EUROGRAPHICS 2006 / E. Gröller and L. Szirmay-Kalos
(Guest Editors)

Volume 25 (2006), Number 3

C-BDAM – Compressed Batched Dynamic Adaptive Meshes for Terrain Rendering

E. Gobbetti[1], F. Marton[1], P. Cignoni[2], M. Di Benedetto[2], and F. Ganovelli[2]

[1] Visual Computing Group, CRS4, Pula, Italy
[2] Visual Computing Group, ISTI CNR, Pisa, Italy

Abstract

We describe a compressed multiresolution representation for supporting interactive rendering of very large planar and spherical terrain surfaces. The technique, called Compressed Batched Dynamic Adaptive Meshes (C-BDAM), is an extension of the BDAM and P-BDAM chunked level-of-detail hierarchy. In the C-BDAM approach, all patches share the same regular triangulation connectivity and incrementally encode their vertex attributes using a quantized representation of the difference with respect to values predicted from the coarser level. The structure provides a number of benefits: simplicity of data structures, overall geometric continuity for planar and spherical domains, support for variable resolution input data, management of multiple vertex attributes, efficient compression and fast construction times, ability to support maximum-error metrics, real-time decompression and shaded rendering with configurable variable level-of-detail extraction, and runtime detail synthesis. The efficiency of the approach and the achieved compression rates are demonstrated on a number of test cases, including the interactive visualization of a 29 gigasample reconstruction of the whole planet Earth created from high resolution SRTM data.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture and Image Generation; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism.

1. Introduction

Real-time 3D exploration of digital elevation models is one of the most important components in a number of practical applications. Nowadays, high accuracy datasets contain billions of samples, exceeding memory size and graphics processing capability of even the highest-end graphics platforms. To cope with this problem, there has been extensive research on output sensitive algorithms for terrain rendering (see section 2). At present time, the most successful techniques for efficiently processing and rendering very large datasets are based on two approaches: adaptive coarse grained refinement from out-of-core multiresolution data structures (e.g., BDAM [CGG*03a], P-BDAM [CGG*03b], 4-8 tiling [HDJ05]) and in-core rendering from aggressively compressed pyramidal structures (e.g., Geometry Clipmaps [LH04]). The first set of methods is very efficient in approximating a planar or spherical terrain with the required accuracy and in incrementally communicating updates to the GPU as the viewer moves, but multiresolution structure footprints typically require out-of-core data management. The second approach, limited to planar domains, uses nested regular grids centered about the viewer. This technique ignores local adaptivity, but is able to exploit structure regularity to compress data so that it typically succeeds in fitting most if not all data structures entirely in core memory, thereby avoiding the complexity of out-of-core memory management for a large class of practical models.

Contribution. In this paper, we describe a compressed multiresolution representation for the management and interactive rendering of very large planar and spherical terrain surfaces. The technique, called Compressed Batched Dynamic Adaptive Meshes (*C-BDAM*), is an extension of the BDAM and P-BDAM chunked level-of-detail hierarchy, and strives to combine the generality and adaptivity of chunked bintree multiresolution structures with the compression rates of nested regular grid techniques. Similarly to BDAM, coarse grain refinement operations are associated to regions in a bintree hierarchy. Each region, called diamond, is formed by two triangular patches that share their longest edge. In BDAM, each patch is a general precomputed triangulated surface region. In the C-BDAM approach, however, all patches share the same regular triangulation connectivity and incrementally encode their vertex attributes when descending in the multiresolution hierarchy. The encoding follows a two-stage wavelet based near-lossless scheme. The

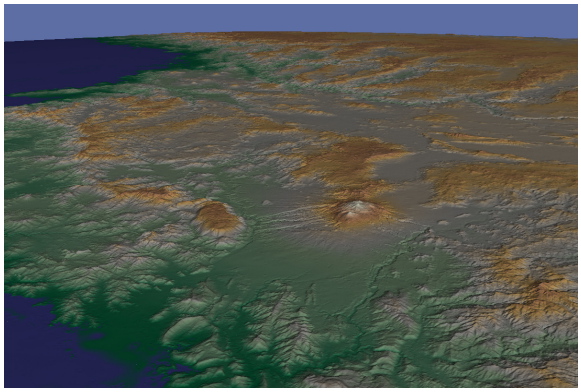


Figure 1: **View of the Earth near Guadalajara.** This 29G samples sparse global dataset is compressed to 0.25bps. At run-time, decompression and normal computation are performed incrementally, allowing interactive full-screen flights at video rates with pixel sized triangles.

proposed approach supports both mean-square error and maximum error metrics allowing to introduce a strict bound on the maximum error introduced in the visualization process. The scheme requires storage of two small square matrices of residuals per diamond, which are maintained in a repository. At run-time, a compact in-core multiresolution structure is traversed, and incrementally refined or coarsened on a diamond-by-diamond basis until screen space error criteria are met. The data required for refining is either retrieved from the repository or procedurally generated to support run-time detail synthesis. At each frame, updates are communicated to the GPU with a batched communication model.

Advantages. The structure provides a number of benefits: overall geometric continuity for planar and spherical domains, support for variable resolution input data, management of multiple vertex attributes, efficient compression and fast construction times, ability to support maximum-error metrics, real-time decompression and shaded rendering with configurable variable level-of-detail extraction, and runtime detail synthesis. 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.

Limitations. The proposed method has also some limitations. As for geometry clipmaps [LH04], the compression method is lossy and assumes that the terrain has bounded spectral density, which is the case for typical remote sensing datasets. Moreover, as for all regular grid approaches, the rendered mesh has a higher triangle count than in the BDAM and P-BDAM schemes, which can exploit irregular connectivity to follow terrain features.

Our approach adopts the philosophy of the BDAM breed of techniques, that we summarize in section 3. The new approach for efficiently encoding planet sized datasets is described in section 4. Section 5 illustrates the pre-processing of digital elevation models, while section 6 is devoted to the presentation of the run-time rendering algorithm. The efficiency of the method has been successfully evaluated on a number of test cases, including the interactive visualization of the Earth created from 3 arcsec SRTM data (section 7).

