Interactive Out-of-core Visualisation of Very Large Landscapes on Commodity Graphics Platform

P. Cignoni[2], F. Ganovelli[2], E. Gobbetti[1], F. Marton[1], F. Ponchio[2], and R. Scopigno[2]

[1] CRS4
POLARIS Edificio 1, C.P. 25, I-09010 Pula (CA), Italy,
WWW: `http://www.crs4.it/vic/`
[2] ISTI-CNR
Area della Ricerca, via G. Moruzzi 1, 56124 PISA, Italy
WWW: `http://vcg.iei.pi.cnr.it/`

Abstract. We recently introduced an efficient technique for out-of-core rendering and management of large textured landscapes. The technique, called Batched Dynamic Adaptive Meshes (*BDAM*), is based on a paired tree structure: a tiled quadtree for texture data and a pair of bintrees of small triangular patches for the geometry. These small patches are TINs that are constructed and optimized off-line with high quality simplification and tristripping algorithms. Hierarchical view frustum culling and view-dependendent texture/geometry refinement is performed at each frame with a stateless traversal algorithm that renders a continuous adaptive terrain surface by assembling out of core data. Thanks to the batched CPU/GPU communication model, the proposed technique is not processor intensive and fully harnesses the power of current graphics hardware. This paper summarizes the method and discusses the results obtained in a virtual fly-through over a textured digital landscape derived from aerial imaging.

1 Introduction

Virtual environment technology has been developing over a long period, and offering presence simulation to users as an interface metaphor to a synthesized world has become the research agenda for a growing community of researchers and industries. The motivation for such a research direction is twofold. From an evolutionary perspective, virtual reality is seen as a way to overcome limitations of standard human-computer interfaces; from a revolutionary perspective, virtual reality technology opens the door to new types of applications that exploit the additional possibilities offered by presence simulation.

Our sense of physical reality is a construction derived from the symbolic, geometric, and dynamic information directly presented to our senses. Sensory simulation is thus at the heart of virtual reality technology and, since vision is generally considered the most dominant sense, the development of efficient methods for generating high quality visual representations of simulated environments is of primary importance.

Real-time 3D exploration of digital elevation models (DEM), which are computerized representations of the Earth's relief, is now one of the most important features

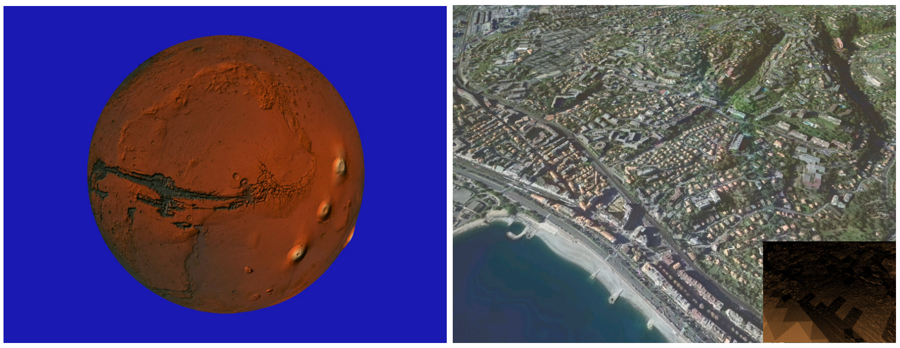


Fig. 1. Flying over reconstructed terrains at different scales: Left: Global reconstruction of Mars. Right: City scale exploration (Nice, France); textured terrain and same view with wireframe showing adaptive rendering tessellation.

in a number of practical applications, that range from scientific simulators, to gaming systems, to virtual storytelling applications.

Rendering large-scale high resolution terrains with high visual and temporal fidelity on a graphics PC platform is, however, a very challenging task, since it is necessary to meet the following requirements on the rendering subsystem:

- **high visual and temporal fidelity**: at least 30 frames per second, with about 1M pixels/frame with color defined at 16 to 24 bits per pixels;
- **large scale high resolution terrains**: texture mapped elevation data (DTM or DEM/DSM). Typical current database sites are represented by elevation grids of about 8Kx8K to 40Kx40K (this means up to billions of vertices per model). Color resolution is generally similar or 2 to 4 times larger;
- **graphics PC platform**: at the time of this writing, this means a one or two-processor machine with 32 bit architecture, large local disk, 1GHz or more Pentium class processor, DDR memory, and good quality commodity graphics board (AGP4X/8X, theoretical peak rendering speed in the order of tens or hundreds of millions of vertices per second).

