

Despite the fact that an entire triangulated restricted quadtree can be represented by one triangle strip, triangle

[1] generalized triangle strips allow swap operations

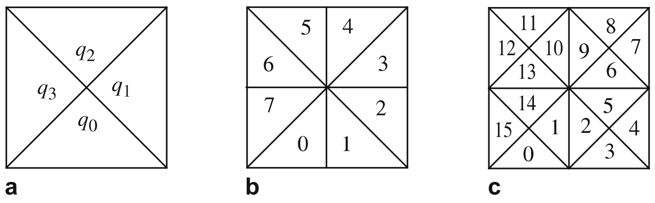

Fig. 13a–c. Recursive quadtree traversal for triangle strip generation. **a** Initial order of triangular quadrants, with **b** *left-first* traversal for odd subdivisions and **c** *right-first* traversal of even subdivision steps

strips are formed for individual blocks only in [34]. For each frame a block-based view-dependent image-space error metric is used (see Sect. 5) to form a (non-restricted) quadtree subdivision S of the terrain. For each block $b \in S$ of this subdivision, a vertex-based error metric is applied to achieve a fine-grain selection of vertices to be included in the triangulation. Furthermore, the vertex dependencies are resolved at this stage to guarantee a conforming triangulation. Finally, for each quadtree block $b \in S$ a triangle strip is generated and used for rendering.

The triangulation method presented in [34] is very efficient in terms of rendering performance. The triangulation algorithm is output-sensitive since the quadtree subdivision is performed top-down and does not need to examine all vertices on the highest resolution. Furthermore, efficient rendering primitives in the form of triangle strips are generated for optimized rendering. Despite the fact that the view-dependent error metric does not provide a guaranteed error bound, it is very efficient in practice and provides good terrain simplification while maintaining plausible visual results.

Restricted quadtree triangulation. The *restricted quadtree triangulation* (RQT) approach presented in [42, 43] is focused on large scale real-time terrain visualization. The triangulation method is based on a quadtree hierarchy as in [57, 58] and exploits the dependency relation presented in [34] to generate minimally conforming quadtree triangulations. Both, top-down and bottom-up triangulation algorithms are given for a terrain height-field maintained in a region quadtree, and where each vertex has an approximation error associated with it. It is observed that the quadtree hierarchy can be defined implicitly on an array of the regular grid input data set by appropriate point indexing and recursive functions, and no hierarchical data structure actually needs to be stored. For such an *implicit quadtree*, this reduces the storage cost effectively down to the elevation and approximation error values per vertex.

As shown in [43], for each point $P_{i,j}$ of the $2^k+1 \times 2^k+1$ height-field grid its level l in the implicit quadtree hierarchy can efficiently be determined by arithmetic and logical operations on the integer index values i and j (see also Fig. 14). Furthermore, it is also observed that the dependency relation of Fig. 12 can be expressed by arithmetic expressions as functions of the points index i, j. The implicit definition of quadtree levels and dependency relations between points by arithmetic functions allows the top-down and bottom-up algorithms presented in [42] to run very quickly and directly on the array of the height-field grid data instead of relying on a hierarchical pointer-based data structure. (See also [56] for efficient operations on quadtrees.)

An optimal output-sensitive triangulation algorithm is presented in [42] that exploits the strict error monotonicity achieved by *error saturation* (see Sect. 6). This allows for a simple top-down vertex selection algorithm, which does not have to resolve any restricted quadtree dependencies or propagate triangle splits at run-time. The proposed saturated error metric guarantees that the set of initially selected vertices for a given threshold automatically satisfy the restricted quadtree constraint and hence allow for a crack-free conforming triangulation.

To improve rendering performance, a triangle strip construction algorithm is presented in [42] that traverses the entire quadtree hierarchy instead of blocks as proposed in [34]. As shown in Fig. 15, the RQT triangle strip that recursively circles counterclockwise around each quadtree block center vertex is a *space filling curve* that visits all triangles exactly once. It also represents a *Hamiltonian* path in the dual graph of the triangulation. This triangle strip can be generated by a depth-first traversal of the quadtree in linear time, proportional to the size of the generated triangle strip. Moreover, the proposed error saturation technique in [42] and the quadtree-based triangle strip generation support a highly efficient unified vertex selection, triangle strip generation and rendering algorithm based on a single depth-first traversal of the implicit height-field quadtree.

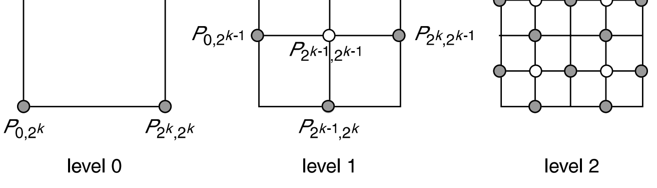

Fig. 14. Implicit quadtree hierarchy and point indexing defined on the height-field grid

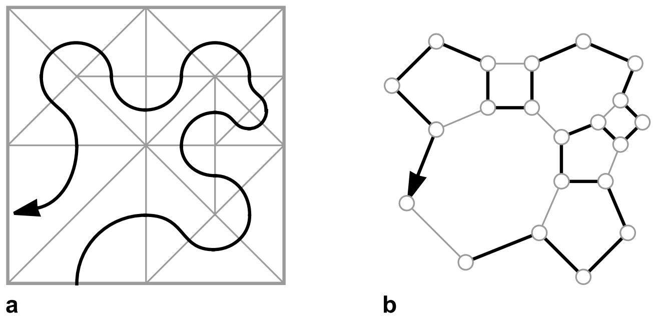

Fig. 15. **a** Generalized RQT triangle strip. **b** Its Hamiltonian path on the dual graph

The 4-8 meshes. The class of *4-8 meshes* [3,60,61] is based on a quadtree subdivision and triangulation as illustrated in Fig. 16, which in its triangulation power is basically equivalent to the other outlined quadtree and triangle bin-tree meshing approaches.

However, instead of a vertex dependency graph as in [34], a *merging domain* M_v is defined for each vertex v in [3] for the purpose of satisfying the triangulation constraints that avoid cracks in the surface mesh. As shown in Fig. 17, the merging domain M_v is basically the transi-

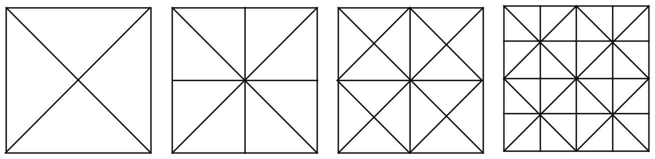

Fig. 16. Recursive 4-8 triangle mesh subdivision

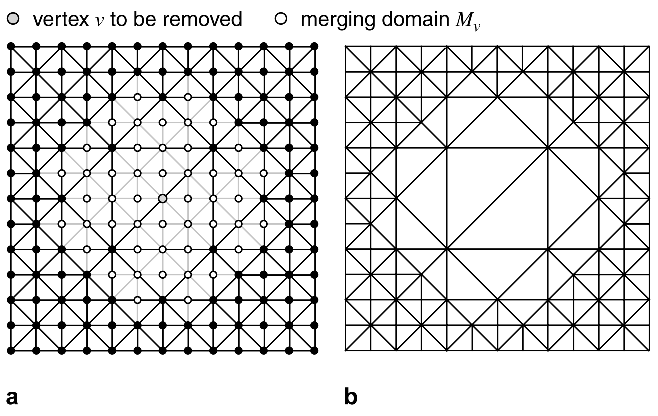

Fig. 17. **a** A vertex v and its merging domain M_v are highlighted. **b** The adaptive triangle mesh after removal of v and M_v

tive hull of all vertices depending on v in the dependency graph [34]. Consequently M_v is used to define all vertices that must be removed from the triangulation jointly with v. And hence, the removal of multiple vertices is constrained by the joint removal of the union of their merging domains. A similar concept, the *splitting domain*, is introduced for inserting vertices into the triangulation.

The triangulation algorithms presented in [2, 3] require $O(n \log n)$ time to refine or merge n nodes. This is in contrast to the algorithms presented in [34] and [42], which can generate an adaptive mesh of n triangles optimally in linear $O(n)$ time.

Irregular quadtree hierarchy. In [60, 61] it has been shown that arbitrary 3D surfaces can adaptively be triangulated by a hierarchical 4-8 triangulation approach, given a parameterization of the manifold surface is known. The *QuadTIN* approach presented in [44] goes one step further and defines a restricted quadtree hierarchy on top of any irregular point set in 2D, i.e., given from a preprocessed *triangulated irregular network* (TIN). As in [60, 61], the idea of QuadTIN [44] is based on the fact that points do not have to lie on a regular grid to allow for a regular hierarchical triangle subdivision as shown in Fig. 18.

At each subdivision step, the diagonal edge of a quadrilateral is not necessarily split at its midpoint, but using a nearby point from the input data set as shown in Fig. 19a. To avoid badly shaped triangles and inversion of orientation, however, the domain for searching for good input vertices is restricted as illustrated in Fig. 19b. If no good candidate vertices exist, artificial *Steiner points* are inserted to guarantee a coherent restricted quadtree triangulation hierarchy.

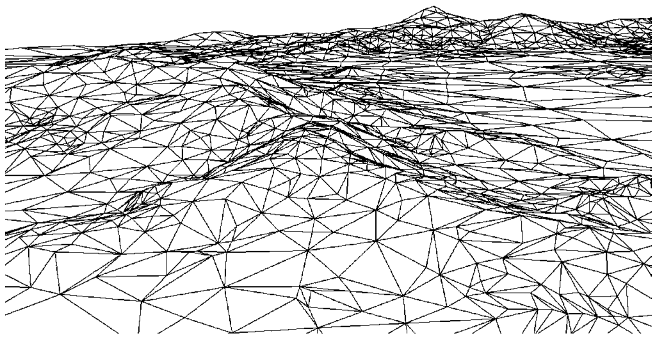
Fig. 20. Adaptive QuadTIN triangulation of an irregular distribution of elevation samples

An example adaptive QuadTIN-based terrain triangulation is shown in Fig. 20, which demonstrates its flexibility to adapt to an irregular input point data set. This added flexibility comes at the expense of extra points inserted into the data set.

4.2 Triangle bin-trees

In this section we discuss triangle bisection-based algorithms which generate equivalent triangulations of grid-digital terrain height-fields as the methods presented previously.

