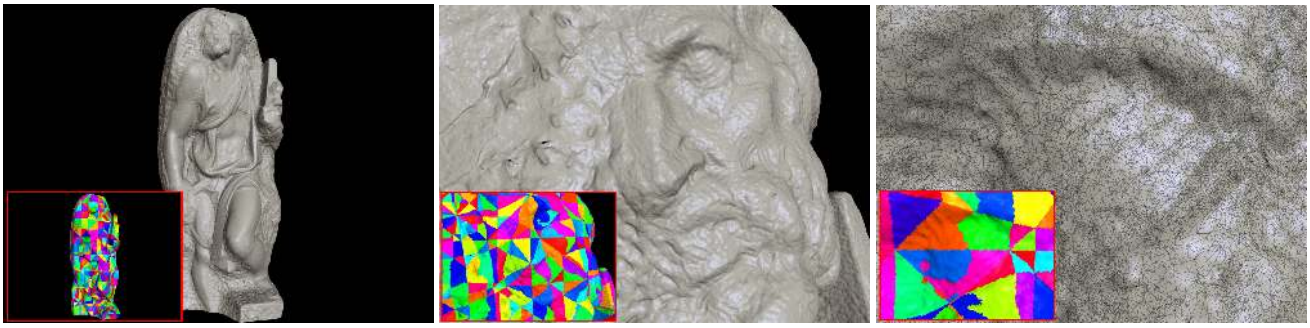
Adaptive TetraPuzzles: Efficient Out-of-Core Construction and Visualization of Gigantic Multiresolution Polygonal Models

Paolo Cignoni ISTI - CNR *
Fabio Ganovelli ISTI - CNR
Enrico Gobbetti CRS4 †
Fabio Marton CRS4
Federico Ponchio ISTI - CNR
Roberto Scopigno ISTI - CNR



(a) 1473 patches (2179K triangles) (b) 1221 patches (1855K triangles) (c) 625 patches (991K triangles)

Figure 1: **View-dependent rendering of the St. Matthew dataset.** The full resolution model contains 373 million triangles and is inspected at over 40 fps on a commodity PC platform. The main images present the mesh rendered with Gouraud shading using 4x Gaussian Multisampling on a 1280x1024 window, while the small inset figures depict the adaptive mesh structure with a different color for each patch. The rightmost image also shows the adaptive triangulation.

Abstract

We describe an efficient technique for out-of-core construction and accurate view-dependent visualization of very large surface models. The method uses a regular conformal hierarchy of tetrahedra to spatially partition the model. Each tetrahedral cell contains a precomputed simplified version of the original model, represented using cache coherent indexed strips for fast rendering. The representation is constructed during a fine-to-coarse simplification of the surface contained in diamonds (sets of tetrahedral cells sharing their longest edge). The construction preprocess operates out-of-core and parallelizes nicely. Appropriate boundary constraints are introduced in the simplification to ensure that all conforming selective subdivisions of the tetrahedron hierarchy lead to correctly matching surface patches. For each frame at runtime, the hierarchy is traversed coarse-to-fine to select diamonds of the appropriate resolution given the view parameters. The resulting system can interatively render high quality views of out-of-core models of hundreds of millions of triangles at over 40Hz (or 70M triangles/s) on current commodity graphics platforms.

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, Level of Detail

*ISTI-CNR, Via Moruzzi 1, 56124 Pisa Italy www:http://vcg.isti.cnr.it/ e-mail:first.last@isti.cnr.it

†CRS4, POLARIS Edificio 1, 09010 Pula, Italy www:http://www.crs4.it/ e-mail:first.last@crs4.it

1 Introduction

The need for interactively inspecting very large surface meshes, consisting of hundreds of millions of polygons, arises naturally in many application domains, including 3D scanning, geometric modeling, and numerical simulation. However, despite the rapid improvement in hardware performance, these meshes largely overload the performance and memory capacity of state-of-the-art graphics and computational platforms. A wide variety of simplification methods and dynamic multiresolution models have been proposed to face the problem, but, unfortunately, none of them is able to perform both scalable simplification and interactive view-dependent visualization of very large meshes without imposing a lossy decimation of the original dataset [Lindstrom 2003]. This is mainly because current methods, heavily CPU bound, are unable to generate model updates at full GPU speed and to efficiently communicate them to the graphics hardware through preferential data paths. This CPU/GPU gap is doomed to widen, since CPU processing power grows at a much slower rate than that of the GPU.

The original contribution of this paper is a solution for interactive and accurate visualization of very large surface models on consumer graphics platforms. The underlying idea of the proposed method is to depart from current point- or triangle-based multiresolution models and adopt a patch-based data structure, from which view-dependent conforming mesh representations can be efficiently extracted by combining precomputed patches. Since each patch is itself a mesh composed of a few thousand triangles, the multiresolution extraction cost is amortized over many graphics primitives, and CPU/GPU communication can be optimized to fully exploit the complex memory hierarchy of modern graphics platforms.

The method uses a conformal hierarchy of tetrahedra generated by recursive longest edge bisection to spatially partition the model. Each tetrahedral cell contains a precomputed simplified version of the original model. The representation is constructed off-line during a fine-to-coarse parallel out-of-core simplification of the surface contained in diamonds (sets of tetrahedral cells sharing their longest edge). Appropriate boundary constraints are introduced in the simplification process to ensure that all conforming selective subdivisions of the tetrahedron hierarchy lead to correctly matching surface patches. At run-time, selective refinement queries based on projected error estimation are performed on the external memory

tetrahedron hierarchy to rapidly produce view-dependent continuous mesh representations by combining precomputed patches. The resulting technique, dubbed *Adaptive TetraPuzzles (ATP)* since it heavily builds on the composition properties of conformal hierarchies of tetrahedra, has the following properties: it is fully adaptive and is able to retain all the original topological and geometrical detail even for massive datasets; it 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; it is strongly GPU bound and over one order of magnitude faster than existing adaptive tessellation solutions on current PC platforms, since its patch based structure successfully exploits on-board caching, cache coherent stripification, compressed out of core representation and speculative prefetching for efficient rendering on commodity graphics platforms with limited main memory; high quality simplified representations can be constructed with a distributed out of core simplification algorithm.

