Planet-Sized Batched Dynamic Adaptive Meshes (P-BDAM)

Paolo Cignoni
ISTI - CNR *

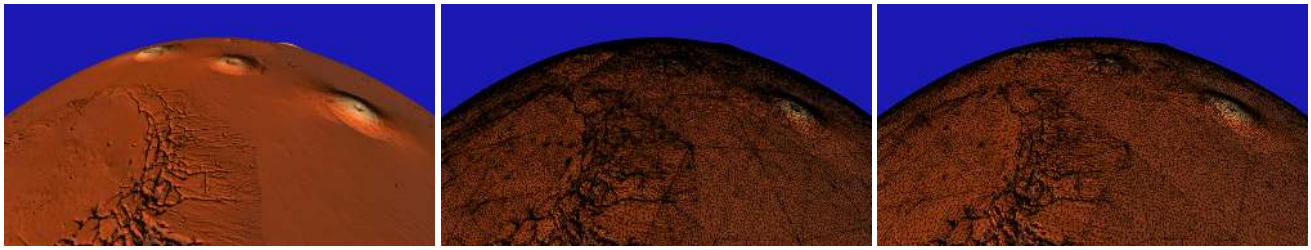
Fabio Ganovelli
ISTI - CNR

Enrico Gobbetti
CRS4 †

Fabio Marton
CRS4

Federico Ponchio
ISTI - CNR

Roberto Scopigno
ISTI - CNR



(a) screen tolerance = 1 pixel, 128 texture tiles, 286 patches (253K triangles)

(b) screen tolerance = 1 pixel, 128 texture tiles, 286 patches (253K triangles)

(c) screen tolerance = 5 pixels, 71 texture tiles, 148 patches (131K triangles)

Figure 1: **View of Mariner Valley and the Tharsis Volcanos, Mars.** Snapshots from an interactive inspection session of a global reconstruction of planet Mars created from Mars Orbiter Laser Altimeter 128 samples/degree data [Smith et al. 2002].

Abstract

We describe an efficient technique for out-of-core management and interactive rendering of planet sized textured terrain surfaces. The technique, called P-Batched Dynamic Adaptive Meshes (*P-BDAM*), extends the BDAM approach by using as basic primitive a general triangulation of points on a displaced triangle. The proposed framework introduces several advances with respect to the state of the art: thanks to a batched host-to-graphics communication model, we outperform current adaptive tessellation solutions in terms of rendering speed; we guarantee overall geometric continuity, exploiting programmable graphics hardware to cope with the accuracy issues introduced by single precision floating points; we exploit a compressed out of core representation and speculative prefetching for hiding disk latency during rendering of out-of-core data; we efficiently construct high quality simplified representations with a novel distributed out of core simplification algorithm working on a standard PC network.

CR Categories: K.6.1 [Computer Graphics]: Picture and Image Generation—; K.7.m [Computer Graphics]: Three-Dimensional Graphics and Realism—.

Keywords: Multiresolution, terrains, huge dataset

1 Introduction

Interactive visualization of huge planet-sized textured terrain datasets is a complex and challenging problem: the size of a high accuracy geometry and texture representation of an entire planet sits in the scale of giga-triangle and giga-texel datasets. This kind of datasets exceeds the capabilities of current hardware and existing multiresolution algorithms.

Various dynamic multiresolution models have been proposed to face the problem of efficient visualization of terrains, usually based on the idea of constructing, on the fly, a coarser adaptively approximated representation to be rendered in place of the complete terrain model. Unfortunately, current dynamic multiresolution solutions are not suitable to handling planet-sized data for various reasons. Most of the existing techniques are very processor intensive: the extraction of an adequate terrain representation from a multiresolution model and its transmission to the graphics hardware is usually the main bottleneck in terrain visualization. Nowadays, consumer graphics hardware is able to sustain rendering rate of tens of millions of triangles per second, but most of current multiresolution solutions fall short of reaching such performance. Moreover, accuracy problems, due to the limited representation range of standard floats, arise when the size of the dataset becomes huge and none of the existing solutions correctly handles this issue without introducing discontinuities or large performance penalties. Lastly, for planet sized dataset also the preprocessing steps for constructing the required multiresolution structure can easily become totally overwhelming, so a parallel scalable solution should be introduced.

The original contribution of this paper is to introduce a novel solution for interactive and accurate visualization of huge textured terrains that improves over current methods in the following areas:

- it manages and renders non flat huge datasets nearly one order of magnitude faster than existing solutions thanks to a batched host-to-graphics communication model;
- it guarantees overall geometric continuity, exploiting programmable graphics hardware to cope with the accuracy issues introduced by single precision floating point numbers;
- it successfully exploits a compressed out of core representation and speculative prefetching for efficient rendering;
- it efficiently handles the construction of high quality simplified representations by using a novel distributed out of core simplification algorithm.

As highlighted in the short overview of the current solutions for interactive visualization of large terrains (Sec. 2), the techniques based on the hierarchy of right triangles are the ones which ensure

*ISTI-CNR, Via Moruzzi 1, 56124 Pisa Italy www:http://vcg.isti.cnr.it/ e-mail:first.last@isti.cnr.it
†CRS4, POLARIS Edificio 1, 09010 Pula, Italy www:http://www.crs4.it/ e-mail:first.last@crs4.it

maximum throughput, while TIN based multiresolution solutions reach maximal accuracy for a given triangle count. In this paper we introduce a new data structure that takes the best of the above approaches. Moreover, our approach efficiently supports the combination of high resolution elevation and texture data in the same framework. Our approach adopts the philosophy of the BDAM technique (Sec. 3), an efficient approach for the rendering of flat large textured terrains. The new structure here proposed, called P-BDAM, to succeed in efficiently rendering planet sized datasets is described in Sec. 4. A distributed out-of-core technique has been designed and tested for constructing P-BDAMs using a generic high quality simplification algorithm (Sec. 5). The efficiency of the P-BDAM approach has been successfully evaluated by showing the accurate interactive visualization of the whole planet Mars created from Mars Orbiter Laser Altimeter data (Sec. 6).

