View-dependent Tetrahedral Meshing and Rendering using Arbitrary Segments

Ralf Sondershaus
WSI / GRIS
University of Tübingen, Germany
sondershaus@gris.uni-tuebingen.de

Wolfgang Straßer
WSI / GRIS
University of Tübingen, Germany
strasser@gris.uni-tuebingen.de

ABSTRACT

We present a meshing and rendering framework for tetrahedral meshes that constructs a multi-resolution representation and uses this representation to adapt the mesh to rendering parameters. The mesh is partitioned into several segments which are simplified independently. A multi-resolution representation is constructed by merging simplified segments and again simplifying the merged segments. We end up with a (binary) hierarchy of segments whose parent nodes are the simplified versions of their children nodes. We show how the segments of arbitrary levels can be connected efficiently such that the mesh can be adapted fast to rendering parameters at run time. This hierarchy is stored on disc and segments are swapped into the main memory as needed. Our algorithm ensures that the adapted mesh can always be treated like a not-segmented mesh from outside and thus can be used by any renderer. We demonstrate a segmentation technnique that is based on an octree although the multi-resolution representation itself does not rely on any paticular segmentation technique.

Keywords: multi-resolution meshes, tetrahedral meshes, view-dependent rendering, volume rendering

1 INTRODUCTION

Tetrahedral meshes are often used as finite element meshes that discretize a volumetric domain for scientific simulations like computational fluid dynamics (CFD). Modern simulation environments typically use meshes that contain millions of tetrahedra.

The simulations carry data along with a tetrahedral mesh. The data values are usually scalar values like temperature or pressure, or vector values like velocity, and can be attached to the vertices, edges, border faces or to the tetrahedra.

The emerging need to visualize the simulation data has introduced tetrahedral meshes to volume visualization. Modern algorithms render up to about one million of projected tetrahedra per second [KQE04] or can extract isosurfaces of about two millions of tetrahedra per second [KSE04]. This is not enough to render a model like the F16 interactively (figure 8) that contains about 6 million tetrahedra.

In this paper, we present an *out-of-core data structure* that enables such a big mesh to be simplified with a small memory footprint. The data structure is built on segments which are swapped efficiently to and from the core memory as needed. Additionally, we introduce a *multi-resolution framework* which is built on a hierarchy of segments and can be used for *interactive volume rendering*.

Our out-of-core data structure partitions the mesh into segments by merging leaves of an octree. The segmentation adapts to the details of the mesh and thereby creates segments which contain similiar number of vertices (and tetrahedra). Subsequent algorithms (like simplifiers) process the mesh one segment after the other and need small memory footprints. As a by-product, the vertices are reordered as they are assigned to the segments which results in a better performance of a subsequent algorithm because it tends to reduce cache-misses.

Based on the out-of-core data structure, we introduce a multi-resolution framework. Many previously published multi-resolution frameworks work with a hierarchy of vertices and decide per frame which vertex is to be split or which edge is to be collapsed in order to refine or coarsen the mesh. Because a huge mesh can contain millions of points, the adaptation of the mesh can be become time-consuming even if priority queues are used. We construct a hierarchy of segments where segments of arbitrary resolution levels can connect to each other (however, they need to share the same border vertices). The hierarchy of segments is used at run-time to adjust the mesh to viewing parameters and performs better than a vertex hierarchy because fewer nodes must be tested to be refined or coarsened and no dependencies between nodes are needed.

For triangle meshes, the big issue is to connect different segments correctly if the segments belong to different resolution levels. Many previous algorithms store dependencies between segments in a directed acyclic graph or restrict the resolution levels to differ by at

Journal of WSCG, ISSN 1213-6972, Vol.14, 2006
Plzen, Czech Republic.
Copyright UNION Agency – Science Press

most one between adjacent segments in order to ensure correct connections between segments. Our model can connect arbitrary resolution levels by fixing the borders between segments.

For tetrahedral meshes, the adjacency information for every tetrahedron must additionally be set in order for subsequent algorithms (like MPVO sorting in volume rendering) to work correctly. Our multi-resolution framework introduces a so-called 0-segment which handles the adaptation of the adjacency information efficiently. The volume renderer always sees just one consistent mesh and can run its sorting and rendering routines as if no multi-resolution mesh is used.

Our multi-resolution model does not depend directly on how the mesh is segmented but can work on any segmentation. Instead of the octree-based structure, other techniques like vertex clustering could be used. Furthermore, we do not rely on edge collapse based simplifications but could use any simplification technique.