Real-time optimally adapting meshes. The *real-time optimally adapting meshes* (ROAM) triangulation method presented in [15] is conceptually very close to [34]. However, it is strictly based on the notion of a *triangle bin-tree* hierarchy as shown in Fig. 21, which is a special case of the *longest side bisection* triangle refinement method described in [51, 52]. This method recursively refines triangles by splitting their longest edge at the *base vertex* (see also Fig. 11).

As shown in Fig. 22, for a refinement operation a pair of triangles are split at the common base vertex of their shared longest edge, and a simplification operation consists of merging two triangle pairs at their common base vertex.

An important observation is that in a conforming triangulation, all neighbors of a triangle t on level l in the

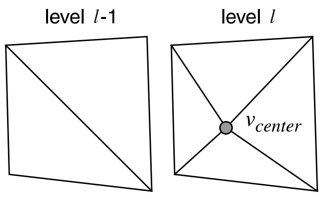

Fig. 18. Irregular recursive QuadTIN subdivision

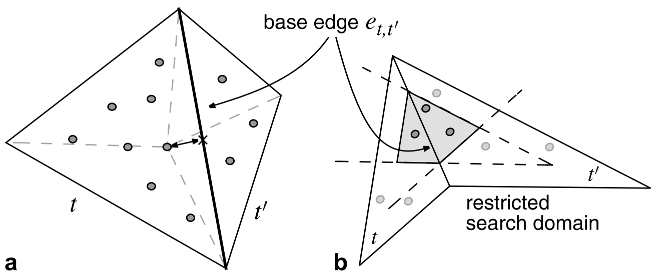

Fig. 19. a Vertex closest to the midpoint of diagonal edge $e_{t,t'}$ is selected for recursive subdivision. **b** Only vertices from a restricted search domain are considered

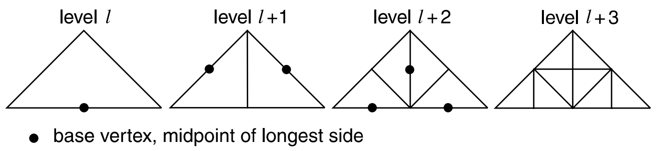

Fig. 21. Binary longest side bisection hierarchy of isosceles triangles with indicated split vertices on the longest side

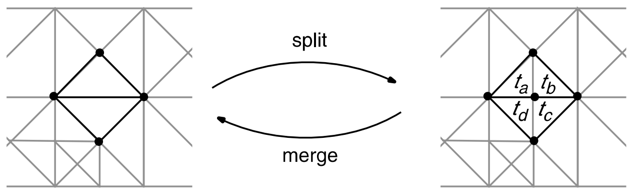

Fig. 22. Split and merge operations on a bin-tree triangulation

bin-tree hierarchy must be either on the same level as t, or on levels $l+1$ or $l-1$ of the bin-tree hierarchy. Therefore, two pairs of triangles t_a, t_b and t_c, t_d sharing the same base vertex can only be merged if they are all on the same level in the bin-tree hierarchy as shown in Fig. 22. Furthermore, a triangle t cannot be split immediately if its neighbor t_{long} across its longest edge is from a coarser level as shown in Fig. 23. In that case, triangle t can only be split if its corresponding neighbor is *forced* to split first. These forced splits are conceptually the same as the split propagation of [58] shown in Fig. 10. Moreover, the dependency relation of [34] in Fig. 12 denotes exactly the same forced split propagation of a bin-tree or restricted quadtree triangulation. All these concepts for assuring a conforming triangulation are equivalent in this context.

The run-time triangulation algorithm of ROAM is based on a greedy algorithm using two priority queues of the triangles $t \in T$ of the current mesh T: The split queue Q_s stores the triangles $t \in T$ according to their priority to be split next, and the merge queue Q_m maintains the mergible triangle pairs of T. For each frame the priority queues Q_m and Q_s are consulted and the current triangle mesh is adaptively refined or simplified accordingly to satisfy the given error threshold τ. The priorities are based on an error metric defined on triangles.

To guarantee a τ-approximation with respect to a particular error metric, the proposed greedy algorithm requires the error metric, and thus the priorities of Q_m and Q_s, to be *strictly monotonic*. This means that the error or priority of any triangle in the bin-tree hierarchy cannot be larger than its parent's priority. This monotonicity requirement limits the direct applicability of many standard error metrics. For example, neither the view-dependent error metric in [34] nor the vertical distance measure of [58] or the Hausdorff distance error metric defined hierarchically on removed vertices or triangles initially satisfy this monotonicity requirement (see also Sect. 6). Special care has to be taken to enforce monotonicity of any error metric by a bottom-up traversal of the triangle bin-tree hierarchy in a preprocess and calculating bounding priorities at each node.

Besides the two main contributions of ROAM which are the priority-queue driven triangle bin-tree-based triangulation method and a screen distortion error metric, the work in [15] contains a number of interesting contributions. A list of twelve criteria is given that generally apply to mesh simplification and in particular to large scale terrain visualization. Furthermore, a few performance enhancements that are implemented in ROAM are described including view frustum culling, incremental triangle strip generation, deferred priority recomputation, and progressive optimization.

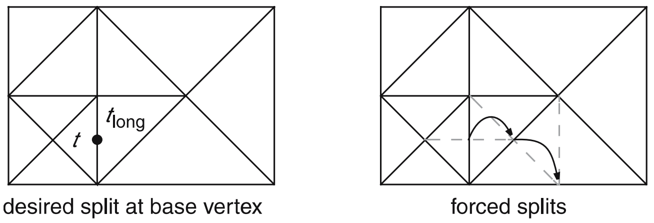

Fig. 23. Propagation of forced triangle splits

Right-triangulated irregular networks. The right-triangulated irregular network (RTIN) as presented in [16] is a multiresolution triangulation framework for the same class of triangle bin-tree meshes [15] as presented above. The RTIN approach is particularly focused on the efficient representation of the binary triangle hierarchy, and fast mesh traversal for neighbor-finding. Starting with a square triangulated by choosing one diagonal, triangles are split recursively at the base vertex or midpoint of their longest edge, identical to the method described above. To guarantee a conforming triangulation without cracks the same propagation of forced splits as shown in Fig. 23 is imposed on the RTIN triangulation. In [16] it is observed that split propagation caused by splitting a triangle t on level l_t cannot cause triangles smaller than t to be split (on levels $l > l_t$), and that at most two triangles on each level $l \leq l_t$ are split. Thus split propagation terminates in the worst case in $O(\log n)$ steps, with n being the size of the triangle bin-tree hierarchy (number of leaf nodes).

One of the main contributions of [16] is an efficient data structure to represent right-triangular surface approximations. Similar to Fig. 11, child triangles resulting from a split are labeled as *left* and *right* with respect to the split vertex of their parent triangle. A binary labeling scheme as shown in Fig. 24 is used in RTIN to identify triangular regions of the approximation. A RTIN triangulation is thus represented by a binary tree denoting the triangle splits and the elevation values (z coordinate) of the triangle vertices. The geographical (x and y) coordinates do not have to be stored for each vertex but can be computed from the triangles label. As noted in [16], a main memory implementation of such a binary tree structure with two pointers and three vertex indices[2] per node is space inefficient if used to represent one single triangulated surface approximation. However, a triangle bin-tree actually represents an entire

[2] This could be reduced to only one vertex index, others are known from parent triangle nodes.

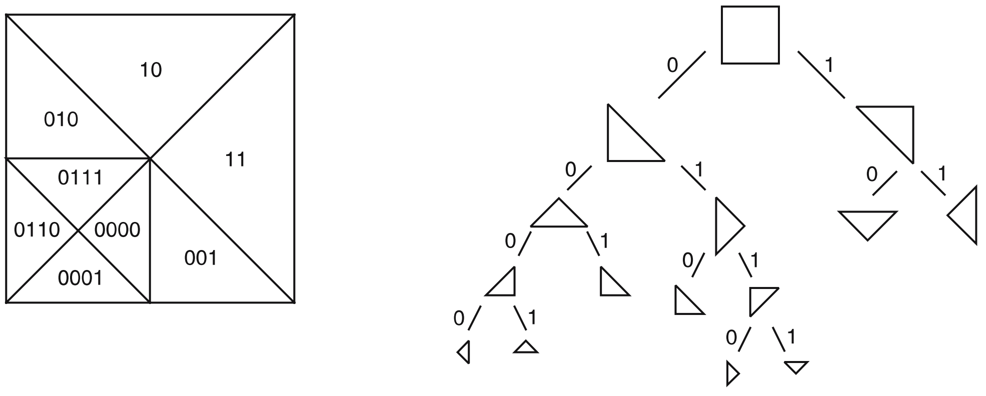

Fig. 24. RTIN triangle bin-tree labelling using 0 for *left* and 1 for *right*

hierarchy of triangulations. To reduce the storage cost of a triangle bin-tree hierarchy it is proposed to remove child pointers by storing the nodes in an array and using an array indexing scheme based on the node labels.

Based on the binary tree representation of the RTIN hierarchy as shown in Fig. 24, an efficient neighbor-finding scheme is the second main contribution of [16]. Given a counterclockwise numbering from v_1 to v_3 of the vertices of triangle t with vertex v_3 being the right-angled vertex, the *i-neighbor* of triangle t is defined as the adjacent triangle t_i that does not share vertex i. Furthermore, the *same-size i-neighbors* of any triangle are the edge adjacent triangles at the same level in the bin-tree hierarchy. For example, triangle 10 in Fig. 24 is the same-size 1-neighbor of triangle 11, and triangle 001 is the 3-neighbor of triangle 0000 but not a same-size neighbor. The neighbor-finding function $N_I(t)$ presented in [16] first finds the same-size i-neighbor of a triangle and then determines the actual i-neighbor for a particular triangulation. The recursive neighbor-finding function $N_I(t)$, that returns the label of the same-size i-neighbor of a given triangle t, is conceptually identical to a recursive tree traversal for finding adjacent regions in any binary space partition (BSP-tree; see also [53, 54]). An efficient non-recursive implementation of $N_I(t)$ based on arithmetic and logical operations is also given in [16].

For terrain visualization, each triangle is assigned an approximation error during the preprocess phase of constructing the RTIN hierarchy. At run-time, starting with the two triangles at the root of the RTIN hierarchy a depth-first traversal recursively splits triangles whose approximation errors exceed a given tolerance threshold. Forced splits are propagated to the corresponding i-neighbors to avoid cracks in the triangulated surface approximation.