2 Related Work

Adaptive triangulations. Adaptive rendering of huge terrain datasets has a long history, and a comprehensive overview of this subject is beyond the scope of this paper. In the following, we will discuss the approaches that are most closely related with our work. Readers may refer to recent surveys [Lindstrom and Pascucci 2002; Pajarola 2002] for further details.

Two main classes of techniques have been proposed to manage and render continuous adaptive terrain representations: a) regular hierarchical structures, b) more general unconstrained triangulations. The first class is generally based on a particular regular refinement scheme called hierarchies of right triangles (HRT) [Evans et al. 2001] or longest edge bisection [Lindstrom and Pascucci 2002], triangle bintree [Lindstrom et al. 1996; Duchaineau et al. 1997], restricted quadtree triangulation [Pajarola 1998; Samet 1990]. The second class of algorithms is based on less constrained triangulations of the terrain (TINs) and includes multiresolution data structures like multi-triangulations [Puppo 1996] adaptive merge trees [Xia and Varshney 1996], hypertriangulations [Cignoni et al. 1997], and the extension of progressive meshes [Hoppe 1997] to the view-dependent management of terrains [Hoppe 1998]. As pointed out and numerically evaluated in [Evans et al. 2001], TIN outperform right triangles hierarchies in terms of number of triangles / error counts, but are more complicated and hardly manage large datasets in real-time. The two classes of methods also differ in how they interact with texture management. Very few techniques exist that full decouple texture and geometry LOD management. To our knowledge, the only general approach is the SGI-specific clipmapping extension [Tanner et al. 1998] and 3DLabs Virtual Textures, which requires, however, special hardware. In general, large scale textures are handled by explicitly partitioning them into tiles and possibly arranging them in a pyramidal structure [Döllner et al. 2000]. Clipping rendered geometry to texture tile domains imposes severe limitations on the geometry refinement subsystem and TIN approaches are more difficult to adapt to this context than HRTs.

Our work aims at combining the benefits of TINS and HRT in a single data structure for the efficient management of multiresolution textured terrain data, that is here extended to planet-sized datasets. A first attempt toward this aim is the work of [Pajarola et al. 2002], where a technique to build a HRT starting from a TIN terrain is presented. The main idea is to adaptively build a HRT following the TIN data distribution. Among the many other differences, in our proposal the advantages of TINS are much better exploited, because each patch is a completely general triangulation of the corresponding domain.

Out-of-core simplification and rendering. Various techniques have been presented to face the problem of huge mesh simplification: with the exception of the clustering solutions based on the Lindstrom approach [Lindstrom 2000; Lindstrom 2003], most of these techniques, such as Hoppe's hierarchical method for digital terrain management [Hoppe 1998] and the octree based structure OEMM [Cignoni et al. 2003a], are based on some kind of mesh partitioning and subsequent independent simplification. Hoppe hierarchically divides the mesh in blocks, simplifies each block by edge-collapse (borders are constrained) and then traverses bottom-up the hierarchical structure by merging sibling cells and again simplifying. In this approach some of the borders remains not simplified until the whole mesh can be loaded entirely in memory. The OEMM avoids this kind of problem, but it does not build a multiresolution structure. On the other hand, the BDAM approach [Cignoni et al. 2003b] allows both the correct independent processing of small sub-portions of the whole mesh and the construction of a multiresolution structure; in this work, we exploit this independence property to efficiently parallelize the simplification process in order to build a multiresolution structure for arbitrarily large meshes.

With respect to the compressed storage of our geometric data, we want to remark that our approach is aimed to obtain the highest decompression performance, rather than attaining high compression ratios like for example [Isenburg and Gumhold 2003]. Moreover, the uncompressed in-core representation, using baricentric coordinates, implicit texture coordinates, interpolated normals, triangles strips and short 16 bit data types is already quite compact. For this reason we have chosen a rather simple approach for coding the geometry on the disk but that can be decompressed without slowing down the rendering process.

Accuracy of the representation. Numerical accuracy issues are one of the most neglected aspects in the management of huge datasets. 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 problem for metric datasets on the Earth and to camera jitter problems in the general case [Reddy et al. 1999]. Typical solutions (e.g. [Lindstrom et al. 1997; Reddy et al. 1999]) are to partition the dataset into multiple tiles, each of them with an associated local, possibly hierarchical, coordinate system, and then use single precision floating points for representing vertex positions in the local reference. At rendering time, tiles are separately rendered in their local rendering systems, performing matrix transformations on the host in double precision to reduce roundoff errors. This solution, however, leads to discontinuity problems at tile borders. Our solution solves the accuracy problem using patch parametric coordinates, exploiting the programmability features of modern GPUs to ensure overall continuity.