2. Related Work

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 briefly discuss the approaches that are most closely related with our work. Readers may refer to well established surveys [LP02, Paj02] for further details.

The vast majority of the 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., [DFMP97, Hop98]) or semi-regular adaptive triangulations (e.g., [LKR*96, DWS*97, Paj98, LP01]). 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 was able to sustain the rendering. More recently, the impressive improvement of the graphics hardware both in term of computation (transformations per second, fill rate) and communication speed (since the introduction of AGP1x to the PCI-E) shifted the bottleneck of the process from the GPU to the CPU. In other words, the approaches which select the proper set of triangles to be rendered in the CPU did not have a sufficient throughput to feed the GPU at the top of its capability. For this reason many techniques proposed to reduce per-triangle workload by composing at run-time pre-assembled optimized surface patches, making it possible to employ the *retained-mode* rendering model instead of the less efficient direct rendering approach. Tiled blocks techniques (e.g., [HM93, WMD*04]), originally designed for external data management purposes, partition the terrain into square patches tessellated at different resolutions. The main challenge is to seamlessly stitch block boundaries. The first methods capable to producing adaptive conforming surfaces by composing precomputed patches with a low CPU cost, were explicitly designed for terrain rendering. RUSTIC [Pom00] and CABTT [Lev02] are extensions of the ROAM [DWS*97] algorithm, in which subtrees of the ROAM bintree are cached and reused during rendering. A similar technique is also presented in [DP02] for generic meshes. BDAM [CGG*03a, CGG*03b] 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 [HDJ05], which store textures and geometry using the same technique.

Recently various authors have concentrated on combining data compression methods with multiresolution schemes

to reduce data transfer bandwidths and memory footprints. Tiled block techniques typically use standard 2D compressors to independently compress each tile. Geometry clipmaps [LH04] organize the terrain height data in a pyramidal multiresolution scheme and the residual between levels are compressed using an advanced image coder that supports fast access to image regions. 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 that can be maintained in main memory even for large datasets. The pyramidal scheme limits however adaptivity. In particular, as with texture clipmap-based methods, the technique works best for wide field of views and nearly planar geometry, and would not apply to planetary reconstructions that would require more than one nesting neighborhood for a given perspective. In [HDJ05], 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 in the BDAM framework.

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 on the real-time view-dependent rendering from the compressed representation; for a recent survey of this field we refer the reader to [AG05].

3. Batched Dynamic Adaptive Meshes

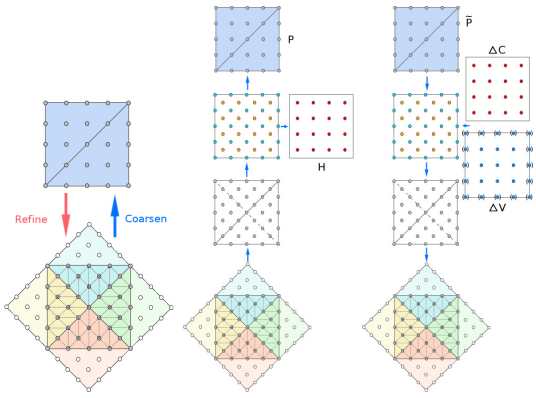
The BDAM approach is a specialization of the more general Batched Multi-Triangulation framework [CGG*05], and is based on the idea of moving the grain of the multiresolution surface model up from points or triangles to small contiguous portions of mesh.

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. 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, it is possible to extract conforming variable resolution mesh representations. BDAM exploits this property to construct a coarse grained level-of-detail structure for the surface of the input model. This is done by associating to each triangle region a small tessellated patch, up to a given triangle count, of the portion of the mesh contained in it. Each patch is constructed so that vertices along the longest edge of the region are kept fixed during a diamond coarsening operation (or, equivalently, so that vertices along the shortest edge are kept fixed when refining). In this way, it is ensured that each collection of small patches arranged as a correct hierarchy of triangular regions generates a globally correct and conforming surface triangulation. These properties are exploited in [CGG*03b] to define an efficient parallel simplification method in which patches composing diamonds at levels of resolution matching the input data are sampled, while coarser level patches contain TINs constructed by constrained edge-collapse simplification of child patches.

4. Compressing Batched Dynamic Adaptive Meshes

In order to construct a BDAM hierarchy, three operations are required: original data sampling, diamond coarsening (simplification), and its reciprocal, diamond refinement. At diamond coarsening time, data is gathered from child patches and simplified while keeping the diamond boundary fixed. Diamond refinement has to undo the simplification operation, pushing new patches into child diamonds. As noted elsewhere [LH04, HDJ04, CGG*05], given current hardware rendering rates, exceeding the hundreds of millions of triangles per second, it is now possible to interactively render scenes with approximately one triangle/pixel thresholds. At this point, controlling triangle shapes during simplification to reduce triangle counts is no longer of primary importance, and it is possible to work on regular grids rather than on irregular triangular networks, replacing mesh simplification/refinement with digital signal processing operations.



(a) Coarsening and refinement (b) Wavelet view (c) Near lossless scheme

Figure 2: **BDAM on a regular grid.** We recast diamond processing in the framework of wavelet update lifting In this framework, diamond coarsening and refinement are associated to wavelet analysis and synthesis.

Wavelet transformation. The structure of a BDAM mesh, when operating on regular grids, is depicted in figure 2(a).

As we can see, the two patches that form a diamond have 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. To support compression, we recast diamond processing in the framework of wavelet update lifting [JDSB03], with the purpose of transforming data into a domain with a sparser representation. In this framework, diamond coarsening is associated to wavelet analysis, while diamond refinement corresponds to wavelet synthesis (see figure 2(b)).

