Coarse-grained Multiresolution Structures for Mobile Exploration of Gigantic Surface Models

Marcos Balsa Rodríguez
CRS4

Enrico Gobbetti
CRS4*

Fabio Marton
CRS4

Alex Tinti
CRS4

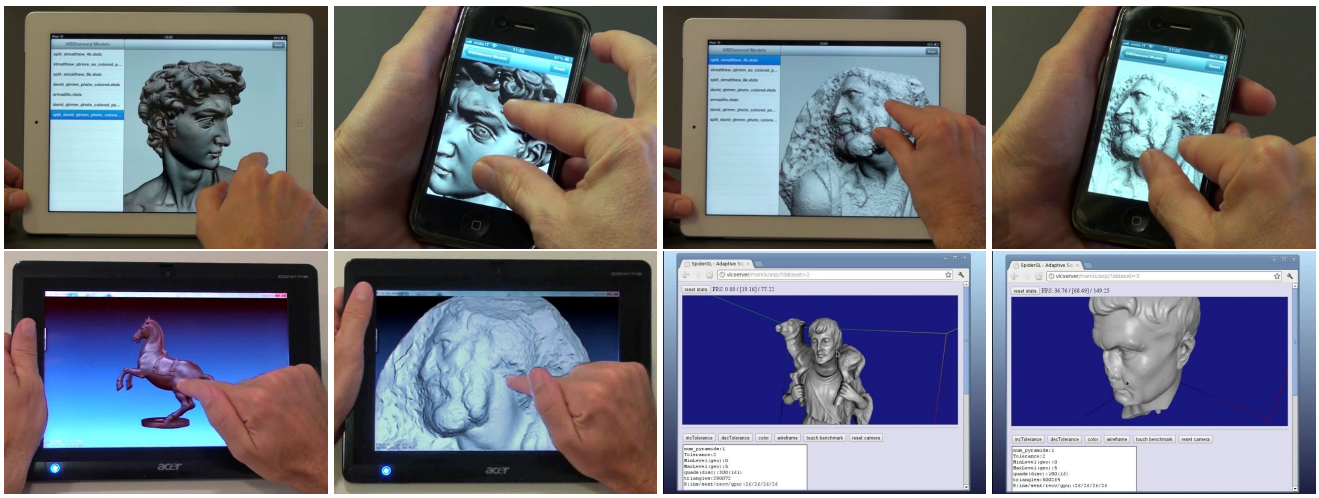

Figure 1: ***Top row:*** *Giga-triangle models interactively explored on a iPhone 4 and on "the new iPad" using the Compact Adaptive TetraPuzzles (CATP) approach.* ***Bottom row:*** *Parameterized scanned models interactively explored using the Adaptive Quad Patches (AQP) approach on a Acer Iconia Tab W501 using native OpenGL implementation (two leftmost images) and on a linux laptop using a WebGL/JavaScript running within Chromium (two rightmost images).*

Abstract

We discuss our experience in creating scalable systems for distributing and rendering gigantic 3D surfaces on web environments and common handheld devices. Our methods are based on compressed streamable coarse-grained multiresolution structures. By combining CPU and GPU compression technology with our multiresolution data representation, we are able to incrementally transfer, locally store and render with unprecedented performance extremely detailed 3D mesh models on WebGL-enabled browsers, as well as on hardware-constrained mobile devices.

CR Categories: I.3.2 [Computer Graphics]: Graphics Systems—Distributed/Network Graphics

Keywords: mobile graphics, LOD, massive models, compression

1 Introduction

With the increasingly widespread diffusion of graphics-enabled mobile terminals with high-quality screens, mobile 3D graphics is attracting a lot of attention. In many domains, detailed high density 3D models are an important ingredient of the information flow that needs to be made available to the public. In cultural heritage, for instance, highly detailed representations are required to reproduce the unique aura of real objects. Such models, however, are heavy, often containing billions of color and geometry samples, and must be experienced in a highly non-linear interactive way. These characteristics impose adaptive streaming and rendering techniques, reasonable compression, and GPU accelerated rendering methods.

Mobile hardware is continuously improving at impressive paces. Screen resolutions are often extremely large. However, mobile 3D graphics hardware is still constrained compared to the desktop counterparts. Major limiting factors are the comparatively low computing powers, low memory bandwidths, small amounts of memory, and limited power supply. Streaming and rendering extremely detailed 3D surface models, thus requires improving state-of-the-art results in a number of technological areas.

