Batched Multi Triangulation

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

ABSTRACT

The Multi Triangulation framework (MT) is a very general approach for managing adaptive resolution in triangle meshes. The key idea is arranging mesh fragments at different resolution in a Directed Acyclic Graph (DAG) which encodes the dependencies between fragments, thereby encompassing a wide class of multiresolution approaches that use hierarchies or DAGs with predefined topology. On current architectures, the classic MT is however unfit for real-time rendering, since DAG traversal costs vastly dominate raw rendering costs. In this paper, we redesign the MT framework in a GPU friendly fashion, moving its granularity from triangles to precomputed optimized triangle patches. The patches can be conveniently tri-stripped and stored in secondary memory to be loaded on demand, ready to be sent to the GPU using preferential paths. In this manner, central memory only contains the DAG structure and CPU workload becomes negligible. The major contributions of this work are: a new out-of-core multiresolution framework, that, just like the MT, encompasses a wide class of multiresolution structures; a robust and elegant way to build a well conditioned MT DAG by introducing the concept of V -partitions, that can encompass various state of the art multiresolution algorithms; an efficient multithreaded rendering engine and a general subsystem for the external memory processing and simplification of huge meshes.

1 INTRODUCTION

In recent years, the large entertainment and gaming market has resulted in major investments in commodity graphics chip technology, leading to state-of-the-art programmable graphics units (GPUs) with greater complexity and computational density than current CPUs. Despite the already impressive performance of current graphics chips, both architectural considerations and the inherently parallel nature of graphics operations, suggest that this trend will not change, and GPU performance increase will continue to outpace CPU performance increase.

However, our ability to generate models that vastly exceed the peak memory and processing capabilities of even the most powerful hardware is a constant in a number of application domains (see, e.g., 3D scanning [14], geometric modeling [24], and numerical simulation [18]), imposing the need for adaptive techniques.

An important class of large scale 3D models is characterized by an extremely dense sampling, with lots of fine geometric details, accompanied by a moderate depth complexity.

The amount of data contained in these models does not allow us neither to render them directly nor to keep them in the main memory. So we need both a level-of-detail strategy, to filter out as efficiently as possible the data that is not contributing to a particular image, and an out-of-core strategy, to supply efficiently to the insufficient amount of main memory.

These models require both multiresolution techniques, because the graphics architecture cannot sustain such amount of data, and out-of-core techniques, because

the combination of out-of-core data management techniques, for handling datasets larger than main memory, with level-of-detail algorithms based on multiresolution structures, to filter out as efficiently as possible the data that is not contributing to a particular image.

Typical examples of these kind of datasets are terrains and scanned models.

Up until recently, the vast majority of view-dependent level-of-detail methods were based on multiresolution structures taking decisions at the triangle/vertex primitive level. This kind of approaches involves a constant CPU workload for each triangle that with current GPU evolution makes the CPU the bottleneck of the whole rendering process. In other words classical multiresolution approaches are not able to choose what has to be rendered fast enough. Given the current hardware trends, this performance bottleneck is doomed to become more and more evident.

To overcome this bottleneck and to fully exploit the capabilities of current graphics hardware is therefore necessary to select and send batches of geometric primitives to be rendered with just a few CPU instructions. Following this approach, various GPU oriented multiresolution structures have been recently proposed, based on the idea of moving the granularity of the representation from triangles to triangle patches [1, 3, 25]. The benefit of these approaches is that the needed per-triangle workload to extract a multiresolution model reduces by orders of magnitude, the small patches can be preprocessed and optimized off line for a more efficient rendering and highly efficient retained mode graphics calls can be exploited for caching the current adaptive model in AGP or video memory. Recent work has shown that the vast performance increase in CPU/GPU communication results in greatly improved frame rates [1, 3, 25].

It must be said that changing multiresolution granularity reduces the model flexibility: In general, more triangles than necessary will rendered to achieve a given accuracy. On the other hand, the rendering time does not depends linearly on the triangle count anymore. Instead, it is strongly influenced by how the triangles are organized in memory and sent to the graphics card.

With this paper we generalize previous recent approaches by proposing a batched multiresolution framework based on the Multi-Triangulation (MT) [22]. The MT is a very general framework that encompasses a wide class of multiresolution algorithms, but, like the techniques proposed in the 90's, it was originally designed to minimize the number of triangles to be rendered, at the expense of CPU time.

In this paper, we redesign the MT in a GPU friendly fashion, by moving the granularity from triangles to optimized triangle patches, and by redefining the construction and rendering algorithm to work on external memory. As a result, we provide a new out-of-core multiresolution scheme that, just like the MT, encompasses a wide class of construction and view-dependent extraction algorithms and that enables the interactive rendering of massive meshes on commodity platforms. Moreover, we introduce a novel robust technique to build a well conditioned multiresolution data structure, based on $\mathcal{V}$*-Partitions* sequences.

[*]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`

The original contribution of this is twofold: 1) a general multiresolution framework (Sec. 3 4) capable of rendering large meshes at interactive rate that fully exploit GPU capabilities and encompasses existing approaches (Section 5) 2) A general subsystem for handling and modifying massive meshes in external memory, (Section 6) the system can be used for the out-of-core construction of the MT, for the efficient rendering of our multiresolution model, but also usable for general purposes mesh healing and processing.

2 Related work

In this section, we briefly survey some of the extensive previous work on the general subjects of mesh simplification, multiresolution models, and selective refinement; we focus mainly on the aspects most closely related to our work: being able to manage large meshes in external memory and trying to group primitives together.

A common characteristic of most adaptive mesh generation techniques is that they spend a great deal of the rendering time to compute the view-dependent triangulation and to communicate the updates to the graphics board. For this reason, many authors have proposed techniques to alleviate popping effects due to small triangle counts [4, 11] or to amortize construction costs over multiple frames [6, 10, 15], improving feedback frequency at the expense of a (much) higher latency.

Our technique reduces instead the 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. The idea of grouping together sets of triangles in order to alleviate the CPU/GPU bottleneck has already been the focus of a number of approaches.