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 a square matrix $P^{(l+1)}$, which is then decomposed into two square matrices: 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 $P^{(l)}$ and a matrix of detail coefficients $H^{(l)}$ can then be computed by the following low- and high-pass filtering operations:

$$P_{ij}^{(l)} = \alpha_{ij} V_{ij}^{(l+1)} + (1-\alpha_{ij}) \mathcal{P}_{ij}^{(V)}(C^{(l+1)}) \quad (1)$$

$$H_{ij}^{(l)} = C_{ij}^{(l+1)} - \mathcal{P}_{ij}^{(C)}(P^{(l)}) \quad (2)$$

Here, $\mathcal{P}_{ij}^{(A)}(B)$ are prediction operators that predict the value at $A(i,j)$ from the values in matrix B, and $0 < \alpha_{ij} \leq 1$ weights between smoothing and pure subsampling. To comply with BDAM diamond boundary constraints, it is sufficient to set $\alpha_{ij} = 1$ for all diamond boundary vertices. Even though weighting could be data dependent (see, e.g., [PH01]), for this paper we assume for speed and simplicity reason a constant weighting of $\alpha_{ij} = \frac{1}{2}$ for all inner points. For prediction, we use a order 4 Neville interpolating filter if all support points fall inside the diamond, and a order 2 filter otherwise [KS00]. These filters predict points by a weighted sum of 12 (order 4) or 4 (order 2) coefficients, and are therefore very fast to compute (see figure 3).

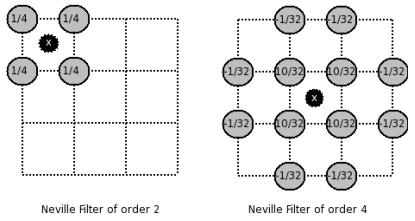


Figure 3: **Neville filters.** The filters predict points by a weighted sum of 12 (order 4) or 4 (order 2) coefficients.

By iterating the analysis process from leaf diamonds up to the roots, all the input data gets 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 vertex and face centered values are computed from the diamond vertex values:

$$C_{ij}^{(l+1)} = H_{ij}^{(l)} + \mathcal{P}_{ij}^{(C)}(P^{(l)}) \quad (3)$$

$$V_{ij}^{(l+1)} = \frac{P_{ij}^{(l)} - (1-\alpha_{ij}) \mathcal{P}_{ij}^{(V)}(C^{(l+1)})}{\alpha_{ij}} \quad (4)$$

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

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. Quantizing the wavelet coefficients to meet a maximum error criterion is however a complex problem, and while there are many excellent wavelet-based lossy compression schemes under the L^2 norm, none of them can offer a tight bound on the maximum reconstruction error of each value, since it is difficult to derive meaningful relations between distortions in the wavelet domains and in the signal domain in the L^∞ sense [YP04]. Using a L^2 norm for general terrain management applications is hardly acceptable, since reconstruction errors are averaged over the entire domain and results suffer from high variance in the quality of data approximation. Therefore, using a L^2 norm can severely bias reconstructions in favor of smooth regions, and does not offer error guarantees for individual approximate answers to variable resolution queries. In C-BDAM, we solve the problem using a simple two-stage near-lossless scheme that ensures that each diamond is within a given maximum value from the original filtered data. In contrast to other two stage schemes from the signal processing literature [YP04], we thus keep under strict control each level's error, and not only the finest level reconstruction, in order to avoid view-dependent rendering artifacts.

In our two-stage scheme, we first compute a full wavelet representation fine to coarse. We then produce a quantized compressed representation in a diamond-by-diamond coarse to fine pass that, at each step corrects the data reconstructed from quantized values by adding a quantized residual. For a diamond at level l, the residuals required for refinement are thus computed as follows:

$$\Delta\tilde{C}_{ij}^{(l+1)} = Quant_\varepsilon(\mathcal{P}_{ij}^{(C)}(P^{(l)}) + H_{ij}^{(l)} - \mathcal{P}_{ij}^{(C)}(\tilde{P}^{(l)})) \quad (5)$$

$$\tilde{C}_{ij}^{(l+1)} = \mathcal{P}_{ij}^{(C)}(\tilde{P}^{(l)}) + \Delta\tilde{C}_{ij}^{(l+1)} \quad (6)$$

$$\Delta\tilde{V}_{ij}^{(l+1)} = Quant_\varepsilon\left(\frac{P_{ij}^{(l)} - \tilde{P}_{ij}^{(l)} - (1-\alpha_{ij})(\mathcal{P}_{ij}^{(V)}(C^{(l+1)}) - \mathcal{P}_{ij}^{(V)}(\tilde{C}^{(l+1)}))}{\alpha_{ij}}\right) \quad (7)$$

$$\tilde{V}_{ij}^{(l+1)} = \frac{\tilde{P}_{ij}^{(l)} - (1-\alpha_{ij})(\mathcal{P}_{ij}^{(V)}(\tilde{C}^{(l+1)}))}{\alpha_{ij}} + \Delta\tilde{V}_{ij}^{(l+1)} \quad (8)$$

where $Quant_\varepsilon(x)$ is a uniform quantizer with step size 2ε. This scheme requires storage of two small square matrices of residuals per diamond, which are maintained in a data repository indexed by diamond id. For a diamond of N^2 vertices, matrix $\Delta\tilde{C}$ is of size $(N-1)^2$, while matrix $\Delta\tilde{V}$ is of size $(N-2)^2$, since, due to BDAM diamond graph constraints,

all boundary values are kept constant (we use a point sampling filter), and corrections are therefore implicitly null.

5. Out-of-core Construction of Massive Variable Resolution Terrain Datasets

The off-line component of our method constructs a multiresolution structure starting from a collection of input terrain datasets, a required diamond size, and a required maximum error tolerance.

Diamond graph construction. The first construction phase – graph construction – generates a diamond DAG, whose roots partition the input domain and whose leaves sample the dataset at a finer or equal density to that of the input dataset. The graph is constructed coarse to fine, by refining the diamond graph a diamond at a time until sample spacing is sufficiently fine. This process makes it possible to efficiently process variable resolution terrain datasets (see figure 4). In our current implementation, the input dataset is described by a hierarchy of layers, ordered by resolution, where each layer contains a number of tiles. For graph construction, each diamond provides the input dataset with the range of coordinates it covers, and receives as a result the sample spacing of the finest resolution tile contained in it. If this number is smaller than the diamond's sample spacing, the diamond is refined.