2 RELATED WORK

Simplification, multi-resolution, and out-of-core techniques are described shortly because they are important to our technique. A small overview to volume rendering techniques is given.

Simplification. We restrict our overview to approaches that are based on edge collapses. For other simplification techniques we refer to mesh decimation [RO96] or TetFusion [CM02].

Popovic et al. [PH97] have extended the Progressive Mesh approach [Hop96] to general simplicial complexes but do not take into account how the underlying scalar field of a tetrahedral mesh is approximated.

Staadt et al. [SG98] have applied the Progressive Mesh approach to tetrahedral meshes. The edge collapses are sorted by an error heap that uses a cost function which considers various errors like scalar field error or volume and shape deformation.

Cignoni et al. [CCM+00] have characterized the field and domain errors of an edge collapse and present various techniques to predict these errors reliably. The field error is introduced by approximating the original scalar field of the mesh whereas the domain error is introduced by reducing the boundary of the mesh.

Kraus et al. [KE00] have simplified non-convex meshes and can change the topology of the mesh during simplification. Chiang [CL03] have preserved the topological structure of isosurfaces of the mesh during simplification.

Garland [GZ05] extended the quadric error metrics to arbitrary simplices (and to tetrahedra in particular) and showed that this approach produces high-quality approximations that automatically take domain and field errors into account.

Out-of-Core Data Structures. Cignoni et al. [CMRS03] use an octree to partition a large tetrahedral (or triangle) mesh into segments. Each segment can be modified independently of the other segments. Simplification algorithms are adapted to process the mesh on a per-segment basis.

Gumhold et al. [GI03] construct a segmentation of a huge triangle mesh by sorting the vertices into a regular grid and merging grid cells into segments that contain approximatively the same number of vertices. Triangles are sorted into the segments according to their center points.

Isenburg et al. [IL05] introduced streaming meshes which are defined as a new file format that interleaves triangle and vertex definitions and does not introduce a vertex until it is indexed by a triangle. Furthermore, it is marked if no subsequent triangle indexes a vertex anymore such that this vertex can be safely deleted. Streaming meshes are highly efficient to both mesh compression and mesh simplification [VCL+05].

Multi-resolution representations. Many multi-resolution representations [CMRS03, DDFM+04, CDFL+04] construct a binary vertex hierarchy by edge collapses. At run-time, a front through the hierarchy defines a valid mesh. Vertices on the front can be split (which refines the mesh) or collapsed (which coarsens the mesh). In order for the mesh to be valid, not all splits or collapses are valid. Geometric and topological conditions need to be checked and the operation is allowed only if the conditions are fulfilled. These conditions can be enforced by performing additional operations which results in additional costs.

Cignoni et al. [CGG+04] use longest-edge bisection of a tetrahedral mesh in order to decompose the spatial domain of a huge polygonal mesh and to construct a hierarchical decomposition of this polygonal mesh. The mesh segments that are contained in a tetrahedra diamond are simplified leaving the border vertices unchanged, i.e. those vertices that connect different segments. Using longest-edge bisection results in a hierarchy which ensures that neighboring segments can differ in at most one resolution level.

Volume Visualization. A standard technique for direct volume rendering of unstuctured tetrahedral meshes is the projected tetrahedra algorithm of [ST90] that has been greatly enhanced by the pre-integration technique of [MHC90, RKE00].

The tetrahedra are sorted from back to front which can be done for all acyclic tetrahedral meshes and because a tetrahedron is always convex. The tetrahedra are projected onto the view plane and decomposed into 1 - 4 triangles according to the positions of the projected vertices. These triangles are rendered using alpha blending from back to front.

3 OUT-OF-CORE DATA STRUCTURE

We design a data structure that is suited to handle large tetrahedral meshes memory efficiently as well as to support volume rendering at run-time. Therefore, the mesh is partitioned into segments such that the segments are stored on disc, can be loaded independently to main memory and can be written back to disc.

Every segment contains a number of vertices (such that a vertex of the original mesh belongs to exactly one segment) and a number of tetrahedra (a tetrahedra of the original mesh belongs to exactly one segment).

The index of a vertex consists of an index-pair (s_i, l_i) with a segment index s_i and a local index l_i which specifies the vertex within the segment s_i. We encode this index-pair as a bit field of 32 bits. A tetrahedron is also addressed as an index-pair with l_i as the local index of the tetrahedron.

