High-quality networked terrain rendering from compressed bitstreams

Fabio Bettio
CRS4

Enrico Gobbetti
CRS4*

Fabio Marton
CRS4

Giovanni Pintore
CRS4

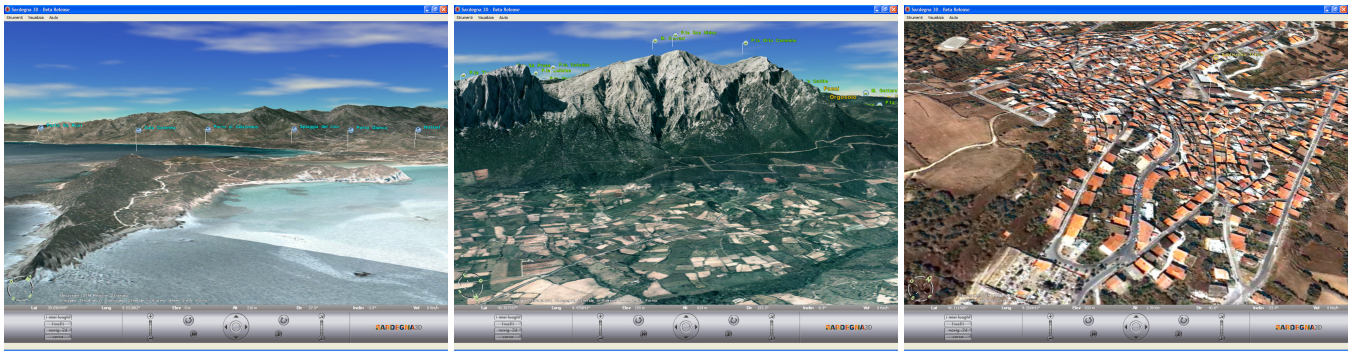


Figure 1: **Real-time exploration of remote terrain models.** Our method is capable of producing high quality seamless renderings of terrain data at high frame rates from highly compressed bitstreams, improving the scalability of servers and the behavior of clients in network environments with narrow bandwidth and low computational power. In this example, we show three frames of an interactive session with an internet geo-viewing tool based on our library exploring a detailed height colored terrain model at high frame rates over an ADSL 4Mbps network. See also figure 7 (color plate).

Abstract

We describe a compressed multiresolution representation and a client-server architecture for supporting interactive high quality remote visualization of very large textured planar and spherical terrains. Our approach incrementally updates a chunked level-of-detail BDAM hierarchy by using precomputed wavelet coefficient matrices decoded from a compressed bitstream originating from a thin server. The structure combines the aggressive compression rates of wavelet-based image representations with the ability to ensure overall geometric continuity for variable resolution views of planar and spherical terrains with no need for run-time stitching. The efficiency of the approach is demonstrated on a large scale interactive remote visualization of global and local terrains on ADSL networks. A library implementing an early version of this work has been incorporated into a widely distributed geo-viewing system with tens of thousands of clients.

CR Categories: I.3.3 [Computer Graphics]: Picture and Image Generation—; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—.

Keywords: Out-Of-Core Algorithms, Network Streaming, Data Compression, Level of Detail

1 Introduction

Real-time 3D exploration of remote digital elevation models built from high resolution imagery and elevation data has long been one of the most important components in a number of practical applications, and extensive research has been carried out in terms of methods and techniques for processing, distributing, and rendering very large datasets. The increased availability of broadband networks and high performance graphics PCs has made this technology, once limited to professional applications, increasingly popular, as testified by the success of Internet geo-viewing tools like Google Earth, NASA WorldWind, and Microsoft Virtual Earth.

The efficient implementation of such tools requires a combination of technologies for adaptively rendering high quality terrain views at high frame rates and techniques for efficiently streaming data from the systems serving the terrain database to a large number of rendering clients.

At the present time, the vast majority of large scale networked terrain visualization systems are based on variations of tiled quads data structures, which partition the terrain into independent square patches tessellated at different resolutions. Since it is not possible to generate seamless variable resolution tilings by combining axis-aligned squares, these methods require run-time work to stitch block boundaries, leading to CPU work and inefficient GPU utilization. It is worth noting that when using independent square tiles, e.g, coming from a quadtree partitioning, stitching cannot be fully incorporated into a preprocess, since view-dependent rendering leads to the need of dealing with variable neighborhoods.

In recent years, very efficient seamless techniques for high quality variable resolution rendering from aggressively compressed datasets have been introduced into the visual simulation domain (e.g., [Losasso and Hoppe 2004; Gobbetti et al. 2006]), but the proposed approaches are tuned to high end graphics boards with local access to the data.

Contribution. In this paper, we describe a compressed multiresolution representation and a client-server architecture for supporting interactive high quality remote visualization of very large textured planar and spherical terrains. Our approach incrementally updates a chunked level-of-detail BDAM hierarchy by using precomputed wavelet coefficient matrices decoded from a compressed bitstream originating from a thin server. The encoding, in contrast to C-BDAM [Gobbetti et al. 2006] follows a reversible integer-to-integer wavelet scheme, which can be constructed using a parallel single pass compression process tuned for near-lossless or mean-square error metrics and can be decoded by clients with low computational power. The methods strives to combine the generality and adaptivity of chunked bintree multiresolution structures with the compression rates of wavelet image compressors and the streaming abilities of tiled quadtree approaches. Similarly to BDAM, coarse grain refinement operations are associated to regions in a bintree hierarchy.

*CRS4 Visual Computing Group, POLARIS Edificio 1, 09010 Pula, Italy www: http://www.crs4.it/vic/ e-mail: {fabio|gobbetti|marton|gianni}@crs4.it

Each region, called diamond, is formed by two triangular patches that share their longest edge and tessellate the model using the same regular triangulation connectivity. The compressed data structure requires storage of a single square matrix of wavelet coefficients per diamond. 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 per diamond wavelet coefficient matrices required for refining are incrementally retrieved from the server in the form of a compressed bitstream. At each frame, updates are communicated to the GPU with a batched communication model.

Advantages. The structure provides a number of benefits: efficient client-server communication using a compressed bitstream; server scalability; efficient one-pass compression and real-time integer-based decompression; overall geometric continuity for planar and spherical domains; support for variable resolution input data; management of multiple vertex attributes. As highlighted in section 2, while other techniques share some of these properties, they typically do not match the capabilities of our method in all of the areas. We are therefore capable of producing high quality seamless renderings of terrain data at high frame rates from highly compressed sources, improving the scalability of servers and the behavior of clients in network environments with narrow bandwidth and low computational power.