As highlighted in the short overview of related work (sec. 2), while certain other algorithms share some of these properties, they typically do not meet the capability of our method in all of the areas. The details of the proposed data structure are presented in section 3, while section 4 describes algorithms for view-dependent refinement and rendering, and section 5 introduces an efficient distributed out-of-core technique for constructing a multiresolution model using a generic high quality simplification algorithm. The efficiency of the approach has been successfully evaluated with a number of large models, including a massive 373 million polygon model of Michelangelo's St. Matthew (section 6).

2 Related Work

Rapidly rendering adaptive representations of large models is a very active research area. In the following, we will discuss the approaches that are most closely related with our work. Readers may refer to recent surveys (e.g., [Chiang et al. 2003]) for further details.

Out-of-core mesh simplification. Various techniques have been presented to face the problem of huge mesh simplification. With the exception of memoryless clustering approaches [Lindstrom 2000; Lindstrom 2003] and stream-based methods [Wu and Kobbelt 2003; Isenburg et al. 2003], most of these techniques, such as Hoppe's hierarchical method for digital terrain management [1998] and the octree based structure OEMM [Cignoni et al. 2003a], are based on some kind of mesh partitioning and subsequent independent simplification. Hoppe hierarchically divides the mesh in blocks, simplifies each block while freezing borders and then traverses the block hierarchy bottom-up by merging sibling cells and again simplifying. In this approach some of the borders remain unchanged until the very last simplification step. OEMM avoids this kind of problem, but it does not build a multiresolution structure. On the other hand, BDAM [Cignoni et al. 2003c] allows both the independent processing of small sub-portions of the whole mesh and the construction of a multiresolution structure, but is limited to height fields. Our work generalizes this approach to arbitrary surfaces, and parallelizes the simplification process in order to efficiently build a multiresolution structure for very large meshes.

View-dependent triangulations. The vast majority of view-dependent simplification methods for general meshes are based on constructing a graph of possible refinement/coarsening operations at the vertex or triangle level. Early methods used edge collapse [Xia and Varshney 1996; Hoppe 1997] or vertex clustering [El-Sana and Varshney 1999; Luebke and Erikson 1997] as primitive operations, and assumed in-core hierarchy construction, limiting their applicability. Few techniques have been presented for both construction and rendering from external memory. Hoppe's hierarchical terrain management method [1998] was an early example, later extended to arbitrary meshes [Prince 2000]. More recently, El-Sana and Chiang [2000] proposed a technique for segmenting the surface and ordering edge collapses to handle block boundaries without explicitly imposed constraints. Both methods have only been tested on models of a few million polygons, and their scalability is unclear. Lindstrom [2003] recently proposed a scheme for out-of-core construction and visualization of multiresolution surfaces based on vertex clustering on a rectilinear octree. While it significantly improves over earlier approaches, it is still unable to retain the fidelity of the original mesh and is heavily CPU bound at rendering time, with a peak performance of 2M triangles/s and asynchronous hierarchy updates at no more than 1 frame/s.

Efficient host-to-graphics communication. All adaptive mesh generation techniques spend a great deal of rendering time to compute the view-dependent triangulation. For this reason, many authors have proposed techniques to alleviate popping effects due to small triangle counts [Cohen-Or and Levanoni 1996; Hoppe 1998] or to amortize construction costs over multiple frames [Duchaineau et al. 1997; Hoppe 1997; Lindstrom 2003]. Our technique reduces instead the per-triangle workload by composing at run-time pre-assembled optimized surface patches. The idea of grouping together sets of triangles in order to alleviate the CPU/GPU bottleneck was presented also in the RUSTIC [Pomeranz 2000], CABTT [Levenberg 2002], and BDAM [Cignoni et al. 2003c] data structures for terrains, and HLOD [Erikson et al. 2001] for general environments. RUSTIC and CABTT are extensions of the ROAM algorithm in which subtrees of the ROAM bintree are cached and reused during rendering. BDAM constructs instead a forest of hierarchies of right triangles, in which each node is a general triangulation of a small surface region. These methods produce adaptive conforming surfaces but are hard to generalize to surfaces with arbitrary topology. HLOD improves instead the classic LOD scene graph by providing multiple precomputed levels of details not only for each model but also for entire subtrees. While related to our method, HLOD focuses on the run-time handling of a large number of small unconnected objects, typical of large CAD assemblies.

Point rendering approaches. An alternative to mesh refinement is to use multi-resolution hierarchies of point primitives to render highly complex scenes in output-sensitive time [Rusinkiewicz and Levoy 2000]. Since current rendering hardware is optimized for triangle rendering, high quality filtered point splatting requires considerable effort, and high visual quality has often to be sacrificed for rendering speed. The latest approaches try to solve this problem by exploiting the programmability features of modern GPUs, leading to impressive peak performances that range from 50M points/second for low quality rendering with unfiltered splats [Dachsbacher et al. 2003] to 10M points/second for high-quality filtering [Botsch and Kobbelt 2003] for in-core models of a few million polygons. By using cache coherent triangle strip primitives, we are able to exceed such rates even for out of core models that are two orders of magnitude larger.

Hierarchies of tetrahedra. In the scientific visualization and finite element literature, much research has been devoted to nested tetrahedral meshes generated by recursive subdivision (see [Cignoni et al. 2003b] for a recent survey). Our data structure is constructed from a recursive partitioning of the input dataset guided by the same regular tetrahedron bisection rule that is often used for modeling and viewing regular 3D grids. We are interested, however, in the space partitioning induced by the multi-level tetrahedralization, rather than in extracting values at mesh vertices as in numerical methods [Maubach 1995] or scientific visualization [Gregorski et al. 2002].