Efficient host-to-graphics communication. A common point of all adaptive mesh generation techniques is that they spend a great deal of the rendering time to compute the view-dependent triangulation. For this reason, many authors have proposed techniques to alleviate popping effects due to small triangle counts [Cohen-Or and Levanoni 1996; Hoppe 1998] or to amortize construction costs over multiple frames [Lindstrom et al. 1996; Duchaineau et al. 1997; Hoppe 1997]. We have recently proposed, instead, to reduce the per-triangle workload by composing at run-time pre-assembled surface patches [Cignoni et al. 2003b]. The idea of grouping together sets of triangles in order to alleviate the CPU/GPU bottleneck was presented also in the RUSTIC [Pomeranz 2000] and in the CABTT [Levenberg 2002] data structures. The RUSTIC method is a extension of the ROAM algorithm in which subtrees of the ROAM bintree are, in a preprocessing phase, statically frozen and saved. The CABTT approach is very similar to RUSTIC, but clusters are dynamically created, cached and reused during rendering. With respect to both methods, our BDAM structure [Cignoni et al.

2003b], among other differences, explicitates the simple edge error property needed for cluster consistency, exploits high quality, fully adaptive triangulation of clusters, cache coherent tri-stripping of clusters for efficient rendering, and multiresolution texturing; finally, it supports out-of-core rendering and construction of huge datasets. In this work, we extend this approach to the efficient rendering of curvilinear patches.

3 Batched Dynamic Adaptive Meshes

As explained in the previous section, most of current multiresolution algorithms are designed to use the triangle as the smallest primitive entity. The main idea behind the Batched Dynamic Adaptive Meshes (*BDAM*) approach 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 paper for further details [Cignoni et al. 2003b].

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 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.

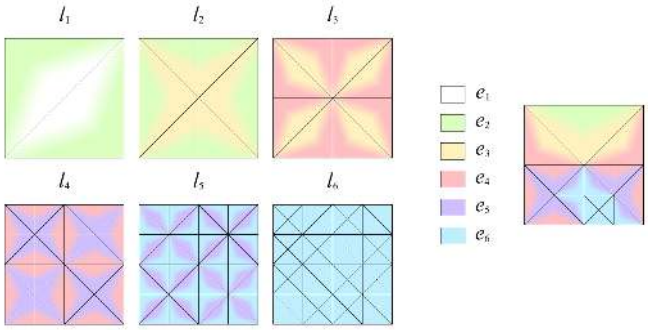


Figure 2: **An example of a BDAM**: each triangle represents a terrain patch composed by many triangles. Colors correspond to different errors; the blending of the color inside each triangle corresponds to the smooth error variation inside each patch.

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 section 5, to design a distributed out-of-core high quality simplification algorithm that concurrently builds all patches. Figure 2 illustrates 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.

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 1998], that supports the extraction of a correct set of triangular patches with a simple stateless refinement visit of the hierarchy, 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.

4 Planet Sized Batched Dynamic Adaptive Meshes (P-BDAM)

The P-BDAM approach exploits many the of ideas introduced in BDAM, and improves the general framework in a number of ways, allowing the correct management of non-flat planet-sized datasets, handling floating point precision issues, and exploiting speculative prefetching and a compressed on-disk representation.

4.1 Planet Partitioning

In order to handle the size and accuracy problems related to planet-sized terrain management, we partition the surface of the planet in a number of square tiles, therefore managing a forest of BDAM hierarchies instead of a single tree. The tiles have an associated (u,v) parameterization, which is used for texture coordinates and to construct the geometry subdivision hierarchy (see figure 3). 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, typically less than 3GB because of memory segmentation). 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 floating point 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.

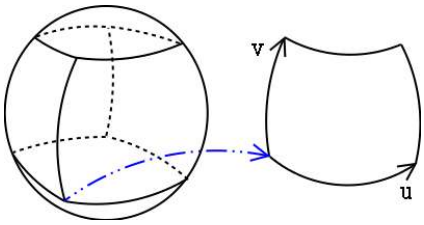


Figure 3: **Data Tiling.** To cope with accuracy and address space limitations, large datasets are decomposed into tiles with a local parameterization.

4.2 Parametric Patch Representation

Our solution to handling the global accuracy problem is based on the approximation of each P-BDAM patch by a displaced triangle and the use of the vertex programming capabilities of modern GPUs to efficiently render the patches with the required accuracy, while unconditionally maintaining geometric continuity.

We represent P-BDAM patches as arbitrary triangulations of points on a displaced triangle. Each P-BDAM base corner vertex contains a pair of texture coordinates $\mathbf{T}_i$, that correspond to the position of the vertex in (u,v) coordinates, as well as a planetocentric position $\mathbf{P}_i$ and a normal vector $\mathbf{N}_i$, that are computed from $\mathbf{T}_i$ at patch construction time as a function of the particular projection used. The vertices $\mathbf{Q}_j$ 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 (see figure 4 left). As for BDAM, the interior of the patch is an arbitrary triangulation of the vertices, that is represented by a cache-coherent generalized triangle strip stored as a single ordered list of vertex indices (see figure 4 right).