We have recently demonstrated how coarse-grained multiresolution techniques combined with compression methods provide a solid basis upon which to create high performance mobile 3D graphics renderers. In this presentation, we illustrate two recently introduced approaches, Compact Adaptive TetraPuzzles (CATP) [Balsa Rodriguez et al. 2013] and Adaptive Quad Patches (AQP) [Gobbetti et al. 2012], which achieve high performance by exploiting different levels of regularity in the multiresolution structure.

These two surface representation approaches share a number of concepts and are integrated in a common compression, streaming, and rendering framework. Our results show that current mobile graphics terminals and/or Web environments are capable with our methods to support, using nowadays networks, the ubiquitous interactive exploration of giga-sample-sized colored 3D models.

2 Related work

Building an efficient system for the exploration of massive 3D meshes on mobile devices requires the improvement and combination of state-of-the-art results in a number of technological areas. In

*CRS4 Visual Computing, Italy — www.crs4.it/vic/ — {name.surname}@crs4.it

the following, we briefly discuss only the approaches most closely related to ours. Readers may refer to established surveys on massive model rendering [Gobbetti et al. 2008], compression [Alliez and Gotsman 2003; Peng et al. 2005], and mobile graphics [Capin et al. 2008] for further details.

While many examples exist for rendering light 3D models on portable platforms (e.g., MeshPad [ISTI-CNR Visual Computing Lab 2012] for meshes or PCL [Marion 2012] for points), exploring massive models on mobile devices is still a hot research topic. Much of the work in model distribution has focused so far on compression of mesh structures rather than adaptive view-dependent streaming. Compressed graphics data potentially enables mobile application to better utilize the limited storage space and bandwidth at all levels of the pipeline. Many mesh compression algorithms offer good performance in compression ratio for both topology and vertex attributes. MPEG-4 [Jovanova et al. 2009] is a reference work in the field, and includes 3D mesh coding (3DMC) algorithms based on topological surgery algorithm [Taubin and Rossignac 1998] and progressive forest split [Taubin et al. 1998].

Classic methods for view-dependent LOD and progressive streaming of arbitrary meshes were built on top of fine-grained updates based on edge collapses or vertex clustering [Xia and Varshney 1996; Hoppe 1997; Luebke and Erikson 1997]. Many compression and streaming formats for the web have been built upon them [Maglo et al. 2010; Blume et al. 2011; Niebling et al. 2010]. These methods, however, are CPU-bound and spend a great deal of rendering time computing a view-dependent triangulation prior to rendering, making their implementation in a mobile setting particularly challenging. With the increasing raw power of GPUs, the currently higher-performance methods typically reduce the per-primitive workload by pre-assembling optimized surface patches [Cignoni et al. 2004; Yoon et al. 2004; Cignoni et al. 2005; Borgeat et al. 2005; Gobbetti and Marton 2004a; Gobbetti and Marton 2004b; Goswami et al. 2013], although this kind of approach has been demonstrated to work on mobile devices until very recently only in the context of point-based rendering [Balsa Rodriguez et al. 2012].

This work illustrates two recently introduced coarse-grained multiresolution mesh approaches: a general compressed multiresolution mesh structure based upon tetrahedral space partitioning (Compact Adaptive TetraPuzzles (CATP) [Balsa Rodriguez et al. 2013]), which improves the Adaptive TetraPuzzles approach [Cignoni et al. 2004] by introducing a compressed GPU-friendly representation that ensures crack-free surfaces while using local quantization, and an efficient image-based mesh representation [Gobbetti et al. 2012], which works for models for which a isometric quad parametrization exist on a simple domain.

Rather than looking for maximum compression, CATP focuses, instead, in computing a representation for geometry that reduces the bandwidth required to transmit it to the graphics subsystem. Hardware-compatible vertex data compression is typically achieved in this context by attribute quantization. Since global position quantization [Calver 2002; Purnomo et al. 2005; Lee et al. 2009] provides poor rate-distortion performance for large meshes, recent efforts have concentrated on local quantization techniques [Lee et al. 2010], which, however, lead to cracks for multiresolution meshes. In CATP, we improve over these local quantization approaches by expressing positions of mesh fragment vertices in the barycentric coordinate system relative to the containing tetrahedron. Hardware-friendly normal compression is achieved through an octahedral parametrization of normals [Meyer et al. 2010]. To our knowledge CATP is the first method fully supporting local quantization in a general adaptive 3D mesh structure.