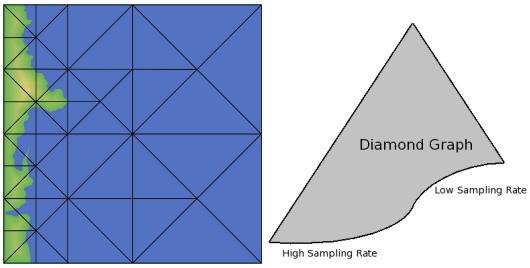


Figure 4: **Variable resolution input.** Diamond graph roots partition the input domain, while leaves leaves sample the dataset at a finer or equal density to that of the input dataset. In this simple example of a sparse input dataset covering emerged land, leafs are denser where data exists (terrain area) and rapidly decrease density where there is no data (blue area).

Fine to coarse filtering. Once the graph is constructed, it is traversed fine to coarse to construct a filtered representation by wavelet analysis (equations 1 and 2). For each filtered diamond, we store in a out-of-core repository both the filtered data $P^{(l)}$ and the detail coefficients $H^{(l)}$. Storing both matrices per diamond allows us to locally have all the information required to compute the reference values in the compression step. When sampling variable resolution input datasets at leaf diamonds, a value is always returned from the higher resolution tile that contains the query coordinate. A null value (typically 0) is returned when data is missing. An alternative approach would be to define transition areas and return weighted averages in case of overlapping tiles.

Fine to coarse compression. Once the filtered data repository is created, the root diamond values are copied to the output repository, and the diamond graph is then traversed coarse to fine to generate the final compressed representation of residuals. At each visited diamond, the reconstructed values of its parent diamonds are retrieved from a auxiliary repository updated while compressing, while the filtered values are retrieved from the repository generated in the first pass. Equations 5 to 8 are used to determine from this data the reconstructed vertex values for the diamond, as well as the residual matrices required to generate children. The relevant temporary and output repository are then updated, and the process continues until all diamonds are visited.

Boundary management. For non-global datasets, some of the diamonds fall on a boundary and are therefore incomplete. This would lead, in general, to deal, during both filtering and compression, not only with square coefficients matrices, but also with triangular ones. To simplify processing, we choose instead the approach of synthetically generating missing data by a symmetric extension (mirroring along the boundary). This data is only virtual, and will never be used for rendering, since no patches will be generated for fully out-of-bounds children.

Compressed data representation. The compression process generates a set of square matrices of integer coefficients that have to be compactly stored in the repository. These matrices are expected to be sparse and mostly composed of very small numbers. A number of efficient entropy coding methods for such matrices have been presented in the image compression literature. For our work, we have adopted a very simple storage scheme that, even though it is far from being optimal, it has the advantage of being straightforward to implement and extremely fast to decompress. In this scheme, the matrix is interpreted as a quadtree and recursively traversed in depth first order until a sub-matrix contains only zeros or is smaller than 2×2 in size. A single bit per split is used to encode the quadtree structure. At non-empty leaves, the coefficients are mapped to positive integers and encoded using a simple Elias gamma code [Eli75], in which a positive integer x is represented by: $1 + \lfloor \log_2 x \rfloor$ in unary (that is, $\lfloor \log_2 x \rfloor$ 0-bits followed by a 1-bit), followed by the binary representation of x without its most significant bit.

Out-of-core parallel construction. The whole construction process is inherently parallel, because the grain of the individual diamond processing tasks is very fine and synchronization is required only at the completion of each level. In this solution, the main memory required for each worker is that required for processing a single diamond, while the coordinator simply needs to reserve enough memory for holding out-of-sync requests in addition to the memory required to store the structure of the diamond graph, which is orders of magnitude smaller than the input data, since each diamond represents thousands of samples.

Managing multiple vertex attributes. Quite often, other vertex attributes need to be mapped over geometry. In our framework, we interpret these attributes as separate layers, and assume that different repositories are created independently for each of the attributes. For the moment, we just support elevation, represented as a scalar value, and color, represented in input as a RGB triple. For color compression, we map colors to the YCoCg-R color space [MS03] to reduce correlation among components, and then compress each component separately. Normals are neither stored in the repository nor passed to the GPU, but directly computed on demand by finite differences from elevation data.

6. Rendering a C-BDAM

Our adaptive rendering algorithm works on a standard PC, and preprocessed data is assumed to be either locally stored (in memory or on a secondary storage unit directly visible to the rendering engine) or remotely stored on a network server (see figure 5).

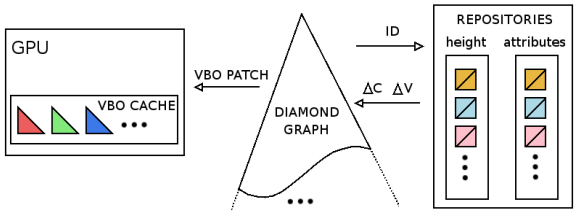


Figure 5: **The rendering pipeline.** The renderer accesses compressed data through a data access layer that hides whether data is local or remote.

Data layout and data access. The result of pre-processing is stored in repositories that contain root diamond values and residual matrices for each diamond. We assume that a main repository contains elevation values, while an optional secondary repository contain color values. The repositories are required to cover the same domain and to use the same diamond patch granularity but can be, and generally will be, of different depth, as the original data is typically provided at different resolutions. Access to a data repository is made through a data access layer, that hides from the renderer whether data is local or remote. This layer internally uses memory mapping primitives for local data, and a TCP/IP protocol for remote data. It makes it possible to asynchronously move in-core a diamond's associated data by fetching it from the repository, to test whether a diamond's data is immediately available, and to move available data to RAM.