The main focus of RTIN is efficient representation of the triangle bin-tree hierarchy and neighbor-finding techniques on the adaptively triangulated surface. Similar to [15, 34, 42], RTIN is efficient in creating an adaptive surface triangulation since its top-down algorithm is output-sensitive. In fact, the RTIN approach is almost identical to the ROAM method and only differs in notation and representation of the triangle bin-tree hierarchy. No detailed algorithms are given in [16] on how to incorporate propagation of forced splits to generate a conforming triangulation.

Right-triangular bin-tree. In [19], the class of restricted quadtree or right-triangular bin-tree triangulations is studied with respect to efficient data storage and processing, search and access methods, and data compression. It is proposed to always manage the data in compressed form, even interactive processing is performed on the compressed data. The multiresolution triangulation framework in [19] follows the binary triangle hierarchy approach as used in [15] and [16]. To prevent cracks in the triangulation resulting from recursive triangle bisection, error saturation is used as presented in [42].

The main contribution of [19] is a compressed representation of the triangle bin-tree hierarchy based on an efficient mesh traversal and triangle numbering scheme. The traversal order of triangles in the bin-tree hierarchy is equivalent to the triangle strip ordering as shown in Fig. 25. Furthermore, each triangle is numbered such that the left child of a triangle with number n receives the number $2n$ and the right child is numbered $2n+1$ if the level l of the parent triangle is odd and vice versa if it is even as shown in Fig. 25. For a given triangle, bit-wise logical operations can be used to compute the adjacent triangle that shares the common refinement vertex. Each vertex is associated with the two numbers of the triangles that it refines.

This ordering and triangle numbering imposes a binary classification of triangles in a conforming bin-tree triangulation into up- or down-triangles. In a depth-first traversal of the bin-tree hierarchy, an *up-triangle* can only be followed by a triangle on the same or higher level (coarser

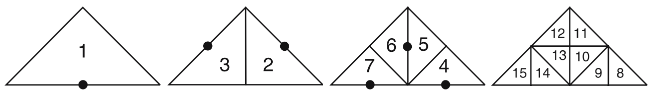

Fig. 25. Triangle numbering

triangle) in the hierarchy. Similarly, a *down-triangle* can only have a neighbor on the same or lower level of the hierarchy. Therefore, the starting triangle and one bit per triangle is sufficient to encode an adaptive bin-tree triangulation. Furthermore, vertices only need to be specified on their first occurrence in the bin-tree traversal. Based on this traversal and numbering technique an efficient compressed representation of a triangle bin-tree hierarchy is proposed. Moreover, it is shown how an arbitrary adaptive triangulation can efficiently be extracted from the code stream that represents the entire bin-tree hierarchy, and that can be read and processed efficiently from disk.

The triangulation algorithm and data structure presented in [19] are particularly tailored towards efficient representation and traversal of the binary triangle hierarchy. The proposed encoding of the triangle bin-tree is very interesting from the point of view that it can be used to access an adaptive triangulation efficiently even if the bin-tree is stored sequentially on disk. The proposed multiresolution framework provides most of the important features such as continuous LOD, fast rendering, and compact representation.

4.3 Discussion

The different multiresolution terrain triangulation approaches reviewed in this section all contribute unique features and improvements to the class of restricted quadtree and bin-tree triangulations. The basic adaptive multiresolution triangulation framework has been introduced in [58]. The approaches of [34] and [42] follow this concept of an adaptive quadtree hierarchy, while the methods presented in [15, 16] and [19] describe the same class of triangulations from the point of a binary triangle subdivision hierarchy.

Very efficient triangulation algorithms are the focus of [34, 35, 42] and [5], which are based on a simple vertex selection strategy, and [15], which is based on a prioritized triangle merge and split concept. Error saturation conforming to the restricted quadtree triangulation constraints introduced in [42] and [19], has been extended to efficient view-dependent error metrics and LOD selection algorithms in [20, 35] and [5]. While effective, most other triangle bin-tree-based approaches are slightly more complex due to recursively splitting triangles and resolving propagated forced splits, and thus have some disadvantages compared to the simple quadtree-based vertex selection algorithms. All surveyed methods are capable of generating smooth adaptive LODs for efficient terrain surface approximation, and, though not explicitly described, RTIN [16] can generate triangle strips for fast rendering.

The main objective of this kind of algorithms was to compute on the CPU the minimum number of triangles to render each frame, so that the graphic board was able to sustain the rendering. More recently, the impressive improvements of the graphics hardware both in terms of computation and communication speed shifted the bottleneck of the process from the GPU to the CPU. In the next section we will show how these methods can be made more efficient in terms of raw triangle throughput by employing cluster-based approaches.

5 Cluster triangulations

The impressive improvement of graphics hardware in terms of computation and communication speed is reshaping the real-time rendering domain. A number of performance and architectural aspects have a major impact on the design of real-time rendering methods.

Today's GPUs are able to sustain speeds of hundreds of millions of triangles per second; this fact has two important implications for real-time rendering methods. First of all, to sustain such speeds, the CPU workload of the adaptive rendered has to be reduced to a few instruction cycles per rendered triangle. Second, since the target rendering speed is two orders of magnitude larger than the number of screen pixels, there is an expectation for high quality scenes with millions of triangles per frame. On classic vertex- or triangle-based structures, managing and storing very large dependency graphs at run-time becomes a major bottleneck, mostly due to random-access traversals with poor cache-coherence. Moreover, current GPUs are optimized for retained mode graphics, and their maximum performance is obtained only when using specific preferential data paths. This typically means using stripified, indexed, and well packed and aligned primitives to exploit on-board vertex caches and fast render routes. In addition, the number of primitive batches (i.e., the number of *DrawIndexedPrimitive* calls) per frame has to be kept low, as driver overhead would otherwise dominate rendering time [64]. Finally, maximum performance is only obtained when rendering from on-board memory. Editing on-board memory introduces however synchronization issues between CPU and GPU, which is a problem for dynamic LOD techniques. In this setting, approaches which select, at each frame, the minimum set of triangles to be rendered in the CPU typically do not have a sufficient throughput to feed the GPU at the top of its capacity, both because of the per-triangle cost and the complexity associated with sending geometry in the correct format through preferential paths. Since the processing rate of GPUs is increasing faster than that of CPUs, the gap between what could be rendered by the graphics hardware and what the CPU is able to compute on-the-fly to generate adaptive triangulations is doomed to widen.

For such reasons many techniques have been recently proposed to reduce the per-primitive workload by composing at run-time preassembled optimized surface patches, making it possible to employ the *retained mode* rendering model instead of the less efficient direct rendering approach for all CPU/GPU communication tasks. The main common point of these methods, that we call here *cluster triangulations*, is that they move the LOD unit up from points or triangles to small contiguous portions of a mesh.

5.1 Tiled blocks

A classic example of a cluster triangulations approach are tiled blocks techniques [26, 63], which partition the terrain into square patches tessellated at different resolutions. A full survey of this subject is beyond this paper, devoted to quadtree approaches. We restrict our presentation to [55], which proposes a combination of tiled blocks and restricted quadtree triangulations. The method strives to improve CPU/GPU communication efficiency by an incremental batched communication of updates. In this approach, the terrain mesh is partitioned into equal tiles of size 257×257, with an overlap of one sample in either direction. For each tile, a fixed set of restricted quadtree meshes of increasing error is generated, resulting in a nested mesh hierarchy per tile. At run-time a specific LOD is selected independently for each tile, and the relevant updates are sent to the GPU. Each finer level is represented by all coarser level vertices plus the additional ones. By caching the current mesh on the GPU, only the additional vertices need to be sent, reducing the required bandwidth by 50%. Since vertices are transferred by groups, efficient vertex array techniques can be employed to boost transfer efficiency. In [55] all vertex data for a given tile is stored in a single vertex array, which grows by a block for each LOD, while the connectivity is stored in a separate per level element array. In order to smoothly transition surface changes, the method exploits the concept of *geomorphing* [27], which interpolates vertex attributes between LODs.
The main challenge for this technique, as for all tiled block techniques, is to seamlessly stitch block boundaries, which requires extra run-time work. In [55] boundaries of neighboring tiles are detected and connected using run-time generated triangles. This need to remesh boundaries is avoided in the quadtree-based techniques that will be presented next. Moreover, the technique is not fully adaptive, and limits simplification to pure subsampling, in order to support progressive vertex transmission.

5.2 Cached triangle bin-trees

RUSTIC [47] and CABTT [31] are both extensions of the ROAM [15] algorithm that improve rendering performance through the addition of coarse-grained on-board caching. RUSTIC is an extension of the basic ROAM algorithm in which preprocessed static subtrees of the ROAM triangle bin-tree are used. The CABTT approach is very similar to RUSTIC, but triangle clusters are dynamically created, cached and reused during rendering. Triangle clusters form the unit for LOD refinement/coarsening operations, and can be cached on the GPU as vertex arrays. Improved performance over ROAM is gained by rendering the meshes as triangle strips. Since all adaptively refined graphs are still ROAM graphs, adaptive triangulations are guaranteed to be conforming.

These methods demonstrate the performance benefits of coarse-grain LOD adaptation, but limited its application to geometry caching. A particular contribution of these methods was to show that, even though the number of triangles per frame increased by a factor of 50%, with respect to ROAM, the overall rendering performance was boosted by a factor of four due to the order of magnitude raw performance increase of the rendering interface.

5.3 Combining regular and irregular triangulations

BDAM [7], P-BDAM [8], and HyperBlock-QuadTIN [30] generalized the caching approach by combining regular and irregular triangulations in the same GPU-friendly framework. The main insight of these methods is to separate the coarse topology of the multiresolution method, managed using semi-regular fine geometry of the objects, managed using triangulations. In other words, the task of the multiresolution structure is to generate adaptive regular partitions of the terrain domain using data-independent techniques, while the task of the geometry is to approximate the data inside the partition with a fixed triangle count mesh with appropriate boundary constraints.

HyperBlock-QuadTIN. QuadTIN [44] is an efficient quadtree-based triangulation approach to irregular input point sets with improved storage cost and feature adaptive sampling resolution. It preserves a regular quadtree multiresolution hierarchy over the irregular input data set (see Sect. 4.1). HyperBlock-QuadTIN [30] extends the basic QuadTIN [44] method by creating a coarse-grained tree structure of blocks that store different triangulation levels. Similar to the clustering performed by RUSTIC [47] and CABTT [31] on ROAM hierarchies but with the additional advantage of direct support of irregular point sets. The construction process starts by a full QuadTIN hierarchy, which is then clustered into fixed size blocks by traversing it coarse to fine. At run-time, the coarse block hierarchy is traversed, and resolution levels are selected on a block-by-block basis. A global crack-free triangulation is ensured by adjusting the selected block levels so that they meet restricted quadtree constraints. A simplified illustration of an example of restricted quadtree blocks of HyperBlock-QuadTIN [30] is given in Fig. 26.