HLOD [7] improves the classic LOD scene graph by providing multiple precomputed levels of details not only for each model but also for entire subtrees. In this approach, conformality of the triangulations between elements of the partition at different resolutions can be guaranteed only by leaving some of the boundaries unsimplified, with obvious scalability and quality problems. Some approaches simply avoid dealing with this kind of problem, limiting themselves to filling the resulting cracks between patches with ad hoc geometry [9], or moving to a complete mesh-less structure [8, 23]

The first methods capable to producing adaptive conforming surfaces by composing precomputed patches were designed for terrain rendering. RUSTIC [21] and CABTT [13] are extensions of the ROAM [6] algorithm, in which subtrees of the ROAM bintree are cached and reused during rendering. A similar technique is also presented in [5] for generic meshes. BDAM [1] constructs a forest of hierarchies of right triangles, in which each node is a general triangulation of a small surface region. These methods are efficient and crack-free, but are limited to 2.5D datasets.

The first approach able to guarantee an adaptive conforming surface for a massive mesh with an arbitrary topology has been presented in the Adaptive TetraPuzzles approach [3] where, by exploiting a 3D tetrahedral embedding of the well-known right triangle hierarchy, the authors extend the results of the BDAM approach to general 3D meshes. A related approach has been presented in the QuickVDR system [25]: the original massive model is partitioned in a hierarchical set of small patches (called clusters) that are independently converted into progressive meshes and merged bottom-up. Additional logic in the management of boundaries between clusters is used to allow the simplification of some cluster boundaries while enforcing the conformality of the resulting mesh. It should be noted that this approach generates a DAG of dependencies between clusters, and thus the whole structure can be considered a particular case of the our batched MT framework. Similarly, TetraPuzzles can also be considered a particular case of the MT framework, in which all dependencies are implicitly encoded in the tetrahedral hierarchy.

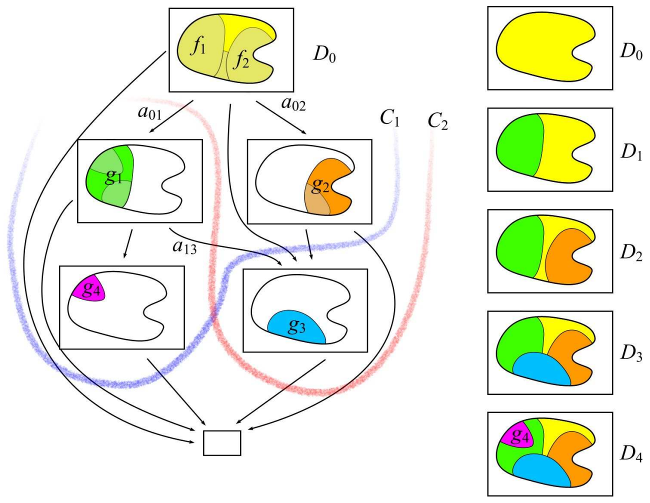


Figure 1: An example of the MT DAG that shows the one-to-one correspondence between the valid subsequences and the valid cuts. Note that the cut C_2 (rendered in red) is not valid because two arc of the cut, a_{01} and a_{13}, are in the same path from the source to the sink.

3 Re-designing the MT for massive meshes

The Multi Triangulation framework (MT)[22] was designed as a general way to formalize and implement multiresolution models based on simplicial complexes, but its basic concepts hold in a more general context. In the following description we summarize the MT framework abstracting from the way the domain is represented and we give conditions that should be satisfied for generating MT whose DAG is not ill-conditioned and can guarantee good performances and flexible adaptivity during extraction.

3.1 The MT Framework on general domains

Let us consider a domain Ω as the subset of $\mathbb{R}^3$ corresponding to the surface to be represented. With the general term *description* we denote all the primitives that can be used to describe this domain, e.g. triangulated surfaces, point sets, parametric surface etc..

Let Ω be our domain and D a description of Ω. The operation of replacing a portion f of D with a new description g, provided that both f and g describe the same part of the domain, is called *pasting* and it is formally written as: $D \oplus g = D \setminus f \cup g$ where f is called *floor* of g and g is called *fragment*.

A general simplification or refinement algorithm can be expressed as the iteration of the following steps, starting with $D_0 = D$ and $i = 0$:

- select a region $f_{i+1} \subseteq D_i$;
- construct a new fragment g_{i+1} s.t. $\Omega(f_{i+1}) = \Omega(g_{i+1})$;
- update: $D_{i+1} = D_i \oplus g_{i+1}$

If the fragment g_{i+1} is a description more accurate than its floor, then this is a refinement algorithm, otherwise it is a simplification algorithm.

When speaking about descriptions represented by a triangulation T we say that, after a pasting operation, T is *conforming*, or in other words correct, if for any pair of triangles in T their intersection is either empty or it is coincident with a vertex or an edge of both triangles.

Note that g_{i+1} replaces its floor, which in turn could have been introduced by previous pasting operations. Therefore the floor of g_{i+1} will be, in general, distributed among several fragments pasted before g_{i+1}. Referring to Figure 1, the floor of g_3 is distributed among D_0, g_1 and g_2 while the floor of g_4 is all contained in g_1. We will refer to this property saying that a fragment g_i *depends* on the fragments intersecting its floor; e.g., referring to Fig. 1, g_4 depends on g_3 while g_3 depends on g_1, D_0 and g_2.

Now consider the whole sequence of pasting operations produced by the sequences of fragments $S = (g_1, \dots, g_n)$ and the corresponding description $D_n = (((D_0 \oplus g_1) \oplus g_2) \oplus \dots) \oplus g_n$. Observe that a pasting $D_i \oplus g_{i+1}$ can be done if and only if D_i contains the floor of g_{i+1}, in other words if S contains all fragments on which g_{i+1} depends. This means that if we take a subsequence of $S' \subseteq S$ such that for each fragment in S' all the fragments on which it depends are also in the subsequence (transitive closure of dependency), then S' is also a sequence of valid pasting that will produce a new representation, possibly different from $D_i \forall i = 1..n$. In the example $D_0 \oplus g_2$, $D_0 \oplus g_1$ and $(D_1 \oplus g_1) \oplus g_4$ are all valid descriptions of the domain.

In the MT framework dependencies between fragments represent a partial ordering and can be encoded in a directed acyclic graph (DAG) where fragments are the nodes. A closed subset (with respect to transitive closure of dependency) of the nodes of the DAG, corresponds to a valid sequence, and it is conveniently encoded with a *cut* on the DAG, i.e. the set of arcs leaving the specified portion of the DAG.

Note that, for completeness, all the leaf nodes containing portions that are not floor of any fragments are connected through an arc to a dummy sink node, so there is no node without leaving arcs except the dummy node, which is never included in a cut.