The only aspect that requires particular care is the computation of planetocentric positions, since all other information is local to the patch. We therefore store $\mathbf{P}_i$ in double precision. At each frame, we render all patches in camera coordinates, simply subtracting the camera position $\mathbf{O}$ from $\mathbf{P}_i$ on the host before converting them to single precision for communicating it to the graphics hardware. 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 we want. In contrast to common linear transformation approaches [Lindstrom et al. 1997; Reddy et al. 1999], 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 (9 subtractions and 9 floating point conversion) 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

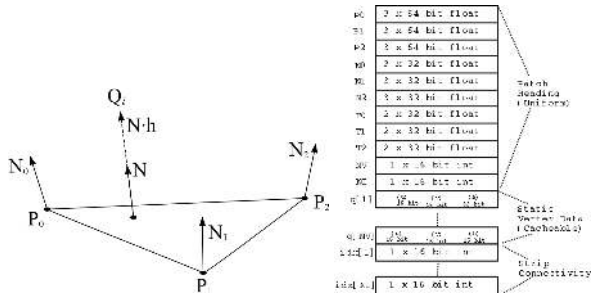


Figure 4: **P-BDAM patch.** Left: P-BDAM patches are represented as arbitrary triangulations of points on a displaced triangle. Right: the memory layout is optimized for rendering using OpenGL vertex array extensions.

data on the GPU, using a simple vertex program (see figure 5). This has the important advantage that, since the the vertices of the internal triangulation are invariant in barycentric coordinates, they can be stored in a static vertex array directly in graphics memory, and the rendering routine can fully benefit of the post-transform-and-lighting cache of current graphics architectures, which is fully exploited when drawing from the indexed representation. In particular, since the vertex program is executed only at cache misses, its cost is amortized over multiple vertices. Figure 4 right illustrates the memory layout employed, that is optimized for rendering using OpenGL vertex array extensions. It is important to note that approx-

```
struct a2v {
  float4 uvh: POSITION; // barycentric position and displacement
};

struct v2f {
  float4 hpos: HPOS; // view normalized position
  float4 tex0: TEX0; // texture coord. for mapping
  float4 tex1: TEX1; // texture coord. for bottom/left clipping
  float4 tex2: TEX2; // texture coord. for top/right clipping
};

v2f vpl_displaced_tripatch(
   a2v vertex,
   uniform float4x4 pvm,
   uniform float4 texture_border_width,
   uniform float4 one_minus_two_texture_border_width,
   uniform float4 P0,   uniform float4 T0,   uniform float4 N0,
   uniform float4 P0P1, uniform float4 T0T1, uniform float4 N0N1,
   uniform float4 P0P2, uniform float4 T0T2, uniform float4 N0N2) {
  v2f result;

  // Uncompress displacement (two shorts to a float)
  float h = vertex.uvh[2]*327.67 + vertex.uvh[3]*0.01;

  // Interpolate using barycentric coordinates
  float4 N = N0 + vertex.uvh[0]*N0N1 + vertex.uvh[1]*N0N2;
  float4 P = P0 + vertex.uvh[0]*P0P1 + vertex.uvh[1]*P0P2 + h*N;
  float4 T = T0 + vertex.uvh[0]*T0T1 + vertex.uvh[1]*T0T2;

  // Compute output position and texture coordinates
  result.hpos = mul(pvm,P);
  result.tex0 = texture_border_width +
                T*one_minus_two_texture_border_width;
  result.tex1 = T;
  result.tex2 = float4(1,1,1,1)-T;
  return result;
}
```

Figure 5: **Vertex program for rendering a P-BDAM patch.** The program performs both vertex unpacking and interpolation.

imating arbitrary map projections with displacements along interpolated normal directions introduces a representation error. However, this error can be fully taken into account by incorporating it

in the object-space representation error computed during simplification. Moreover, this error rapidly converges to zero for arbitrary map projections as geodetic lines converge to straight lines.

4.3 Adaptive rendering

Rendering a P-BDAM structure is similar to rendering a sequence of BDAM trees. Before rendering, the vertex program of figure 5 is installed in the graphics hardware. For each of the partitions that compose the planet, 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. At the end, the vertex program is disabled and OpenGL rendering can proceed as usual for non-terrain data. 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. The installed vertex program performs data unpacking and conversion from barycentric to Cartesian/texture coordinated.

4.4 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. As in BDAM, 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 (see [Cignoni et al. 2003b] for details). 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.

Patch Compression. Several clever schemes have been developed to concisely encode triangle strip connectivity, in as few as 1-2 bits per triangle (e.g. [Isenburg 2001]). As a result, the major portion of a compressed BDAM patch goes to storing internal mesh vertex barycentric positions and offsets. Standard mesh compression solutions typically combine quantization, local prediction, and variable-length delta encoding [Alliez and Desbrun 2001; Bajaj et al. 1999; Chow 1997; Deering 1995]. In this work, we have chosen a simple solution that favors mesh decompression speed over compression ratio. We quantize the barycentric coordinates to 12 bits (sufficient to encode over 16M positions in a triangular patch), reorder vertices in strip occurrence order, delta encode them, and compress the result using the LZO lossless compression method[1]. This typically achieves compresses data to less than 50% of original size, while supporting a decompression rate of over 15M triangles/second on typical PC hardware (see results section).

[1]LZO is a data compression library based on a Lempel Ziv variant which is suitable for data decompression in real-time. The library source is available from `http://www.oberhumer.com/opensource/lzo/`

Speculative prefetching. The external memory component of P-BDAM, as for BDAM [Cignoni et al. 2003b] and SOAR [Lindstrom and Pascucci 2002], is based on associating to each terrain partition a file dynamically mapped to a read-only logical address space, and on rearranging terrain data so that it can be accessed in a memory coherent manner. Since in typical terrain exploration tasks the viewer's expected near future positions can be extrapolated with good accuracy based on the last few positions, we can further hide the disk latency by prefetching geometry and texture data that will soon be accessed. The prefetching routine, that may be executed in parallel to the rendering thread or sequentially as an idle task, executes the same refinement algorithm as the adaptive rendering code, taking as input the predicted camera position instead of the current one. When the refinement terminates, instead of rendering the patches or binding the textures, it simply checks whether the required graphics objects are in the cache. If not, it advises the operating system kernel that the pages containing their representation will likely be accessed in the near future. On Linux, this is done by executing the `madvise` system call, to instruct the kernel that it would be advantageous to asynchronously read the indicated pages ahead if they are not already in core. This technique blends well with the virtual memory based external memory management subsystem. In particular, the main rendering code doesn't need to be aware of the prefetching component, and we exploit the extensive performance optimizations of the operating system's virtual memory manager, such as reordering of requests to reduce seek access time and overlapping of computation and disk access [Gorman 2003].