Limitations. The proposed method has also some limitations. As for state-of-the-art rendering systems from compressed datasets [Losasso and Hoppe 2004; Gobbetti et al. 2006], the compression method is lossy and assumes that the terrain has bounded spectral density, which is the case for typical remote sensing datasets. In contrast to [Gobbetti et al. 2006] aggressively compressed datasets are unable to guarantee absolute error tolerances. We consider this acceptable for a streaming application. Unlike tiled quadtree approaches, the method provides faster seamless high quality renderings and higher compression rates at the expense of a pre-processing of the entire database.

Our approach adopts the philosophy of the BDAM breed of techniques. The new approach for efficiently encoding and streaming terrain datasets is described in section 3. Section 4 illustrates the client-server architecture. The efficiency of the method has been successfully evaluated on a number of test cases, discussed in section 5.

2 Related work

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

Efficient adaptive rendering. The vast majority of adaptive terrain rendering approaches have historically dealt with large triangle meshes computed on the regularly distributed height samples of the original data, using either irregular (e.g., [De Floriani et al. 1997; Hoppe 1998]) or semi-regular adaptive triangulations (e.g., [Lindstrom et al. 1996; Duchaineau et al. 1997; Pajarola 1998; Lindstrom and Pascucci 2001]). The main objective of this kind of algorithms was to compute the minimum number of triangles to render each frame, so that the graphic board would be able to sustain the rendering. More recently, the impressive improvement of the graphics hardware shifted the bottleneck from the GPU to the CPU. For this reason many techniques were proposed to reduce per-triangle workload by composing pre-assembled optimized surface patches at run-time. Tiled blocks techniques (e.g., [Hitchner and Mcgreevy 1993; Wahl et al. 2004]), originally designed for external data management purposes, and now ubiquitous for networked applications, partition the terrain into square patches tessellated at different resolutions. The main challenge is to seamlessly stitch block boundaries, using run-time generated geometry. The first methods capable of producing adaptive conforming surfaces by composing precomputed patches with a low CPU cost were explicitly designed for terrain rendering. RUSTIC [Pomeranz 2000] and CABTT [Levenberg 2002] are extensions of the ROAM [Duchaineau et al. 1997] algorithm, in which subtrees of the ROAM bintree are cached and reused during rendering. A similar technique is also presented in [DeCoro and Pajarola 2002] for generic meshes. BDAM [Cignoni et al. 2003a; Cignoni et al. 2003b] constructs a forest of hierarchies of right triangles, where each node is a general triangulation of a small surface region, and explicitates the rules required to obtain globally conforming triangulations by composing precomputed patches. A similar approach, but described in terms of a 4-8 hierarchy, is described in [Hwa et al. 2005], which store textures and geometry using the same technique.

Streaming and compression. Various authors have recently 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, limiting the achievable compression rates. Many compression algorithms have been proposed in the image processing field that are able to provide high compression ratios, image streaming, and access to image portions. JPEG2K is a standard with outstanding capabilities. Image based techniques, however, do not meet all the requirements for a terrain rendering applications. In particular, the rapid assembly of variable resolution approximations is not supported, as fast extraction of blocks requires, at best, server side work, and produces axis-aligned tiles. Kim and Ra proposed a networked visualization algorithm combined with wavelet-based compression that can work with a thin server [Kim and Ra 2004]. In their approach, a restricted quadtree mesh model is updated in real time by using wavelet coefficients decoded from a compressed bitstream. However, per-vertex adaptation limits GPU efficiency and the proposed blockwise processing method requires access to multiple blocks at reconstruction time. Geometry clipmaps [Losasso and Hoppe 2004] organize the terrain height data into a pyramidal multiresolution scheme and the residual between levels is compressed using an advanced image coder that supports fast access to image regions [Malvar 2000]. Storing in a compressed form just the heights and reconstructing at runtime both normal and color data (using a simple height color mapping) provides a very compact representation. However, the pyramidal scheme limits adaptivity, and the technique works best for wide field of views and nearly planar geometry. Moreover, CPU or GPU processing is required to blend clipmap levels. An extension to spherical datasets has been recently proposed [Clasen and Hege 2006], but it requires heavy per-vertex trigonometric GPU computations. A networking version of the planar-only version has been proposed [Deb and Narayanan 2006], but it is tuned for a small number of clients per server, as the server module has to explicitly track the dynamic state of each client. In [Hwa et al. 2005], 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. Our work proposes an efficient method for incorporating compression of height and texture information into the BDAM framework. A similar approach has been taken in the CBDAM framework [Gobbetti et al. 2006], which recasts diamond processing in the framework of wavelet update lifting [Jr. et al. 2003], with the purpose of transforming data into a domain with a sparser representation amenable to efficient compression. In this framework, diamond coarsening is associated to wavelet analysis, while diamond refinement corresponds to wavelet synthesis. This approach supports maximum error metrics control but requires storage of two matrix coefficients per diamond, and a

two-step construction process. By contrast, we recast BDAM in the framework of standard wavelet lifting [Kovacevic and Sweldens 2000] and propose a lighter integer-to-integer implementation more suited to a networking environment with heterogenous clients.

We have to note that many other authors have explored the problem of creating compressed representations for geometric data, but in most of these cases the focus is on compression ratio rather than the real-time view-dependent rendering from the compressed representation; for a recent survey of this field we refer the reader to [Alliez and Gotsman 2005].

3 Data representation

In high quality real-time networked terrain viewing applications the client has to rapidly adapt a seamless level-of-detail terrain representation in response to viewer motion, using data coming from the server. In this context, a compressed data representation must possess a number of requirements: ability support random access to different portions of the dataset; resolution scalability for progressive resolution increase; hierarchical structuring able to support variable resolution seamless LOD; high compression ratio, to reduce transmitted data; low server side complexity, to ensure scalability with number of cliens; low client side decoding complexity, to support heterogeneous clients. We meet these requirements by proposing a data representation based on an integer wavelet lifting transformation of the BDAM structure.

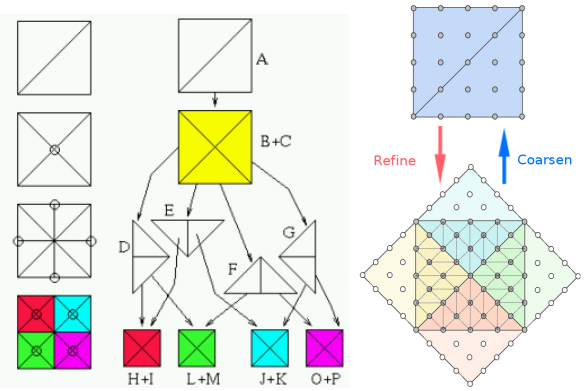


Figure 2: **BDAM on a regular grid.** Left: a diamond hierarchy. Right: diamond coarsening and refinement operations on a regular grid.