Let a_{ij} be the intersection between the floor of g_j and g_i (see Fig. 2). Then, for each arc (g_i, g_j), a_{ij} represents the part of the description that is replaced by pasting g_j in a subsequence that contains g_i. This means that, given a cut on the DAG, $\cup_{k \in cut} a_k$ corresponds to the result of pasting all the fragments in the corresponding subsequence. This is decisive in terms of efficiency because it means that we never need to actually compute all the pasting operations of a given subsequence, we can obtain the correct result by simply combining together all the descriptions associated with the arcs leaving the nodes included by the DAG.

Updating a Cut Once you have a valid cut, corresponding to a description that fits your multiresolution needs, it is possible to update the current representation by means of *refinement/coarsening* operations over the cut itself. A *refinement* consists of replacing an arc in the cut with the forward star of its head node. Consider the cut represented in Figure 1 and suppose to execute *Refine* on the arc a_{23}. It means to replace it with the arcs in the forward star of g_3. Since g_3 depends on g_1 and g_1 is outside the current cut, we need to refine arc a_{01} as well. In other words the nodes in the backward star of a newly inserted node are recursively visited to ensure that they enter the subsequence. Similarly, *coarsening* an arc means to replace the forward star of its tail node with its backward star. Note that this operation is *legal* only if all the arcs in the forward star of the tail node are also in the cut. *Refinement/coarsening* operations allow to continuously adapt the representation error to the current application needs.

In the original MT implementation, that uses edge contraction as primitive operations, to move the cut forward a node, which requires the execution of several instructions, merely means to replace 8–10 triangles, i.e., the region of influence of an edge. Our goal is to spend this time for replacing order of thousands triangles. This requires the definition of primitive operations that work on larger mesh regions. These operations cannot be arbitrary, but have to fulfill a number of conditions in order to guarantee good performance and local adaptivity during extraction.

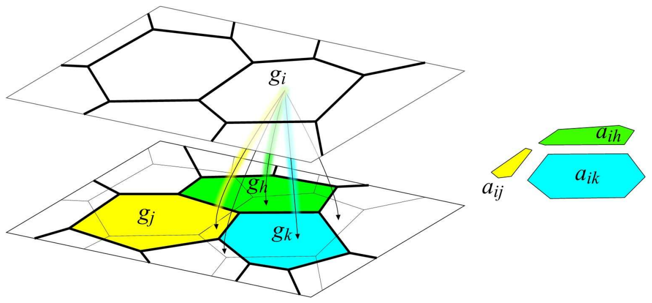


Figure 2: In a MT we associate to each arc (g_i, g_j) of the DAG the description a_{ij} replaced by pasting g_j in a subsequence that contains g_i.

3.2 Well behaving DAG's

Selecting a new region f to be substituted in the MT building algorithm, means to create a new node of the DAG and the set of arcs corresponding to its backward star. So how we select such a region strongly affects the topology of the DAG that influences the efficiency of the process of extracting a description from the MT.

In the following we will show two worst case examples to explain the characteristic that the DAG should fulfill. In Figure 3 top, a DAG is derived from a series of refinement steps where the floor of the fragment g_{i+1} overlaps the fragment g_i. As a result, if the arc a_4 is refined, in order to provide more detail in the region g_4, then all the other arcs have to be refined as well, even if they are related to distant regions of the domain. In other words, the error does not increase smoothly as the distance from g_4 increase, but the extracted representation is in fact all at the lowest error. A similar case happens if the floor of a fragment intersects too many other fragments, i.e. if the backward star of a node is too big (see Figure 3 bottom).

The following two conditions ensure that a DAG is well conditioned:

1) the length of a path connecting the root to any leaf is logarithmic in the number of nodes and

2) the diameter of a fragment decreases geometrically with the distance of the corresponding node from the root of the DAG.

In the case of meshes the *pasting* operation implies that the old region f and the new g must have the same boundary, so in the simplification or refinement algorithm the boundary $(closure(D - f) \cap f)$ must be preserved. It is important to note that if many fragments share the same boundary, there will be no simplification along this border. We must then add a third condition:

3) a fragment should not share boundaries with all the fragments intersecting its floor.

Various multiresolution schemes have been presented in literature that fulfills the above requirements, like for example the right triangle hierarchies exploited in the BDAM approach [1]. In the next section we propose a new general scheme for building MT with well behaving DAG's which satisfies the conditions stated above.

4 The $\mathcal{V}$-Partition Multiresolution Model

The techniques presented in the previous sections allow us to manage and simplify a massive mesh with a patch-wise approach. In this section we introduce *$\mathcal{V}$-Partitions*, a general scheme for the definition of a sequence of coarser and coarser partitions over a

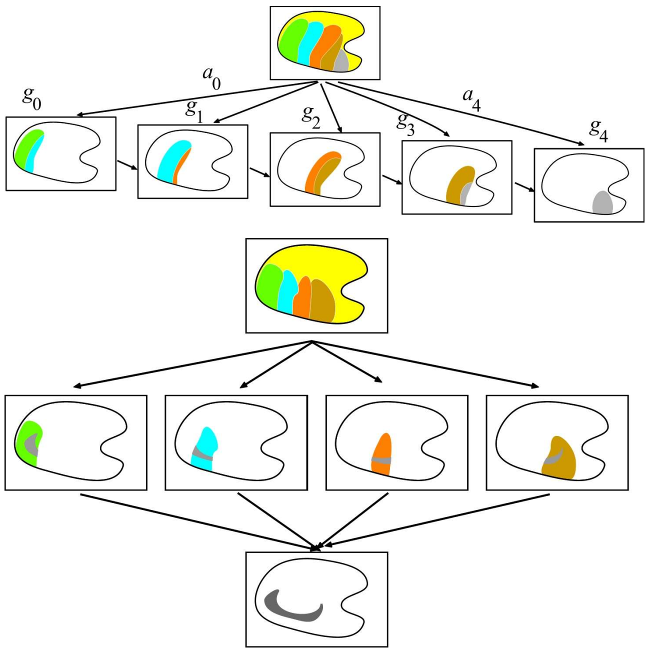
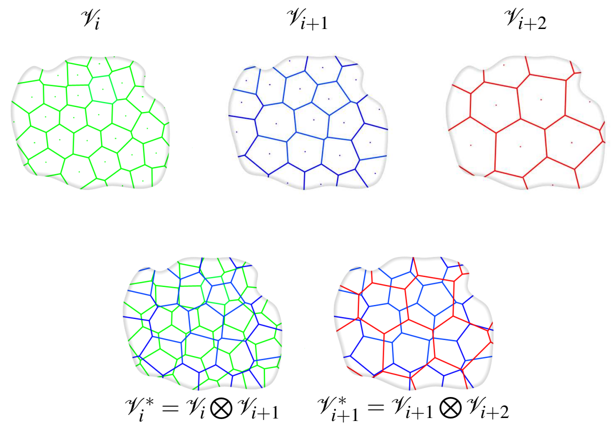


Figure 3: Two examples of malconditioned DAGs.