Batched dynamic adaptive meshes (BDAM). The BDAM approach [7] seamlessly combines the benefits of TINs and restricted quadtree triangulation in a single data structure for multiresolution terrain modeling. BDAM is a specialization of the more general batched multitriangulation framework [9]. It is based on the idea of exploiting the

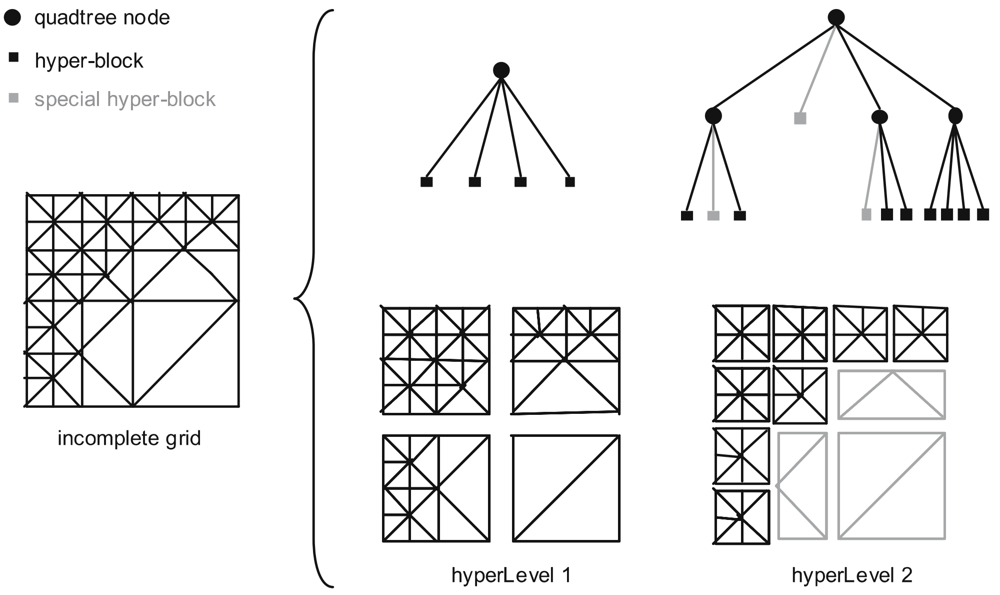

Fig. 26. Adaptive elevation grid and the corresponding LOD hyper-blocks of levels 1 and 2

partitioning induced by a recursive subdivision of the input domain in a *hierarchy of right triangle* clusters to generate a coarse-grained multiresolution structure. The partitioning consists of a forest of triangle bin-trees (see also Sect. 4.2) covering the input domain.

The partitioning consists of replacing a triangular region σ with two triangular regions obtained by splitting σ at the midpoint of its longest edge [51, 52]. To guarantee that a conforming mesh is always generated after a bisection, the two triangular regions sharing σ's longest edge are split at the same time. These pairs of triangular regions are called *diamonds* and cover a square. The dependency graph encoding the multiresolution structure is thus a DAG with at most two parents and at most four children per node (conceptually the same as in [34]).

This structure has the important property that, by selectively refining or coarsening it on a diamond-by-diamond basis, it is possible to extract conforming variable resolution mesh representations. BDAM exploits this property to construct a coarse-grained LOD structure. This is done by associating to each triangle region σ a small triangle patch, up to a given triangle count, of the portion of the surface contained in it. Each patch is constructed so that vertices along the longest edge of the region are kept fixed during a diamond coarsening operation (or, equivalently, so that vertices along the shortest edge are kept fixed when refining). In this way, it is ensured that each mesh composed by a collection of small triangle patches arranged as a triangle bin-tree generates a globally correct and conforming triangulation (see Fig. 27).

At run-time, the LOD is chosen by a triangle bin-tree refinement over the triangle patches (based on saturated error [7, 8] or incremental refinement based on a dual queue technique [9]). The selection cost is thus amortized over patches of thousands of triangles.

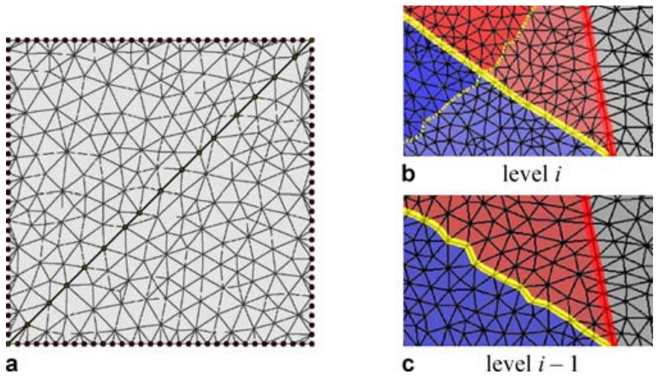

Fig. 27. a BDAM triangle clusters of a diamond structure. Coarsening of two diamonds in **b** to one in **c** with coarsening vertices along the shared boundary (in *yellow*). Highlighted vertices (in *red*) shared with neighboring diamonds remain unchanged

The highest resolution triangle patches sample the input data at a matching resolution, while coarser level patches contain TINs constructed by constrained edge-collapse simplification of child patches. In a preprocess, simplification is carried out fine-to-coarse level-by-level, and independently for each diamond. The whole simplification process is inherently massively parallel, because the grain of the individual task is very fine and synchronization is required only at the completion of each bin-tree level (see also Fig. 28).

5.4 The 4-8 mesh cluster hierarchies

An approach similar to BDAM, but described in terms of a 4-8 mesh hierarchy and optimized for regular grids is introduced in [28]. The authors remark that, with current rendering rates, it is now possible to render adaptive

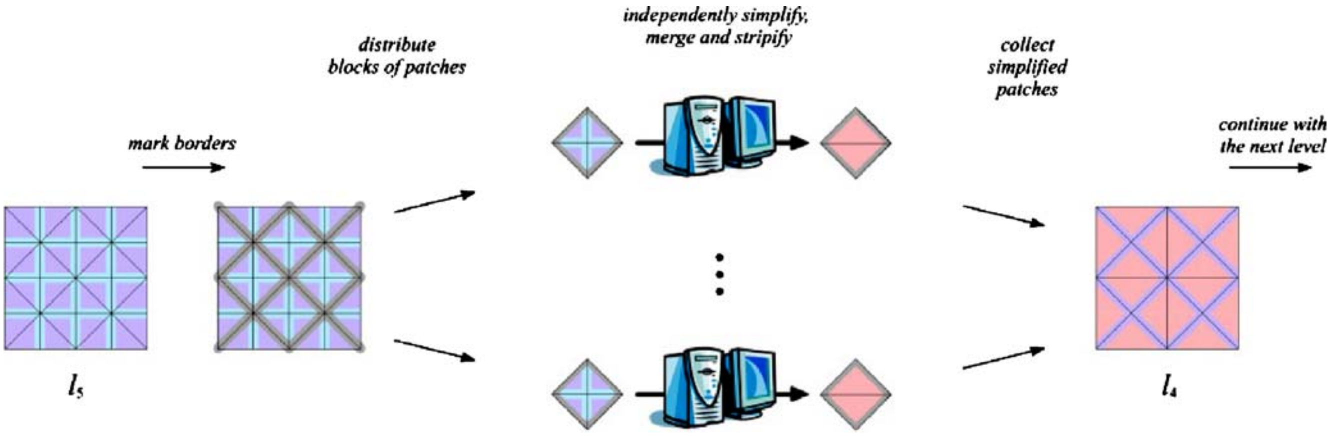
mark borders
distribute blocks of patches
independently simplify, merge and stripify
collect simplified patches
continue with the next level
l_5
l_4

Fig. 28. Construction of a BDAM through a sequence of (parallel) simplification and marking steps. Each *triangle* represents a terrain patch composed by many triangles, as in Fig. 27

scenes with triangles that have a projected size of one or a few pixels. At this point, it is no longer desirable to make triangles non-uniform in screen space due to variations in surface roughness, since this will only lead to subpixel triangles and thus to artifacts. The authors therefore rewrite the BDAM approach in terms of regular grids, replacing geometric patch simplification with low-pass filtering. In addition, while the original BDAM work encoded the hierarchy with triangle bin-trees, this work explicitly encodes the graph of diamonds, and incrementally refines and coarsens it using ROAM's dual queue incremental method. Another contribution of the work is that geometry and texture are handled in the same framework. That is, both geometry and textures are treated as small regular grids, called tiles, defined for each diamond in the hierarchy. Each grid corresponds to two patches sharing the main diagonal. The relative density of the grids are adjusted to maintain a fixed ratio of texels per triangle.

6 LOD error metric

In this section we review the major error metrics that have been proposed for the discussed terrain triangulation algorithms.

6.1 Object-space approximation error

To render deformed parametric surfaces, several recursive subdivision criteria are given in [62] that take into account local curvature, intersection of surfaces, and silhouette boundaries. While these subdivision criteria are not directly applicable to terrain height-fields, the local curvature criterion, or flatness, is similar to other geometric approximation error metrics used for terrain triangulation.

The approximation error proposed in [58] is the vertical distance of a removed vertex with respect to its linear interpolation provided by the parent node as shown in Fig. 29. The error of vertex B is its vertical distance to the average elevation of A and C. An example of mergible nodes, with respect to Sect. 4.1, is given in Fig. 29. Given that the approximation error of all removed vertices (outlined points in Fig. 29b) is within the given tolerance, and given that no other neighboring nodes violate the restricted quadtree constraint, the nodes and triangles of Fig. 29a can be merged into the larger node Fig. 29b.

A major problem of the proposed bottom-up quadtree initialization in [58] is the computation of the approximation error metric. While the vertical distance of a removed vertex (B in Fig. 29) with respect to its linear interpolation (line between A and C in Fig. 29) in the immediate parent node may be below a given error threshold τ, it is not clear that this removed vertex is within the distance τ to the final result of an iterative bottom-up merging process. As shown for a 2D example in Fig. 30, this error metric is not monotonic. In fact, the resulting simplified surface based on this method does not interpolate the removed vertices within a bounded distance.