BDAM exploits the partitioning induced by a recursive subdivision of the input domain in a *hierarchy of right triangles*. The partitioning consists of a binary forest of triangular regions, whose roots cover the entire domain and whose other nodes are generated by triangle bisection. This operation consists in replacing a triangular region σ with the two triangular regions obtained by splitting σ at the midpoint of its longest edge. In order to guarantee that a conforming mesh is always generated after a bisection, the (at most) 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. This structure has the important property that, by selectively refining or coarsening it on a diamond by diamond basis while keeping the boundary vertices fixed, it is possible to extract conforming variable resolution mesh representations. The structure of a BDAM mesh, when operating on regular grids, is depicted in figure 2. As we can see, the two patches that form a diamond have their vertices placed on a uniform square grid, while the vertices of the refined patches are placed at the centers of the square cells of the grid. Diamond coarsening has thus to remove cell-centered values and filter vertex-centered ones, while diamond refinement has to undo the operation.

In order to represent data in a domain more amenable to compression, in this work diamond coarsening is associated to wavelet analysis, while diamond refinement corresponds to wavelet synthesis (see figure 3.

Data compression is performed by iterating a wavelet analysis process starting from the leaf level storing the original data and terminating at the root. The analysis process for constructing a level l diamond starts by gathering vertex data from child diamonds at level $l+1$ (or from the input dataset) into two interleaved square matrices of values: a matrix $V^{(l+1)}$ of vertex centered values, and a matrix $C^{(l+1)}$ of cell-centered values. A new set of vertex values $L^{(l)}$ and a matrix of detail coefficients $H^{(l)}$ can then be computed by the following high- and low-pass filtering operations:

$$H_{ij}^{(l)} = V_{ij}^{(l+1)} - \mathscr{P}_{ij}(C^{(l+1)}) \quad (1)$$

$$V_{ij}^{(l)} = C_{ij}^{(l+1)} + \frac{1}{2}\mathscr{U}_{ij}(H^{(l)}) \quad (2)$$

Here, $\mathscr{P}(.)$ is a prediction operator that predicts a vertex centered value from neighboring cell centered values, and $\mathscr{U}(.)$ is an update operator that averages the deltas on neighboring cell-centered values to update a cell-centered value. In order to comply with BDAM diamond boundary constraints, it is sufficient to set $\mathscr{U}_{ij}(.) = 0$ for all diamond boundary vertices, which ensures that boundary vertices remain unmodified by diamond filtering.

For prediction, we use a order 4 Neville interpolating filter if all support points fall inside the diamond, and an order 2 filter in the opposite case [Kovacevic and Sweldens 2000]. For update, we always use an order 2 filter. These filters predict points by a weighted sum of 12 (order 4) or 4 (order 2) coefficients, use only integer operations, and are therefore very fast to compute (see figure 3).

By iterating the analysis process from leaf diamonds up to the roots, all the input data get filtered up to the coarsest scale. The resulting wavelet representation is multiscale, as it represents the original terrain dataset with a set of coarsest scale coefficients associated to the root diamonds, plus a set of detail coefficients with increasingly finer resolution. During synthesis it is possible to produce variable resolution representations by refining a diamond at a time, using a process that simply reverses the analysis steps. At diamond refinement time, first face and vertex-centered values are computed from the coarser level diamond vertex values:

$$C_{ij}^{(l+1)} = V_{ij}^{(l)} - \frac{1}{2}\mathscr{U}_{ij}(H^{(l)}) \quad (3)$$

$$V_{ij}^{(l+1)} = H_{ij}^{(l)} + \mathscr{P}_{ij}(C^{(l+1)}) \quad (4)$$

Then, this data is reassembled and scattered to child diamonds.

The approach is similar to that of nonseparable wavelets applied to a quincunx lattice [Kovacevic and Sweldens 2000]. However, by constraining the boundary vertices of each diamond to remain fixed, and by restricting all other filters to use only diamond internal values, we can efficiently support variable resolution reconstructions with independent diamond operations. Moreover, these constraints do not induce a tiling effect, as no boundary remains locked for more then one level due to the properties of the BDAM hierarchy [Cignoni et al. 2003a].

Lossless, near-lossless, and lossy data compression. For typical terrain datasets, the wavelet representation produces detail coefficients that decay rapidly from coarse to fine scale, as most of the shape is captured by the prediction operators. By entropy coding the detail coefficients it is thus possible to efficiently compress the input dataset. The achievable lossless compression ratio is however limited. More aggressive lossy compression can be achieved by quantizing detail coefficients or discarding small ones. In C-BDAM [Gobbetti et al. 2006], the problem is solved by using a

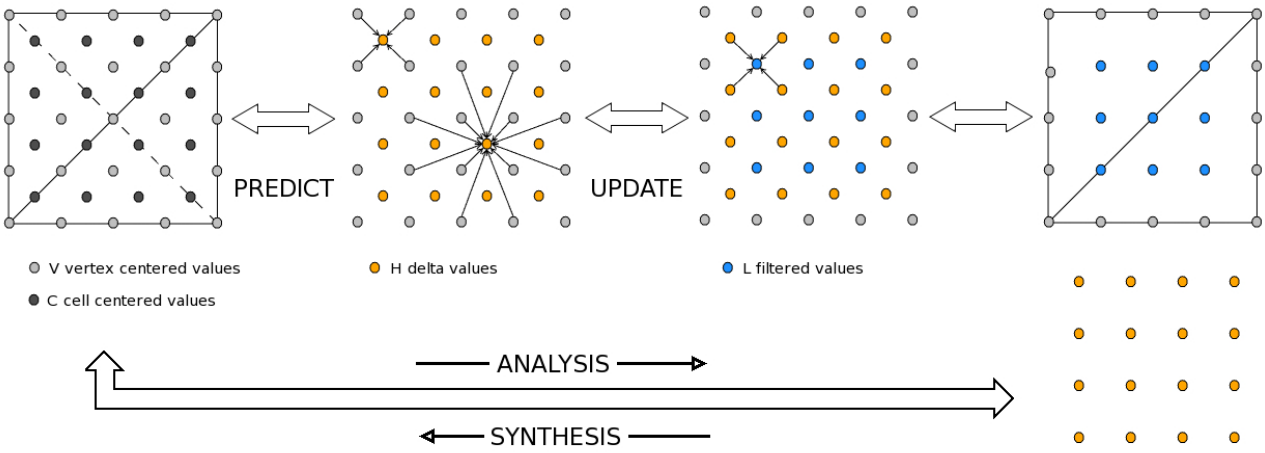


Figure 3: **Wavelet transformation in a BDAM diamond.** Diamond coarsening removes cell-centered values and filters vertex-centered ones, while diamond refinement performs the inverse operation.

two-stage near-lossless scheme which ensures that each diamond is within a given maximum value of the original filtered data, but requires a non reversible wavelet transformation based on update lifting, a two-step compression process, as well as storage and computation of two matrices per node. In this work, tuned for internet application with heterogeneous clients, we have opted for a more straightforward approach directly based on the reversible integer-to-integer wavelet representation. In order to obtain a compressed representation, we quantize coefficients with a uniform quantizer without deadzones and then entropy code the quantized results with the same quadtree-based approach used in [Gobbetti et al. 2006]. Lossless and near-lossless compression can be obtained by applying the quantization only to leaf diamonds. This approach is much simpler, but tuned for lossless or RMS control.