5 Building a P-BDAM

A P-BDAM structure is constructed starting from a generic height field representing the surface of a planet. There are two main tasks that have to be accomplished: preparing the input data in order to build the lowest level of the P-BDAM hierarchy and performing the bottom-up simplification that builds the patches of the upper levels.

Data preparation and resampling. Size and accuracy constraints impose a partitioning of the data, that must be done in a way that no triangle crosses partition/texture borders. No resampling is strictly necessary, since for all map projections it is possible to rearrange the original data samples in order to meet the constraints. The most commonly used maps, however, do not evenly sample the planet surface, and are often discontinuous at the poles. As a first step, we thus in general prefer to resample data using a cubical map projection. Starting from the cubemap tile structure shown in Fig. 3, we recursively subdivide each face. The new vertices are placed onto the surface of the planet and recursive subdivision stops when each patch is small enough to sample the original data with the required precision. The final triangles are then grouped in batches to form the leaves of the geometry bintrees.

Out-of-core Geometric Simplification. Once all the leaves of all the bintrees have been generated, we perform a bottom-up simplification to build the patches of the higher levels. The construction of our multiresolution model is carried out as an iterated sequence of mark, distribute, independent simplify and patch merging steps. During the mark phases, we constrain the vertices on the longest edges of the patches to remain unmodified during the subsequent simplification step. In order to maintain geometric continuity, planet partitions are considered as a single surface. This marking subdivides the surface in independent blocks, each one formed by four triangular patches, possibly from different planet partitions, joined by their short edges. This allows to simplify all the blocks in parallel. The simplification is targeted to halve the

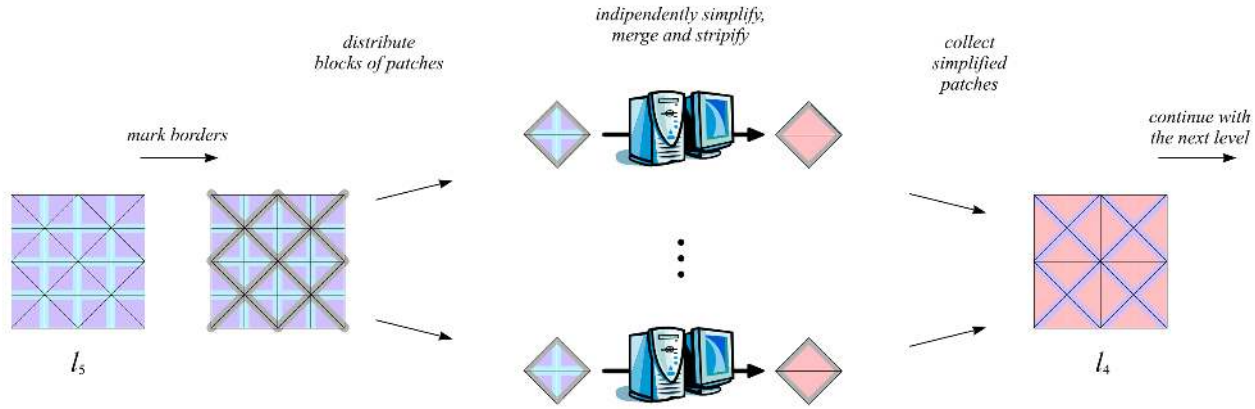


Figure 6: Construction of a BDAM through a sequence of simplification and marking steps. Each triangle represents a terrain patch composed by many triangles. Colors correspond to different errors as in Fig 2.

number of vertices inside the block, so, once simplified, the triangles of a block can be partitioned in two patches which respect the patch size constraint. The simplification process takes into account that the two resulting patches must have no triangle crossing the borders, so the vertices on the diagonal are forced to remain on it. Figure 6 illustrates this process: each triangle color corresponds to an increasing approximation error. With bold gray lines we show the mesh portion marked as un-modifiable (that is a local portion of the mesh that remains unaffected by the subsequent simplification step). Note that each triangular patch has the longest edge of the error (color) of the previous level, hence the error distribution of each patch, shown in figure 2, is respected. The whole simplification process is inherently parallel, because the grain of the individual tasks is very fine and synchronization is required only at the completion of each bintree level. This is very important, because the high quality simplification of gigatriangles meshes is a very time consuming task, that would require days to be completed on a single machine. Individual patch pairs are seimplified using greedy edge collapse driven by the quadric error metric [Garland and Heckbert 1997]. Then, the error of the simplified mesh is taken as the maximum difference with the original one in planetocentric reference system (normal to the planet surface). To perform this computation quickly by exploiting graphics hardware, we render the original and simplified meshes under a perspective projection originating from the center of the planet and we evaluate the difference among the corresponding depth buffers. In the preprocessing step, textures are also sampled and converted into a hierarchical structure by a bottom-up filtering process followed by a compression to the DXT1 format. As for geometry, error nesting is ensured by a final bottom-up pass that combines all object space errors of neighboring and descendant tiles. Further details on the construction process can be found in [Cignoni et al. 2003b].