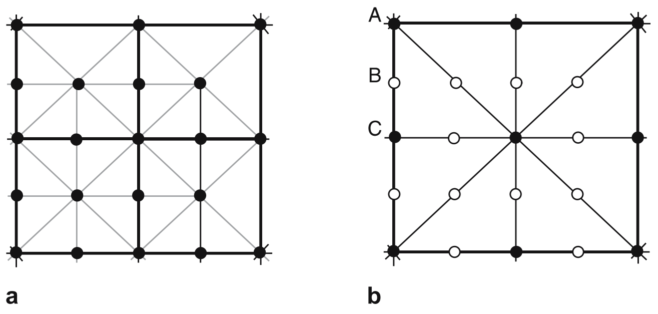

Fig. 29a,b. Initial four leaf nodes shown in **a** that are merged in **b** with the outlined points denoting the removed vertices in the merged node

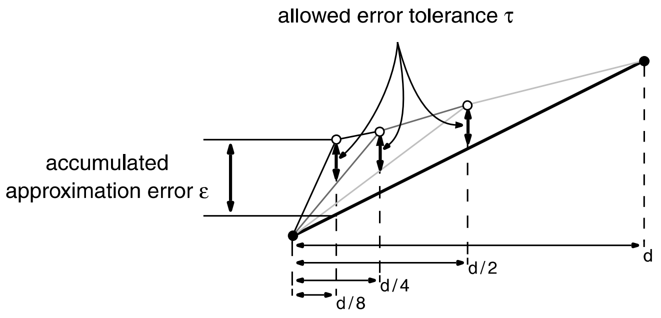

Fig. 30. Merging of nodes satisfying the approximation error threshold locally may result in intolerably large accumulated errors with respect to the final result

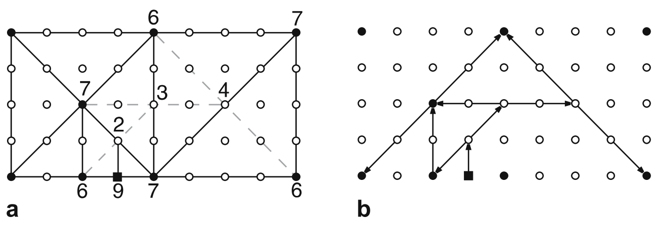

Fig. 31. Initial error metric shown in **a** for selected vertices, *white* vertices are below and *black* vertices above the error threshold $\tau = 5$. Forced splits are indicated with *dashed gray lines*. Propagation of error saturation shown in **b** for the vertex causing the forced splits

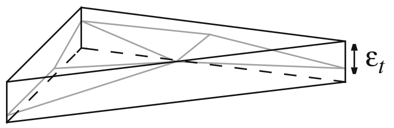

Fig. 32. The thickness of a bounding wedgie defines an object-space geometric approximation error

However, the top-down triangulation approach in [58] computes the distance to the original surface for each vertex with respect to the current adaptive restricted quadtree surface approximation. Therefore, no accumulation of errors beyond the given threshold τ can occur, and the reconstructed surface map is a correct τ-approximation.

A similar vertical distance measure has been used in [16] and [42], modified to satisfy the monotonicity requirement outlined in [15]. However, in contrast to [16], and also [15], which define the error metric on triangles, the RQT approach [42] defines the error metric on vertices. If precomputed per triangle it is straight forward to make the error metric monotonic, setting it to the maximum distance of vertices within the domain of the triangle. However, a geometric approximation error attribute has to be stored for each triangle that can be formed by the adaptive multiresolution triangulation method. This can be quite a costly approach in terms of memory usage as this number is several times larger than the number of input elements (elevation values). The per-vertex error metric proposed in [42] eliminates this memory cost.

It has been observed in [42, 43] and [41] that for object-space geometric error metrics the dependency graph shown in Fig. 12 can be encoded into the error metric itself by a technique known as *error saturation*. As demonstrated in Fig. 31a, the selection of a particular vertex P (black square) due to its error value $\varepsilon = 9$, exceeding the allowed tolerance $\tau = 5$, causes several forced triangle splits (dashed grey lines). To avoid such forced splits, error values are propagated and maximized along the dependency graph, as shown in Fig. 31b. This error saturation is performed in the preprocess: Each vertex stores the maximum value of all propagated errors and its own computed error, and propagates this maximum further along the dependency graph. This preprocess can be implemented by a simple traversal over the grid-digital elevation values. Therefore, a fast top-down selection of vertices according to their saturated error metric directly yields an adaptive and conforming triangulation of a restricted quadtree, without the need of enforcing any quadtree constraints, forced splits or resolving dependency relations. This error saturation technique has also been observed in [22] and can be applied in various ways to enforce constraints on multiresolution hierarchies such as topology preservation in isosurface extraction [21].

Other geometric distance metrics, instead of the vertical offset measure, must be treated in a similar way to preserve monotonicity for an efficient output-sensitive top-down adaptive mesh refinement approach.

An object-space geometric approximation error metric is defined in [15] by calculating for each triangle t in the bin-tree hierarchy the thickness ε_t of a bounding *wedgie* that encloses all children of its subtree as shown in Fig. 32. This measure bounds the maximal deviation of a simplified mesh with respect to the full resolution input data; however, it has to be computed and stored for every triangle that can possibly be defined by the multiresolution hierarchy. This basic object-space approximation bound is input to a view-dependent image-space error metric as discussed in the following section.

6.2 Image-space approximation error

A static object-space geometric error metric is not sufficient to adaptively simplify terrain for perspective rendering. This is because far away regions must be simplified more aggressively than nearby areas. As this depends on the observer's location and changes continually, the error metric must be defined dynamically and view-dependently.

In [34] the definition of an efficient view-dependent image-space error metric has been proposed that determines removal or inclusion of vertices for a given viewpoint. As illustrated in Fig. 33, the basic idea of this error metric is that triangle pairs can be merged if the change in

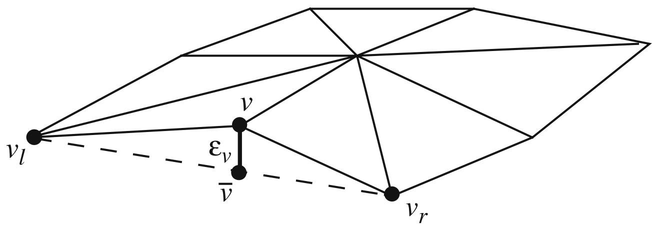

Fig. 33. Vertical distance ε_v between removed base vertex v and its linear interpolation $\bar{v}$

slope ε_v at the removed base vertex v projected into screen space is smaller than a given threshold τ. The line segment $\varepsilon = v - \bar{v}$ between the removed base vertex v and its linear interpolation $\bar{v} = (v_l + v_r)/2$ is perspectively projected onto the screen space viewing plane as ρ_v. If ρ_v is smaller than the tolerance τ then the vertex v can be removed and the corresponding triangle pairs merged. Note that the projected delta segment ρ_v is not defined with respect to the highest resolution mesh or the current LOD mesh but rather based on the adjacent vertices v_l and v_r of the next lower resolution in the quadtree. Therefore, although hardly noticeable in practice, the metric as defined in [34] suffers from the same limitations as the error computation of the bottom-up triangulation method presented in [58] and does not provide a guaranteed error bound on the final triangulation. For this, the error metric must either be saturated correctly, or defined and maximized on each triangle with respect to the full resolution mesh.

For efficient block-based mesh simplification, the view-dependent image-space error metric is extended to entire quadtree blocks in [34]. In particular, if for a quadtree region R the maximum delta projection of all higher resolution vertices within R is smaller than the threshold τ then they can be ignored. For an axis-aligned bounding box of a quadtree block R and given viewing parameters, one can compute the smallest elevation delta ε_l and largest ε_h of that box that when projected onto screen may exceed τ. Therefore, if the maximum vertical error $\varepsilon_{\max}$ of all vertices $v \in R$ is smaller than ε_l then R can be replaced by a lower LOD block, and if $\varepsilon_{\max}$ is larger than ε_h then R has to be refined into smaller blocks. Otherwise the screen space projected errors ρ_v of all vertices $v \in R$ have to be computed and compared to τ individually.

The thickness ε_t of a bounding wedgie as introduced in [15] (see Fig. 32) can be used to estimate the maximal image-space distortion ρ_t of a triangle t for view-dependent simplification similar to the approach presented in [34]. Consequently, for any given triangulation T, its image-space distortion can be bounded by the maximum projected length ρ_t of all triangles $t \in T$. Additionally to this image-space distortion error metric, [15] proposes several other mesh refinement and simplification measures such as: backface detail reduction, surface normal distortion, texture-coordinate distortion, silhouette preservation, view frustum culling, atmospheric or depth attenuation, and region of interest.

In [35, 36] and [20] it has been observed that view-dependent error metrics can also, in a sense, conservatively be saturated similar to [42] for object-space measures. This works if the image-space error metric ρ_v of a vertex v is based on a static geometric approximation error ε_v, which is perspectively projected into image-space (divided by d_v given the distance d_v of the vertex v to the viewer). For this to work, additionally to ε_v, a conservative bounding sphere radius r_v is needed for each vertex. This attribute r_v defines a nested bounding sphere hierarchy on the restricted quadtree vertex dependency graph [35, 36]. A vertex v will be selected for the current LOD triangulation if its conservative image-space error $\rho_v = \frac{\varepsilon}{d_v - r_v}$ is larger than the given threshold τ.

In SMART [5] the same basic error metric and view-dependent vertex selection criterion $d_v < \frac{\varepsilon_v}{\tau} + r_v$ gives rise to a τ-sphere defined for each vertex by the radius $r_v^\tau = \frac{\varepsilon_v}{\tau} + r_v$. Hence vertex selection is simplified to all vertices whose τ-spheres contain the viewpoint. Furthermore, it is elaborated in [5] that a so-called τ-save-distance can dynamically be maintained, which bounds for each vertex the deviation of the viewpoint that does not change the LOD level of the vertex. This concept allows for optimized LOD computations as well as efficient vertex caching, and results in significantly improved LOD meshing and rendering performance.