4 Client-server framework

Our compressed representation forms the basis of a scalable terrain streaming system able to adapt to client characteristics and to exploit available network bandwidth. A block diagram of our system's design is presented in figure 4. In the following sections, we discuss the data storage and server side distribution component, as well as the client-side streaming and rendering methods.

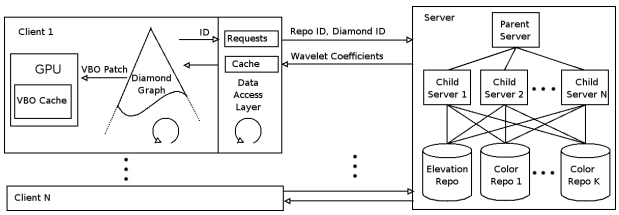


Figure 4: **Client-server architecture.**

4.1 Data storage and distribution

Diamond data repositories. Compressed data is stored in repositories that contain root diamond values and wavelet coefficient matrices for each diamond. We assume that each repository contains a single elevation or color value per sample. In a single client renderer, multiple repositories can be combined, provided that they meet consistency constraints. The repositories are required to cover the same domain but, in contrast to [Gobbetti et al. 2006] may use different diamond patch granularity. In particular, we found it profitable to use twice as many color samples than elevation samples per diamond side. This means that at rendering time we associate a texture diamond with $(2N+1)^2$ samples to each geometry diamond with $(N+1)^2$ samples.

Moreover, repositories can be, and generally will be, of different depths, as the original datasets are typically provided at different resolutions (generally, color has a higher sampling rate than elevation). Data can be requested from the repositories using a unique identifier which is the integer 3D center position of the selected diamond. For a planar terrain, we use a single root diamond with 2D coordinates, while for spherical datasets we use a cube map representation with one root diamond per cube face.

Server design and implementation. The server is able to manage different repositories, and does not differentiate among color or elevation components. From the server's point of view, a repository is just a database with a unique key for indexing a block of bytes containing an encoded bitstream representing a compressed wavelet coefficient matrix. In order to increase server-side scalability, no component is present in the server, whose only behavior is to return a block of bytes if present. This approach makes it possible to leverage existing database components instead of being forced to implement a storage manager. In this work, storage management is done through Berkeley DB, and data serving is done through an Apache2 server extended with an appropriate module.

Berkeley DB is a widely tested open source embeddable database which provides a fast, scalable, transactional database engine with industrial grade reliability and availability, able to manage up to terabytes of data. Moreover, the amount of per-process replicated cache is configurable, and different instances of the same database are able to share index memory, thus reducing memory load for servers dealing in parallel with hundreds of clients. Apache2 is a secure, efficient and extensible open source server that provides many HTTP services in sync with the current HTTP standards. Besides, it is scalable, multithreaded and includes features like persistent server processes and proxy load balancing, which are essential for the performance of our application.

A custom apache module manages a connectionless protocol based on HTTP. Clients requests include database name and diamond identifiers. The module parse the request, queries the associated database, and sends in response either a compressed bitstream extracted from the database and encoding a matrix of wavelet detail coefficients, or an empty message if no data is available (and thus the diamond is a leaf that has no refinement data associated to it).

This approach based on open-source components is simple to im-

plement and maintain, and provides very good performance. In particular, the concurrency features of apache are exploited to handle thousands of clients in parallel through the forking of multiple child servers sharing the same database.

4.2 Multi-threaded clients for remote visualization

Incremental diamond graph refinement. The client manages a diamond graph which represents the terrain and its relative texture from the coarser representation (given by the root) to the current view dependent representation, which is the current cut of the graph. Elevation and texture come from different repositories, that are matched at run-time. Both color and elevation are represented using the same technique.

For the sake of interactivity, the multiresolution extraction process should be able to support a constant high frame rate, given the available time and memory resources. This means that the algorithm must be able to fulfill its task within a predetermined budget of time and memory resources, always ending with a consistent result, or, in other words, it must be interruptible. Our extraction method uses a time-critical variation of the dual-queue refinement approach [Duchaineau et al. 1997] to maintain a current cut in the diamond DAG by means of refinement and coarsening operations. The graph with the reconstructed values of all the diamonds from the root until the current cut is kept incore.

Each diamond has an associated error equal to the maximum between the estimated mean projected edge length of its triangles and the mean projected size of the associated texels in case of colored datasets. In the typical case, more texels are associated to each triangles, as we typically diamonds with more samples for color than for elevation. This means that, at rendering time, the renderer will seek to produce images with a prescribed number of texels/pixel for colored datasets. Since rendering is very efficient, the typical rendering tolerance is in the range of 1 texel/pixel. This makes it possible to use a very simple error metric, stated purely in terms of sampling density, without the need to resort to a complex estimation of perceivable features.

A coarsen operation is performed on a diamond above the cut if its projected error is lower than a user selected screen threshold and all its children are leaves, while we perform a refine operation on a diamond of the cut when its error is larger than the threshold, and all its parents have been already refined. This means that coarsening is obtained just by moving up the cut in the diamond graph, and freeing unreferenced memorym and does not require particular computation. In the refinement operation, instead, children data must be constructed, starting from current color and elevation data and requesting detail coefficients from the server. If detail coefficients are not immediatly available, the refinement of that diamond stops and refinement of other diamonds in the queue is performed until the queue is empty or the elapsed time has exhausted the time budget. While waiting for data coming from the server, the renderer keeps using the current terrain approximation. Latency in data arrival can be reduced by using a prefetching approach (see below). When new data come from the server, the client takes care of decompressing the bitstream into a matrix of wavelet detail coefficients, then uses them to reconstruct the actual values to be stored in the multiresolution structure.