$\mathcal{V}_i$
$\mathcal{V}_{i+1}$
$\mathcal{V}_{i+2}$
$\mathcal{V}_i^* = \mathcal{V}_i \bigotimes \mathcal{V}_{i+1}$
$\mathcal{V}_{i+1}^* = \mathcal{V}_{i+1} \bigotimes \mathcal{V}_{i+2}$

Figure 4: Top: three consecutive partitions of a sequence, bottom: The two set of patches used for the simplification step obtained by intersection of two consecutive $\mathcal{V}$-partitions $\mathcal{V}_i^*, \mathcal{V}_{i+1}^*$

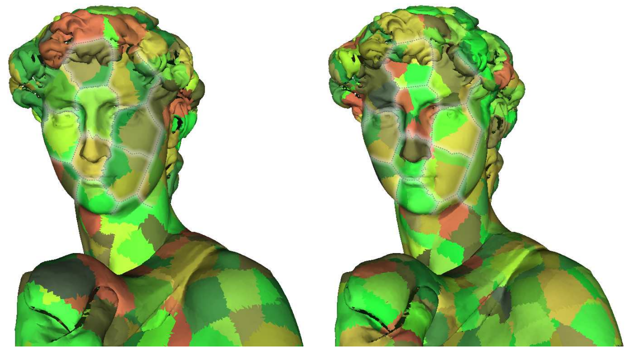


Figure 5: Two consecutive $\mathcal{V}$-partitions $\mathcal{V}_i^*, \mathcal{V}_{i+1}^*$ of the David mesh. The enhanced lines show some of the borders of the partition $\mathcal{V}_{i+1}$ that remains unchanged between the two steps.

massive mesh that we will use to define sets of patches, that can be simplified and merged together to form a well-behaving MT DAG.

Let Ψ be a partition of the space into $k = |\Psi|$ disjoint regions $\Psi = \{\psi_1, \ldots, \psi_k\}$. Given a rule that uniquely assigns a triangle t to a region of space, e.g. the one where the barycenter of t falls, it can be applied to a mesh T to subdivide it into a set of k conforming triangulations $\Psi(T) = \{T_1^\Psi, \ldots, T_k^\Psi\}$. Given two partitions $\Psi = \{\psi_i\}, \Phi = \{\phi_j\}$, we denote with $\Psi \bigotimes \Phi$ the partition resulting by the *crossing* of the two partitions, defined as:

$$\Psi \bigotimes \Phi = \bigcup_{i=0..|\Psi|, j=0..|\Phi|} \{\psi_i \cap \phi_j\}$$

informally speaking $\Psi \bigotimes \Phi$ is the partition that you obtain by overlaying the two partitions.

Proposition 1 *Consider a sequence of coarser and coarser partitions $\Psi_0, \ldots, \Psi_n$ and the sequence of partitions obtained by crossing them $\Psi_i^* = \Psi_i \bigotimes \Psi_{i+1}$. You can assemble the elements of the Ψ_{k+1} partitions in two different ways by using either the elements of Ψ_k^* or the elements of Ψ_{k+1}^*.*

This property is the central point of our multiresolution approach: coarser partitions corresponds to coarser mesh resolutions and we use the elements of the various Ψ_i^* partitions to assemble conforming triangulations with varying resolution according to the MT rules explained in Sec. 3.

The key idea is that we perform the simplification process at discrete steps: one for each Ψ_i^* partition. To ensure that we obtain conforming triangulations we have to take some care: when simplifying from step i to step $i+1$, first we assemble the patches of Ψ_i^* to build patches of Ψ_{i+1}, then we simplify them without touching the borders of Ψ_{i+1} patches and finally we split the result of simplification according to the Ψ_{i+1}^* partition. Assuming that the partitions in the sequence are coarser and coarser, we exploit the simplification step to keep the *density* of triangles contained in each element of the partition as close to constant as possible.

This approach is illustrated in Figure 4 top, that shows three consecutive partitions $\mathcal{V}_i, \mathcal{V}_{i+1}, \mathcal{V}_{i+2}$, and figure 4 bottom, that shows the two partitions $\mathcal{V}_i^*, \mathcal{V}_{i+1}^*$ resulting, respectively, from $\mathcal{V}_i^* = \mathcal{V}_i \bigotimes \mathcal{V}_{i+1}$ and $\mathcal{V}_{i+1}^* = \mathcal{V}_{i+1} \bigotimes \mathcal{V}_{i+2}$.

As an example let us describe the i-th simplification step according to figures 4: we start with our mesh that is partitioned according to $\mathcal{V}_i^*$ and we consider the all patches of $\mathcal{V}_i^*$ corresponding to a single region of $\mathcal{V}_{i+1}$, i.e. the blue ones in Fig 4; this mesh portion is independently simplified, keeping the blue border unchanged. At the end of the simplification, we save this simplified mesh portion (that corresponds to a blue patch) split according to the patches of the new partition $\mathcal{V}_{i+1}^*$ (the red lines of Fig. 4). Informally speaking patch borders with the same color always match. Once all the regions of $\mathcal{V}_i$ have been processed, the mesh is partitioned according to $\mathcal{V}_{i+1}^*$, and we can thus start the next simplification step by processing all the $\mathcal{V}_{i+2}$ regions.

4.1 Partition Sequences and MT

The simplification process sketched above can be directly interpreted in terms of local operations and fragments. In a simplification step i we perform a set of $|\mathcal{V}_{i+1}|$ local actions substituting each $\mathcal{V}_{i+1}$ patch partitioned according to $\mathcal{V}_i^*$ with the a patch with the same border but a simplified interior and partitioned according to $\mathcal{V}_{i+1}^*$. In terms of MT the $\mathcal{V}_{i+1}$ patch, partitioned according $\mathcal{V}_{i+1}^*$, is a fragment whose floor is the same patch but partitioned according to $\mathcal{V}_i^*$.

A DAG built using a *partition sequence* where the number of the elements in each partition decreases geometrically will satisfy condition 1. To satisfy condition 2 we need that the partition elements are distributed uniformly. In such a case the floor of each fragment of $\mathcal{V}_{i+1}$ intersects a roughly constant number of fragments of $\mathcal{V}_i$ and therefore the size of the backward star of a node will be roughly constant.