Because all vertices and tetrahedra need to be addressable with this address space, the number of vertices and tetrahedra within a segment should be balanced.

The multi-resolution model (section 5) adds new segments to the mesh that are simplified versions of the original segments. Every simplified segment has an error associated with it such that the volume renderer can compute which segment is to be swapped in and out from the main memory. In order to achieve a good error estimation for each segment, we need segments that contain vertices with similiar attribute values and tetrahedra with similiar sizes.

Using these objectives as a starting point, we construct the segmentation as follows.

3.1 Construction

Although we could use the techiques of [CMRS03] or [GI03], we found both only partly applicable to our models. An octree partitions the vertices of a mesh fast and robust. Its leaves reflect the density of the points in the mesh which can be significantly different in different areas of a tetrahedral mesh (for an example see figure 8, left). But the number of vertices that are sorted into cells can differ highly such that some cells contain almost no vertices whereas other contain many. This leads to segments of different sizes which can result in memory and address space defraction.

A grid has the disadvantage that the user must specify its resolution and that dependending on the resolution the variation of the number of vertices of the grid cells differ highly. But it is easy to be implemented out-of-core. It is mandatory to combine cells afterwards to form segments in order to obtain balanced segments. For tetrahedral meshes, the size of the grid cells needs to be very small in order to catch the fine-detailed areas of the mesh which results in a huge number of cells that contain data.

We combine both approaches. First, an octree is constructed for the vertices of the mesh. Afterwards, the leaves of the octree are considered as nodes in a (undirected) graph. The edges of the graph reflect the neighborhood of the leaves but contain edges between cells only that share a face and have similiar sizes (we restrict the size difference to be at most two). This ensures that areas of the mesh with a similiar point density are merged into one segment because the size of an octree leaf reflects the point density of the mesh.

A graph partitioning algorithm (we use Metis 4.0) is called which constructs a partition of the graph. Every partition forms now a segment and consists of a collection of leaves of the octree. Every leaf belongs to exactly one segment.

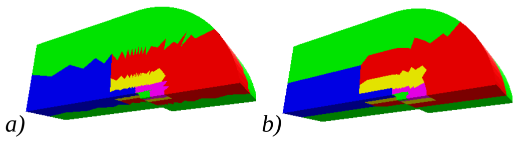

Figure 1: *Segmentations that sort the tetrahedra (a) according to their center vertices into the segmentation, or (b) according to the smallest segment index of their four vertices.*

Afterwards, we sort the tetrahedra into the segments as follows. The four vertices of a tetrahedron are assigned to their segments. The tetrahedron is assigned to the segment of the vertex with the smallest segment index. We do not need to compute the center of the segment (as it is done by [CMRS03] and [GI03]) and found this method to produce well-shaped borders that are sufficient for our purposes, see figures 1 and 2.

The user specifies the average number of vertices that are to be stored in each segment. The octree is constructed to contain at most this number of vertices in its leaves. The graph matching combines leaves into segments such that a balanced segmentation is achieved.

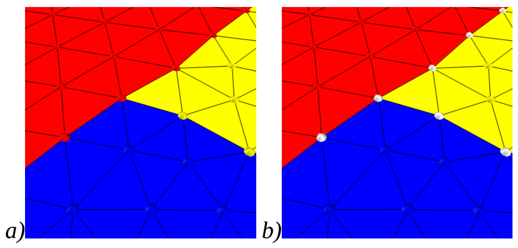

Figure 2: *(a) A tetrahedron is sorted to the segment of its index with the smallest segment index. Red is smaller than yellow which is smaller than blue. Note that a tetrahedron may reference vertices from other segments. (b) The white vertices on the boundary belong to the* 0*-segment as described in section 5.*

Because subsequent algorithms process the mesh by traversing the segments one after another, the segments need to be stored in an order that neighboring segments are traversed together. Therefore, we sort the segments by their minimal points, first in x-, followed by y- and by z-coordinates. More elaborated techniques could be applied here.

The segments are stored in a single file on disc in the order of the sortation. For every segment, we store

1. The geometry of the vertices using the local ordering of the vertices within the segment.
2. The attributes of the vertices (if any).
3. For every tetrahedron its four vertex index-pairs.
4. The attributes of the tetrahedra (if any).
5. For every tetrahedron its four adjacent tetrahedra.
6. The indices of all segments that are incident to this segment (i.e. share at least one vertex).