The above requirements indicate that scene complexity is much larger that what can be handled by brute force methods. Since there is no upper bound on the complexity of a scene visible from a specific viewpoint, occlusion and view frustum culling techniques are not sufficient alone for meeting the performance requirements dictated by the human perceptual system. Achieving this goal requires the ability to trade rendering quality with speed. Ideally, this time/quality conflict should be handled with adaptive techniques, to cope with the wide range of viewing conditions while avoiding worst case assumptions.

Current dynamic multiresolution algorithms are, however, very processor intensive, and fail to provide adequate performance for large datasets. In particular, multiresolution rendering techniques were designed for the previous generation machines, and are generally too processor intensive. The potential of current GPUs is far from being fully exploited, and, on current platforms, performance is directly related to CPU processing power, which grows at a much slower rate than GPU's one. Current state-of-the-art techniques, based on hierarchical techniques (e.g., [4]) use only about 10% to 20% of the available graphics power for large datasets and do not offer efficient means to preserve sharp features. Moreover, in most applications that need interactive terrain visualization, managing the terrain multiresolution structure is not the only task that must be accomplished by the system: dynamics simulation, AI of other agents, database queries, and other jobs have to be carried out in the same time. For this reason, using current multiresolution solutions, the size of the terrain representation that can be constructed/updated and rendered for each frame is severely limited by the CPU processing power and by far smaller than the GPU capabilities.

We recently introduced a technique, named Batched Dynamic Adaptive Meshes (BDAM), that is based on the idea to move the grain of the multiresolution models up from triangles to small contiguous portions of mesh in order to alleviate the processing load and to better exploit current graphics hardware. The technique is based on a paired tree structure: a tiled quadtree for texture data and a pair of bintrees of small triangular patches for the geometry. These small patches are TINs that are constructed and optimized off-line with high quality simplification and tristripping algorithms. A hierarchical view frustum culling and view-dependendent texture/geometry refinement can be performed, at each frame, with a stateless traversal algorithm that renders a continuous adaptive terrain surface by assembling these small patches. An out-of-core technique has been designed and tested for constructing BDAMs using a generic high quality simplification.

Thanks to the batched CPU/GPU communication model, the proposed technique is not processor intensive and fully harnesses the power of current graphics hardware. The remainder of this paper summarizes the method and discusses the results obtained in a virtual flythrough over a textured digital terrain model of the city of Nice, France.

2 Methods and tools

Most of current multiresolution algorithms are designed to use the triangle as the smallest primitive entity. The main idea behind our approch is to adopt a more complex primitive: small surface patches composed of a batch of a few hundreds of triangles. The benefits of this approach are that the per-triangle workload to extract a multiresolution model is highly reduced and the small patches can be preprocessed and optimized off-line for a more efficient rendering. We summarize here the main concepts behind BDAM. Please refer to the original papers for further details [1, 2].

In BDAM, the small patches form a hierarchy of right triangles (HRT) that is coded as a binary tree. This representation can be used to easily extract a consistent set of contiguous triangles which cover a particular region with given error thresholds. These small triangular patches can be *batched* (hence the name) to the graphics hardware in the most efficient way. Therefore, each bintree node contains a small chunk (in the range of 256..8K) of contiguous well packed tri-stripped triangles. To ensure the correct matching between triangular patches, BDAM exploits the right triangle hierarchy property that each triangle can correctly connect to: triangles of its same level; triangles of the next coarser level through the longest edge; and triangles of the next finer level through the two shortest edges.

To guarantee the correct connectivity along borders of different simplification levels, triangular patches are built so that the error is distributed as shown in figure 2: each triangle of the bintree represents a small mesh patch with error e_k inside and error e_{k+1} (the error corresponding to the next more refined level in the bintree) along the two shortest edges. In this way, each mesh composed by a collection of small patches arranged as a correct bintree triangulation still generates a globally correct triangulation. This simple edge error property is exploited, as explained in the original paper [2], to design a distributed out-of-core high quality simplification algorithm that concurrently builds all patches.