4.2 Building Partition Sequences using $\mathcal{V}$-partitions

In practice we need to find an effective sequence of partitions that we can use to build our patch based MT. The sequence of partitions must be roughly uniform and coarser and coarser. It should be noted that the sequence of partitions does not need to adapt to the geometric characteristics of the mesh (like curvature or density). In this approach the adaptivity is handled during the MT traversal. If a portion of the mesh presents more feature its simplification will yield an higher error and therefore during the MT traversal that portion will be maintained at a finer resolution.

We propose to build the partitioning using a Voronoi like approach. Given a set of 3D points $Q = \{v_0, \dots, v_k\}$, called seed set, we define the $\mathcal{V}$-partition of a mesh T into patches $\mathcal{V}_Q = \{Q_0^T, \dots, Q_k^T\}$ by defining Q_i^T as the patch composed by the faces that are nearest to the seed point v_i. Note that it is not required for these patches to be composed of a single connected component.

To build the multiresolution model, we need a sequence of seed sets $Q_0, \dots Q_k$, and the corresponding $\mathcal{V}$-partitions $\mathcal{V}_0 \dots \mathcal{V}_k$, of decreasing granularity.

We propose two approaches for building the sequence of seed sets, the first one generates a regular partitioning while the second one generates a sequence of irregular partitions.

4.2.1 Regular $\mathcal{V}$-partitioning

A simple and effective method is to use a regular recursive seed distribution scheme. Consider the two dimensional case, illustrated in figure 6: we start by placing vertices on a regular grid, obtaining a partition in squares, then we continue placing seeds on the midpoint of the edges of these squares obtaining another finer partition in squares (tilted 45 degree), and so on. With this approach the $\mathcal{V}_i^*$ partitions forms the well known triangle bintree hierarchy, and the simplification strategy that we obtain is quite similar to the one used by the BDAM approach [1]. This approach can directly be extended to the three-dimensional case by considering a regular grid and placing seeds onto a) cube centers, b) face centers, c) edge centers. The sequence of partitions $\mathcal{V}_i$ that we obtain (where Voronoi regions are cubes and octahedra) forms the same patterns of the *diamonds* of the Slow Growing Subdivision scheme [20] used also in the recent *TetraPuzzle*[3] approach, but the sequence of *crossed* partitions $\mathcal{V}_i^*$ is not a simplicial complex, but it is formed by convex cells built by adjacent tetrahedra. Many other recursive 2D subdivision schemas can be obtained by regular seeds placement, like for example hexagonal subdivisions [12] shown in Fig. 7 (also described as *dual* $\sqrt{3}$ *subdivisions* in [19]).

4.2.2 Irregular $\mathcal{V}$-partitioning

Beside the above technique here we present a simple approach for finding a sequence of seed sets by an I/O efficient sampling the original surface able to manage huge surfaces. We assume that the number of seeds is much smaller that the original surface and can reasonably be kept in core. Note that assuming patches of $\approx 1k$ triangles, this means that massive meshes of more than one giga triangles can reasonably be managed.

We start by choosing a average radius of the patch r and then we sequentially scan the surface triangles, adding the barycenter of the triangle t to the seed set Q every time a triangle t of the stream is farther than r from all the other points of Q. Then, in a second

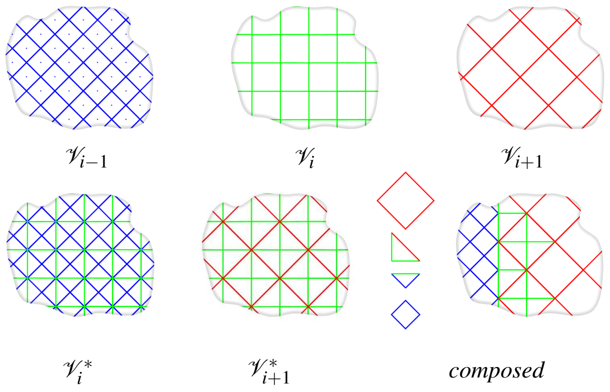


Figure 6: The sequence of partions obtained placing seeds on centers and corners of a square grid generate the well known bintree hierarchy used for 2d terrain multiresolution models.

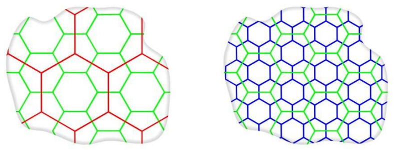


Figure 7: The sequence of partitions obtained recursively placing seeds on vertexes of a hexagonal grid.

sequential scan, we remove small patches and we apply a step of Lloyd's Voronoi relaxation [16] by moving the seeds towards the barycenter of their region. A sequence of coarser and coarser partitions can be obtained by simply choosing a sequence of increasing radii $r_0, \dots r_i$. Since the process of scanning the whole mesh is the dominant one the creation of all the various seed sets Q_i can be done in parallel during the same mesh scan.

At the end of this process we have a sequence of seed sets Q_i that subdivide the original surface into coarser and coarser partitions $\mathcal{V}_i$ where cells, within each partition, have approximatively the same number of triangles. This is done by decreasing the triangle count, from a level to the next, by the same ratio as the number of seeds. These partitions will be used, as described above, to build the $\mathcal{V}_i^*$ partitions that identify the patches at the basis of our multiresolution approach.

5 Out of core, time critical rendering

For the sake of interactivity the multiresolution extraction process should be able to support a constant 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 algorithm uses two threads: *ExtractRender* and *PatchServer*. The ExtractRender thread is responsible for finding the correct cut in the DAG and for filling a *OperationList*; this list contains the needed coarsening/refinements of the cut, e.g. the patches that must be removed/inserted from the current description. The *PatchServer thread* is responsible for loading in main memory the needed patches without blocking the ExtractRender thread.

The Extraction thread keeps updated the current cut 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*

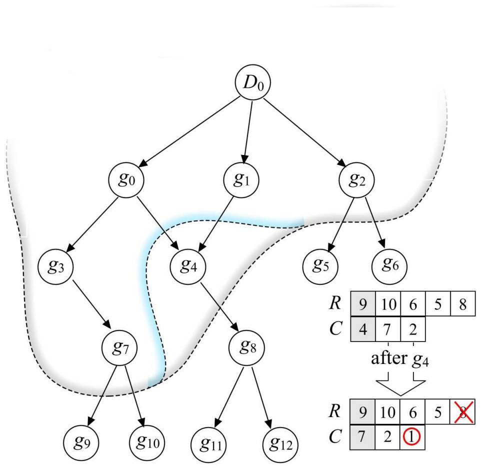


Figure 8: Example of cut and corresponding heaps.

and the *RefinementHeap*.