The AQP methods employs, instead, a solution that encodes much of the shape and appearance of a model into a texture. This is also the goal of *geometry images* [Gu et al. 2002; Sander et al. 2003]. which enable the powerful GPU rasterization architecture to process geometry in addition to images, and the networking component to rely on already existing and optimized libraries for compression and streaming of images. Geometry images focus on reparametrizations of meshes onto regular grids, while we focus on developing a specific multiresolution structure on top of a reparameterized model. Our quad-based parametrization leads in addition to a tighter texture packing and a simple handling of chart boundaries. We also adapt semi-uniform adaptive patch tessellation [Dyken et al. 2009] to handle collections of quad patches at different LODs with textured detail. Whereas previous adaptive GPU mesh refinement approaches are typically used to amplify coarse geometry, our end-to-end framework is designed to faithfully reproduce a resampled high-resolution model.

3 Regularity in multiresolution models

Networked massive model renderers have to employ methods for filtering out as efficiently as possible the data that is not contributing to a particular image. and to adaptively load and render them, efficiently using bandwidth and local resources by combining compression and data management methods. A compressed continuous level-of-detail (LOD) model, i.e., a compact description of multiple representations of a single shape supporting the extraction of representations with varying accuracies in different regions, is the key element for providing the necessary degrees of freedom to achieve run-time adaptivity. The basic ingredients of a such a model are a *base mesh*, that defines the coarsest approximation to the 3D model surface, a set of compactly represented *updates*, that, when incrementally loaded and applied to the base mesh, provide variable resolution mesh-based representations, and a *dependency relation* among updates, which allows combining them to extract consistent intermediate representations. Different specialized multiresolution models, of various efficiency and generality, are obtained by mixing and matching different instances of all these ingredients.

In the most general (and common) case, the multiresolution model is based on a fully irregular approach in which the base mesh is an irregular triangulation with unrestricted connectivity, and updates are encoded as changes to regions of this triangulations. Because of their flexibility, fully irregular approaches are theoretically capable of producing the minimum complexity representation for a given error measure. However, this flexibility comes at a price. In particular, mesh connectivity, hierarchy, and dependencies must explicitly be encoded, and simplification and coarsening operations must handle arbitrary neighborhoods. By imposing constraints on mesh connectivity and update operations it is possible to devise classes of more restricted models that are less costly to store, transmit, render, and simpler to modify. This is because much of the information required for all these tasks becomes implicit, and often, because stricter bounds on the region of influence of each local modification can be defined.

In this paper, we illustrate two recently introduced approaches that achieve performance by introducing some regularity in the representation. The first example is the Compact Adaptive TetraPuzzles (CATP) method, introduced by Balsa et al. [2013], based on irregular triangulations but exploiting the regularity of a hierarchical volumetric space partitioning to construct a compact and seamless multiresolution model. The second example, the Adaptive Quad Patches (AQP) approach introduced by Gobbetti et al. [2012] adopts, instead, a fully regular approach by parameterizing and resampling the mesh into an image-based structure.

4 CATP: Compact Adaptive TetraPuzzles

The CATP method, introduced by Balsa et al. [2013], builds on Adaptive TetraPuzzles (*ATP*) [Cignoni et al. 2004] by using a regular conformal hierarchy of tetrahedra to spatially partition the input 3D model and to arrange mesh fragments at different resolution in an implicit diamond graph.

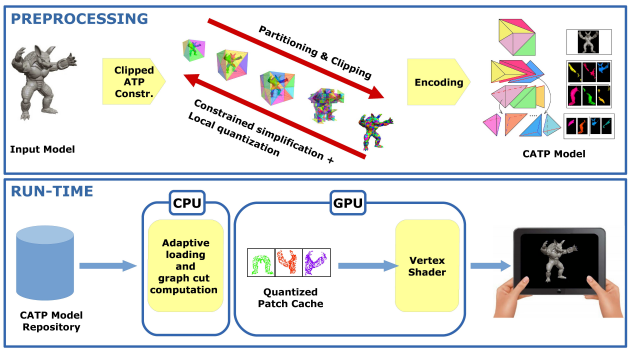

Figure 2: ***CATP overview*** *A regular conformal hierarchy of tetrahedra spatially partitions the input 3D model and arranges mesh fragments at different resolution in an implicit diamond graph. Tetrahedra not only partition but also clip the original triangulation, ensuring that each mesh fragment is fully contained within its bounding tetrahedron. Local quantization with appropriate boundary constraints can thus be used to compress the model. At run-time, clients maintain an adaptive local graph cut, and rendering is performed directed on compressed data, which is dequantized in the vertex shader.*