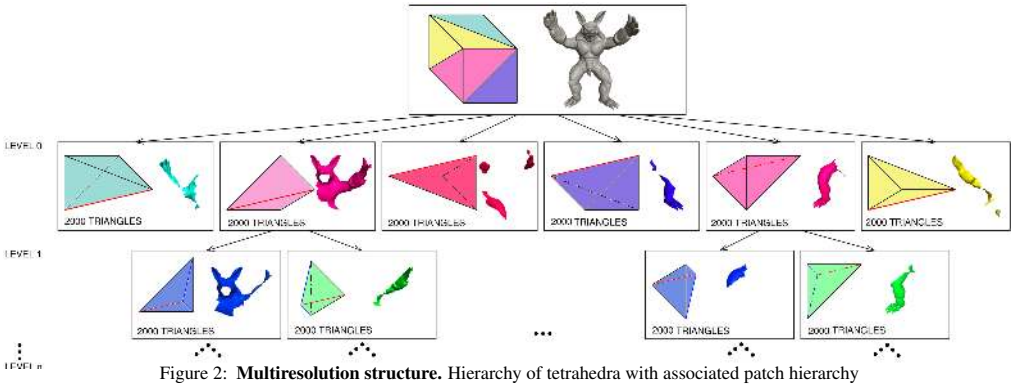
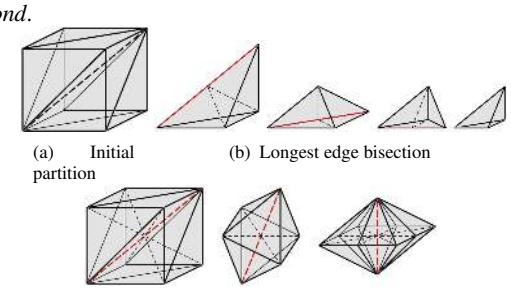


Figure 2: **Multiresolution structure.** Hierarchy of tetrahedra with associated patch hierarchy

3 Multiresolution model

A multiresolution surface model supporting view-dependent rendering must encode the steps performed by a mesh refinement or coarsening process in a compact data structure from which a virtually continuous set of variable-resolution meshes can be efficiently extracted. Our approach 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. The benefits of this approach are that the workload required for a unit refinement/coarsening step is amortized on a large number of triangle primitives, and that the small patches can be optimized off-line for best performance.

To avoid making assumptions on a particular topological genus or subdivision connectivity of the input mesh, we exploit the partitioning induced by a recursive volumetric subdivision of the mesh bounding volume in a *hierarchy of tetrahedra* (see figure 3). The partitioning consists of a binary forest of tetrahedra, whose roots correspond to six tetrahedra around a major box diagonal and whose other nodes are generated by tetrahedron bisection. This operation consists in replacing a tetrahedron σ with the two tetrahedra obtained by splitting σ at the midpoint of its longest edge by the plane passing through such point and the opposite edge in σ. To guarantee that a conforming tetrahedral mesh is always generated after a bisection, all the tetrahedra sharing their longest edge with σ are split at the same time. Such a cluster of tetrahedra is called *diamond*.

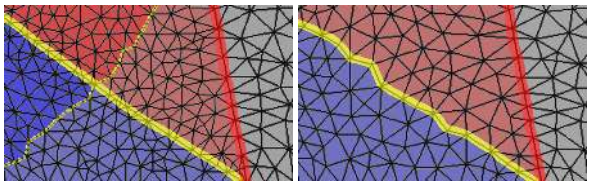


(a) Initial partition (b) Longest edge bisection

(c) Diamond types

Figure 3: **Hierarchy of tetrahedra for space partitioning.** The longest edge is highlighted in red, while next level edges are light-dashed.

The hierarchy of tetrahedra 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 volumetric mesh representations.We exploit this property to construct a level-of-detail structure for the surface of the input model. The basic idea (see figure 2) is to generate from the tetrahedral structure a hierarchy of surface representations. We first partition the input model triangles among the leaf tetrahedra. We then recursively associate to each non-leaf tetrahedron a simplification, up to a given triangle count, of the portion of the mesh contained in its two children, along with all the information required for evaluating view dependent errors.



(a) level i (b) level $i-1$

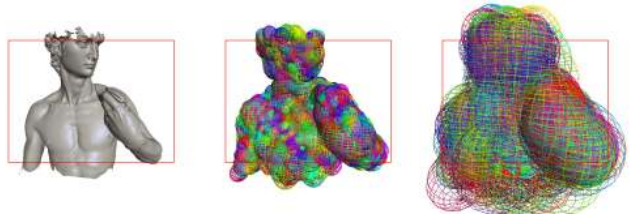
Figure 4: **Generating conforming triangulations.** The four patches at the left of figure 4(a) are part of the same diamond, and are simplified into the two patches at the left of figure 4(b) when coarsening the mesh. The generation of a conforming triangulation is ensured by locking the vertices shared with the neighboring diamond (highlighted in red), and by consistently simplifying the vertices shared by different patches in the diamond (highlighted in yellow).