6.3 Discussion

The error metric of triangle bin-tree approaches is defined on the triangles in the binary hierarchy. Due to the property of a binary tree having roughly $2n$ nodes for n leaf nodes and a triangle mesh having $2n$ triangles for n vertices, storage of a triangle-based error metric requires maintaining about $4n$ error values. In contrast, the quadtree-based approaches define the error metric on vertices and only require n error values to be stored. Simple geometric approximation error metrics based on vertical displacement can be found in [16, 42, 58] and [19]. More sophisticated view-dependent error metrics such as screen space distortion are discussed in [34] and [15], and saturated view-dependent error metrics are presented in [35] and [5]. Projection of a global geometric approximation error metric into image-space will be most efficient for large scale terrain visualization in practice. In [15] it was observed that an error metric must be hierarchically monotonic to guarantee ε-bounded approximations. RTIN [16] and RQT [42] in object-space as well as SOAR [35, 36] and SMART [5] in image-space provide such monotonic geometric error metrics.

The type or error metric and error representation has thus important consequences also on structure size and efficiency. Arguably the most space efficient representation of a multiresolution triangulation of a height-field is an implicit hierarchical structure, embedded in an array, with a saturated error metric defined on the grid of elevation values as proposed in [42] and [35]. This representation does not require any information to be stored that describes the structure of the multiresolution hierarchy, and only needs the elevation and error values for each grid point. Furthermore, such an elevation grid can also efficiently be partitioned and stored on a remote server as shown in [42] and [43], or mapped linearly to disk as demonstrated in [35]. However, this fully implicit representation is only possible if the tree is complete, i.e., if the input data is a uniformly sampled square.

Other related techniques for the efficient representation and compression of a triangle bin-tree hierarchy are discussed in [16] and [19]. However, both approaches use triangle-based error metrics, which are space inefficient due to the large number of error values that have to be stored. Efficient LOD-based spatial access and triangulation is discussed in [42], and extraction of an adaptive triangulation in a sequentially stored and compressed triangle bin-tree representation is considered in [19]. Very efficient representations are further achieved in cluster-based triangulation approaches such as [7, 28, 63], since errors and other structural information is only stored per cluster.

7 System issues

In this section we want to briefly review a few system and database level aspects of terrain visualization in conjunction with the LOD triangulation and rendering algorithms discussed so far. This includes topics such as dynamic scene management, progressive or incremental meshing, data storage and retrieval, or client-server architectures that are important for large scale real-time terrain visualization systems.

7.1 Dynamic scene management

Most of the discussed real-time terrain triangulation and visualization algorithms assume the entire terrain data set to be accessed directly in virtual memory and do not explicitly consider dynamically loading terrain from disk or from a database server. Also, most algorithms can dynamically extract a particular LOD triangle mesh from a hierarchical multiresolution data structure holding the terrain data.

Fully main-memory resident approaches generally generate a space-LOD query for each rendered frame given the current view frustum and LOD tolerance threshold t settings. This query is generally answered using the multiresolution terrain triangulation hierarchy. Efficient recursive top-down LOD selection and triangulation algorithms for real-time terrain rendering are presented in [15, 16, 34, 42, 58], of which [34] and [42] address the issue of out-of-core data management and are discussed in the following section.

Specifically designed for fast real-time LOD triangulation and rendering in main memory is the system presented in [15] (ROAM). As discussed in Sect. 4.2, the run-time triangulation algorithm of ROAM is based on a greedy algorithm that maintains two priority queues, the split queue Q_s and the merge queue Q_m. For each frame the priority queues Q_m and Q_s are used to incrementally simplify and refine the current triangle mesh to reach a triangulation that satisfies the given error threshold t. The priorities of Q_s and Q_m are based on the error metric defined on the triangles.

The ROAM terrain rendering system [15] is designed to support guaranteed frame rates in an interactive visualization application. Despite the maintenance of priority queues at run-time, which requires order $O(n \log n)$ cost for each update, the method is efficient as it is output-sensitive (for monotonic error metrics) and because the triangulation can be updated incrementally between rendered frames. In addition to the basic algorithms, a couple of system-level issues are discussed as well such as reducing the amount of CPU time spent on updating priorities between frames, or limiting the number of split and merge operations to bound the triangle count and guarantee consistent frame rates.

Clustered triangulation approaches typically adapt their representation by traversing an in-core structure that represents the coarse-grained multiresolution models. BDAM [7], and P-BDAM [8] use a top-down refinement approach based on saturated errors and bounding volumes similar to SOAR [35, 36]. The 4-8 texture hierarchy system [28] use instead the dual queue approach of ROAM [15]. All these systems maintain in-core the dependency graph (for a conforming triangulation) and incrementally fetch from external memory the required LOD data. The choice of the particular refinement strategy is less important for clustered triangulation approaches than for the other methods surveyed here, since the run-time dependency graph size is output-sensitive and small, and all operations are amortized over thousands of rendered triangles.

7.2 Out-of-core data organization

While early systems such as VGIS [29, 33, 34], ViRGIS [27, 42, 45, 46] and TerraVision II [50], as well as extremely successful viewers such as Google Earth or NASA Worldwind, generally manage the terrain data as a set of rectangular elevation grid tiles, more recent approaches [5, 35, 36] use clever indexing and, when possible, memory mapping techniques.

In tile-based systems the terrain data can easily exceed the main (or even virtual) memory capacity of the workstation used for rendering as the data is dynamically loaded on-demand from disk. VGIS [29, 33, 34] maintains the terrain data on disk partitioned into a hierarchy of blocks of 129×129 vertices each. Hence at runtime, retrieval of the terrain data from disk is based on block access at fixed grid resolutions. In main memory a partial global terrain quadtree is maintained, and updated dynamically by loading elevation data blocks on demand from disk. Adaptive simplification is performed from this in-core data for each frame based on the view-dependent block- and vertex-level error metrics discussed in Sect. 4.1.

In ViRGIS [42, 45, 46], a tiled sliding window concept is applied that dynamically maintains a fraction of the entire data set in main memory, similar to Fig. 1. A dynamic scene manager dynamically updates the set of visible tiles, by loading from disk on-demand, and maintains each tile itself as a RQT. To avoid excessive loading from disk, a strategy of *deferred cumulative updates* is proposed, which incrementally updates grid tiles in-core based on the required additional LOD. A multi-client capable terrain server manages the elevation data in a quadtree database structure, supporting LOD-based rectangular range queries as well as LOD-interval range queries for incremental tile updates. Given a rectangular query range R an adaptive triangulation for any specified LOD-interval can be retrieved as indicated in Fig. 34. In that process, the boundary ∂R of the query region R is resolved such that a conforming triangulation of the query region R is generated.

Instead of using grid tiles to partition the elevation data, [4] combines spatial grouping with a LOD priority to cluster elevation data on disk. Starting with a simple group of vertices of the restricted quadtree triangulation hierarchy, a cluster is formed by recursively adding same, or similar LOD child nodes until the size limit for a single cluster is reached, or the LOD priority of the vertices in the cluster exceeds a tolerated evenness bound. Hence each cluster forms a part of the quadtree hierarchy structure and preserves spatial selectivity as well as uniform LOD distribution within the cluster.

In [27] the entire terrain data set is block-partitioned into quadratic patches on disk. Each patch may be pre-simplified to a minimum tolerance and stored on disk, however, block boundaries are preserved at the finest resolution to guarantee conforming triangulations. In main memory, the interior of each quadratic block is adaptively simplified for each frame. Simplification across the highly tessellated block boundaries is performed in a second stage, after block-internal simplification, to reduce artifacts between block regions.

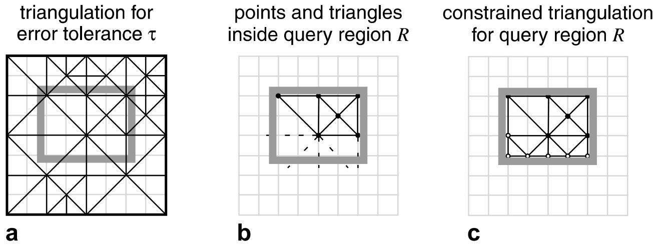
triangulation for error tolerance τ
points and triangles inside query region R
constrained triangulation for query region R
a
b
c

Fig. 34a–c. Rectangular range query shown in **a** and initial vertex selection given in **b**. RQT constraints are enforced on the range query as shown in **c**

A different approach for out-of-core memory management of multiresolution data has been presented in [35, 36], which relies purely on the virtual memory management functionality of modern operating systems. The basic principle is to sequentially order the grid-digital elevation samples based on a hierarchical, recursively defined *space-filling curve* indexing scheme [1]. The space-filling property of such an index preserves spatial proximity between index neighbors, and the hierarchical definition, e.g., of the z-curve index as used in [35] provides a basic LOD ordering. The multiresolution restricted quadtree, or bin-tree triangulation hierarchy is thus mapped to a linear data layout that can be stored on external memory. The out-of-core data management is then solved by *memory mapping* this file at run-time to an array data structure. View-dependent adaptive LOD triangulation and real-time rendering can then be carried out fully in (virtual) main memory without specific out-of-core data access mechanisms.

Clustered triangulation approaches obtain their efficiency by moving the granularity of all LOD operations from individual vertices or triangles to small mesh portions. This reduces memory needs, since less dependency information has to be stored, and offers the possibility to optimize the throughput by exploiting block-transfer features and compression at the level of individual mesh portions. As a representative example, the BDAM and P-BDAM [7, 8] systems encode the hierarchy of right triangles that guide their multiresolution partitioning as a triangle bin-tree, and store the geometry associated to each bin-tree region in an out-of-core patch repository, which is accessed on a patch-by-patch basis. This repository is constructed in a preprocessing step by a distributed algorithm that builds the patches bottom-up using edge collapse simplification with appropriate boundary constraints. Patches are stored in the repository in a packed stripified form ready for rendering. Similarly to [36], the data layout is optimized to improve memory coherency by sorting patches by level and spatial position. Spatial sorting is realized using an indexing function based on a space-filling curve. A separate index, kept in-core, establishes the relation between triangle bin-tree regions and stored mesh patches. At run-time, the most recently used patches are cached on the GPU using a LRU strategy, while the new patches are retrieved by accessing the repository through memory-mapping primitives. When dealing with textured terrains, a tiled texture quadtree, stored in compressed DXT format is overlaid on the geometry.

The 4-8 texture hierarchy system [28] improves over the previous approach by integrating geometry and texture in the same framework. In this case, the diamond region is used in the data structure rather than the bin-tree triangles. Both geometry and textures are treated as small regular grids, called tiles, defined for each diamond in the hierarchy and paged-in from disk on-demand. Loading a new diamond corresponds to loading two patches sharing the main diagonal. For efficient input and output, files and disk blocks are laid out using a diamond indexing scheme based on the Sierpinski space-filling curve. In [23], the client and data access components are separated to support thin clients and network servers.