Refinement algorithm. 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 [DWS*97] to maintain a current cut in the diamond DAG by means of refinement and coarsening operations. For this purpose we store the set of operations that are feasible given the available budget and compatible with the current cut, in two heaps: the *CoarseningHeap* and the *RefinementHeap*, with a priority dictated by the screen space error. The heaps are reinitialized at each frame, given the current cut and the current view parameters. When initializing the refinement heap, only operations for which data is immediately available are considered. In case of missing data, an asynchronous request is posted to the data access layer, which will be responsible of fetching the needed patches without blocking the extraction thread. The renderer maintains in-core all nodes of the diamond graph above the current cut, in a streamlined format suitable for communicating with the GPU and for quickly evaluating visibility and screen space error.

For each node, we maintain an object space error, the diamond corner information, links to parents and children, an oriented bounding rectangle, and one set of fixed size vertex arrays for each of its two patches. Each time a refinement or coarsening operation is performed, the current cut is updated. For coarsening, this simply amounts to freeing the unreferenced memory, while for refinement children data must be constructed, starting from current vertex data and detail coefficients retrieved from the repositories, or, optionally, procedurally generated (see figure 6). In our current implementation, all the decompression steps are performed on the CPU. The fact that we are able to rapidly generate details is exploited to render scenes where color and elevation are not available at the same resolution in the repository. In that case, data is fetched from repository where available, and generated otherwise. Moreover, the fact that coarsening operations are basically no cost, is exploited to support non-monotonic error metrics in a time-critical setting. To that end, the extraction algorithm performs refinement operations until the graph is locally optimal or a time-out occurs, while coarsening has the role of a garbage collector and is performed only once in a while to remove unneeded detail.

Elevation, colors, and normals When constructing a given node, data is required for elevation, and optionally for color and normals. Since our target is to render very small (pixel sized) triangles, texture maps offer limited benefits, since no attribute interpolation would be employed anyway. In the most simple implementation, we have thus the option of managing everything using per-vertex attributes. In addition, since vertices are tightly packed, the normals required for shading can be effectively computed at refinement time from vertex elevation rather than managed in an external repository. Consistently with BDAM diamond constraints, boundary normals are kept fixed, while the others are computed by central differences from elevation data. Communication with

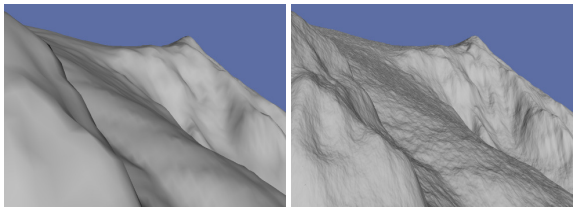


Figure 6: **Procedural Detail Synthesis.** On the left, refinement stops when no more data is available in the repository, i.e., when reaching the input datasets original sampling density; on the right, artificial detail is synthesized by generating procedural detail coefficients (in this case, simply a zero-centered uniform random number less than 30% of triangle edge length for ΔV, and 0 for ΔC.

the GPU is made exclusively through a retained mode interface, which reduces bus traffic by managing a least-recently-used cache of patches maintained on-board as OpenGL *Vertex Buffer Object*. GPU memory and bandwidth costs are further reduced by compactly coding vertex data, and using vertex shaders to perform decompression. All data is specified in local (u,v) patch coordinates. Since we are using regular grids, (u,v) locations as well as triangle strip indices are shared among all patches and stored only once in GPU memory. A single per patch 3-component vector stores elevation $h(u,v)$ and its derivatives dh/du and dh/du, from which positions and normals are computed on the GPU. In a textureless approach, a second optional vertex array stores color in RGB8 format. A texture-based approach can provide a better anisotropic filtering at the cost of higher implementation complexity.

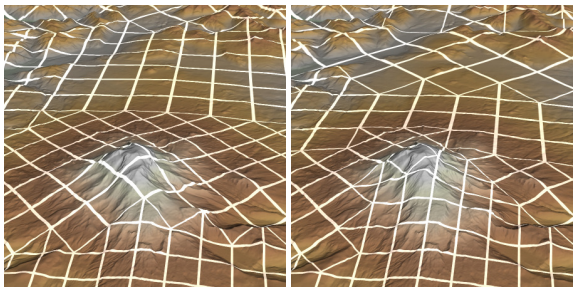


Figure 7: **View-space error control.** A distance based error metric (left) is less efficient in distributing detail than a data dependent measure based on projected diamond size (right).

View space and object space errors. As for other graph based methods, the presented technique is not limited to using distance based LODs, but can employ configurable data-dependent metrics. In the examples presented in this paper, we use oriented bounding boxes as bounding volume primitives, and we simply use the average triangle areas as a measure of object space error. Thanks to the computation of the $L^{\inf}$ error during compression, and, given the relatively modest size of the diamond graph, the use of elaborate metrics at run-time does not slow down the rendering process. Bounding boxes are computed on-the-fly each time a diamond is updated with a new patch during a refine. View space errors are estimated by dividing the projected size of the box on the screen by the number of triangles contained in a diamond. Using a such a data depended measures provides higher quality results than the simple distance based techniques that have to be employed in pyramidal schemes (see figure 7).

7. Implementation and Results

An experimental software library and a rendering application supporting the C-BDAM technique have been implemented on Linux using C++ with OpenGL and NVIDIA Cg. We have extensively tested our system with a number of large terrain datasets. In this paper, we discuss the results obtained on three terrain datasets with different characteristics.