The above *diamond-by-diamond* property is sufficient for guaranteeing conforming tetrahedral meshes, but, when switching from tetrahedra to the small patches associated to them, the correct connectivity along borders of patches at different simplification levels must be guaranteed by imposing appropriate constraints during simplification. This is efficiently done by carrying out bottom-up construction on a diamond by diamond basis. When building a simplified representation for all tetrahedra in a given diamond, the borders of a patch contained in a tetrahedron are of three possible kinds: (a) *diamond-internal borders*, i.e., borders connecting it with other triangles contained in tetrahedra of the same diamond; (b) *diamond-external borders*, i.e., borders connecting it with triangles contained in tetrahedra of other diamonds; (c) *original borders*, i.e., borders of the original mesh. Since all tetrahedra in a diamond are, by definition, always split/merged at the same time, handling borders of kind (a) just requires that the mesh contained in a diamond is simplified as a single unit, while borders of kind (b) need to be kept fixed to ensure connectivity with all possible neighbors, and borders of type (c) do not need any special handling. The main idea behind these rules is to associate a merge operation of the internal edges of a diamond with the overall simplification of the patch strictly inside the diamond. Thus, when we coarsen our tetrahedral mesh by merging the split-edges inside a diamond, we can safely substitute the patches inside the involved tetrahedra with the ones of the merged diamond: by construction they share the same border and therefore correctly match with the rest of the surface. In this way, we ensure that each mesh composed by a collection of small patches arranged as a correct hierarchy of tetrahedra generates a globally correct surface triangulation (see figure 4). It is worth mentioning that, unlike other hierarchical simplification approaches [Hoppe 1998; Prince 2000], these constraints have little effect on overall simplification quality, since constrained vertices alternate from diamond-internal to diamond-external throughout the hierarchy, and are locked only when in diamond-external state. Moreover, the fact that each diamond is simplified independently can be exploited, see section 5, to design a parallel out-of-core high quality simplification algorithm.

4 View-dependent rendering

The adaptive rendering algorithm is based on a top-down refinement of the tetrahedra hierarchy, designed to fully exploit the rendering capabilities of modern graphics accelerators through batched primitive rendering. Its main components can be considered a generalization of the BDAM approach [Cignoni et al. 2003c] to arbitrary surfaces.

Data organization. Since the algorithm is designed to work in a standalone PC architecture (as opposed to a distributed, network-based solution), we assume that all data is stored locally on a secondary storage unit visible to the rendering engine. Our approach, similarly to recent large scale terrain visualization approaches [Lindstrom and Pascucci 2002; Cignoni et al. 2003c], is based on optimizing the data layout to improve memory coherency and on accessing external memory geometry data through system memory mapping functions, demanding to the operating system the task of loading, when needed, the requested data. To maximize memory locality, we have thus chosen to represent our nested subdivision as a forest of binary trees, and to extract conforming meshes without requiring neighbor finding we employ a saturation technique [Ohlberger and Rumpf 1998]. Therefore, each tree is stored as a memory mapped linear array, and each of its nodes, corresponding to a particular tetrahedron, contains just the following information: a reference to the associated patch data (vertex attributes and connectivity) in a patch repository; the tight bounding sphere and bounding cone of normals for the patch; the saturated model space error and bounding sphere of the neighborhood; the index of child nodes in the linear arrays, which correspond to the two tetrahedra generated by bisection. To minimize the number of page faults, data storage order reflects traversal order. All data in the tree and in the corresponding patch repository is therefore sorted by level, then by geometric proximity, by ordering the nodes in a given level by increasing Morton code [Samet 1990] of their center point. The external size of geometric representation is also reduced by storing each patch in compressed form (see section 5).



(a) (b) (c)

Figure 5: **Refinement and culling.** The model is refined to a screen space error tolerance of one pixel and is culled against the red rectangle. The spheres used for culling are in sub-figure 5(b), while the saturated spheres used for refinement are in figure 5(c)

Refinement algorithm. With this structure, variable resolution rendering is implemented by simple stateless top-down traversals of the binary trees, that combine view-frustum, backface, and contribution culling (see figure 5). As we recurse the hierarchy, we test if the current node is invisible or fully backfacing by checking the tight bounding sphere and cone of normals of the associated patch against the current view volume. If so, we simply stop, culling away the entire branch of the tree. If the node is potentially visible, we test whether its patch is an accurate enough representation by measuring its saturated screen space error. If so, we can render the associated patch, otherwise we continue the recursive refinement with the node's children.

Saturated screen space error is the quantity that guides refinement. We obtain a consistent upper bound by measuring the apparent size of a sphere with diameter equal to the saturated object space errors and centered at the saturated bounding sphere point closest to the viewpoint. The refinement condition, once the point closest to the viewpoint is found, requires only one multiplication to check if the ratio of error sphere radius to distance is larger than the screen space threshold. The extraction of a consistent mesh is ensured by constructing the saturated object space errors/spheres so that they are equal for all the tetrahedra in a diamond and monotonically decreasing when descending the hierarchy. Since the projection to screen space is also monotonic, the diamond composition property ensures that we always extract a conforming volumetric mesh, and our patch construction rules ensure that we generate a globally correct surface triangulation.

Host to graphics processing unit communication. To take advantage of spatial and temporal coherency, it is worth spending time to build an optimal rendering representation for each patch, that can be efficiently reused on subsequent frames, instead of using direct rendering each time. This is the only way to harness the power of current graphics architectures, that heavily rely on extensive on board data caching. We have thus combined our refinement method with a memory manager based on a simple LRU strategy, that explicitly manages graphics board memory, using OpenGL's *Vertex Buffer Objects* extension. Each time we need to render a patch, we reuse the cached version if present, otherwise we render it and cache its representation in place of the oldest one. The transformation from compressed external memory representation to an efficient graphics format happens only at cache faults. The primitive geometric element is a patch composed of multiple triangles, that is heavily optimized during pre-processing using cache-coherent tri-stripping. Since we use an indexed representation, the post-texture-and-lighting cache of current GPUs is fully exploited.

Speculative prefetching. Since the disk is by far the slowest hardware component, we can further hide data access latency by prefetching from compressed external memory the geometry that will soon be accessed. The prefetching routine, that may be executed in parallel to the rendering thread or sequentially as an idle task, executes the same refinement algorithm as the adaptive rendering code, taking as input the predicted camera position instead of the current one. When the refinement terminates, instead of rendering the patches, it simply checks whether the required graphics objects are in pages already in core. If not, it advises the operating system kernel that the pages containing their representation will likely be accessed in the near future and that it would be advantageous to asynchronously read them ahead. This technique is easy to implement on Linux with the `mincore/madvise` system calls. Furthermore, the main rendering needs not to be aware of the prefetching component, and we exploit the extensive performance optimizations of the operating system's virtual memory manager, such as reordering of requests to reduce seek access time and overlapping of computation and disk access.