In Fig. 2 we show an example of these properties. In the upper part of the figure we show the various levels of a HRT and each triangle represents a terrain patch composed by many graphics primitives. Colors correspond to different errors; the blending of the colors inside each triangular patch corresponds to the smooth error variation inside each patch. When composing these triangular patches using the HRT consistency rules, the color variation is always smooth: the triangulation of adjacent patches correctly matches.

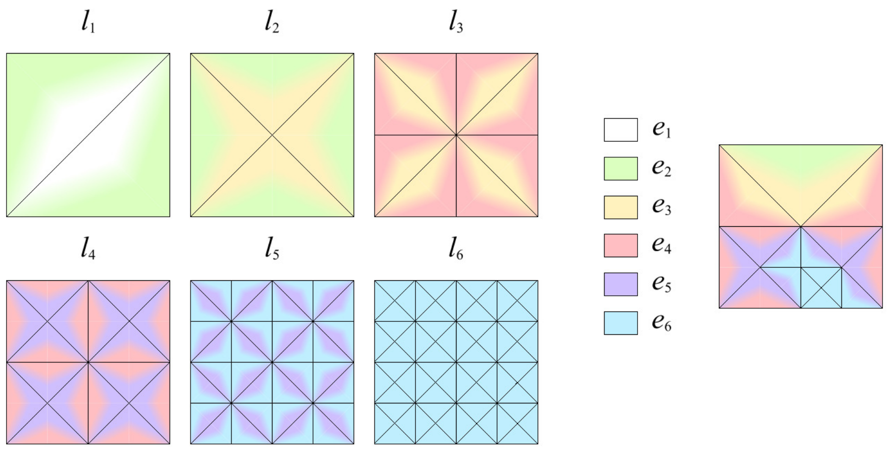


Fig. 2. An example of a BDAM: each triangle represents a terrain patch composed by many triangles. Colors correspond to different errors; the blending of the color inside each triangle corresponds to the smooth error variation inside each patch.

Texture and geometry trees. To efficiently manage large textures, the BDAM approach partitions them into tiles before rendering and arranges them in a multiresolution structure as a tiled texture quadtree. Each texture quadtree element corresponds to a pair of adjacent geometry bintree elements. The roots of the trees cover the entire dataset, and both trees are maintained off-core using a pointerless structure that is mapped at run time to a virtual memory address range. During rendering, the two trees are processed together. Descending one level in the texture quadtree corresponds to descending two levels in the associated pair of geometry bintrees. This correspondence can be exploited in the preprocessing step to associate object-space representation errors to the quadtree levels, and in the rendering step to implement view-dependent multiresolution texture and geometry extraction in a single top-down refinement strategy.

Errors and bounding volumes. To easily maintain the triangulation coherence BDAM exploits the concept of nested/saturated errors, introduced by Pajarola[6], that supports the extraction of a correct set of triangular patches with a simple stateless refinement visit of the hierarchy [6, 4] that starts at the top-level of the texture and geometry trees and recursively visits the nodes until the screen space texture error becomes acceptable. The object-space errors of the patches are computed directly during the preprocessing construction of the BDAM. Once these errors have been computed, a hierarchy of errors that respect nesting conditions is constructed bottom up. Texture errors are computed from texture features, and, similarly, are embedded in a corresponding hierarchy. For the rendering purpose, BDAM adopts a tree of nested volumes that is also built during the preprocessing, with properties very similar to the two error rules: 1) bounding volume of a patch include all children bounding volumes; 2) two patches adjacent along hypotenuse must share the same bounding volume which encloses both. These bounding volumes are used to compute screen space errors and also for view frustum culling.

Large dataset partitioning. In order to handle the size and accuracy problems related to large dataset management, we partition input data in a number of square tiles, therefore managing a forest of BDAM hierarchies instead of a single tree. This effectively decomposes the original dataset into terrain tiles. It should be noted, however, that the tiles are only used to circumvent address space and accuracy limitations and do not affect other parts of the system. In particular, errors and bounding volumes are propagated to neighboring tiles through the common edges in order to ensure continuity for the entire dataset. The tiles have an associated (u, v) parameterization, which is used for texture coordinates and to construct the geometry subdivision hierarchy. The number and size of the tiles is arbitrary and depends only on the size of the original dataset. In particular, we make sure that the following constraints are met: (a) a single precision floating point representation is accurate enough for representing local coordinates (i.e. there are less than 2^{23} texels/positions along each coordinate axis); (b) the size of the generated multiresolution structure is within the data size limitations imposed by the operating system (i.e. less than the largest possible memory mapped segment).