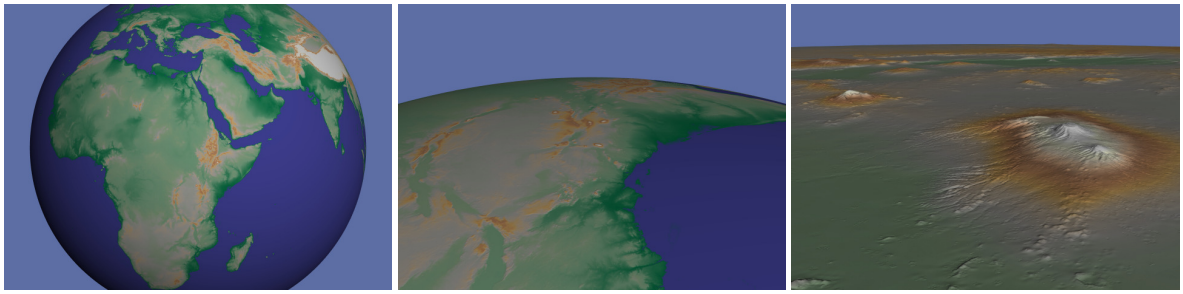
The main dataset is a the global reconstruction of the planet Earth from SRTM data at 3 arcsec resolution (one point every $90m$ at the Equator. Source: CGIAR Consortium for Spatial Information. *srtm.csi.cgiar.org*). The dataset is very large (29 billion samples) and provides an example of variable 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 two other models are local terrain datasets. The first one, is the standard 265 million samples Puget Sound 10m DTM dataset presented here for the sake of comparison with previous work (source: USGS and Peter Lindstrom. *www.cc.gatech.edu/projects/large_models/ps.html*). The second one is a 67 million samples dataset covering the city of Paris at $1sample/m$, textured with an high resolution 872 million samples ortho-photo (source: ISTAR CityView database. *www.istar.com*). The Paris elevation dataset is a worst case situation, as it is a model derived by sampling at 1m spacing a vector representation of building outlines, and, as such contains lots of discontinuities. The texture is a typical city-scale high resolution photograph, and is here to demonstrate the ability to handle color information within our framework.

Preprocessing. All the preprocessing tests were executed 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.

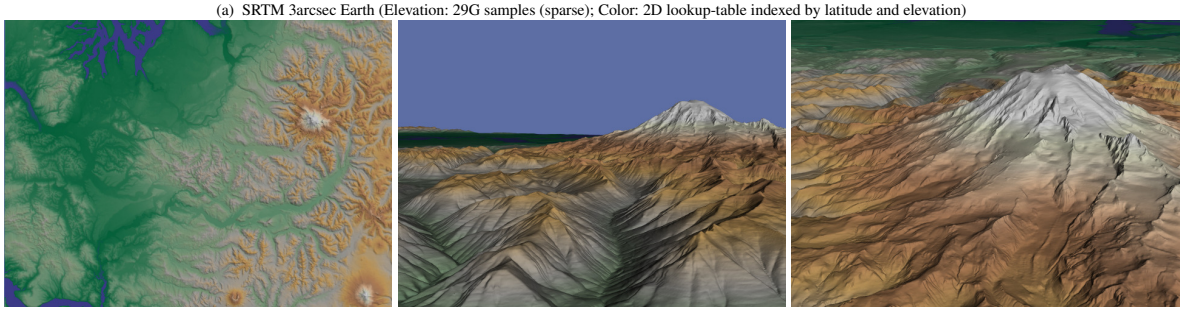
	Sample #	XY Step	Tolerance	Time	Output Size	bps	Rate
Puget Sound	265 M	10 m	1 m RMS	11 m	9.1 MB	0.28	57:1
Puget Sound	265 M	10 m	1 m AMAX	10 m	19.2 MB	0.61	26:1
Earth SRTM	29 G	90 m	16 m AMAX	30 h 38 m	869.9 MB	0.25	64:1
Paris	67 M	1 m	0.1 m AMAX	6 m	14.6 MB	1.80	9:1
Paris color	872 M	0.25 m	10/255 AMAX	1 h 17 m	254.2 MB	2.45	10:1

Table 1: **Numerical results for terrain preprocessing.** Preprocessing results for tests datasets.

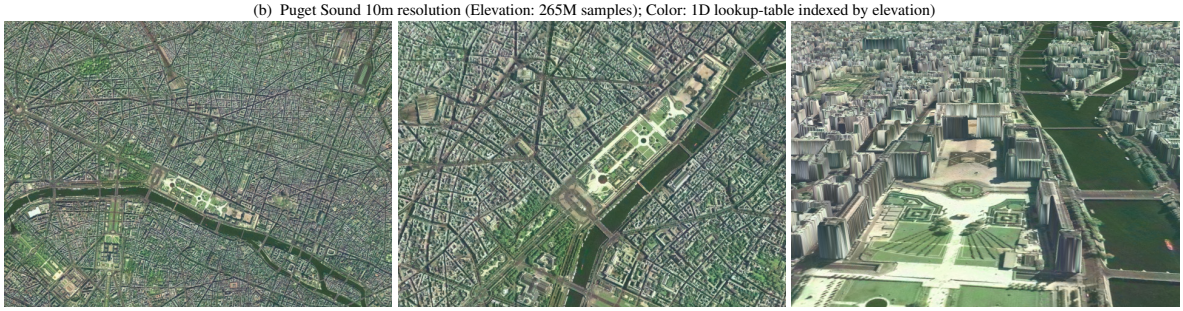
Table 1 lists numerical results for our out-of-core preprocessing method for the Puget Sound and the Earth datasets.



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



(b) Puget Sound 10m resolution (Elevation: 265M samples); Color: 1D lookup-table indexed by elevation)



(c) Paris (Elevation 67M samples; Color 872M samples)

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 pixel sized triangles.

We constructed all multiresolution structures with a prescribed diamond dimension of 64×64 vertex side which gives a good granularity for the multiresolution structure. Each triangular patch is composed by 4K triangles. Preprocessing time is quite equally subdivided into the filtering and the encoding phases. In all the results here presented 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 dominated by 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 260K to 450K input samples per second, more than 7 time faster than P-BDAM [CGG*03b]. This is because regular grid filtering and compression is faster than general mesh simplification. Still, processing speed is about 4 times slower than 4-8 hierarchies[HDJ05], which, however, does not perform compression, and about 2 times slower than Geometry Clipmaps [LH04], which, however, generates half the number of resolution levels.

Compression rates. The compression rates are presented in table 1. The tolerances used for compressing were chosen to provide near-lossless results. In the Puget Sound and in the Paris dataset the chosen tolerances correspond to 1% percent of the sample spacing, while in the Earth dataset the tolerance is $16m$, which corresponds to the vertical accuracy of the input dataset. For the color dataset, we imposed a maximum error tolerance of $10/255$ per component, which is comparable with the error introduced by the S3TC compression algorithm, commonly used in terrain rendering applications (see, e.g, [CGG*03a]).