5 Construction

The off-line component of our method constructs a multiresolution structure starting from a high resolution mesh. As input, we assume that the mesh is represented as a *triangle soup*, i.e., a flat list of triangles with direct vertex information, and that a list of boundary vertices is also available. This representation is commonly employed for out-of-core methods, and can be derived in a I/O efficient way from the more common indexed mesh representation using standard external memory graph techniques [Chiang et al. 1995].

5.1 Mesh Partitioning

The first phase – mesh partitioning – generates a binary forest of tetrahedra, whose roots partition the mesh bounding box and whose leaves contain less than a predefined number of mesh triangles.

The forest is built in a top-down fashion, through recursive insertion of mesh triangles by starting from an initial subdivision of the mesh bounding box into six tetrahedra around a major box diagonal. When a new triangle is inserted, we locate the leaf that contains its center point and, if the number of triangles already contained in it does not exceed the maximum, we simply insert the new one into the associated triangle bucket. Otherwise, we refine the hierarchy by tetrahedron bisection and recursively continue the insertion procedure. Each time a tetrahedron is split, all the triangles in the associated bucket are reassigned to its children by a recursive application of the insertion procedure. The end result is a tetrahedron graph, that describes the subdivision structure using a DAG of diamonds [Pascucci 2002], and a set of triangle buckets associated with leaf tetrahedra that cover the mesh. The graph, rather small since each node typically contains a few thousand triangles, is for efficiency reasons maintained in main memory, while triangle buckets are stored in secondary memory.

This simple partitioning scheme, that clusters triangles solely based on the location of their center point, does not adapt to surface features, and, as for all spatial clustering methods, could lead to patches with exceedingly complex boundaries. Even though we have not found this to be a problem in practice, we plan to improve partitioning in a second pass, that adaptively reassigns triangles among leaf tetrahedra after the first partitioning. This kind of approach demonstrated its efficiency in recent out-of-core simplification methods (e.g., [Shaffer and Garland 2001]).

5.2 Level-of-detail hierarchy construction

The second and final phase – simplification – completes the volumetric structure with a hierarchy of surface representations by recursively associating to each non-leaf tetrahedron a fixed triangle count simplification of the portion of the mesh contained in its two children, along with all the information required for evaluating view dependent errors. This is efficiently done by carrying out bottom-up construction on a diamond by diamond basis. For leaf diamonds, we retain all the fidelity of the original mesh and directly produce an optimized representation for each tetrahedron from the stored triangle buckets. For non-leaf diamonds, we retrieve from the repository the triangles associated to the children of all involved tetrahedra, merge them in a single mesh, that is then simplified so that each involved tetrahedron contains less than a predefined number of triangles. The repository is then updated by erasing the buckets associated to child tetrahedra before saving parent ones.

Diamond simplification. As explained in section 3, to ensure that the patches contained in the diamond's tetrahedra match with all the possible neighbors at different simplification levels, we lock all the vertices on the diamond external boundary. These are easily identified, without maintaining special connectivity information, as the end points of all edges shared by only one triangle of the diamond patch, which were not part of the original model boundary. To avoid introducing new boundary points and simplify bookkeeping, we also constrain original boundary vertices to be removed only by a collapse with another original boundary vertex. Note that any simplification technique can be adopted as long as it allows the choice of the number of triangles for different regions of the output mesh and the specification of vertex constraints. This means for example that, if needed by a particular application, it is possible to exactly preserve the original mesh topology and therefore preserve any existing parametrization. On the other hand, it is also possible to decide to simplify more aggressively by closing holes or clustering vertices. In our case, we have implemented a variation of Hoppe's method for simplifying meshes with appearance attributes [Hoppe 1999], that combines a quadric error metric with regularization penalties for improving sampling regularity and mesh quality in regions of null quadric error. Moreover, to ensure that each tetrahedron in the diamond is simplified to a predefined number of triangles, the simplifier maintains a separate collapse queue for each tetrahedron. We put a edge collapse in the queue of each of the tetrahedrons where it removes a triangle. During iterative simplification, collapses are taken in order of increasing cost from the queues of the tetrahdera that still need to be simplified, and all queues are maintained up-to-date after each simplification step.

Errors and bounds. After simplification, model space errors, bounding spheres and normal cones have to be computed and saturated for each tetrahedron to complete the structure. For errors, we have currently taken the common approach of deriving the model space error ε directly from the quadric metric ε_q (see, e.g., [Lindstrom 2003]). We employ the simple formula $\varepsilon = s\sqrt[3]{\varepsilon_q}$, where s is an empirical scale factor for converting to world units (see, e.g., [Lindstrom 2003]). The scale factor is determined prior to rendering time by finding the smallest value of s leading to no image difference in a fixed number of random views, when setting the screen space tolerance below 1 pixel. Bounding spheres and normal cones, are, instead, computed and saturated using optimal methods from computational geometry for finding the minimum enclosing ball of points (for leafs) and minimum enclosing ball of balls (for all other nodes) [Fischer and Gärtner 2003].