In order to traverse the tetrahedra of the mesh, we store adjacency indices (number 5 above). Every tetrahedron t stores one index-triple (s_i, t_i, c_i) for each of its vertex index-pairs that points to the tetrahedron that is opposite to the vertex. The index-triple encodes the segment index s_i, the local index t_i (within the segment s_i) and a code $c_i \in \{0,1,2,3\}$. The code specifies the vertex inside the adjacent tetrahedron that is opposite to the shared face.

The interface of the data structures allows for loading a particular segment as well as to request a single vertex or tetrahedron such that the segment that this vertex belongs to is automatically loaded. A Last-Recently-Used queue keeps track of all loaded segments and stores segments that have been changed back to disc if they are not needed any more.

4 SIMPLIFICATION

Using our data structure, the simplifier traverses the mesh one segment after the other and simplifies each segment. Our simplifier uses edge collapses and is steered by a priority queue that sorts all possible collapses of a segment by the error that they introduce. The error is evaluated by using quadric error metrics of Garland et al. [GZ05].

A quadric error metric measures the squared geometric and attribute distances of points to hyperplanes that are spanned by the (original) tetrahedra. A special penalty error can be introduced at boundary vertices in order to preserve the boundary of the mesh well.

The vertices $\mathbf{v} = (v_x, v_y, v_z)$ of the mesh and their attributes $f_{i,v}$ are embedded into a $3+k$ dimensional space with $\mathbf{p} = p(v_x, v_y, v_z, f_{1,v}, \ldots, f_{k,v})$. For every tetrahedron $t = (\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3)$, a local coordinate system $\mathbf{e}_0$, $\mathbf{e}_1$, $\mathbf{e}_2$ is constructed by orthonormalizing the vectors $\mathbf{p}_1 - \mathbf{p}_0$, $\mathbf{p}_2 - \mathbf{p}_0$, and $\mathbf{p}_3 - \mathbf{p}_0$.

Then, the quadric error function Q of a tetrahedron can be written with a symmetric, positive semi-definite matrix $\mathbf{A}$, a vector $\mathbf{b}$, and a scalar value c as

$$Q(\mathbf{x}) = \mathbf{x}^T \mathbf{A} \mathbf{x} - 2\mathbf{b}^T \mathbf{x} + c$$

with

$$\mathbf{A} = \mathbf{I} - \sum_{i=0}^{2} \mathbf{e}_i \mathbf{e}_i^T \qquad \mathbf{b} = \mathbf{A}\mathbf{p} \qquad c = \mathbf{p}^T \mathbf{A} \mathbf{p}$$

where $\mathbf{p}$ is the barycenter of the tetrahedron.

The quadric error of a vertex approximates the sum of the squared distances to its incident hyperplanes and can be computed by summing all matrices $\mathbf{A}$ component-wise as well as all vectors $\mathbf{b}$ and all scalars c of the incident tetrahedra.

For an edge collapse, the metrics (i.e. $\mathbf{A}$, $\mathbf{b}$, and c) of both collapsing vertices are summed component-wise. The point that minimizes this quadratic error function can be found by solving the linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$. Because $\mathbf{A}$ is symmetric and positive semi-definite, a cg solver can be used that takes the middle point of the collapsing edge as starting point.

5 MULTI-RESOLUTION MODEL

The data structure and the simplifier presented so far can be used to simplify a tetrahedral mesh as a whole. But for view-dependent rendering the mesh needs to be adapted to viewing parameters and thus different simplification (or resolution) levels need to be merged into one mesh at run-time.

Therefore, we build a hierarchy of segments where each segment can connect to its neighbors regardless of their resolution levels. We first describe how the hierarchy is constructed and show after that how the adjacency information between segments can be updated.

5.1 Construction

The segmented tetrahedral mesh is stored on disc as described above. We create new segments to build up a hierarchy as follows and append the new segments at the end of the file. Each segment stores all the information described in section 3.

First, we copy each segment of the original mesh and simplify the copies (as described above) independently. The vertices that are referenced by different segments remain unchanged (i.e. the vertices on the boundary between at least two segments). We end up with a coarse approximation of each segment where the segments are still connected at the original resolution level. The simplified copies are inserted as parents of their original segments into the hierarchy, see figure 6.