The priority in the heaps is given by the screen space error associated with the operation: the first operation in the RefinementHeap is the feasible refinement with the largest screen space error, while the first in the CoarseningHeap is the coarsening with the smallest error that can be done on the current cut while maintaining the desired screen space error.

The algorithm performs refinement operations until possible, and coarsening operations otherwise. Whenever an operation is done, new operations will possibly be inserted in the heaps.

Figure 8 shows a cut example and the corresponding heaps. For example, if operation C_4 (move the cut before the node g_4) is performed C_1 is inserted in the RefinementHeap and R_8 is invalidated, since is no more feasible. Note that the inverse operation R_4 would now be feasible, but it would never been chosen and hence is not even inserted.

Figure 9 shows the the algorithm ExtractRender thread. The basic block of the algorithm is the estimation of the time needed to perform an operation, implemented in the function *CheckBudget*.

The time for the ExtractRender thread is the time for rendering, which is estimated as linearly proportional to the number of rendered triangles. Thus, for each operation (P_{out}, P_{in}), we update the number of rendered triangles RT with $RT = RT + |P_{in}| - |P_{out}|$. If RT exceeds the maximum number of triangles the operation cannot be done.

Similarly, the PatchServer thread time is dominated by the time needed to load the patches from the disc. In the worst case, loading time is dominated by disk seek latency, that we assume bounded by a constant found by experiment.

The extraction algorithm always tries to apply the refinement with the lowest error within the current budget; when the budget does not allows it, it tries to apply the coarsening with the greatest error. The function *AddToOperationsList* inserts the operation passed as argument in *OperationList*, which will be read by the *PatchServer*. Once *Extraction* is ended, *OperationsList* contains the list of operations that can be done. The procedure *Apply* simply runs through this list and performs the operations. To perform an operation means to free the memory allocated for the patches to be removed and to have the patches to be sent to the GPU in main memory, which can require a loading from disk if they are not already present. If the budget time ends before the whole list is scanned, which may happen if the time estimation was optimistic, then the algorithm returns and is guaranteed that the representation is conformal even if not to the required accuracy.

```
ExtractRender() {
  Extract();
  Apply();
  Prefetch();
}

Extract() {
do {
    op = null;
    if( CheckBudget(RefinementHeap.root ) )
        op =pop(RefinementHeap.root);
    else
        if( CheckBudget( CoarseningHeap.root ) )
            op =pop(CoarseningHeap.root);

    if(op!=null){
        AddToOperationsList(op);
        UpdateBudgets(op);
        }
    }
    while( ( render_budget > 0 ) &&
                   (load_budget > 0) && (op!=null))
}

Apply() {
    while((clock() < endFrame) &&  (!OperationsList.empty() ) ) {
    op = RemoveFirst(OperationsList);
    Perform(op);
    }
}

Prefetch(){
 if ( clock() < endFrame ) {
       execute Perform on the operations in the heaps
       RefinementHeap and CoarseningHeap
    }
}
```

Figure 9: Interruptible rendering cycle.

If there is still time after the execution of *Apply*, the remaining time is used for pre-fetching the patches that will be probably used in the next frame. At this stage, the pre-fetching strategy is very simple and it consists of loading in memory the patches *around* the current cut, which is easily achieved by applying all the operations in the heaps.

5.1 View space and object space errors

In order to obtain a view-dependent multiresolution representation where the mesh resolution adapts with current viewing needs, we need a view-dependent measure of the error. The screen space error associated with a patch is derived at run time in a way similar to [1] using an object-space view-independent error measure and the bounding sphere of each patch. This object-space view-independent value is projected in screen space to obtain the error using the bounding sphere. The relation between the error of the arcs of the graph is preserved by imposing that each bounding sphere encloses the bounding sphere of all the arcs in the subgraph.

Common measures used to quantify the error of a single patch are based on the Hausdorff distance between simplified and original mesh [1, 25, 3]. We have chosen a simpler strategy: we just use the average edge length of the triangle of the patch. We made this choice on the basis of the following considerations: the initial meshes are dense and we use an highly accurate simplification algorithm which produce roughly uniform meshes; this means that the average edge length is monotonic along the levels of the DAG, as opposed to the Hausdorff distance based error which often needs to be corrected in order to respect the partial order among the node of the DAG: transform the average length of a triangle edge view space (in pixels units) and, given the fine tessellations created, the visual fidelity is no more dominate by the geometric error but by the surface shading, as observed in [17]. These considerations were confirmed using the geometric error computed during simplification with no noticeable difference.

Model	triangle number	time	size ply	size MT
Lucy	28,055,742	54m	520	854
David 1mm	56,230,343	97m	1,154	1,640
S.Matthew 0.25mm	372,767,445	357m	7,611	11,600

Table 1: Numerical results for the construction of the MT (in minutes) the time are relative to a small cluster of 4 pc's.

6 A SUBSYSTEM TO HANDLE MASSIVE MESHES

Our framework requires an out-of-core mesh manager that allows us to handle massive meshes patch by patch, reflecting the MT concept of local modification in terms of patches instead of triangles. The steps for the construction of the MT are: select a set of patches (the floor of a local modification e.g. all the patches composing an element of the partition), load and modify the triangle mesh associated with the patches, change the patch structure defining a new set of patches, and then save them.

Therefore we store the whole mesh as a collection of independent sub-meshes called *patches*. Each disk-stored *patch* contains an indexed representation of a small portion of the mesh with a copy of all the referenced vertices. *Boundary vertices*, that are shared between patches, are replicated but identified for easier patch processing. For each patch p we maintain the list L of all the vertices that have external dependencies. L entries are triplets (v_p, q, v_q), denoting, for each vertex v_p in p, the patch q that refers to it and its position v_q inside q.

When a set of patches P is requested for being processed and modified in-core, we exploit these lists to efficiently unify vertex indexes and to mark the vertices that are referred by not loaded patches. When the in-core mesh portion has to be written back, the user can change the mesh partitioning scheme, defining a new set of patches that covers the same mesh portion. In this case, we also update the boundary lists of the patches that are not loaded but referring to vertices in the current portion P.

Once you have a partition sequence, starting from the finer partition we have to load patches in memory, to simplify them and effectively build the whole MT structure. Moreover with this scheme it is rather simple to perform out of core mesh healing processes like smoothing and small hole filling.