Conversion into final format. Each patch is stored on disk in a compressed representation from which a version optimized for efficient rendering can be rapidly extracted. On current graphics architectures, the most efficient mesh representation is the *cache coherent indexed stripification*. In this representation, vertex attributes are stored in vertex arrays, and triangulation topology is specified with a generalized triangle strip ordered such that vertex cache miss rate is minimized. We greedily compute this ordering by first decomposing the mesh into strips of length approximately equal to vertex cache size, and, then, starting with the strip with the maximum number of border vertices, incrementally adding the other strips, always choosing the one with the minimum cache miss to strip length ratio. For disk storage, topology is compressed using a mesh encoding scheme preserving stripification [Isenburg 2001], while vertex attributes are optionally quantized and entropy encoded. For this paper, that emphasizes the ability of our method to preserve all the original details, we chose a 3x24 bits/position and 32 bit/normal quantization, which correspond to no loss.

Network parallel construction. The hierarchy construction phase dominates, by far, the overall processing cost. The whole process is however 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 a network parallel implementation, the coordinator process traverses all the diamonds bottom up and by geometric proximity, distributes the diamond processing job to a number of workers, which execute it and send the result back to the coordinator for updating the repository and generating output file. Load balancing is ensured if the coordinator initially seeds each worker with a single diamond and subsequently always sends a new job request to the work that sent back a result. The generation of the output file in the correct order can be ensured with a small buffer for handling out-of-sync output requests. 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.

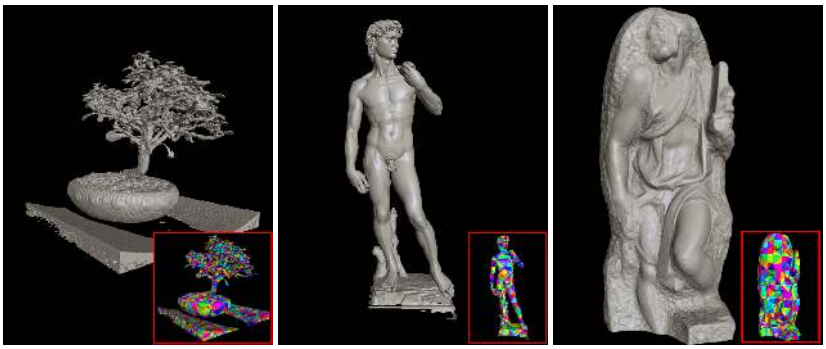


Figure 6: **Test models.** The main images show the models as presented to the user during interactive inspection session, while the inset images illustrate the mesh structure. Left: Bonsai isosurface (6.4M triangles); Middle: David 2mm (8.3M triangles) and 1mm (56M triangles); Right: St. Matthew 0.25mm (373M triangles).

6 Results

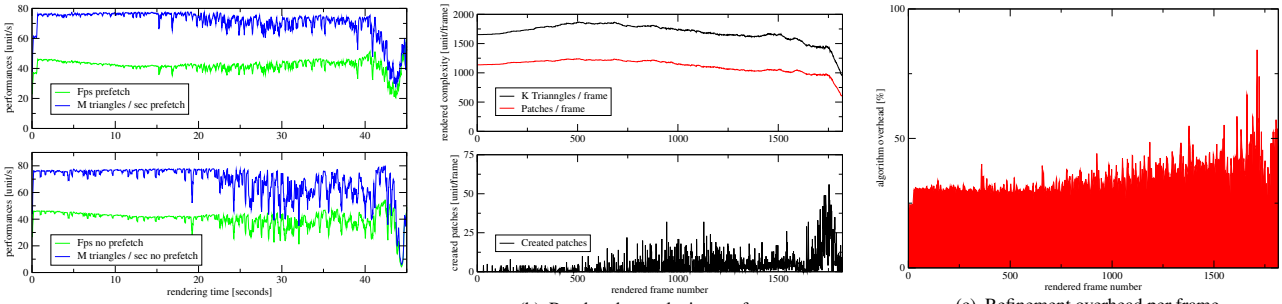
An experimental software library and a rendering application supporting the technique have been implemented on Linux using C++ with OpenGL and the MPICH MPI implementation. We have extensively tested our system with a number of large surface models. The quantitative and qualitative results discussed here are restricted to the freely available models of figure 6.

Model	Triangles	CPU	Partition	Bottom-up construction			Total time	Disk usage (MB)		
				I/O	Process	Total		in	tmp	out
Bonsai	6,317,116	1+1	45	42	1,654	1,696	1,741	289	808	76
		1+4	40	63	435	498	538			
		1+8	41	44	239	283	324			
		1+14	42	63	114	177	219			
David 2mm	8,277,479	1+1	59	93	3,583	3,676	3,735	379	1,059	158
		1+4	60	94	836	930	990			
		1+8	62	144	366	510	572			
		1+14	62	148	216	364	426			
David 1mm	56,230,343	1+1	477	2,335	21,687	24,022	24,499	2,574	6,850	967
		1+4	501	2,299	3,981	6,280	6,781			
		1+8	583	2,086	1,758	3,844	4,427			
		1+14	477	2,238	879	3,117	3,594			
St. Matthew	372,767,445	1+1	3,234	15,722	73,299	89,021	92,255	17,063	41,289	5,887
		1+4	3,400	16,639	19,917	36,556	39,956			
		1+8	3,201	15,876	11,020	26,896	30,097			
		1+14	3,215	16,863	7,712	24,575	27,790			

Table 1: **Numerical results for out-of-core construction.** Tests performed on a network of PCs. All times are in seconds.