7.3 Compression

Various authors have concentrated on combining data compression methods with multiresolution schemes to reduce data transfer bandwidths and memory footprints. Tiled block techniques typically use standard 2D compressors to independently compress each tile. In [28], the authors point out that, when using a 4-8 hierarchy, the rectangular tiles associated to each diamond could be also compressed using standard 2D image compression methods.

Geometry clipmaps [37] organize the terrain height data in a pyramidal multiresolution scheme and the residual between levels are compressed using an advanced image coder that supports fast access to image regions [40]. Storing in a compressed form just the heights and reconstructing at run-time both normal and color data (using a simple height color mapping) provides a very compact representation that can be maintained in main memory even for large data sets. The method is possibly the current state-of-the-art in terms of compression rates.

The compressed batched dynamic adaptive meshes (C-BDAM) technique [23], an extension of the BDAM and P-BDAM chunked level-of-detail hierarchy, strives to combine the generality and adaptivity of chunked bin-tree multiresolution structures with the compression rates of nested regular grid techniques. Similarly to BDAM, coarse-grain refinement operations are associated to regions in a bin-tree hierarchy. Each region, called a diamond, is formed by two triangular patches that share their longest edge. In BDAM, each patch is a general precomputed triangulated surface region. In the C-BDAM approach, however, all patches share the same regular triangulation connectivity and incrementally encode their vertex attributes when descending in the multiresolution hierarchy. The encoding follows a two-stage wavelet-based near-lossless scheme in which lossy wavelet predictions are corrected to keep approximated values within user imposed bounds. The approach supports both mean-square error and maximum error metrics allowing to introduce a strict bound on the maximum error introduced in the visualization process. The scheme requires storage of two small square matrices of residuals per diamond, which are maintained in a repository. At run-time, a compact in-core multiresolution structure is traversed, and incrementally refined or coarsened on a diamond-by-diamond basis until screen space error criteria are met. The data required for refining is either retrieved from the repository or procedurally generated to support run-time detail synthesis. At each frame, updates are communicated to the GPU with a batched communication model.

The main take home message of the C-BDAM work is that it is not necessary to use non-adaptive techniques, such as geometry clipmaps, to incorporate aggressive compression in a high performance view-dependent terrain renderer. This comes, however, at the cost of increased implementation complexity.

7.4 Numerical accuracy

Numerical accuracy issues are one of the most neglected aspects in the management of huge data sets. Sending positions to the graphics hardware pipeline needs particular care, given that the highest precision data-type is the IEEE floating point, whose 23 bit mantissa leads to noticeable vertex coalescing problems for metric data sets on the Earth and to camera jitter problems in the general case [50]. In P-BDAM [8], BDAM's structural properties that guarantee overall geometric continuity are exploited for planetary sized rendering applications. Programmable graphics hardware is in particular exploited to cope with the accuracy issues introduced by single precision floating point numbers, resulting in the first fully hardware accelerated system able to provide submetric positioning accuracy on the Earth.

The method uses as basic primitive a general triangulation of points on a displaced triangle (see Fig. 35). Each corner vertex contains a pair of parametric coordinates T_i, that correspond to the position of the vertex in (u, v) coordinates, as well as a planetocentric position P_i and a normal vector N_i, computed from T_i during the patch construction preprocess as a function of the particular projection used. The vertices of the internal triangulation are stored by specifying a barycentric coordinate and an offset along the interpolated normal direction, and all the information required at rendering time is linearly interpolated from the base corner vertex data. As for BDAM, the interior of the patch is an arbitrary triangulation of the vertices, that is represented by

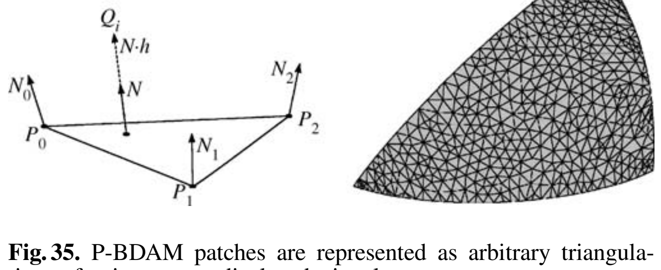

Fig. 35. P-BDAM patches are represented as arbitrary triangulations of points over a displaced triangle

a cache-coherent generalized triangle strip stored as a single ordered list of vertex indices. The only aspect that requires particular care is the computation of planetocentric positions, since all other information is local to the patch. P-BDAM therefore stores P_i in double precision. At each frame, all patches are rendered in camera coordinates, simply subtracting the camera position from P_i on the host before converting them to single precision for transfer to the GPU. This way a single reference frame is used for each frame, and positional accuracy decreases with the distance from the camera, which is exactly what is needed. In contrast to common linear transformation approaches [33, 50], neighboring patches remain unconditionally connected because displaced vertex values only depend on the common base corner vertices (along the edges, the weight for the opposite vertex is null). The conversion cost (nine subtractions and nine floating point conversions) is negligible, since it is amortized over all the internal triangles. Moreover, the transformation from barycentric to Cartesian/texture coordinates can be efficiently computed from corner data on the GPU. This has the important advantage that since the vertices of the internal triangulation are invariant in barycentric coordinates, they can be cached in a static vertex array directly in graphics memory. Moreover, the rendering routine can fully benefit from the post-transform-and-lighting cache of current graphics architectures, which is fully exploited when drawing from the indexed representation.

8 Conclusions

The investigation of multiresolution methods to dynamically adapt rendered model complexity has been, and still is, a very active computer graphics research area, which is obviously impossible to fully cover in a short survey. In this article, we analyzed the most common semi-regular multiresolution approaches for grid-digital terrain models. Despite the slightly increased size of the produced LOD triangle meshes compared to fully irregular approaches, the semi-regular multiresolution methods described in this paper are among the best choices for real-time visualization of very large scale height-field data sets. The various reviewed approaches provide different alternatives in data structures, triangulation algorithms, error metrics, dynamic scene management and rendering methods that can be exploited for an optimized implementation.

Models based on tiled blocks and nested regular grids are generally simple to implement and maintain, and offer optimized interfaces to the graphics hardware at the cost of limited adaptivity and/or approximation quality and/or domain generality. Quadtree and triangle bin-tree triangulations offer a sound mathematical basis upon which efficient dynamic structures providing fully adaptive conforming triangulations can be programmed. Cluster-based approaches, that build upon this basis, have recently shown how these methods can efficiently harness the performance of current commodity graphics platforms, at the cost of a slight reduction in adaptivity.

Even though the domain is mature and has a long history, open problems remain. In particular, while networked and out-of-core rendering systems have been demonstrated for most of the structures discussed in this survey, limited solutions have been proposed for fully out-of-core data construction. Moreover efficient techniques for incrementally updating an already constructed multiresolution hierarchy are still to be devised.

References

1. Asano, T., Ranjan, D., Roos, T., Welzl, E., Widmayer, P.: Space filling curves and their use in the design of geometric data structures. Theor. Comput. Sci. **181**, 3–15 (1997)
2. Balmelli, L., Ayer, S., Vetterli, M.: Efficient algorithms for embedded rendering of terrain models. In: Proceedings IEEE International Conference on Image Processing ICIP 98, pp. 914–918 (1998)
3. Balmelli, L., Liebling, T., Vetterli, M.: Computational analysis of 4-8 meshes with application to surface simplification using global error. In: Electronic Proceedings of the 13th Canadian Conference on Computational Geometry (CCCG) (2001)
4. Bao, X., Pajarola, R.: LOD-based clustering techniques for optimizing large-scale terrain storage and visualization. In: Proceedings SPIE Conference on Visualization and Data Analysis, pp. 225–235 (2003)
5. Bao, X., Pajarola, R., Shafae, M.: SMART: an efficient technique for massive terrain visualization from out-of-core. In: Proceedings Vision, Modeling and Visualization (VMV), pp. 413–420 (2004)
6. Baumann, K., Döllner, J., Hinrichs, K., Kersting, O.: A hybrid, hierarchical data structure for real-time terrain visualization. In: Computer Graphics International, pp. 85–92 (1999)
7. Cignoni, P., Ganovelli, F., Gobbetti, E., Marton, F., Ponchio, F., Scopigno, R.: BDAM: batched dynamic adaptive meshes for high performance terrain visualization. In: Proceedings EUROGRAPHICS, pp. 505–514 (2003).
8. Cignoni, P., Ganovelli, F., Gobbetti, E., Marton, F., Ponchio, F., Scopigno, R.: Planet-sized batched dynamic adaptive meshes (P-BDAM). In: Proceedings IEEE Visualization, pp. 147–155. IEEE Press, Washington, DC (2003)
9. Cignoni, P., Ganovelli, F., Gobbetti, E., Marton, F., Ponchio, F., Scopigno, R.: Batched multi triangulation. In: Proceedings IEEE Visualization, pp. 207–214. IEEE Press, Washington, DC (2005)
10. Cignoni, P., Montani, C., Scopigno, R.: A comparison of mesh simplification algorithms. Comput. Graph. **22**(1), 37–54 (1998)
11. De Floriani, L., Kobbelt, L., Puppo, E.: A survey on data structures for level-of-detail models. In: N. Dodgson, M. Floater, M. Sabin (eds.) Advances in Multiresolution for Geometric Modelling, Mathematics and Visualization, pp. 49–74. Springer, Berlin Heidelberg New York (2004)
12. De Floriani, L., Magillo, P., Puppo, E.: Building and traversing a surface at variable resolution. In: IEEE Visualization'97 (1997)