Note that this approach it is somewhat independent from how patches are actually stored. This allow to use the same structure also for rendering purposes, just changing the final format of stored patches. In this case for sake of rendering efficiency we can store patches as optimized triangle strips, with precomputed normals and then individually compressed using a quick decompression algorithm [1].

7 RESULTS AND DISCUSSION

The results presented in this section relate three dense meshes of increasing size: the Lucy (28M tri), the Michelangelo's David (56M tri) and Michelangelo's S.Matthew (370M tri), all of them coming from the Stanford repository.

7.1 Preprocessing

The preprocessing was done on a cluster of PC's on an Ethernet 10Mb and 100Mb moderately loaded. The network speed resulted unimportant since the computation is dominated by the CPU time to perform patch simplification, analogously to what reported by QDVR and Tetrapuzzles.

[1]we used the *minilzo* compression library available at `http://www.oberhumer.com/`.

Since the number of triangles is halved at each iteration the number of triangles contained in the whole dataset is almost twice than the number of original triangles, while, as can be seen in Table 1, the disk occupation of the MT is roughly 40% percent higher than the original dataset. This should not be surprising, since the original mesh is in a raw format while all the MT patches are stripified. These numbers are comparable to QVDR [25] which requires 13.992GB for the same S.Matthew model, and Adaptive Tetrapuzzles [3] (5.887 GB).

7.2 Rendering

The rendering performance was evaluated over several inspections, rotating and abruptly zooming in and out the model. All the tests were done with window size 800x600 on a Windows machine equipped with an AMD Athlon 64, 2 GHz, 512 MB Ram, SCSI hard disk, bus AGP 8x and graphics card GeForce 6800 GT. In all cases the in core memory limit to store all the patches was set to approx. 90 MB. In the case of the S.Matthew model, this is less that 1% of the total data size, showing the effectiveness of the out of core data management strategy.

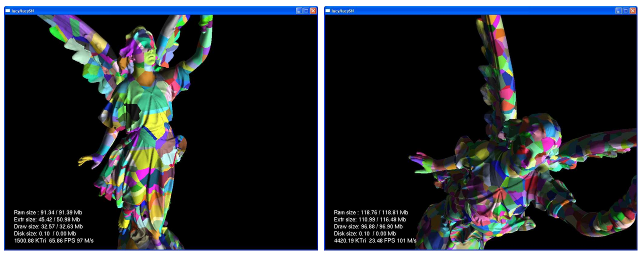
Our algorithm is able to render around 4M triangles per frame at 35 fps with a pixel precision, computed as the average length of the triangles projected onto the screen. The Adaptive Tetrapuzzles approach sustains an average rendering rate of 70 millions of triangles per second on a Linux equipped pc with a GeForce FX 5800 Ultra Graphics, which should produce results comparable to our method on the same hardware setting. QVDR, instead, sustains 771k triangles per frame at 17fps on a GeForce Ultra FX 5950 GPU, but also implements occlusion culling which is not strictly necessary for the kind of meshes discussed in the paper.

8 CONCLUSION AND FUTURE WORK

We have presented a GPU friendly multiresolution framework providing high visual quality as well as efficient rendering. 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 are efficiently extracted and batched to the GPU by simply combining precomputed patches.

Our main contributions are: a general framework for building efficient out-of-core multiresolution models that fully exploit the capabilities of current consumer graphics hardware; a general out-of-core patch-based mesh management system on which the framework can be efficiently implemented; a parallel out-of-core, high quality, simplification algorithm; a proof-of-concept implementations of novel multiresolution models that produce well conditioned multiresolution structures and fit in the above framework.

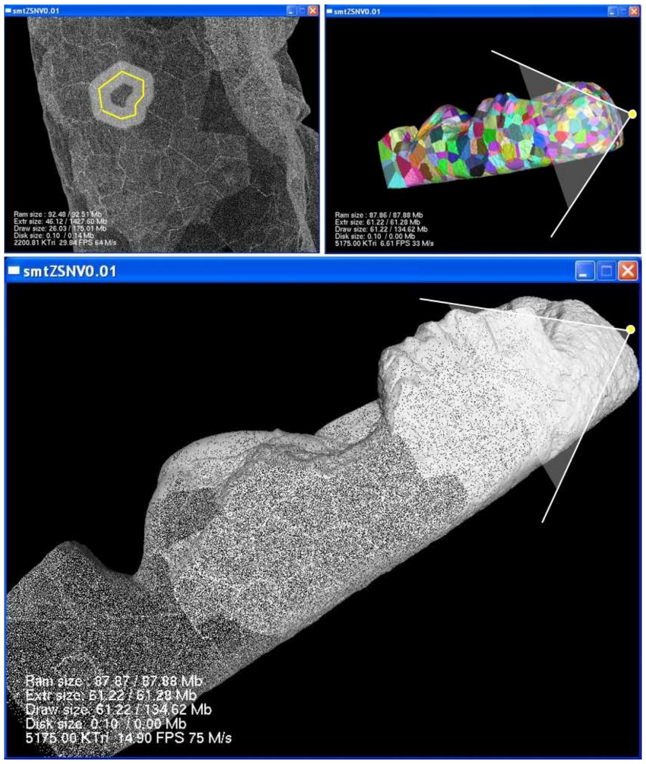
The implementation of the framework proves it comparable, in terms of speed and data preservation, to the ad-hoc state-of-the-art clustered multiresolution models it generalizes.



```
Ram size : 91.34 / 91.39 Mb
Extr size: 45.42 / 50.98 Mb
Draw size: 32.57 / 32.63 Mb
Disk size: 0.10  / 0.00 Mb
1500.88 KTri  65.86 FPS 97 M/s
```

```
Ram size : 118.76 / 118.81 Mb
Extr size: 110.99 / 116.48 Mb
Draw size: 96.88 / 96.90 Mb
Disk size: 0.10  / 0.00 Mb
4420.19 KTri  23.48 FPS 101 M/s
```

Figure 10: The Lucy model adaptively rendered with the batched MT.



```
Ram size : 92.48 / 92.51 Mb
Extr size: 46.12 / 1427.60 Mb
Draw size: 26.03 / 175.01 Mb
Disk size: 0.10  / 0.14 Mb
2200.81 KTri  29.84 FPS 64 M/s
```

```
Ram size : 87.86 / 87.88 Mb
Extr size: 61.22 / 61.28 Mb
Draw size: 61.22 / 134.62 Mb
Disk size: 0.10  / 0.00 Mb
5175.00 KTri  6.61 FPS 33 M/s
```

```
Ram size : 87.87 / 87.88 Mb
Extr size: 61.22 / 61.28 Mb
Draw size: 61.22 / 134.62 Mb
Disk size: 0.10  / 0.00 Mb
5175.00 KTri  14.90 FPS 75 M/s
```

Figure 11: The S. Matthew model adaptively rendered with the batched MT.