Preprocessing. Table 1 lists numerical results for our out-of-core preprocessing method for a number of runs on all the test datasets. The tests were executed on a moderately loaded network of PCs running Linux 2.4. Each PC has two CPU Athlon 2200+ CPUs, 1GB DDR memory, a 70GB ATA 133 hard disk, and a Ethernet 100 Mb/s network connection. We constructed all multiresolution structures with a prescribed maximum leaf size of 4000 triangles/tetrahedron for the partitioning phase and an average non-leaf size of 2000 triangles/tetrahedron for the bottom-up construction phase. To test parallel performance, all tests were repeated with 1, 4, 8, and 14 workers. Overall processing times range from about 3K-4K triangles/s for 1 CPU to 15K-30K triangles/s for 14 CPU. As a point of reference, current state-of-the-art high quality out-of-core simplification methods achieve simplification rates up to 70K triangles/s [Chiang et al. 2003]. Our speed is currently slower. This is mainly because we generate a full multiresolution structure, as opposed to a single small model, and simplification is only a single step of diamond processing, that also includes cache-coherent stripification, mesh compression, and optimal bound computation, each costing as much as simplification. As the number of CPUs increases, construction time, initially dominated by diamond processing, starts to be dominated by raw I/O. The almost linear reduction in processing time shows the efficiency of the distributed approach. The large I/O overhead is due to the repeated access to the temporary triangle repository during bottom-up construction, stored on a slow IDE disk. Similarly to competing methods [Lindstrom 2003], temporary storage size is a constant multiple (roughly a factor of 3) of input size, since we need to store at most the partitioning of the input in the leaves of the tetrahedra hierarchy. Our current version uses a pessimistic fixed-size bucket per leaf, and stores triangles in raw uncompressed form. We anticipate that reducing the size of the repository with compressed dynamically sized buckets would radically decrease I/O cost. Not included in the table is the maximum resident memory usage of the method, which is low and constant for workers (26 MB each) and variable for the master process, due to the in-core graph layout data structure and to disk buffer caches used by the operating system (for a maximum of 280 MB on all runs).

Adaptive rendering. We evaluated the rendering performance of the technique on a number of inspection sequences on all test datasets, using a Linux PC with a Intel Xeon 2.4 GHz, 2GB RAM, a Seagate ST373453LW 70 GB ULTRA SCSI 320 hard drives, AGP 8x and NVIDIA GeForce FX 5800 Ultra graphics. The quantitative results presented here in details were collected during a 45 seconds inspection of the largest model (the 373M triangles reconstruction of Michelangelo's St. Matthew). The session, performed using a window size of 800x600 pixels, hardware full scene antialiasing (4x Gaussian Multisampling), and a screen tolerance of ± 2 pixels (i.e., a refinement epsilon of 4), was designed to be representative of typical mesh inspection tasks and to heavily stress the system, and includes rotations and rapid changes from overall views to extreme close-ups. To further emphasize the quality of the view-dependent simplification, we used glossy material properties and a single off-center positional light placed slightly above and to the right of the camera. The qualitative performance of our adaptive renderer is also illustrated in an accompanying video, that shows live recordings of the analyzed flythrough sequence, and of similar sequences with the other datasets. In all cases, during live sessions there were practically no visible artifacts due to adaptive rendering, since the error measure, even though empirically derived, is very conservative. To fully test our out-of-core data management components, the benchmarks were started with all data off core and disk buffers flushed. During the entire walkthrough, the resident set size of the application is maintained at roughly 144 MB, i.e. less than 3% of out-of-core data size, demostrating the effectiveness of out-of-core data management. Figure 7(a) illustrates the rendering performance of the application, both with and without speculative prefetching. The speculative prefetching version executed an additional refinement step per frame to prefetch pages that are likely to be accessed in the near future, using a linear prediction of camera position with a half a second look-ahead. The prefetching version is smoother, due to the success of prefetching in hiding the latency due to page faults. The only noticeable jitters with the prefetching version (visible in the graph near the end of the sequence) correspond to rapid accelerations of the path while zooming. In the prefetch-



(a) Rendering rates per frame (b) Rendered complexity per frame (c) Refinement overhead per frame

Figure 7: **Rendering Performance Evaluation.** To fully test our out-of-core data management components, benchmarks were started with all data off core and disk buffers flushed.

ing version, we were able to sustain an average rendering rate of over 70 millions of triangles per second, with peaks exceeding 78 millions. By comparison, on similar machines, Lindstrom's [2003] multiresolution vertex clustering method's peak performance was measured at roughly 3 millions of triangles per second, with a representation update latency of up to 1s, even though the model was radically downsampled during preprocessing to only 3M polygons. The increased performance of our approach is due to the larger granularity of the structure, that amortizes structure traversal costs over many graphics primitives, reduces AGP data transfers through on-board memory management and fully exploits the post-transform-and-lighting cache with optimized indexed triangle strips. This favorably compares also with the latest point rendering approaches, that render in-core models of a few million polygons at 10M-50M points/second depending on filtering quality [Dachsbacher et al. 2003; Botsch and Kobbelt 2003]. The overhead of the prefetching and rendering code, measured by repeating the test without executing OpenGL calls, is only about 30% of total frame time (Fig. 7(c)), and is mostly due to external data access (mainly I/O wait and patch decompression). This demonstrates that we are GPU bound even when handling extremely large out-of-core data sets. Rendered scene granularity is illustrated in figure 7(b): even though the peak complexity of the rendered scenes exceeds 1.8M triangles per frame, the number of rendered graphics primitives per frame remains relatively small, never exceeding 1240 patches per frame, which are maintained on graphics memory. Since we are able to render such complex scenes at high frame rates, it is possible to use very small pixel thresholds, virtually eliminating popping artifacts, without the need to resort to costly geomorphing features. Figure 8 is an example of the extremely detailed representation that can be inspected in real-time using our technique.

Figure 8: **David 1mm hand close-up.** Model rendered at ± 1 pixel screen tolerance with 841 patches and 1172K triangles at 50 fps on a 1280x1024 window with 4x Gaussian Multisampling, one positional light and glossy material. Note the very fine geometric and illumination details.

7 Conclusions

We have presented an efficient technique for end-to-end out-of-core construction and view-dependent visualization of very large surface models on commodity graphics platforms. The proposed solution consists in an innovative combination of volumetric subdivision, mesh compression and simplification, out-of-core data management, and batched rendering techniques, resulting in an unprecedented spatiotemporal quality in the interactive rendering of massive models.