Multithreaded data access for fecthing and prefetching wavelet coefficients. A multithreaded data access layer hides to the application the technique used for accessing the terrain repository. In our current implementation, we use a HTTP/1.1 persistent connection approach and optionally employ HTTP pipelining. The combination of these two techniques improve bandwidth usage and reduce network latency, while keeping the protocol simple from API point-of-view, since clients benefit from an underlying connection-based implementation hidden under a reliable connectionless interface. The pipelining approach allows multiple HTTP requests to be written together out to the socket connecting client and server without waiting for the corresponding responses. The client then waits for the responses to arrive in the order in which they were requested. The act of pipelining the requests can result in a dramatic improvement in response times, especially over high latency connections. Pipelining can also dramatically reduce the number of TCP/IP packets, since it is possible to pack several HTTP requests into one TCP/IP packet, reducing the burden on IP routers and networks. This is particularly important in our setting, since diamond data is higly compressed (e.g., few tens of bytes per diamond), and multiple requests are typically issued at each frame. With a typical TCP/IP MSS (maximum segment size) in the range of 536 to 1460 bytes, a single packet can pack multiple requests to the server as well as multiple responses.

The client data access layer is subdivided into two threads that communicate through a shared cache of wavelet coefficients indexed by diamond ids to hide network latencies. The main thread requests the wavelet coefficients which are needed to refine the diamond graph from the current point of view. Requests are pushed in a prioritity queue. At the end of the frame, only as many new requests as those allowed by the estimated network bandwidth are issued and managed by a separate network access thread, and the remaining ones are ignored. Since the dual queue incremental refinement approach ensures that requests are issued sorted coarse to fine and by estimated projected error, and that unhandled requests are repeated at each frame, a simple limited memory first-in/fist-out queue induces a request ordering that is both I/O efficient and ensures to download the most relevant data as soon as possible. In addition, this method makes it possible to incorporate prefetching to request refinement data before it is needed. This is done by pushing to the request queue also a set of diamonds which are not scheduled for refinement, but are likely to be refined in the near future. Since these diamonds are at the end of the queue, they are considered only if all true refinement requests can be managed whithin the current fetching budget.

The second thread managing the network receives the compressed bitstream from the server and decodes it to wavelet coefficients before storing them in the shared cache. Moving all the decoding in this second thread reduces CPU work in the main one, which will receive directly wavelet coefficient matrices in a form usable for the wavelet synthesis operation required to update the diamond graph. If color and elevation data are not available at the same resolution it could happen that requests are issued for a diamond that is present only in one of the two repositories. In that case, this fact is hidden to the main diamond refinement thread, and the missing wavelet data are procedurally generated by the client to be able to refine the graph to the maximum depth of the color and elevation datasets. In our current implementation, this procedurally generated matrix can either be a null one, or a matrix of zero-centered uniform random numbers for high frequency detail synthesis.

When we request a diamond which is not present in either of the two datasets, the server returns two null bitstreams. This diamond is marked as empty, so that none of the successive requests of this data is sent to the server. A cache of identifiers of empty diamonds is stored in the data access layer; when the diamond graph requests an empty diamond nothing is returned, and thus the processed diamond is no further refined.

Rendering process At the end of the incremental refinement process, the leafs of the graph contain the current terrain approximation, in a format suitable for graphics rendering. Communication with the GPU is made exclusively through a retained mode interface, which reduces bus traffic by managing a least-recently-used cache of triangular patches and textures maintained on-board. Since we have to manage heterogeneous clients with varying graphics capabilities, in our implementation at the beginning of rendering we

detect which kind of GPU is available thus tuning the graphics approach that will be used to enable our application to work also on low-end GPUs or in the extreme case, in software only environments.

As stated above, in contrast to C-BDAM, we decouple geometry resolution from color resolution and put the color information into texture images, instead of using per-vertex color. It is thus possible to decouple color resolution from geometry resolution by using different diamond sizes.

On non programmable graphics systems, the computation of all the vertex information (position and eventually normal) starting from an array of heights and from the boundary corner of the patch, is performed by the CPU before sending data from the diamond graph to the GPU in the form of a Vertex Buffer Object. On high-end hardware this decompression of vertex information is performed directly by vertex shaders, allowing the application to move compressed data from the CPU to the GPU. A similar approach is taken for colors, for which the decompression from the YCoCg-R colorspace used for compression is performed either in CPU or on a fragment shader depending on the capabilities of the graphics board.

We manage a cache of Vertex Buffer Objects in the GPU to exploit spatial and temporal coherence, reusing the same data for several frames without the need of moving it again to the GPU. A backup solution is also used for the CPU - GPU data comunication if no Vertex Buffer Objects are available, and it is based on the standard Vertex Arrays which are provided since OpenGL 1.1.

5 Implementation and results

An experimental software library and a rendering application supporting the C-BDAM technique have been implemented on Linux and Windows platforms using C++ with OpenGL and NVIDIA Cg. An early version of this library has been incoporated into a widely distributed regional geo-viewing system with tens of thousands of clients[1]. As for most contemporary goeviewing systems, the client application enriches the digital terrain model with different kinds of static and dynamic geocoded data. A snapshot of the GUI is depicted in figure 6.

We have extensively tested our system with a number of large terrain datasets. In this paper, we discuss the results obtained on two terrain datasets with different characteristics. The first dataset is a global reconstruction of the planet Earth from SRTM data at 3 arcsec resolution (one point every $90m$ at the Equator. Source: CGIAR Consortium for Spatial Information. *srtm.csi.cgiar.org*). The dataset is reasonably large (29 billion samples) and provides an example of variable resolution planetary data, as it is described by a sparse set of tiles providing data only for emerged land in the 66S-66N latitude range. The total size of this dataset is 58GB uncompressed. The second dataset is a reconstruction of the island of Sardinia textured with a high resolution ortho-photo. Elevation data has a $10m/sample$ resolution (800MB uncompressed), while the texture has a $1m/sample$ resolution (123GB uncompressed).

Preprocessing and compression rates All the preprocessing tests were run on a single PC running Linux 2.6.15 with two Dual Core AMD Opteron 1.8 GHz processors, 2 GB of RAM, SATA 10000 rpm, 150 GB hard drive.