The leaves of the multiresolution structure contain representation of the full resolution original model, while inner nodes contain simplified representations of the geometry with approximatively half of the number of triangles contained into the children. In contrast to ATP, tetrahedra not only partition but also clip the original triangulation, ensuring that each mesh fragment is fully contained within its bounding tetrahedron, and that vertices shared with neighboring fragments always lie on a tetrahedron's face. The tetrahedral structure can thus be exploited in a compression scheme by encoding vertex positions with tetrahedral barycentric coordinates, i.e., expressing vertex positions as combinations of a tetrahedron's corners. These coordinates can be quantized locally for each tetrahedra. Continuity among adjacent tetrahedra is ensured when quantizing positions, by the fact that barycentric coordinates of vertices lying on tetrahedra faces are expressed as combination of only the three corners defining that face, which are the same among neighboring tetrahedra in a conformal hierarchy. Such an approach introduces for the first time local quantization in a general adaptive 3D mesh structure

Quantized vertex coordinates are encoded together with normals and colors into a compact GPU-friendly 64bit representation suitable for direct rendering, where 3 bytes are used for position, 2 bytes for normal and 3 bytes for color. Position is parameterized with 4 barycentric coordinates, but only 3 components are required since the four components sum to one. Normals are encoded using the *octahedron normal vector* approach [Meyer et al. 2010], which maps unit vectors to two parametric coordinates. This encoding consists in projecting the normals onto the octahedron by normalizing them with the 1-norm. The octahedron is unwrapped to a square and the $[u, v]$ parameters in the plane are quantized to 8 bits, leading to sub-degree precision [Meyer et al. 2010]. Decompression is numerically stable and requires only few basic operations which can be executed in the vertex shader. The remaining 3 bytes are used for a RGB representation of colors. This 64bit/vertex encoding provides an extremely compact aligned representation that can be efficiently accessed on current GPUs, which typically require vertex data to be aligned on 32 bit boundaries.

For storage and network distribution, further compression is obtained on top of the compact GPU-friendly representation by exploiting local data coherence using a low-complexity coding approach based on a wavelet transformation followed by entropy coding of coefficients (see Balsa et al. [2013] for details).

On a mobile client, an adaptive rendering approach incrementally updates the representation that best fits the current point of view and retrieves the required data from a remote server in an incremental fashion. Differently from ATP, where the multiresolution structure was encoded by six binary trees of disjoint tetrahedra, the CATP run-time structure is based on *diamonds*. Each diamond is composed by the set of all the tetrahedra sharing their longest edge. Using a diamond based structure, see Weiss and De Floriani [2010], dependencies are implicitly encoded into the hierarchy, and refinement is interruptible, producing a conforming mesh also when children data is not available. This feature perfectly fits in a mobile rendering architecture, where it is common to experience data fetch delays due to bandwidth limitations.

The working set is kept to a fixed small size in a GPU friendly representation by performing entropy decoding at network loading time and storing data on the mobile device directly in the the compact 64-bit/vertex GPU format suitable for direct rendering through specific *shaders* that do the decoding directly in the *Vertex Shader*. Since color is already in the RGB24 format, the only extra work that needs to be performed is the linear transformation of positions from local barycentric coordinates and the decoding of normals from the two quantized octahedral map coordinates.

Advantages The method is fully adaptive and able to retain all the original topological and (within quantization limits) geometrical detail of the original meshes. The method is not limited to meshes of a particular topological genus or with a particular subdivision connectivity and preserves geometric continuity of variable resolution representations at no run-time cost. CPU and GPU cooperate for decompression, and a shaded rendering of colored meshes is performed at interactive speed on the GPU directly from an intermediate compact representation that uses only 8bytes/vertex, therefore coping with both memory and bandwidth limitations. Keeping data compact up to rendering time is of particular importance with current mobile devices, with extremely large screen resolutions, which dictate large rendering working sets, but limited main memory sizes. For instance, the current iPad generation sports a 3Mpixel display, but has a RAM capacity of only 1GB. Moreover, by decoding compressed data on-the-fly on graphics hardware, we can not only reduce local memory consumption, but also power consumption, thanks to reduced memory access and data transmission through the system bus.

Limitations The regularity of the subdivision structure does not adapt to geometric complexity. In addition, the mesh simplification approach repeatedly merges nearby surface points based on error minimization considerations. The method, thus, perform best for highly tessellated surfaces that are otherwise relatively smooth and topologically simple, since it becomes difficult, in other cases, to derive good "average" merged properties and to generate subdivision with a reduced amount of boundary vertices/patch.

Device compatibility The method poses minimal constraints on the hardware, and just requires standard vertex shaders to execute vertex dequantization. On all tested platforms, we achieved maximum performance when data is stored in an interleaved array of 8 bytes per vertex.