6 Results

An experimental software library and a planet rendering application supporting the P-BDAM technique has been implemented and tested on Linux and Windows NT machines.

The test case discussed in this paper is the interactive exploration of a near-global topographic map of planet Mars created from Mars Orbiter Laser Altimeter data. The dataset covers latitudes from 88 North to 88 South and the full longitude range at 128 samples per degree [Smith et al. 2002]. The original maps are provided in simple cylindrical projection. To avoid discontinuity problems at the Poles, we reprojected the maps using a cubical projection, maintaining roughly the same number of points. The resulting dataset is composed of 6 tiles of 13313^2 sample points, for a total of over 1G vertices and 2G triangles. The terrain was textured using six 16384^2 shaded relief textures, for a total of over 1.5G RGB texels.

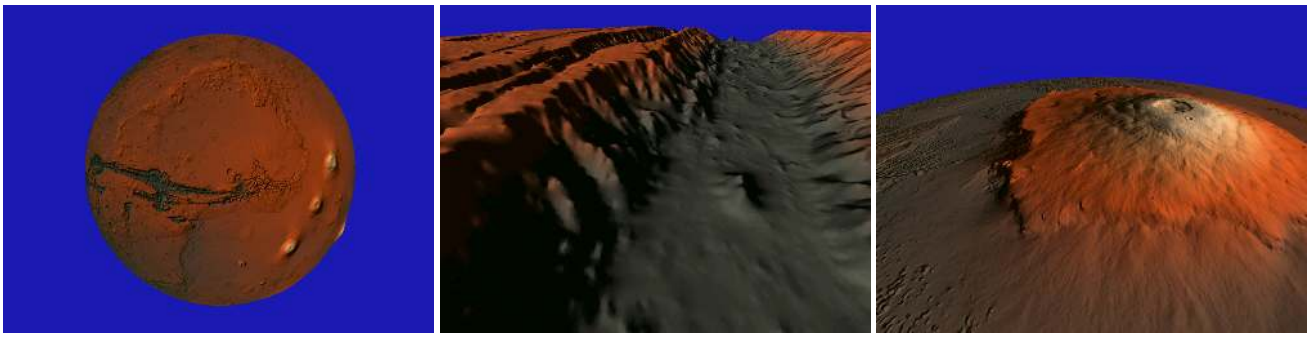
6.1 Preprocessing

The dataset was partitioned into six partitions (corresponding to the six faces of the cubical projection map). The resampling step took about one hour for the texture and 36 min for the geometry. The preprocessing time for geometry is largely dominated by the simplification task, that was for this reason parallelized. Thanks to the asynchronous nature of our parallel simplification algorithm we are able to successfully run the process with no constraint on the performance of PC and/or network connection. The equipment used for the experiments was just a standard PC network composed by the author's workstations: five 1.6GHz PCs connected by a 10Mb Ethernet network.

For geometry, we generated 12 bintrees, with leaf nodes containing triangular patches of 26x26 vertex side at full resolution and interior nodes with a constant vertex count of 512. The simplification process built the multiresolution structure starting from the leaves of the bintrees in 6h10min (wall clock time). Each bintree occupied at the end of the process 737 Mb for a total 8.63 GB needed to store the whole geometry on disk before patch compression. From some experiments on smaller sized datasets we can estimate the speed up obtained by our parallel pre-processing to be about 3.9 on the five PC network. The compression of the geometry bintrees took only 20 minutes (executed in parallel) and reduced dataset size to roughly 4.5 GB. For textures, we used a tile size of 256x256 texels, which produced six 7 level quadtrees and compressed colors using the DXT1 format. Texture preprocessing, including error propagation, took roughly one hour per tile on a single processor and produced a structure occupying 1.2 GB on disk for the entire dataset. Processing time is largely dominated by texture compression.

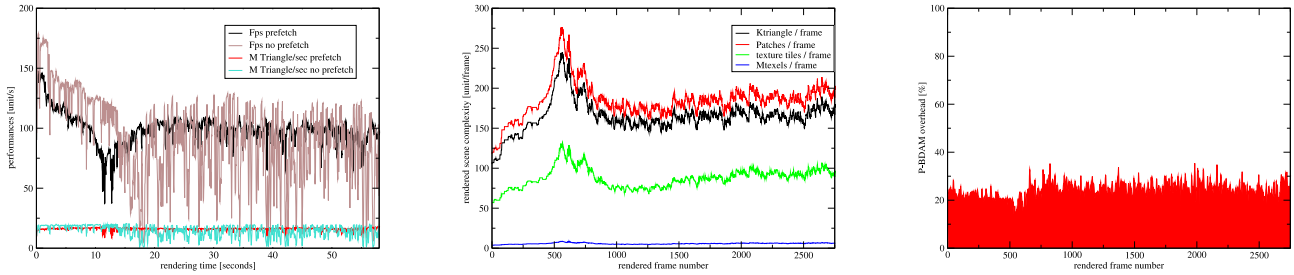
6.2 View-dependent Refinement

We evaluated the rendering performance of the P-BDAM technique on a number of flythrough sequences. The quantitative results presented here were collected during a 58 seconds high speed fly se-



(a) Frame 100: Approaching Mars (b) Frame 1240: Inside Mariner Valley (c) Frame 2780: Arriving at the Olympus Mons

Figure 7: **Selected flythrough frames.** Screen space error tolerance set to 3 pixels.