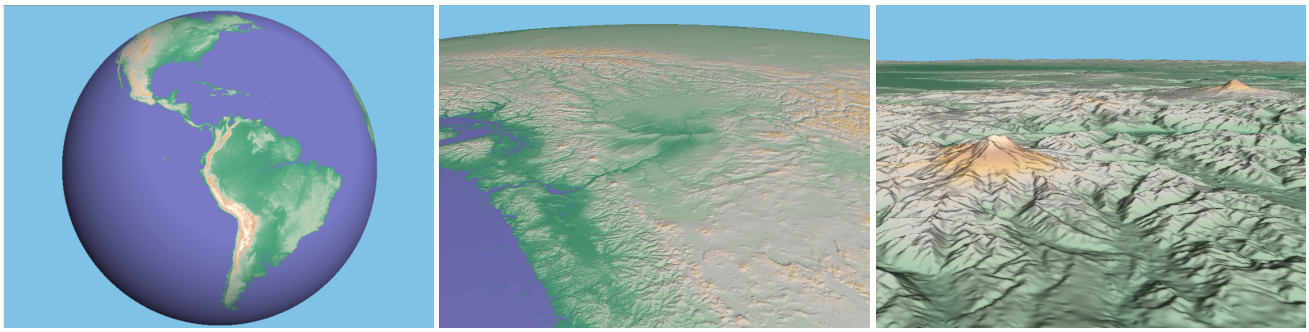
We constructed all geometry multiresolution structures with a prescribed diamond dimension of 33×33 samples for the elevation and 65×65 samples for color. Each diamond is thus composed by two patches formed by 1K triangles each and is covered by a texture with two texels per triangle edge. Preprocessing planet earth (29G input samples) took 9.5 hours, while preprocessing the Sardina dataset took 6 minutes for the elevation (400M input samples), and 30.5 hours for the colors (41G samples). In all the results presented here we make use of four threads (one per CPU core), with a speed-up of 3X with respect to the sequential version. The sub-linear speed-up is due to the fact that time is taken up with I/O operations on the input, temporary and output files. We expect a performance increase when distributing them on multiple disks. Processing speed ranges from 360K samples/s for color up to 850K samples/s for height, more than 3 time faster than C-BDAM [Gobbetti et al. 2006] This is mainly because we reduced the processing to one pass, we generate roughly half the temporary data, and we compute only one matrix per diamond instead of two matrice using integer only operations.

The tolerances used for compressing were chosen to provide near-lossless results, using root mean square error tolerance to drive the compression. In both the elevation datasets we used a prescribed maximum RMS error tolerances of 10% of sample spacing: 9 meters for Planet Earth and 1 meter for the Sardinia heightfield, while colors have been compressed imposing a RMS error of 11% per component which gives an error lower than what is achieved for the S3TC compression algorithm, commonly used in terrain rendering. The achieved bit rate for planet Earth is $0.13bps$ (bits per sample) (or $0.296bps$ considering the Berkeley DB overhead). Sardinia has a bit rate of $0.23bps$ for elevation ($0.68bps$ with Berkeley DB overhead), and $2.67bps$ ($3.743bps$) for color. The color bit rate is higher because elevation data is much smoother. It should be noted that we imposed a strict tolerance in preprocessing to conserve all extremely sharp features and to test the near lossless behavior of the compressor. In any case, the compression is similar to that of other wavelet based image compressors, with the additional advantage of ability to produce seamless variable resolution representation at neglible costs. By using larger kernels for prediction and update operators and using a more elaborate entropy encoder it would be possible to further improve compression rates, at the expense, however, of decoding speed and code complexity. The figures above show that the Berkely DB database has a noticeable overhead due to its generality, e.g., for supporting variable size keys, and data paging organization. It should be considered, however, that for a networked application this is not an issue, since the stream bit rate is independent of storage overhead.

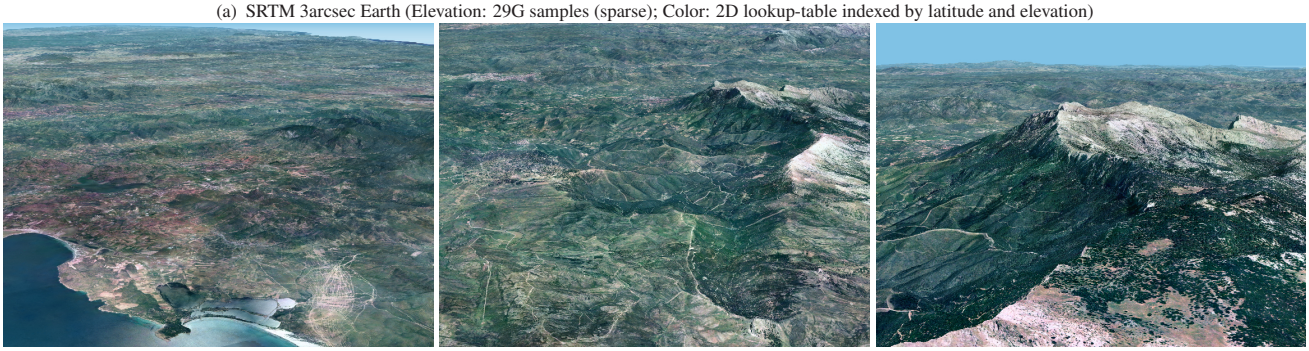
An important consideration here comes from a comparison with current tile-based technology. Current tiling systems apply compression at the tile level. To achieve high compression rates, they are forced to use tile sized much larger than our diamond sizes. This is because, otherwise, not enough context in a single tile would be present to offer compression possibilities. For instance, Microsoft Virtual Earth has a compression rate of about $3bps$ for the color dataset using a tile size of $256x256$ and a lossy JPEG encoding. Using such a large tile size (over 16x the number of samples in one of our diamonds) forces the application to refine the model at a much larger granularity. This implies that more data is required to achieve the same approximation quality, and that refinement is not as smooth as in our solution. Reducing the tile size is not an option, since, for instance, the same JPEG compression applied to tiles of $64x64$ samples produces a bit rate of $4.5bps$. Note also that our representation is critically sampled and has twice the refinement levels, while a standard tile-based replicates data among levels and offers only power of two refinement.

Adaptive rendering. The rendering tests were run on low- and medium-end PCs with NVIDIA, ATI or Intel graphics boards and CPUs ranging from AMD64 to Pentium3 800MHz. In all cases we are able to sustain interactive rendering rates ranging from 20Hz to over 100Hz with pixel-sized texels datasets or triangles when no color layer is displayed. The qualitative performance of our adaptive renderer is illustrated in an accompanying video that shows live recordings of flythrough sequences. The video was recorded at 640x480 window size due to recording equipment constraints.

[1] Viewer available from: `http://www.sardegna3d.it` - at the time of this writing, over 40'000 users are served by two AMD64 servers.



(a) SRTM 3arcsec Earth (Elevation: 29G samples (sparse); Color: 2D lookup-table indexed by latitude and elevation)



(b) Sardinia 10m/1m (Elevation: 10m resolution; Color: 1m resolution)

Figure 5: **Inspection sequences: selected frames.** All images were recorded live on a AMD 1.8 GHz PC with 2 GB RAM and PCI Xpress NVIDIA Geforce 7800GT graphics using a tolerance of 1triangle/pixel for the shaded model and 1texel/pixel for the phototextured model. See also figure 8 (color plate).

Network streaming. Extensive network tests have been performed on all test models, on an ADSL 4Mbit/s connection as well as on an intranet. Tests have been made for the viewer application under interactive control, as well as using synthetic benchmarks to push the streaming layer to the limit.