The three datasets have been preprocessed using maximum absolute error tolerance to drive the compression. For the sake of comparison with previous work based on Geometry Clipmaps [LH04], we also compressed the Puget Sound dataset using RMS error control. In this case, our result of $0.28bps$ is comparable with a Geometry Clipmap [LH04] representation of the same dataset ($0.26bps$), especially since we provide the double of resolution levels. It is inter-

esting to note that the maximum error in this case goes up to $18m$, showing the inherent lack of local control when using a L^2 norm to drive compression. When using a maximum error tolerance, the bit rate increases to $0.61bps$, but all points are guaranteed to be within $1m$ from the original.

The bit rate for planet Earth is even better ($0.25bps$), even though compression rates are given considering the whole amount of memory occupied by the stored dataset versus the number of samples present in the leaves. The Earth dataset is sampled in the poles area and over the sea, where there is no input data, at the minimum resolution imposed by the continuity constraints of the multiresolution structure. Sampling rate in this variable resolution input test case varies from $53m$ to $23Km$, since our graph adapts to the sparseness of the input grid. This way, compression rates appear worse than what they would be when using a full resolution input of constant elevation for flat areas.

To test the performance in an extreme case, we applied our compressor to the Paris elevation dataset. The bit rate increases to $1.80bps$, which illustrates that the method works much better for terrains which are locally smooth. Compression results for the texture are similar, since texture in this case is also dominated by very sharp boundaries, due to buildings and their shadows. Nonetheless, the bit rate for the Paris dataset is still considerably better than what reported for uncompressed terrain representations, which are typically at least one order of magnitude larger [LH04]. Handling discontinuities, e.g., through adaptive lifting [PH01] or a geometric bandelet approach [PM05], is a promising avenue of future work.

Adaptive rendering. The rendering tests were executed on a medium end single PC running Linux 2.6.15 with single AMD 1.8 GHz CPU, 2 GB of RAM, SATA 10000 RPM 150 GB hard drive and PCI Xpress NVIDIA Geforce 7800GT graphics. These tests were performed with a window size of 1280x1024 and a target of pixel-sized triangles.

We evaluated the rendering performance of the technique on a number of flythrough sequences on all the terrain datasets, with a window size of 1280x1024 and a target of pixel-sized triangles. The qualitative performance of our adaptive renderer is illustrated in an accompanying video that shows live recordings of flythrough sequence. The video was recorded at 800x600 windows size due to recording equipment constraints. The sessions were designed to be representative of typical flythrough and to heavily stress the system, and include abrupt rotations and rapid changes from overall views to extreme close-ups. In addition to shading the terrain, we apply a color, coming from an explicit color channel (Paris dataset), a 1D look-up table indexed by elevation (Puget Sound), or a 2D look-up table indexed by elevation and latitude (Earth).

The average frame rates is around 90Hz, while the minimum frame rate never goes below 60Hz using a 1280x1024 window. For all planar datasets, the average triangle throughput is $130M\Delta/s$, and its peak performance is $156M\Delta/s$. In the flythrough of planet Earth, we achieve a average performance of $100M\Delta/s$ and a peak of $125M\Delta/s$. The difference in performance between planar and spherical datasets is due to the different complexity in vertex shaders, which transforming the height information in a vertex position, and compute also the normal values starting from tangent information. These operations are simpler in the planar shader, than in the spherical shader. Normal computation is the most costly operation. With a simpler shader which computes only position and no shading information the maximum reachable performance is $190M\Delta/s$ in both cases. This increase in performance demonstrates that the technique is GPU bound. Even in the current implementation, the triangle rate is high enough to render over $4M\Delta/frame$ at interactive rates. It is thus possible to use very small pixel thresholds, effectively using pixel sized triangles, virtually eliminating popping artifacts without the need to resort to geomorphing techniques.

Network streaming. Some network tests have been performed on all test models, on a ADSL 1.2Mbps connection using the TCP/IP protocol to access the data. As illustrated by the accompanying video, the rendering rate remains the same as the local file version, only the asynchronous updates arrive with increased latency due to the network connections. Since only few diamonds per frame needs update, and diamond data is extremely compressed, the flythrough quality remains excellent.

8. Conclusions

We have described a compressed multiresolution representation for the management and interactive rendering of very large planar and spherical terrain surfaces. Similarly to BDAM, coarse grain refinement operations are associated to regions in a bintree hierarchy. In the C-BDAM approach, all patches share the same regular triangulation connectivity and incrementally encode their vertex attributes using a quantized representation of the difference with respect to values predicted from the coarser level. As illustrated by our experimental result, the structure provides a number of practical benefits: overall geometric continuity for planar and spherical domains, support for variable resolution inputdata, management of multiple vertex attributes, efficient compression and reasonably fast construction times, ability to support maximum-error metrics, real-time decompression and shaded rendering with configurable variable level-of-detail extraction, as well as runtime detail synthesis.

The main take home message of this paper is that it is possible to combine the generality and adaptivity of batched dynamic adaptive meshes with the compression rates of nested regular grid techniques. Although the current implementation already gives satisfactory results, which are state-of-the-art both in terms of compression rates and of rendering speed, there are still open issues that need to be investigated,

besides incremental improvements to the various filtering and compression components, which have been chosen here mostly because of simplicity. A first important avenue of research is to determine whether it is possible to obtain in practice a max-error approximation directly in a one-stage wavelet approximation. A second important future work is to improve treatment of discontinuities, verifying whether current state-of-the-art adaptive techniques are fast enough to be employed in a real-time application.

Acknowledgments. This research is partially supported by the European projects CRIMSON (RTD contract SEC4-PR-011500) and Epoch (NoE contract IST-2002- 507382).