Top-down view-dependent refinement and rendering. For each of the partitions that compose the dataset, we map its data structure into the process address space, render the structure using a stateless top-down refinement procedure, then delete the mapping for the specified address range. The refinement procedure starts at the top level of the texture and geometry trees of a given tile and recursively visits their nodes until the screen space texture error becomes acceptable or the visited node bounding sphere is proved off the viewing frustum. While descending the texture quadtree, corresponding displaced triangle patches in the two geometry bintree are identified and selected for processing. Once the texture is considered detailed enough, texture refinement stops. At this point, the texture is bound and the algorithm continues by refining the two geometry bintrees until the screen space geometry error becomes acceptable or the visited node is culled out. Patch rendering is done by converting the corner vertices to camera coordinates and binding them along with associated normals and texture coordinates to the appropriate uniform parameters, prior to binding varying vertex data and drawing an indexed triangle strip. With this method, each required texture is bound only once, and all the geometry data covered by it is then drawn, avoiding unnecessary context switches and minimizing host to graphics bandwidth requirement.

Memory management. Time-critical rendering large terrain datasets requires real-time management of huge amounts of data. Moving data from the storage unit to main memory and to the graphics board is often the major bottleneck. We use both a data layout aimed at optimizing memory coherence and a cache managed using a LRU strategy for caching the most recent textures and patches directly in graphics memory. Since the disk is, by far, the slowest component of the system, we have further optimized the external memory management component with mesh compression and speculative prefetching (see [2] for details).

3 Results

An experimental software library and a terrain rendering application supporting the BDAM technique has been implemented and tested on Linux and Win32 platforms. The results were collected on a Linux PC with dual Intel Xeon 2.4 Ghz, 2GB RAM, two Seagate ST373453LW 70 GB ULTRA SCSI 320 hard drives, AGP 8x and NVIDIA GeForce Fx Ultra graphics.

The test case discussed in this paper concerns the preprocessing and interactive exploration of a terrain dataset of the Nice metropolitan area [1]. We used a 8K x 8K elevation grid with 100 centimeter horizontal resolution. On this terrain, we mapped a 16K x 16K RGB texture with 50 centimeter resolution.

3.1 Preprocessing

The input dataset was transformed to multiresolution by our texture and geometry processing tools. For textures, we used a tile size of 256x256 pixels, which produced a 7 level quadtree and compressed colors using the DXT1 format. Texture preprocessing, including error propagation, took roughly two hours and produced a structure occupying 132 MB on disk. Processing time is dominated by texture compression. For geometry, we generated two 17 levels bintrees, with leaf nodes containing triangular patches of 32x32 vertex side at full resolution and interior nodes with a constant vertex

[1] dataset: courtesy of ISTAR high resolution cartographic database.

count of 600. Geometry preprocessing, that included optimized tristrip generation and data compression, has been performed roughly in 3.5 hours, producing a multiresolution structure which occupies 321 MB. Size has been reduced by compression to about 60 % of the size of the original model. The full resolution model is made of 46M triangles. For the sake of comparison, Hoppe's view dependent progressive meshes [3], that, like BDAMs, support unconstrained triangulation of terrains, need roughly 380MB of RAM and uses 190MB of disk space to build a multiresolution model of a simplified version of 7.9M triangles of the Puget Sound dataset [2]. Preprocessing times are similar to BDAM times. By contrast, SOAR [5] geometry data structure, which is based on a hierarchy of right triangles, takes roughly 3.4 GB[3] on disk for the processed data set, but is much faster to compute since the subdivision structure is data independent.