Implementation and results Using a Core i7 GPU, input models from the Digital Michelangelo Repository are processed at about 24k triangles/s, achieving a data compression rate between 40 and 50 bits/vertex (including topology and entropy quantization). Rendering is performed directly on the compact vertex array, with on-the-fly GPU decoding from the 64bits/vertex data representation. We have implemented the method both on Android and iOS devices. On a 3rd generation iPad we are able to sustain an average throughput of 30 Mtriangles/second, while on a low-end iPhone 4 the throughput is 2.8 Mtriangles/second. High-detail interactive exploration is achieved in both cases (with average frame rates ranging from 10 to 30 fps), since the lower performance of the iPhone4 is balanced by the much reduced screen resolution. We have measured performance with a wireless connection of a Linksys WAP 200 802.11 b/g access point 54 Mbps, as well as with UMTS/HSPA connections. The wireless network was shared among many clients, and we measured its peak performance to be 17 Mbps. Applications performs data fetching asynchronously in a separate thread to avoid delaying interactive rendering, and have a peak network bandwidth usage of about 4.8Mbps for an iPad connected to the wireless network and 3.3Mbps for the UMTS/HSPA connection. iPhone performance is about 1.5x slower, due to the lower CPU performance, which leads to increased decoding time. This peak network utilization occurs only upon startup or at abrupt image changes. In any case, interactive performance is guaranteed even for current broadband networks. More detailed results are available in the original paper [Balsa Rodriguez et al. 2013].

5 AQP: Adaptive Quad Patches

AQP [Gobbetti et al. 2012] introduces a remote rendering approach based in which a large class of textured geometric models are converted into a fully regular compact multiresolution image-based representations suitable for storage, distribution, and real-time rendering on modern commodity/web platforms. The approach can be seen as step aiming at bridging the gap that currently exists from general-purpose meshes to rendering oriented structures based on real-time tessellation with normal/bump maps, which are typical of modern gaming platform but currently require considerable human effort to create.

The pipeline takes as input a dense point sampling of the original model. This kind of sampled representation can be created from a large variety of models - point clouds, meshes, or parametric objects. A two-manifold triangular mesh is first fit to the point cloud using a surface reconstruction and topology cleaning step, and the resulting two-manifold mesh is parameterized. Reconstruction and topological filtering are achieved through Poisson reconstruction [Kazhdan et al. 2006], while parameterization on a simple quad-based domain is achieved by first constructing an almost isometric triangle mesh parameterization through abstract domains [Pietroni et al. 2010], which maps the original mesh to a simplified parametrization domain made of equilateral triangles, and then converting the result domain triangulation into a collection of 2D square regions by adding a vertex in the barycenter of each triangle and building a quad for each edge. The final parameterization domain D thus consists in a small collection of almost isometric square patches. Since each of these patches can be sampled on a grid with lines parallel to its sides, storing 3D positions, normals and colors of the associated point on the mesh in a $N \times N$

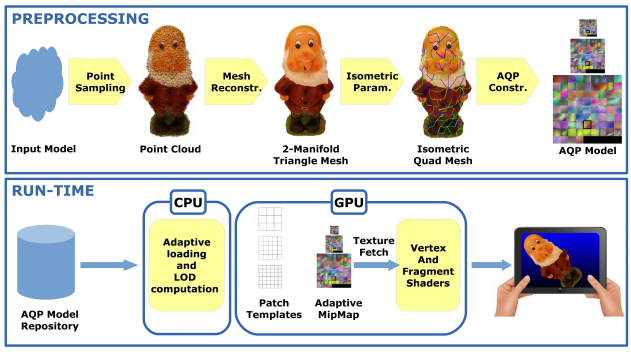

Figure 3: ***AQP overview*** *Input models are transformed into two-manifold meshes whose parametrization domain is a small collection of 2D square regions, encoded into a very compact multiresolution structures composed of variable resolution quad patches. At run-time, we adaptively stream and store geometry and color in a tightly packed mipmapped texture atlas. Seamless variable resolution shape representations are rendered using a GPU-accelerated adaptive tessellation algorithm with negligible CPU overhead.*

square patch, the overall shape representation consists of M square patches of $N \times N$ samples. This regular structure is then encoded into a compact multiresolution structure composed of variable resolution quad patches assembled in 2D images. Geometry and color are stored in a tightly packed multiresolution texture atlas, which can be streamed over the network for generating variable resolution shape representations using a GPU-accelerated adaptive tessellation algorithm. On the client side, the texture atlas is stored as a texture MipMap. Seamless surface rendering is substantially performed by the GPU through a vertex/fragment shader pair, leaving to the CPU only the tasks of selecting the proper level of details for each patch, and of querying missing data from a server. To produce a continuous representation patches must match perfectly along the edges, so they must have the same level of resolution for each edge. The patch LOD evaluation thus first computes the desired LOD for each edge of the quad by projecting it to the screen and comparing it with the desired screen tolerance. Then, each quad patch LOD is set to the maximum (finest) of the 4 edge LODs. We then adapt semi-uniform adaptive patch tessellation [Dyken et al. 2009] to draw quad patches with textured detail with constant resolution at the interior and edge stitching on the boundary. This is achieved by rendering patch templates, and fetching geometry and color details using texture fetches.