The combination of compression with HTTP pipelining has proven particularly effective. Even in the less favorable situation of low-latency high speed network connection, this combination leads to a speed-up of 3x over a non-pipelining approach. Synthetic benchmark designed to push the data access layer to the limit have shown that a single client is able to pull from the server over 12.8$Mbps$, versus 4.5$Mbps$ for a non-pipelining approach. It is interesting to note that the performance improvement is also due to the improved packing of data into TCP/IP packets. As a result of this better packing, the pipelining version trasmits on average over 30% less TCP/IP packets than the non-pipelining one, noticeably reducing burden on IP routers and networks. Speed-ups for the pipelining version are noticeably better for long distance communications, where network latency is also reduced by the fact that multiple HTTP requests are sent to the server without waiting for the corresponding responses.

In an interactive setting, the network usage of a single client is obviously lower than the peak achievable performance cited above. Since rendering is progressive and, on average, viewpoint motion is smooth, only few diamonds per frame need to be refined, and only data for diamonds not already cached are requested to the server. We have measured the bandwidth required by a client to provide a "no delay" experience in low altitude flights at typical interactive speeds over the Sardinia dataset. We measured an average bit rate of 1600$Kbps$ for exploration of areas not previously seen, and peaks of 2400$Kbps$ at viewpoint discontinuities, i.e., when the application has to refine the model all the way to the new viewpoint and the refinement algorithm has to always push new diamonds to refine in the request queue because of non incremental update. The above average bit rate corresponds to refining over 133 diamonds per second, i.e., over 5 diamonds per frame at a 25Hz refresh rate, which means that within 2s the entire screen can be refined.

By introducing client-side or server-side bandwidth limitations, it is possible to reduce burden on network and server, making the system more scalable while maintaining a good interactive quality. Due to the high compression rate and refinement efficiency, we have found that limiting the speed to 300$Kbps$ produces nice interactive results. In that case, delays in case of rapid motion become visible, but with little detriment to interaction (e.g., only 10s are needed to produce a full quality full screen image from scratch).

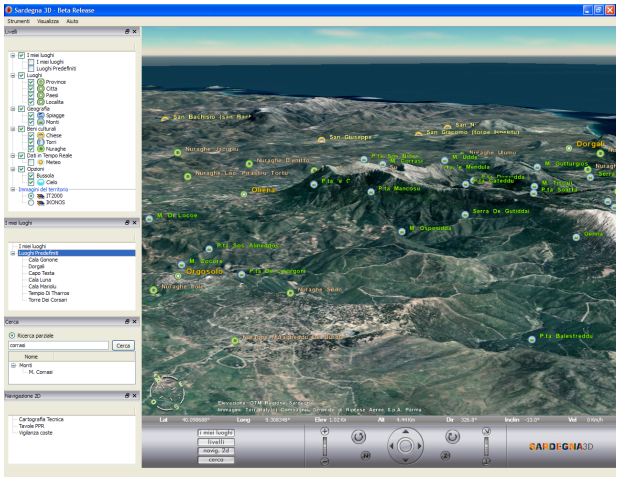


Figure 6: **Snapshot of GUI of a regional geoviewing client.** Thanks to the efficiency of our approach, two AMD64 servers are able to manage a community of over 40'000 users.

6 Conclusions

We have described a compressed multiresolution representation and a client-server architecure for supporting interactive high quality remote visualization of very large textured planar and spherical terrains. Our approach incrementally updates a chunked level-of-detail BDAM hierarchy by using precomputed wavelet coefficient matrices decoded from a compressed bitstream originating from a thin server. As illustrated by our experimental results, the structure provides a number of practical benefits, as it combines the aggressive compression rates of wavelet-based image representations with the ability to ensure overall geometric continuity for variable resolution views of planar and spherical terrains with no need for run-time stitching.

This work shows that it is possible to combine the generality and adaptivity of batched dynamic adaptive meshes with the compression rates of wavelet image encoders and the network friendliness of tiled block techniques. Our current work is concentrating in improving the capability to preprocess very large datasets using cluster-parallel techniques.

Acknowledgments. This research is partially supported by the CYBERSAR and CSR-VIC-TERRAIN3D projects. The authors wish to thank the Autonomous Region of Sardinia Authorities for their support, CORE Soluzione Informatiche for system integration, and CGR for providing orthophoto datasets.

References

ALLIEZ, P., AND GOTSMAN, C. 2005. Recent advances in compression of 3d meshes. In *Advances in Multiresolution for Geometric Modelling*, Springer, M. S. N.A. Dodgson, M.S. Floater, Ed.

CIGNONI, P., GANOVELLI, F., GOBBETTI, E., F.MARTON, PONCHIO, F., AND SCOPIGNO, R. 2003. BDAM: Batched dynamic adaptive meshes for high performance terrain visualization. *Computer Graphics Forum 22*, 3 (Sept.), 505–514.

CIGNONI, P., GANOVELLI, F., GOBBETTI, E., MARTON, F., PONCHIO, F., AND SCOPIGNO, R. 2003. Planet-sized batched dynamic adaptive meshes (P-BDAM). In *IEEE Visualization*, 147–154.

CLASEN, M., AND HEGE, H.-C. 2006. Terrain rendering using spherical clipmaps. In *Proc. EuroVis*, 91–98.

DE FLORIANI, L., MAGILLO, P., AND PUPPO, E. 1997. Building and traversing a surface at variable resolution. In *Proceedings IEEE Visualization 97*, 103–110.

DEB, S., AND NARAYANAN, P. 2006. Streaming terrain rendering. In *Proc. SIGGRAPH 2006 Sketches*.

DECORO, C., AND PAJAROLA, R. 2002. Xfastmesh: fast view-dependent meshing from external memory. In *VIS '02: Proceedings of the conference on Visualization '02*, IEEE Computer Society, Washington, DC, USA, 363–370.

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.

GOBBETTI, E., MARTON, F., CIGNONI, P., BENEDETTO, M. D., AND GANOVELLI, F. 2006. C-BDAM – compressed batched dynamic adaptive meshes for terrain rendering. *Computer Graphics Forum 25*, 3 (September). Proc. Eurographics 2006.

HITCHNER, L. E., AND MCGREEVY, M. W. 1993. Methods for user-based reduction of model complexity for virtual planetary exploration. In *Proc SPIE*, vol. 1913, 622–636.

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

HWA, L. M., DUCHAINEAU, M. A., AND JOY, K. I. 2005. Real-time optimal adaptation for planetary geometry and texture: 4-8 tile hierarchies. *IEEE Transactions on Visualization and Computer Graphics 11*, 4, 355–368.