Although the current implementation gives satisfactory results, there are still issues that will require further work. The first is the incorporation of an ad-hoc speculative pre-fetching of patches, while at the present which patches are pre-fetched is not bound to the camera movement. A second, more intriguing, goal is to efficiently handle the color information. At the present state, color coordinates can be assigned per-vertex basis and of course any technique to preserve this information during the simplification process (for example [2]) can be adopted. Nonetheless, it is foreseeable that an ad hoc solution complying the characteristics of our framework is needed.

References

[1] P. Cignoni, F. Ganovelli, E. Gobbetti, F.Marton, F. Ponchio, and R. Scopigno. BDAM: Batched dynamic adaptive meshes for high performance terrain visualization. *Computer Graphics Forum*, 22(3):505–514, Sept. 2003.

[2] P. Cignoni, C. Montani, C. Rocchini, R. Scopigno, and M. Tarini. Preserving attribute values on simplified meshes by re-sampling detail textures. Technical Report IEI-B4-37-12-98, IEI – C.N.R., Pisa, Italy, Dic. 1998.

[3] Paolo Cignoni, Fabio Ganovelli, Enrico Gobbetti, Fabio Marton, Federico Ponchio, and Roberto Scopigno. Adaptive tetrapuzzles: efficient out-of-core construction and visualization of gigantic multiresolution polygonal models. *ACM Trans. Graph.*, 23(3):796–803, 2004.

[4] Daniel Cohen-Or and Yishay Levanoni. Temporal continuity of levels of detail in delaunay triangulated terrain. In *IEEE Visualization '96*. IEEE, October 1996. ISBN 0-89791-864-9.

[5] Christopher DeCoro and Renato Pajarola. Xfastmesh: fast view-dependent meshing from external memory. In *VIS '02: Proceedings of the conference on Visualization '02*, pages 363–370, Washington, DC, USA, 2002. IEEE Computer Society.

[6] M.A. Duchaineau, M. Wolinsky, D.E. Sigeti, M.C. Miller, C. Aldrich, and M.B. Mineev-Weinstein. ROAMing terrain: Real-time optimally adapting meshes. In *Proceedings IEEE Visualization '97*, pages 81–88. IEEE, October 1997.

[7] Carl Erikson, Dinesh Manocha, and William V. Baxter III. HLODs for faster display of large static and dynamic environments. In *Proc. SIGGRAPH*, pages 111–120, 2001.

[8] E. Gobbetti and F. Marton. Layered point clouds – a simple and efficient multiresolution structure for distributing and rendering gigantic point-sampled models. *Computers & Graphics*, 28(6), December 2004.

[9] M. Guthe, P. Borodin, Á. Balázs, and R. Klein. Real-time appearance preserving out-of-core rendering with shadows. In A. Keller and H. W. Jensen, editors, *Rendering Techniques 2004 (Proceedings of Eurographics Symposium on Rendering)*, pages 69–79 + 409. Eurographics Association, June 2004.

[10] H. Hoppe. View-dependent refinement of progressive meshes. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 189–198. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7.

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

[12] Ioannis Ivrissimtzis, Malcolm Sabin, and Neil Dodgson. On the support of recursive subdivision. *ACM Transactions on Graphics*, 23(4):1043–1060, 2004.

[13] Joshua Levenberg. Fast view-dependent level-of-detail rendering using cached geometry. In *Proceedings IEEE Visualization '02*, pages 259–266. IEEE, Oct 2002.

[14] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M.Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, and D. Fulk. The digital michelangelo project: 3D scanning of large statues. In *Siggraph 2000,pages 131–144*.

[15] Peter Lindstrom. Out-of-core construction and visualization of multiresolution surfaces. In *ACM 2003 Symposium on Interactive 3D Graphics*, pages 93–102,239, April 2003.

[16] S. Lloyd. Least squares quantization in pcm. *Information Theory, IEEE Transactions on*, 28(2):129–137, 1982.

[17] Frank Losasso and Hugues Hoppe. Geometry clipmaps: terrain rendering using nested regular grids. *ACM Transactions on Graphics*, 23(3):769–776, August 2004.

[18] A. A. Mirin, R. H. Cohen, B. C. Curtis, W. P. Dannevik, A. M. Dimits, M. A. Duchaineau, D. E. Eliason, D. R. Schikore, S. E. Anderson, D. H. Porter, P. R. Woodward, L. J. Shieh, and S. W. White. Very high resolution simulation of compressible turbulence on the IBM-SP system. In *Supercomputing '99*. .

[19] Peter Oswald and Peter Schroeder. Composite primal/dual $\sqrt{3}$-subdivision schemes. *Comput. Aided Geom. Des.*, 20(3):135–164, 2003.

[20] Valerio Pascucci. Slow growing subdivision (sgs) in any dimension: Towards removing the curse of dimensionality. *Computer Graphics Forum*, 21(3):451–460, September 2002.

[21] Alex A. Pomeranz. Roam using surface triangle clusters (rustic). Master's thesis, University of California at Davis, 2000.

[22] E. Puppo. Variable resolution terrain surfaces. In *Proceedings Eight Canadian Conference on Computational Geometry, Ottawa, Canada*, pages 202–210, August 12-15 1996.

[23] S. Rusinkiewicz and M. Levoy. QSplat: A multiresolution point rendering system for large meshes. In *Comp. Graph. Proc., Annual Conf. Series (SIGGRAPH 00)*, pages 343–352. ACM Press, July 24-28 2000.

[24] Gokul Varadhan and Dinesh Manocha. Out-of-Core rendering of massive geometric datasets. In Robert Moorhead, Markus Gross, and Kenneth I. Joy, editors, *Proc. of the 13th IEEE Visualization 2002 Conference (VIS-02)*, pages 69–76. IEEE Computer Society, October 2002.

[25] Sung-Eui Yoon, Brian Salomon, Russell Gayle, and Dinesh Manocha. Quick-vdr: Interactive view-dependent rendering of massive models. In *VIS '04: Proceedings of the IEEE Visualization 2004 (VIS'04)*, pages 131–138. IEEE Computer Society, 2004.