The resulting rendering subsystem has negligible CPU overhead and is heavily built on top of consolidated 2D image representations. It can thus be efficiently implemented both on conventional commodity platforms and on the newly emerging scripting platforms for the web. In contrast to AQP, all the encoding/decoding of data can be done using standard image libraries, a major advantage in scripting environments.

Advantages The AQP pipeline is fully automatic and targets densely tessellated models, such as those created by 3D scanning or modeling systems such as ZBrush. The simplicity of a regularly remeshed representation has many benefits. In particular, it reduces random memory accesses and eliminates the indirection and storage of per triangle vertex indices and per vertex texture coordinates. The resulting representation is compact, can be built on top of existing image representations, and is very well suited to streaming. Due to the negligible run-time CPU overhead, real-time performance is achieved both on conventional GPU platforms us-

ing OpenGL, and on the emerging web-based environments based on WebGL. Promising applications of the technology range, thus, from the automatic creation of rapidly renderable objects for local and online games to the set-up of browsable 3D models repositories in the web.

Limitations AQP not general purpose, but targets only restricted classes of meshes defining closed objects with large components (i.e., typical solid objects without fine topological detailssuch as small handles). As for other compressed streamable formats, the method does not strive to exactly replicate the original geometry and color, but only to visually approximate them in a faithful way. As a result, and similarly to compressed video/image formats, the representation is lossy, and thus not applicable in situations where precise measures of the original geometry are required (e.g., CAD systems). Moreover, an efficient full-GPU implementation of the method requires vertex texture fetch (VTF) capabilities, which are currently only available on selected mobile platforms.

Device compatibility considerations There are a variety of GPU chipsets from different vendors present in current mobile devices. From those, the most extended ones include the PowerVR SGX 5xx GPUs present in Apple devices and many Android phones, and the Qualcomm Adreno family integrated in Android HTC devices and many of the latest high-end Android devices. The hardware present in those two GPU families fully support OpenGL ES 2.0 and, in their latest versions, OpenGL ES 3.0. Unfortunately, while on Android devices the drivers typically expose VTF capabilities,the drivers from iOS 4.x and on does not. Rendering this feature unavailable in iPhone/iPad/iPod devices. The latest T6xx version of the ARM Mali GPU, integrated in the Nexus 10, also includes support for VTF. A number of tablet devices, in addition, use chipsets with discrete or integrated GPUs originally designed for netbooks, which fully support VTFs. An example are chips from the AMD Radeon HD series used in a number of Windows-based tablets.

Implementation and results The methods has been implemented for both mobile clients using native codes and web-based environments, using JavaScript and OpenGL. In order to guarantee full compatibility among platforms and minimum decoding overhead, we have used plain PNG encoding for storage and transport of quad patches, achieving a compression rate of about 24bps for colored models. At the expense of portability, higher compression rates can be achieved with formats such as DXT1 and DXT5. Rendering performance has been evaluated both for in-browser rendering (desktop machine) using WebGL and mobile rendering using native implementation. Using a Chromium browser, on a 1.6 GHz laptop equipped with an NVidia Geforce GTX 260M with 1 GB video memory, we achieve a throughput of of 34.2MTri/s, for an average frame-rate of 37fps for a 750x350 in-browser window and a tolerance of 1 pixel. Coming to mobile platforms, on a Acer Iconia Tab W501, with a AMD Radeon HD 6290 graphics adapter with 384MB DDR3, we achieve a peak throughput of 27.5MTri/s using the native C++/OpenGL implementation, which is perfectly adequate to guarantee real-time performance. The combination of incremental multiresolution refinement with compression also reduces network bandwidth, which was measured to range from $312Kbps$ for exploration of areas not previously seen, and peaks of $2.8Mbps$ at viewpoint discontinuities. Interactive mobile applications are thus possible both for wireless connections to typical ADSL lines, and for current mobile broadband network such as UMTS/HSPA. More detailed results, with a focus on the WebGL implementation, are available in the original paper [Gobbetti et al. 2012].