JR., R. L. C., DAVIS, G. M., SWELDENS, W., AND BARANIUK, R. G. 2003. Nonlinear wavelet transforms for image coding via lifting. *IEEE Transactions on Image Processing 12*, 12, 1449–1459.

KIM, J. K., AND RA, J. B. 2004. A real-time terrain visualization algorithm using wavelet-based compression. *The Visual Computer*, 20, 67–85.

KOVACEVIC, J., AND SWELDENS, W. 2000. Wavelet families of increasing order in arbitrary dimensions. *IEEE Transactions on Image Processing 9*, 3, 480–496.

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. 2001. Visualization of large terrains made easy. In *Proc. IEEE Visualization 2001*, IEEE Press, 363–370, 574.

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.

LOSASSO, F., AND HOPPE, H. 2004. Geometry clipmaps: terrain rendering using nested regular grids. *ACM Trans. Graph 23*, 3, 769–776.

MALVAR, H. S. 2000. Fast progressive image coding without wavelets. In *Data Compression Conference*, 243–252.

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.

WAHL, R., MASSING, M., DEGENER, P., GUTHE, M., AND KLEIN, R. 2004. Scalable compression and rendering of textured terrain data. In *Journal of WSCG*, UNION Agency/Science Press, Plzen, Czech Republic, vol. 12.

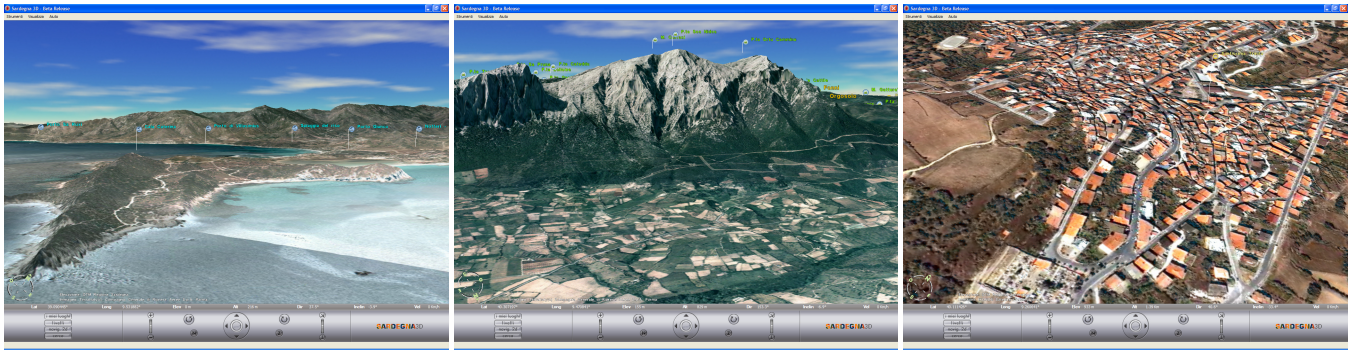
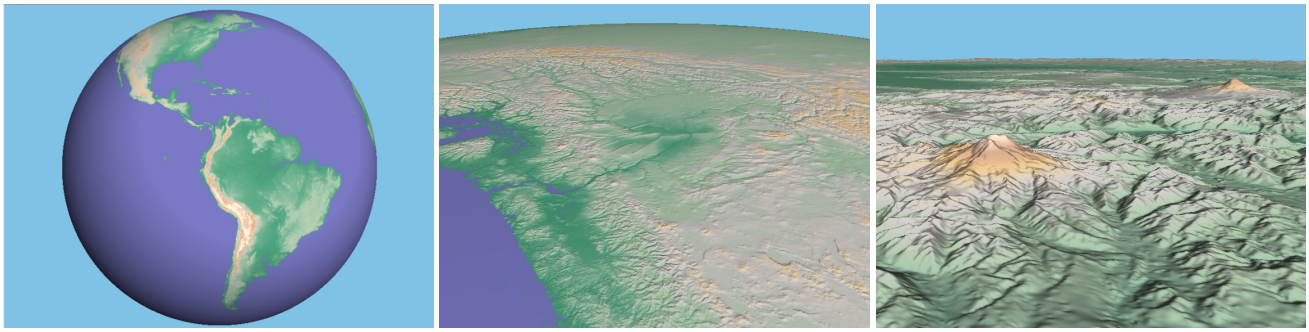
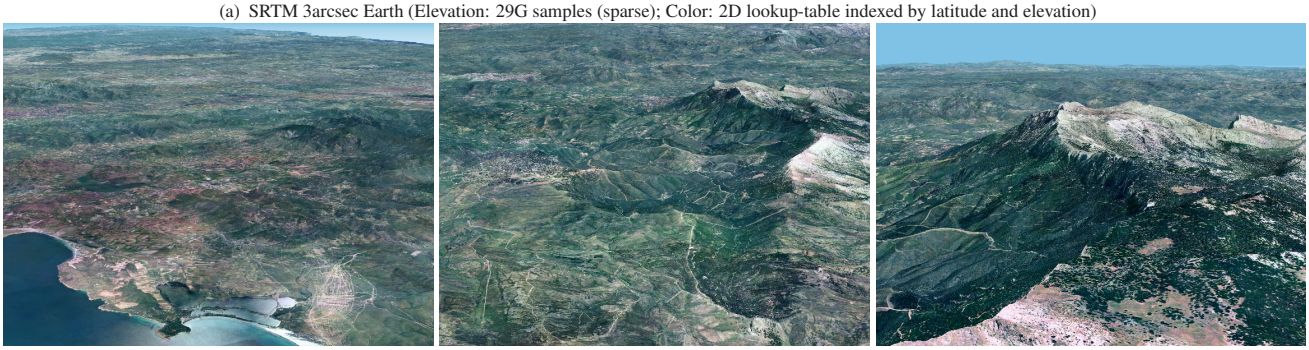


Figure 7: **Real-time exploration of remote terrain models.** Our method is capable of producing high quality seamless renderings of terrain data at high frame rates from highly compressed bitstreams, improving the scalability of servers and the behavior of clients in network environments with narrow bandwidth and low computational power. In this example, we show three frames of an interactive session with an internet geo-viewing tool based on our library exploring a detailed height colored terrain model at high frame rates over an ADSL 4Mbps network.



(a) SRTM 3arcsec Earth (Elevation: 29G samples (sparse); Color: 2D lookup-table indexed by latitude and elevation)



(b) Sardinia 10m/1m (Elevation: 10m resolution; Color: 1m resolution)

Figure 8: **Inspection sequences: selected frames.** All images were recorded live on a AMD 1.8 GHz PC with 2 GB RAM and PCI Xpress NVIDIA Geforce 7800GT graphics using a tolerance of 1triangle/pixel for the shaded model and 1texel/pixel for the phototextured model.