References

[AG05] ALLIEZ P., GOTSMAN C.: Recent advances in compression of 3d meshes. In *Advances in Multiresolution for Geometric Modelling* (2005), N.A. Dodgson M.S. Floater M. S., (Ed.), Springer. 3

[CGG*03a] CIGNONI P., GANOVELLI F., GOBBETTI E., F.MARTON, PONCHIO F., SCOPIGNO R.: BDAM: Batched dynamic adaptive meshes for high performance terrain visualization. *Computer Graphics Forum 22*, 3 (Sept. 2003), 505–514. 1, 2, 8

[CGG*03b] CIGNONI P., GANOVELLI F., GOBBETTI E., MARTON F., PONCHIO F., SCOPIGNO R.: Planet-sized batched dynamic adaptive meshes (P-BDAM). In *IEEE Visualization* (2003), pp. 147–154. 1, 2, 3, 8

[CGG*05] CIGNONI P., GANOVELLI F., GOBBETTI E., MARTON F., PONCHIO F., SCOPIGNO R.: Batched multi triangulation. In *Proceedings IEEE Visualization* (October 2005), IEEE Computer Society Press, pp. 207–214. 3

[DFMP97] DE FLORIANI L., MAGILLO P., PUPPO E.: Building and traversing a surface at variable resolution. In *Proceedings IEEE Visualization 97* (Phoenix, AZ (USA), October 1997), pp. 103–110. 2

[DP02] DECORO C., PAJAROLA R.: Xfastmesh: fast view-dependent meshing from external memory. In *VIS '02: Proceedings of the conference on Visualization '02* (Washington, DC, USA, 2002), IEEE Computer Society, pp. 363–370. 2

[DWS*97] DUCHAINEAU M., WOLINSKY M., SIGETI D., MILLER M., ALDRICH C., MINEEV-WEINSTEIN M.: ROAMing terrain: Real-time optimally adapting meshes. In *Proceedings IEEE Visualization '97* (Oct. 1997), IEEE, pp. 81–88. 2, 6

[Eli75] ELIAS P.: Universal codeword sets and representations of the integers. *IEEE Trans. Inform. Theory 21*, 2 (Mar. 1975), 194–203. 5

[HDJ04] HWA L. M., DUCHAINEAU M. A., JOY K. I.: Adaptive 4-8 texture hierarchies. In *Proceedings of IEEE Visualization 2004* (Los Alamitos, CA, Oct. 2004), IEEE, Computer Society Press, pp. 219–226. 3

[HDJ05] HWA L. M., DUCHAINEAU M. A., JOY K. I.: Real-time optimal adaptation for planetary geometry and texture: 4-8 tile hierarchies. *IEEE Transactions on Visualization and Computer Graphics 11*, 4 (2005), 355–368. 1, 2, 3, 8

[HM93] HITCHNER L. E., MCGREEVY M. W.: Methods for user-based reduction of model complexity for virtual planetary exploration. In *Proc SPIE* (1993), vol. 1913, pp. 622–636. 2

[Hop98] HOPPE H.: Smooth view-dependent level-of-detail control and its aplications to terrain rendering. In *IEEE Visualization '98 Conf.* (1998), pp. 35–42. 2

[JDSB03] JR. R. L. C., DAVIS G. M., SWELDENS W., BARANIUK R. G.: Nonlinear wavelet transforms for image coding via lifting. *IEEE Transactions on Image Processing 12*, 12 (2003), 1449–1459. 4

[KS00] KOVACEVIC J., SWELDENS W.: Wavelet families of increasing order in arbitrary dimensions. *IEEE Transactions on Image Processing 9*, 3 (2000), 480–496. 4

[Lev02] LEVENBERG J.: Fast view-dependent level-of-detail rendering using cached geometry. In *Proceedings IEEE Visualization '02* (Oct 2002), IEEE, pp. 259–266. 2

[LH04] LOSASSO F., HOPPE H.: Geometry clipmaps: terrain rendering using nested regular grids. *ACM Trans. Graph 23*, 3 (2004), 769–776. 1, 2, 3, 8, 9

[LKR*96] LINDSTROM P., KOLLER D., RIBARSKY W., HODGES L., FAUST N., TURNER G.: Real-time, continuous level of detail rendering of height fields. In *Comp. Graph. Proc., Annual Conf. Series (SIGGRAPH 96), ACM Press* (New Orleans, LA, USA, Aug. 6-8 1996), pp. 109–118. 2

[LP01] LINDSTROM P., PASCUCCI V.: Visualization of large terrains made easy. In *Proc. IEEE Visualization 2001* (Oct. 2001), IEEE Press, pp. 363–370, 574. 2

[LP02] LINDSTROM P., PASCUCCI V.: Terrain simplification simplified: A general framework for view-dependent out-of-core visualization. *IEEE Transaction on Visualization and Computer Graphics 8*, 3 (2002), 239–254. 2

[MS03] MALAVAR H., SULLIVAN G.: YCoCg-R: A color space with RGB reversibility and low dynamic range. In *JVT ISO/IEC MPEG ITU-T VCEG*, no. JVT-I014r3. JVT, 2003. 6

[Paj98] PAJAROLA R.: Large scale terrain visualization using the restricted quadtree triangulation. In *Proceedings of Visualization '98* (1998), D. Elbert H. Hagen H. R., (Ed.), pp. 19–26. 2

[Paj02] PAJAROLA R.: *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 2002. 2

[PH01] PIELLA G., HEIJMANS H. J. A. M.: An adaptive update lifting scheme with perfect reconstruction. In *ICIP (3)* (2001), pp. 190–193. 4, 9

[PM05] PEYRÉ G., MALLAT S.: Surface compression with geometric bandelets. *ACM Trans. Graph 24*, 3 (2005), 601–608. 9

[Pom00] POMERANZ A. A.: *ROAM Using Surface Triangle Clusters (RUSTiC)*. Master's thesis, University of California at Davis, 2000. 2

[WMD*04] WAHL R., MASSING M., DEGENER P., GUTHE M., KLEIN R.: Scalable compression and rendering of textured terrain data. In *Journal of WSCG* (Plzen, Czech Republic, Feb. 2004), vol. 12, UNION Agency/Science Press. 2

[YP04] YEA S., PEARLMAN W. A.: A wavelet-based two-stage near-lossless coder. In *Proc. ICIP* (2004), pp. 2503–2506. 4