(a) Rendering rates per frame (b) Rendered complexity per frame (c) P-BDAM refinement overhead per frame

Figure 8: **Performance Evaluation.** To fully test our out-of-core data management components, all benchmarks were started with all data off core and disk buffers flushed.

quence that included an approach to the planet, a low-altitude flyover of the Mariner Valley and the Tharsis volcano region and a simulated landing on top of Olympus Mons. The average speed of the flight exceeds Mach 700. The flight, performed using a window size of 640x480 pixels and a screen tolerance of 3 pixels, was designed to heavily stress the system, as it includes abrupt rotations and changes from overall views to close-ups. The results were collected on a Linux 2.4 PC with two AMD Athlon MP 1600MHz processors, 2GB RAM, a NVIDIA GeForce4 Ti4600 graphics board, and a MAXTOR 6L060J3 60GB IDE disk. The qualitative performance of our adaptive renderer is further illustrated in an accompanying video, that shows live recordings of the analyzed flythrough sequence (Fig. 7). To fully test our out-of-core data management components, the benchmarks were started with all data off core and disk buffers flushed. During the entire walkthrough, the resident set size of the application is maintained at roughly 98 MB, i.e. less than 2% of data size (5.7 GB), demostrating the effectiveness of out-of-core data management.

Figure 8(a) illustrates the rendering performance of the application, both with and without speculative prefetching. The speculative prefetching version executed an additional refinement step per frame to prefetch pages that are likely to be accessed in the near future, using a linear prediction of camera position with a half a second look-ahead. The prefetching version is much smoother, due to the success of prefetching in hiding the latency due to page faults. The only noticeable jitters with the prefetching version (visible in the graph near second 12) correspond to rapid rotations and accelerations of the path. By contrast, the non prefetching version has a much lower average performance and frequently stalls during rendering due to paging latency. In the prefetching version, we were able to sustain an average rendering rate of roughly 16 millions of textured triangles per second, with peaks exceeding 18.5 millions. By comparison, on the same machine, SOAR [Lindstrom and Pascucci 2002] peak performance for a 4Kx4K subset of dataset was measured at roughly 3.3 millions of triangles per second, even though SOAR was using a smaller single resolution texture of 2Kx2K texels and did not handle surface curvature. The increased performance of the P-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, reducing the overhead due to the vertex program.

The time overhead of P-BDAM structure traversal, measured by repeating the test without executing OpenGL calls, is only about 22% of total frame time (Fig. 8(c)), demonstrating that we are GPU bound even for handling extremely large out-of-core data sets. Rendered scene granularity is illustrated in figure 8(b): even though the peak complexity of the rendered scenes exceeds 240K triangles and 8.5M texels per frame, the number of rendered graphics primitives per frame remains relatively small, never exceeding 280 patches and 130 texture blocks per frame. Since we are able to render such complex scenes at high frame rates, it is possible to use very small pixel thresholds, virtually eliminating popping artifacts, without the need to resort to costly geomorphing features.

7 Conclusions

We have presented an efficient technique for out-of-core management and interactive rendering of planet sized textured terrain surfaces. The proposed framework introduces several advances with

respect to the state of the art: thanks to a batched host-to-graphics communication model, we outperform current adaptive tessellation solutions in terms of rendering speed; we guarantee overall geometric continuity, exploiting programmable graphics hardware to cope with the accuracy issues introduced by single precision floating point representations; we successfully exploit a compressed out of core representation and speculative prefetching for hiding disk latency during rendering of out-of-core data; we efficiently handle the construction of high quality simplified representations by using a novel distributed out of core simplification algorithm working on a standard PC network.

Acknowledgments

This research is partially supported by the V-PLANET project (European RTD contract IST-2000-28095). We are grateful to the NASA MOLA Science Team for making their data available.

References

ALLIEZ, P., AND DESBRUN, M. 2001. Progressive compression for lossless transmission of triangle meshes. In *SIGGRAPH '2001 Conference Proceedings*, 198–205.

BAJAJ, C. L., PASCUCCI, V., AND ZHUANG, G. 1999. Progressive compression and transmission of arbitrary triangular meshes. In *Proceedings of the 1999 IEEE Conference on Visualization (VIS-99)*, ACM Press, D. Ebert, M. Gross, and B. Hamann, Eds., 307–316.

CHOW, M. M. 1997. Optimized geometry compression for real-time rendering (color plate S. 559). In *Proceedings of the 8th Annual IEEE Conference on Visualization (VISU-97)*, IEEE Computer Society Press, Los Alamitos, R. Yagel and H. Hagen, Eds., 347–354.

CIGNONI, P., PUPPO, E., AND SCOPIGNO, R. 1997. Representation and visualization of terrain surfaces at variable resolution. *The Visual Computer 13*, 5, 199–217.

CIGNONI, P., MONTANI, C., ROCCHINI, C., AND SCOPIGNO, R. 2003. External memory management and simplification of huge meshes. *IEEE Transactions on Visualization and Computer Graphics 9*, (in press).

CIGNONI, P., GANOVELLI, F., GOBBETTI, E., MARTON, F., PONCHIO, F., AND SCOPIGNO, R. 2003. BDAM – batched dynamic adaptive meshes for high performance terrain visualization. *Computer Graphics Forum 22*, 3 (Sept.). To appear.

COHEN-OR, D., AND LEVANONI, Y. 1996. Temporal continuity of levels of detail in delaunay triangulated terrain. In *IEEE Visualization '96*, IEEE. ISBN 0-89791-864-9.