3.2 View-dependent Refinement

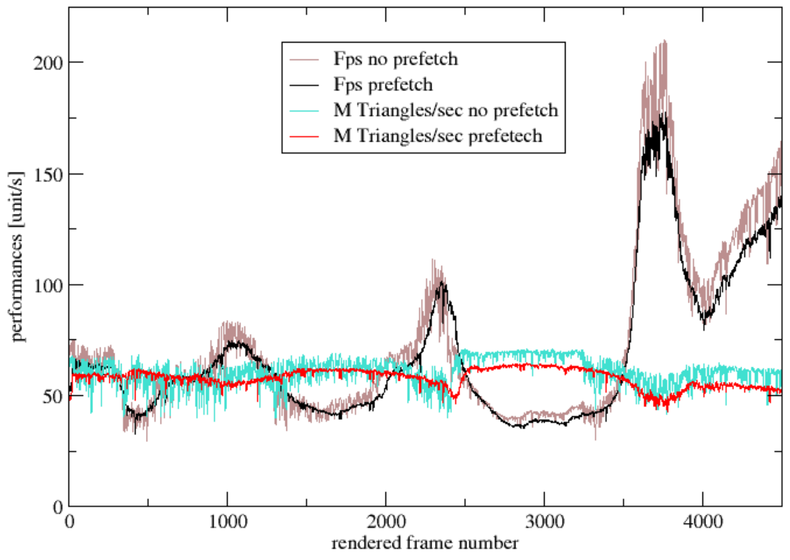


Fig. 3. Performance Evaluation. Rendering rates per frame with and without data prefetching. Note how the prefetch version presents a smoother behaviour.

[2] The dataset at various resolution is freely available from http://www.cc.gatech.edu/projects/large_models/ps.html

[3] The version of SOAR used in this comparison is v1.11, available from http://www.cc.gatech.edu/∼lindstro/software/soar/

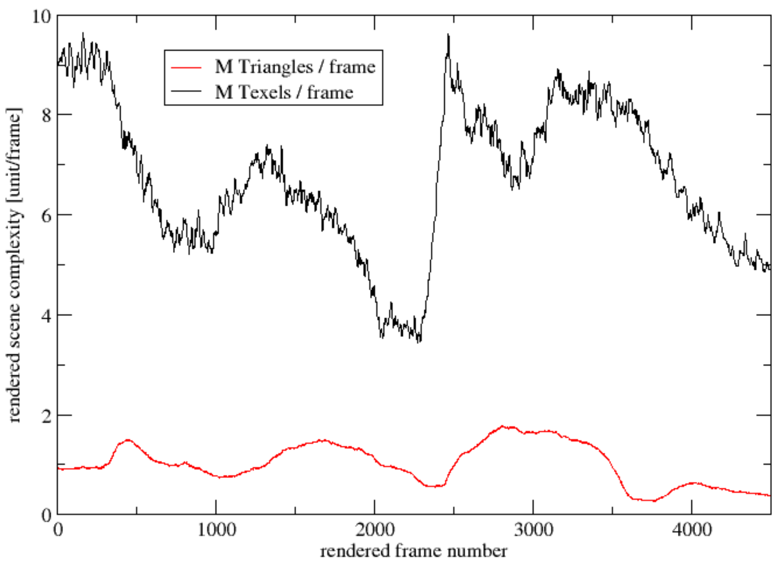


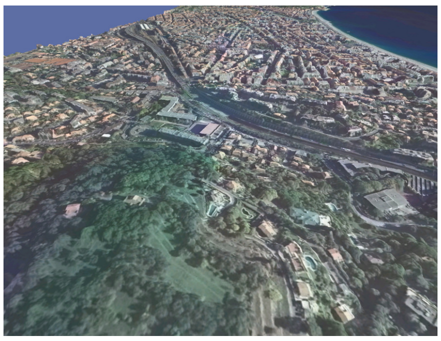
Fig. 4. Complexity evaluation. Rendered complexity per frame.

We evaluated the performance of the BDAM technique on a number of flythrough sequences. The quantitative results presented here were collected during a 100 seconds high speed fly-over of the data set with a window size of 800x600 pixels and a screen tolerance of 1.0 pixel. The qualitative performance of our view-dependent refinement is further illustrated in an accompanying video[4], that shows the live recording of the analyzed flythrough sequence (Fig. 5).

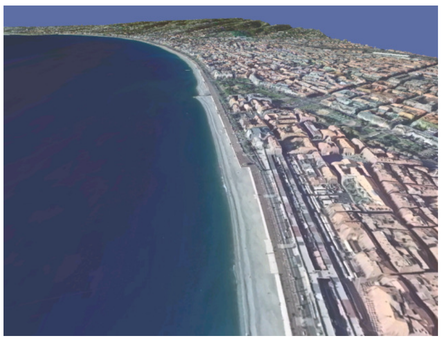
Figure 3 illustrates the rendering performance of the application. We were able to sustain an average rendering rate of roughly 60 millions of textured triangles per second, with peaks exceeding 71 millions, which are close to the peak performance of the rendering board (Fig. 3 left). By comparison, on the same machine, SOAR peak performance was measured at roughly 5.5 millions of triangles per second, even though SOAR was using a smaller single resolution texture of 2Kx2K texels. The increased performance of the BDAM approach is due to the larger granularity of the structure, that amortizes structure traversal costs over many graphics primitives, reduces AGP data transfers through on-board memory management and fully exploits the post-transform-and-lighting cache with optimized indexed triangle strips.