6 Conclusions

We have briefly illustrated our experience in creating scalable systems for distributing and rendering gigantic 3D surfaces on common handheld devices. Our methods are based on compressed streamable coarse-grained regular or irregular multiresolution structures. By exploiting structure regularity, and combining CPU and GPU compression technology with our multiresolution data representation, we are able to incrementally transfer, locally store and render with unprecedented performance extremely detailed 3D mesh models on WebGL-enabled browsers, as well as on selected hardware-constrained mobile devices.

Both presented techniques exploit GPU hardware to render directly from compressed data. This compression-domain rendering approach is promising, but the currently limited feature sets of mobile GPUs (e.g., lack of tessellation units or low performance in texture fetches), severely limits the kind of approaches that can be implemented. However, the current hardware trend is to deploy architectures that finally bring feature/API parity to PCs, next-gen consoles, and smartphones. For instance, the announced Tegra4 GPU (NVIDIA Logan architecture) is announced to support DirectX11, OpenGL 4.4, OpenGL ES 3.0, hardware tessellation, and CUDA 5. This convergence promises to foster a new generation of mobile graphics applications in which complex scenes are rendered directly from compact GPU-friendly models without prior decompression.

Acknowledgments. This work is partially supported by the EU FP7 Program under the DIVA project (REA Agreement 290277). We also acknowledge the contribution of Sardinian Regional Authorities. We thank Fabio Ganovelli and Marco Di Benedetto from ISTI-CNR for their important contribution in the implementation of the AQP approach. Models are courtesy of Stanford University, the Digital Michelangelo Project, and ISTI-CNR.

References

ALLIEZ, P., AND GOTSMAN, C. 2003. Recent advances in compression of 3D meshes. In *Advances in Multiresolution for Geometric Modelling*, Springer-Verlag, 3–26.

BALSA RODRIGUEZ, M., GOBBETTI, E., MARTON, F., PINTUS, R., PINTORE, G., AND TINTI, A. 2012. Interactive exploration of gigantic point clouds on mobile devices. In *The 14th International Symposium on Virtual Reality, Archaeology and Cultural Heritage*, 57–64.

BALSA RODRIGUEZ, M., GOBBETTI, E., MARTON, F., AND TINTI, A. 2013. Compression-domain seamless multiresolution visualization of gigantic meshes on mobile devices. In *Proc. ACM Web3D*, 99–107.

BLUME, A., CHUN, W., KOGAN, D., KOKKEVIS, V., WEBER, N., PETTERSON, R., AND ZEIGER, R. 2011. Google Body: 3D human anatomy in the browser. In *ACM SIGGRAPH 2011 Talks*, 19:1.

BORGEAT, L., GODIN, G., BLAIS, F., MASSICOTTE, P., AND LAHANIER, C. 2005. GoLD: interactive display of huge colored and textured models. *ACM Trans. Graph. 24*, 3, 869–877.

CALVER, D. 2002. Vertex decompression in a shader. *ShaderX: Vertex and Pixel Shader Tips and Tricks*, 172–187.

CAPIN, T., PULLI, K., AND AKENINE-MOLLER, T. 2008. The state of the art in mobile graphics research. *IEEE Computer Graphics and Applications 28*, 4, 74–84.

CIGNONI, P., GANOVELLI, F., GOBBETTI, E., MARTON, F., PONCHIO, F., AND SCOPIGNO, R. 2004. Adaptive TetraPuzzles

– efficient out-of-core construction and visualization of gigantic polygonal models. *ACM Trans. Graph. 23*, 3, 796–803.

CIGNONI, P., GANOVELLI, F., GOBBETTI, E., MARTON, F., PONCHIO, F., AND SCOPIGNO, R. 2005. Batched multi triangulation. In *Proc. IEEE Visualization*, 207–214.

DYKEN, C., REIMERS, M., AND SELAND, J. 2009. Semi-uniform adaptive patch tessellation. *Computer Graphics Forum 28*, 8, 255–263.

GOBBETTI, E., AND MARTON, F. 2004. Layered point clouds. In *Proc. Eurographics Symposium on Point Based Graphics*, 113–120,227.

GOBBETTI, E., AND MARTON, F. 2004. Layered point clouds: A simple and efficient multiresolution structure for distributing and rendering gigantic point-sampled models. *Computers & Graphics 28*, 1, 815–826.

GOBBETTI, E., KASIK, D., AND YOON, S.-E. 2008. Technical strategies for massive model visualization. In *Proc. ACM SPM*, 405–415.