Besides improving the proof-of-concept implementation, we plan to extend the presented approach in a number of ways. In particular, we plan to explore more aggressive and lossy compression techniques and to investigate error metrics taking into account the effect of simplification on the shading of the surface. We are also currently incorporating occlusion culling techniques, useful for datasets with a high depth complexity, and we plan to introduce more sophisticated shading/shadowing techniques. Given the performance of the method, we are confident that multi-pass or vertex/fragment program techniques will become readily applicable to gigantic datasets.

The proposed approach, as well as recently introduced techniques such as BDAM [Cignoni et al. 2003c], can be seen as a step in a larger effort of adapting the classic general multiresolution models to modern GPUs, by moving the granularity of the atomic operations of multiresolution models from triangles/points to small surface portions. In our opinion, this is the most promising way to harness the power of current graphics architectures, that heavily rely on extensive on board caching of indexed primitives. Moreover, our results demonstrate that this approach blends also well with other important large model optimizations, such as compression and prefetching techniques. We envision an entire breed of multiresolution mesh rendering algorithms adapted to this kind of mesh representation, and we are currently defining a general framework for experimenting with them.

Acknowledgments. This research is partially supported by the V-PLANET project (EU RTD contract IST-2000-28095). We are grateful to the anonymous reviewers for their thorough work and to the Stanford Graphics Group, the Digital Michelangelo project, and the University of Stuttgart for making benchmark datasets available.

References

BOTSCH, M., AND KOBBELT, L. 2003. High-quality point-based rendering on modern GPUs. In *Proc. Pacific Graphics*, 335–343.

CHIANG, Y.-J., GOODRICH, M. T., GROVE, E. F., TAMASSIA, R., VENGROFF, D. E., AND VITTER, J. S. 1995. External-memory graph algorithms. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, 139–149.

CHIANG, Y.-J., EL-SANA, J., LINDSTROM, P., PAJAROLA, R., AND SILVA, C. T. 2003. Out-of-core algorithms for scientific visualization and computer graphics. IEEE Visualization 2003, Tutorial 4 Course Notes.

CIGNONI, P., MONTANI, C., ROCCHINI, C., AND SCOPIGNO, R. 2003. External memory management and simplification of huge meshes. *IEEE Transactions on Visualization and Computer Graphics 9*, 525–337.

CIGNONI, P., DE FLORIANI, L., PASCUCCI, V., ROSSIGNAC, J., AND SILVA, C. T. 2003. Multiresolution modeling, visualization, and compression of volumetric data. IEEE Visualization 2003, Tutorial 3 Course Notes.

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

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

DACHSBACHER, C., VOGELSGANG, C., AND STAMMINGER, M. 2003. Sequential point trees. In *Proc. SIGGRAPH*, 657–662.

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.

EL-SANA, J., AND CHIANG, Y.-J. 2000. External memory view-dependent simplification. *Computer Graphics Forum 19*, 3 (Aug.), 139–150.

EL-SANA, J., AND VARSHNEY, A. 1999. Generalized view-dependent simplification. *Computer Graphics Forum 18*, 3, 83–94.

ERIKSON, C., MANOCHA, D., AND BAXTER, W. 2001. HLODs for faster display of large static and dynamic environments. In *Proc. ACM Symposium on Interactive 3D Graphics*, 111–120.

FISCHER, K., AND GÄRTNER, B. 2003. The smallest enclosing ball of balls: combinatorial structure and algorithms. In *Proceedings of the nineteenth conference on Computational geometry*, ACM Press, 292–301.

GREGORSKI, B., DUCHAINEAU, M., LINDSTROM, P., PASCUCCI, V., AND JOY, K. I. 2002. Interactive view-dependent rendering of large IsoSurfaces. In *Proc. IEEE Visualization*, 475–484.

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

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

HOPPE, H. 1999. New quadric metric for simplifying meshes with appearance attributes. In *Proceedings of the 10th Annual IEEE Conference on Visualization (VIS-99)*, ACM Press, New York, pages 59–66.

ISENBURG, M., LINDSTROM, P., GUMHOLD, S., AND J.SNOEYINK. 2003. Large mesh simplification using processing sequences. In *Proc. IEEE Visualization*.

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

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

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

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

LINDSTROM, P. 2003. Out-of-core construction and visualization of multiresolution surfaces. In *ACM 2003 Symposium on Interactive 3D Graphics*, 93–102,239.

LUEBKE, D., AND ERIKSON, C. 1997. View-dependent simplification of arbitrary polygonal environments. In *ACM Computer Graphics Proc., Annual Conference Series, (SIGGRAPH 97)*, 199–208.

MAUBACH, J. 1995. Local bisection refinement for *n*-simplicial grids generated by bisection. *SIAM Journal of Scientific Computing 16*, 210–227.

OHLBERGER, M., AND RUMPF, M. 1998. Adaptive projection operators in multiresolution scientific visualization. *IEEE Transactions on Visualization and Computer Graphics 4*, 4, 344–364.

PASCUCCI, V. 2002. Slow growing subdivision (SGS) in any dimension: Towards removing the curse of dimensionality. *Computer Graphics Forum 21*, 3, 451–460.

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

PRINCE, C. 2000. *Progressive Meshes for Large Models of Arbitrary Topology*. Master's thesis, Department of Computer Science and Engineering, University of Washington, Seattle.

RUSINKIEWICZ, S., AND LEVOY, M. 2000. QSplat: A multiresolution point rendering system for large meshes. In *Comp. Graph. Proc., Annual Conf. Series (SIGGRAPH 00)*, ACM Press, 343–352.

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

SHAFFER, E., AND GARLAND, M. 2001. Efficient adaptive simplification of massive meshes. In *Proc. IEEE Visualization 2001*, IEEE Press, 127–134.

WU, J., AND KOBBELT, L. 2003. A stream algorithm for the decimation of massive meshes. In *Proc. Graphics Interface*, 185–192.

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