13. De Floriani, L., Marzano, P., Puppo, E.: Multiresolution models for topographic surface description. Vis. Comput. **12**(7), 317–345 (1996)
14. De Floriani, L., Puppo, E., Magillo, P.: A formal approach to multiresolution modeling. In: R. Klein, W. Straßer, R. Rau (eds.) Geometric Modeling: Theory and Practice, pp. 302–323. Springer, Berlin Heidelberg New York (1997)
15. Duchaineau, M., Wolinsky, M., Sigeti, D.E., Miller, M.C., Aldrich, C., Mineev-Weinstein, M.B.: ROAMing terrain: Real-time optimally adapting meshes. In: Proceedings IEEE Visualization, pp. 81–88 (1997)
16. Evans, W., Kirkpatrick, D., Townsend, G.: Right-triangulated irregular networks. Algorithmica **30**(2), 264–286 (2001)
17. Falby, J., Zyda, M., Pratt, D., Mackey, L.: NPSNET: hierarchical data structures for real-time 3-dimensional visual simulation. Comput. Graph. **17**(1), 65–69 (1993)
18. Garland, M.: Multiresolution modeling: survey and future opportunities. Eurographics State of The Art Report (STAR), Aire-la-Ville, Switzerland (1999)
19. Gerstner, T.: Multiresolution compression and visualization of global topographic data. Tech. Rep. 29. Institute for Applied Mathematics, University of Bonn (1999)
20. Gerstner, T.: Top-down view-dependent terrain triangulation using the octagon metric. Tech. Rep. Institute of Applied Mathematics, University of Bonn (2003)
21. Gerstner, T., Pajarola, R.: Topology preserving and controlled topology simplifying multiresolution isosurface extraction. In: Proceedings IEEE Visualization, pp. 259–266. IEEE Press, Washington, DC (2000)
22. Gerstner, T., Rumpf, M., Weikard, U.: Error indicators for multilevel visualization and computing on nested grids. Comput. Graph. **24**(3), 363–373 (2000)
23. Gobbetti, E., Marton, F., Cignoni, P., Benedetto, M.D., Ganovelli, F.: C-BDAM: compressed batched dynamic adaptive meshes for terrain rendering. Comput. Graph. Forum **25**(3), 333–342 (2006)
24. Hebert, D., Kim, H.: Image encoding with triangulation wavelets. In: Proceedings of SPIE, vol. 2569, pp. 381–392. SPIE, Bellingham, WA (1995)
25. Heckbert, P.S., Garland, M.: Survey of polygonal surface simplification algorithms. In: SIGGRAPH 97 Course Notes 25 (1997)
26. Hitchner, L.E., McGreevy, M.W.: Methods for user-based reduction of model complexity for virtual planetary exploration. In: Proceedings Symposium on Electronic Imaging, pp. 1–16. SPIE, Bellingham, WA (1993)
27. Hoppe, H.: Smooth view-dependent level-of-detail control and its application to terrain rendering. In: Proceedings IEEE Visualization, pp. 35–42. IEEE Press, Washington, DC (1998)
28. Hwa, L.M., Duchaineau, M.A., Joy, K.I.: Real-time optimal adaptation for planetary geometry and texture: 4-8 tile hierarchies. IEEE Trans. Vis. Comput. Graph. **11**(4), 355–368 (2005)
29. Koller, D., Lindstrom, P., Ribarsky, W., Hodges, L.F., Faust, N., Turner, G.: Virtual GIS: A real-time 3D geographic information system. In: Proceedings IEEE Visualization, pp. 94–100. IEEE Press, Washington, DC (1995)
30. Lario, R., Pajarola, R., Tirado, F.: Hyperblock-QuadTIN: Hyper-block quadtree based triangulated irregular networks. In: Proceedings IASTED Invernational Conference on Visualization, Imaging and Image Processing (VIIP), pp. 733–738 (2003)
31. Levenberg, J.: Fast view-dependent level-of-detail rendering using cached geometry. In: Proceedings IEEE Visualization, pp. 259–266. IEEE Press, Washington, DC (2002)
32. Lindstrom, P., Koller, D., Hodges, L.F., Ribarsky, W., Faust, N., Turner, G.: Level-of-detail management for real-time rendering of phototextured terrain. Tech. Rep. TR 95-06, Graphics, Visualization, and Usability Center, Georgia Tech (1995)
33. Lindstrom, P., Koller, D., Ribarsky, W., Hodges, L., Faust, N.: An integrated global GIS and visual simulation system. Tech. Rep. GVU Technical Report 97-0, Georgia Tech Research Institute (1997)
34. Lindstrom, P., Koller, D., Ribarsky, W., Hodges, L.F., Faust, N., Turner, G.A.: Real-time, continuous level of detail rendering of height fields. In: Proceedings ACM SIGGRAPH, pp. 109–118. ACM SIGGRAPH, New York (1996)
35. Lindstrom, P., Pascucci, V.: Visualization of large terrains made easy. In: Proceedings IEEE Visualization, pp. 363–370. IEEE Press, Washington, DC (2001)
36. Lindstrom, P., Pascucci, V.: Terrain simplification simplified: a general framework for view-dependent out-of-core visualization. IEEE Trans. Vis. Comput. Graph. **8**(3), 239–254 (2002)
37. Losasso, F., Hoppe, H.: Geometry clipmaps: terrain rendering using nested regular grids. ACM Trans. Graph. **23**(3), 769–776 (2004)
38. Luebke, D.: A developer's survey of polygonal simplification algorithms. IEEE Comput. Graph. Appl. **21**(3), 24–35 (2001)
39. Luebke, D., Reddy, M., Cohen, J.D., Varshney, A., Watson, B., Huebner, R.: Level of Detail for 3D Graphics. Morgan Kaufmann, San Francisco, CA (2003)
40. Malvar, H.S.: Fast progressive image coding without wavelets. In: Data Compression Conference, pp. 243–252 (2000)
41. Ohlberger, M., Rumpf, M.: Adaptive projection operators in multiresolution scientific visualization. IEEE Trans. Vis. Comput. Graph. **5**(1), 74–93 (1999)
42. Pajarola, R.: Large scale terrain visualization using the restricted quadtree triangulation. In: Proceedings IEEE Visualization, pp. 19–26 (1998)
43. Pajarola, R.: Large scale terrain visualization using the restricted quadtree triangulation. Tech. Rep. 292, Dept. of Computer Science, ETH Zürich (1998)
44. Pajarola, R., Antonijuan, M., Lario, R.: QuadTIN: Quadtree based triangulated irregular networks. In: Proceedings IEEE Visualization, pp. 395–402. IEEE Press, Washington, DC (2002)
45. Pajarola, R., Ohler, T., Stucki, P., Szabo, K., Widmayer, P.: The Alps at your fingertips: virtual reality and geoinformation systems. In: Proceedings International Conference on Data Engineering (ICDE), pp. 550–557. IEEE Press, Washington, DC (1998)
46. Pajarola, R., Widmayer, P.: Virtual geoexploration: concepts and design choices. Int. J. Comput. Geom. Appl. **11**(1), 1–14 (2001)
47. Pomeranz, A.A.: ROAM Using Surface Triangle Clusters (RUSTiC). Dissertation, University of California at Davis (2000)
48. Puppo, E.: Variable resolution terrain surfaces. In: Proceedings of the 8th Canadian Conference on Computational Geometry, pp. 202–210 (1996)
49. Puppo, E.: Variable resolution triangulations. Comput. Geom. **11**(3–4), 219–238 (1998)
50. Reddy, M., Leclerc, Y., Iverson, L., Bletter, N.: TerraVision II: visualizing massive terrain databases in VRML. IEEE Comput. Graph. Appl. **19**(2), 30–38 (1999)
51. Rivara, M.C.: A discussion on the triangulation refinement problem. In: Proceedings of the 5th Canadian Conference on Computational Geometry, pp. 42–47 (1993)
52. Rivara, M.C.: A discussion on mixed (longest-side midpoint insertion) delaunay techniques for the triangulation refinement problem. In: Proceedings of the 4th International Meshing Roundtable, pp. 335–346 (1995)
53. Samet, H.: Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS. Addison Wesley, Reading, MA (1989)
54. Samet, H.: The Design and Analysis of Spatial Data Structures. Addison Wesley, Reading, MA (1989)
55. Schneider, J., Westermann, R.: GPU-friendly high-quality terrain rendering. J. WSCG **14**(1–3), 49–56 (2006)
56. Schrack, G.: Finding neighbors of equal size in linear quadtrees and octrees in constant time. Comput. Vision Graph. Image Process. Image Underst. **55**(2), 221–230 (1992)
57. Sivan, R.: Surface modeling using quadtrees. Tech. Rep. CS-TR-3609, University of Maryland, College Park, Computer Vision Laboratory, Center for Automation Research (1996)
58. Sivan, R., Samet, H.: Algorithms for constructing quadtree surface maps. In: Proceedings of the 5th International

Symposium on Spatial Data Handling, pp. 361–370 (1992)
59. Ulrich, T.: Rendering massive terrains using chunked level of detail. In: Super-size-it! Scaling up to Massive Virtual Worlds (ACM SIGGRAPH Tutorial Notes). ACM SIGGRAPH, New York (2000)
60. Velho, L.: Using semi-regular 4-8 meshes for subdivision surfaces. J. Graph. Tools **5**(3), 35–47 (2001)
61. Velho, L., Gomes, J.: Variable resolution 4-k meshes: concepts and applications. Comput. Graph. Forum **19**(4), 195–214 (2000)
62. Von Herzen, B., Barr, A.H.: Accurate triangulations of deformed, intersecting surfaces. In: Proceedings ACM SIGGRAPH, pp. 103–110. ACM SIGGRAPH, New York (1987)
63. Wahl, R., Massing, M., Degener, P., Guthe, M., Klein, R.: Scalable compression and rendering of textured terrain data. J. WSCG **12**, 521–528 (2004)
64. Wloka, M.: Optimizing the graphics pipeline. In: Programming Graphics Hardware (Eurographics 2004 Tutorial No. 4). Eurographics Association, Aire-la-Ville, Switzerland (2004)

RENATO B. PAJAROLA received a Dr. sc. techn. in Computer Science in 1998 from the Swiss Federal Institute of Technology (ETH) Zürich. After a postdoc at Georgia Tech he joined the University of California Irvine in 1999 as an Assistant Professor. Since 2005 he has been an Associate Professor in Computer Science at the University of Zürich, leading the Visualization and MultiMedia Lab. His research interests include real-time 3D graphics, scientific visualization and interactive 3D multimedia. He is a frequent committee member and reviewer for top conferences and journals.

ENRICO GOBBETTI is the founder and director of the Visual Computing (ViC) group at the Center for Advanced Studies, Research, and Development in Sardinia (CRS4). Prior to joining CRS4, he was with the Swiss Federal Institute of Technology in Lausanne, the University of Maryland Baltimore County and the Center of Excellence in Space Data and Information Sciences (NASA/CESDIS). His research interests span many areas of computer graphics. He served as program committee member and reviewer for international conferences and journals. He holds an Engineering degree (1989) and a Ph.D. degree (1993) in Computer Science from the Swiss Federal Institute of Technology in Lausanne (EPFL).