Rendered scene granularity is illustrated in figure 4 with the peak complexity of the rendered scenes reaching 1.7M triangles and 9.6M texels per frame. Since we are

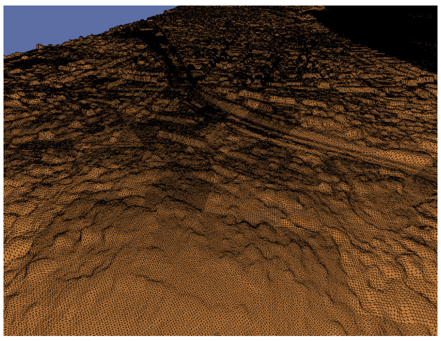
[4] The videos are available from `http://www.crs4.it/vic/multimedia/`.



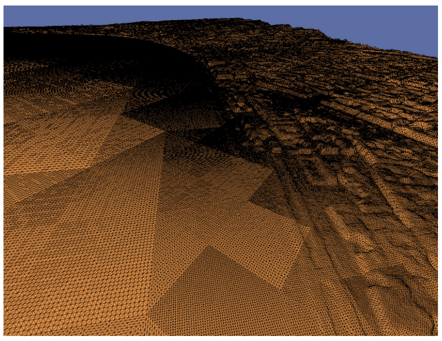
(a) Frame 1800



(b) Frame 2800



(c) Frame 1800 wireframe



(d) Frame 2800 wireframe

Fig. 5. Selected flythrough frames. Screen space error tolerance set to 3.0 pixels.

able to render such complex scenes at high frame rates (30 to 210 Hz for the entire test path, Fig. 4), it is possible to use very small pixel threshold, virtually eliminating popping artifacts, without the need to resort to costly geomorphing features. Moreover, since TINs are used as basic building blocks, the triangulation can more easily adapt to high frequency variations of the terrain, such as house walls, than techniques based on regular subdivision meshes.

4 Conclusions

We have presented an efficient technique for out-of-core management and interactive rendering of large scale textured terrain surfaces. The main idea behind the method is to move the grain of the multiresolution models up from triangles to small contiguous portions of mesh in order to alleviate the processing load and to better exploit current graphics hardware. The results demonstrate that, thanks to the batched host-to-graphics communication model, performance is limited by graphics processor speed. The technique is thus well suited for a range of interactive applications, including virtual storytelling systems requiring the interactive rendering of massive outdoor sets.

References

1. Paolo Cignoni, Fabio Ganovelli, Enrico Gobbetti, Fabio Marton, Federico Ponchio, and Roberto Scopigno. BDAM – batched dynamic adaptive meshes for high performance terrain visualization. *Computer Graphics Forum*, 22(3), September 2003. To appear.
2. Paolo Cignoni, Fabio Ganovelli, Enrico Gobbetti, Fabio Marton, Federico Ponchio, and Roberto Scopigno. Planet–sized batched dynamic adaptive meshes (p-bdam). In *Proceedings IEEE Visualization*, Conference held in Seattle, WA, USA, October 2003. IEEE Computer Society Press. To appear.
3. H. Hoppe. Smooth view-dependent level-of-detail control and its aplications to terrain rendering. In *IEEE Visualization '98 Conf.*, pages 35–42, 1998.
4. P. Lindstrom and V. Pascucci. Visualization of large terrains made easy. In *Proc. IEEE Visualization 2001*, pages 363–370, 574. IEEE Press, October 2001.
5. P. Lindstrom and V. Pascucci. Terrain simplification simplified: A general framework for view-dependent out-of-core visualization. *IEEE Transaction on Visualization and Computer Graphics*, 8(3):239–254, 2002.
6. R. Pajarola. Large scale terrain visualization using the restricted quadtree triangulation. In H. Rushmeier D. Elbert, H. Hagen, editor, *Proceedings of Visualization '98*, pages 19–26, 1998.