DEERING, M. 1995. Geometry compression. In *Comp. Graph. Proc., Annual Conf. Series (SIGGRAPH 95), ACM Press*, 13–20.

DÖLLNER, J., BAUMANN, K., AND HINRICHS, K. 2000. Texturing techniques for terrain visualization. In *IEEE Visualization '00 (VIS '00)*, IEEE, Washington - Brussels - Tokyo, 227–234.

DUCHAINEAU, M., WOLINSKY, M., SIGETI, D., MILLER, M., ALDRICH, C., AND MINEEV-WEINSTEIN, M. 1997. ROAMing terrain: Real-time optimally adapting meshes. In *Proceedings IEEE Visualization '97*, IEEE, 81–88.

EVANS, W., KIRKPATRICK, D., AND TOWNSEND, G. 2001. Right triangulated irregular networks. *Algorithmica 30*, 2 (Mar), 264–286.

GARLAND, M., AND HECKBERT, P. 1997. Surface simplification using quadric error metrics. In *SIGGRAPH 97 Conference Proceedings*, Addison Wesley, Annual Conference Series, 209–216.

GORMAN, M. 2003. Understanding the linux virtual memory manager. Tech. rep., University of Limerick, IE.

HOPPE, H. 1997. View-dependent refinement of progressive meshes. In *SIGGRAPH 97 Conference Proceedings*, Addison Wesley, T. Whitted, Ed., Annual Conference Series, ACM SIGGRAPH, 189–198. ISBN 0-89791-896-7.

HOPPE, H. 1998. Smooth view-dependent level-of-detail control and its aplications to terrain rendering. In *IEEE Visualization '98 Conf.*, 35–42.

ISENBURG, M., AND GUMHOLD, S. 2003. Out-of-core compression for gigantic polygon meshes. In *ACM Siggraph Conference Proceedings 2003*. to appear.

ISENBURG, M. 2001. Triangle strip compression. *Computer Graphics Forum 20*, 2, 91–101. ISSN 1067-7055.

LEVENBERG, J. 2002. Fast view-dependent level-of-detail rendering using cached geometry. In *Proceedings IEEE Visualization '02*, IEEE, 259–266.

LINDSTROM, P., AND PASCUCCI, V. 2002. Terrain simplification simplified: A general framework for view-dependent out-of-core visualization. *IEEE Transaction on Visualization and Computer Graphics 8*, 3, 239–254.

LINDSTROM, P., KOLLER, D., RIBARSKY, W., HODGES, L., FAUST, N., AND TURNER, G. 1996. Real-time, continuous level of detail rendering of height fields. In *Comp. Graph. Proc., Annual Conf. Series (SIGGRAPH 96), ACM Press*, 109–118.

LINDSTROM, P., KOLLER, D., RIBARSKY, W., HODGES, L. F., DEN BOSCH, A. O., AND FAUST, N. 1997. An integrated global gis and visual simulation system. Tech. Rep. 97-07, Graphics, Visualization and Usability Center, Georgia Tech, USA.

LINDSTROM, P. 2000. Out-of-core simplification of large polygonal models. In *Comp. Graph. Proc., Annual Conf. Series (SIGGRAPH 2000), ACM Press*, Addison Wesley, 259–262.

LINDSTROM, P. 2003. Out-of-core construction and visualization of multiresolution surfaces. In *ACM 2003 Symposium on Interactive 3D Graphics*.

PAJAROLA, R., ANTONIJUAN, M., AND LARIO, R. 2002. Quadtin: Quadtree based triangulated irregular networks. In *Proceedings IEEE Visualization '02*, IEEE, 395–402.

PAJAROLA, R. 1998. Large scale terrain visualization using the restricted quadtree triangulation. In *Proceedings of Visualization '98*, H. R. D. Elbert, H. Hagen, Ed., 19–26.

PAJAROLA, R. 2002. Overview of quadtree based terrain triangulation and visualization. Tech. Rep. UCI-ICS TR 02-01, Department of Information, Computer Science University of California, Irvine, Jan.

POMERANZ, A. A. 2000. *ROAM Using Surface Triangle Clusters (RUSTiC)*. Master's thesis, University of California at Davis.

PUPPO, E. 1996. Variable resolution terrain surfaces. In *Proceedings Eight Canadian Conference on Computational Geometry, Ottawa, Canada*, 202–210.

REDDY, M., LECLERC, Y., IVERSON, L., BLETTER, N., AND VIDIMCE, K. 1999. Modeling the digital earth in vrml. In *In Proceedings of SPIE - The International Society for Optical Engineering, vol. 3905*.

SAMET, H. 1990. *Applications of Spatial Data Structures*. Addison Wesley, Reading, MA.

SMITH, D., NEUMANN, G., ARVIDSON, R. E., AND GUINNESS, E. A., 2002. Mars global surveyor laser altimeter mission experiment gridded data record. NASA Planetary Data System, MGS-M-MOLA-5-MEGDR-L3-V2.0, Oct.

TANNER, C. C., MIGDAL, C. J., AND JONES, M. T. 1998. The clipmap: A virtual mipmap. In *SIGGRAPH 98 Conference Proceedings*, Addison Wesley, M. Cohen, Ed., Annual Conference Series, ACM SIGGRAPH, 151–158.

XIA, J., AND VARSHNEY, A. 1996. Dynamic view-dependent simplification for polygonal models. In *IEEE Visualization '96 Proc.*, R. Yagel and G. Nielson, Eds., 327–334.