GOBBETTI, E., MARTON, F., BALSA RODRIGUEZ, M., GANOVELLI, F., AND DI BENEDETTO, M. 2012. Adaptive Quad Patches: an adaptive regular structure for web distribution and adaptive rendering of 3D models. In *Proc. ACM Web3D*, 9–16.

GOSWAMI, P., EROL, F., MUKHI, R., PAJAROLA, R., AND GOBBETTI, E. 2013. An efficient multi-resolution framework for high quality interactive rendering of massive point clouds using multi-way kd-trees. *The Visual Computer 29*, 1, 69–83.

GU, X., GORTLER, S. J., AND HOPPE, H. 2002. Geometry images. *ACM Trans. Graph. 21*, 3, 355–361.

HOPPE, H. 1997. View-dependent refinement of progressive meshes. In *Proc. ACM SIGGRAPH*, 189–198.

ISTI-CNR VISUAL COMPUTING LAB, 2012. MeshLab for iOS: A powerful easy-to-use 3D mesh viewer for iPad and iPhone. www.meshpad.org.

JOVANOVA, B., PREDA, M., AND PRETEUX, F. 2009. MPEG-4 Part 25: A graphics compression framework for xml-based scene graph formats. *Image Commun. 24*, 1-2, 101–114.

KAZHDAN, M., BOLITHO, M., AND HOPPE, H. 2006. Poisson surface reconstruction. In *Proc. SGP*, 61–70.

LEE, H., LAVOUÉ, G., AND DUPONT, F. 2009. Adaptive coarse-to-fine quantization for optimizing rate-distortion of progressive mesh compression. In *Proc. VMV*, 73–82.

LEE, J., CHOE, S., AND LEE, S. 2010. Compression of 3D mesh geometry and vertex attributes for mobile graphics. *Journal of Computing Science and Engineering 4*, 3, 207–224.

LUEBKE, D., AND ERIKSON, C. 1997. View-dependent simplification of arbitrary polygonal environments. In *Proc. ACM SIGGRAPH*, 199–208.

MAGLO, A., LEE, H., LAVOUÉ, G., MOUTON, C., HUDELOT, C., AND DUPONT, F. 2010. Remote scientific visualization of progressive 3D meshes with X3D. In *Proc. ACM Web3D*, 109–116.

MARION, P., 2012. Point cloud streaming to mobile devices with real-time visualization. www.pointclouds.org.

MEYER, Q., SUESSMUTH, J., SUSSNER, G., STAMMINGER, M., AND GREINER, G. 2010. On floating-point normal vectors. *Computer Graphics Forum 29*, 4, 1405–1409.

NIEBLING, F., KOPECKI, A., AND BECKER, M. 2010. Collaborative steering and post-processing of simulations on hpc resources: Everyone, anytime, anywhere. In *Proc. ACM Web3D*, 101–108.

PENG, J., KIM, C.-S., AND JAY KUO, C. C. 2005. Technologies for 3D mesh compression: A survey. *J. Vis. Comun. Image Represent. 16*, 6, 688–733.

PIETRONI, N., TARINI, M., AND CIGNONI, P. 2010. Almost isometric mesh parameterization through abstract domains. *IEEE Transactions on Visualization and Computer Graphics 16*, 4, 621–635.

PURNOMO, B., BILODEAU, J., COHEN, J. D., AND KUMAR, S. 2005. Hardware-compatible vertex compression using quantization and simplification. In *Proc. ACM Graphics Hardware*, 53–61.

SANDER, P. V., WOOD, Z. J., GORTLER, S. J., SNYDER, J., AND HOPPE, H. 2003. Multi-chart geometry images. In *Proc. SGP*, 146–155.

TAUBIN, G., AND ROSSIGNAC, J. 1998. Geometric compression through topological surgery. *ACM Trans. Graph. 17*, 2, 84–115.

TAUBIN, G., GUÉZIEC, A., HORN, W., AND LAZARUS, F. 1998. Progressive forest split compression. In *Proc. ACM SIGGRAPH*, 123–132.

WEISS, K., AND DE FLORIANI, L. 2010. Simplex and diamond hierarchies: Models and applications. In *Eurographics 2010 - State of the Art Reports*, 113–136.

XIA, J., AND VARSHNEY, A. 1996. Dynamic view-dependent simplification for polygonal models. In *Proc. IEEE Visualization*, 327–334.

YOON, S.-E., SALOMON, B., GAYLE, R., AND MANOCHA, D. 2004. Quick-vdr: Interactive view-dependent rendering of massive models. In *Proc. IEEE Visualization*, 131–138.