Secondly, we construct an undirected graph whose nodes are the (simplified) segments and whose edges are between every pair of (simplified) segments that share at least one tetrahedral face. The nodes are weighted by the number of vertices in their segments. We find a matching of the graph using a greedy algorithm and end up with pairs of nodes (segments).

Each pair is merged into a new segment and the new segment is simplified. The vertices that have connected

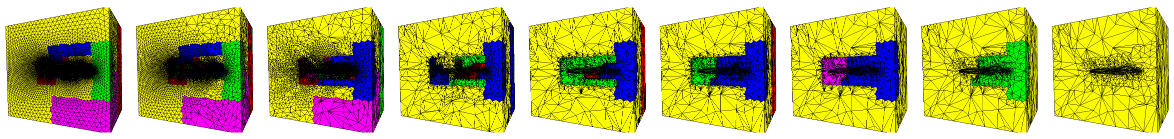

Figure 3: *The construction process of the NASA fighter dataset merges two segments into a new segment which is simplified. The colors of the segments can change from picture to picture.*

both segments belong to the new segment now and are simplified. The vertices that are on the boundaries between the new segments remain unchanged as shown in figure 4.

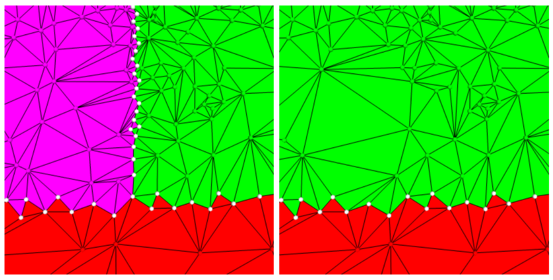

Figure 4: *The purple and the green segment are merged and simplified. The shared vertices are removed but the vertices to the red segment are left unchanged (white vertices).*

Iterating this strategy leads to the algorithm in figure 5. Every iteration constructs the neighboring graph and computes pairs of segments. Both segments of a pair are merged into a new segment which is simplified. The hierarchy is constructed by building a (at most levels binary) tree where the new segment is the parent of both merged segments as shown in figure 6.

Because the border vertices between segments are left unchanged, different resolution levels automatically connect to each other. Figure 3 shows the construction process for the NASA fighter dataset.

```
For every segments $s$
    $s_{new}$ = copy segment $s$;
    add $s_{new}$ as parent of $s$;
    simplify $s_{new}$;
While the number of roots $> 1$
    construct the node graph $G$;
    find pairwise matching $M$ of $G$;
    For every pair $(s_1, s_2)$ in $M$
        $s_{new}$ = merge segments $s_1$ and $s_2$;
        add $s_{new}$ as parent of $s_1$ and $s_2$;
        simplify $s_{new}$;
```

Figure 5: *Every iteration finds pairs of segments that are merged into a new segment and simplified.*

The final mesh segments are written in breadth-first order to disc together with the hierarchy. Using a breadth-first order enhances file accesses because neighboring segments are likely to be stored near together. Each segment is stored as described in section 3.

5.2 Usage

Refining one segment means to replace the mesh of the segment by the meshes of its children. Both children can now be refined independently of each other. For instance, the first child could be refined again and again while the second child remains unchanged. This would lead to a high resolution level that automatically connects to the low resolution level of the second child (but with the limitation that both the high and the low resolution segments need to share the same vertices on their border).

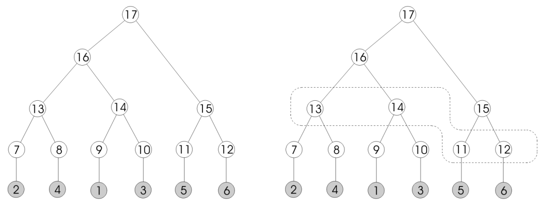

Figure 6: *The segment hierarchy stores how the segments are merged into their parent segments (left). The segments of the original mesh are gray. A front through the hierarchy corresponds to a valid mesh (right).*

Coarsening a segment means to replace the segment itself as well as its sibbling (in the binary hierarchy) by the coarser mesh of the parent.

A valid mesh is defined as a list of segments within the hierarchy that has exactly one segment in every path from the root down to the leaves as shown in figure 6 (right). We call this list a segment front. This corresponds to the well-known vertex front in vertex-based multi-resolution models [DDFM$^+$04].

5.3 The 0-segment

The meshes of the segment front form a valid tetrahedral mesh. During rendering, the adjacencies between all tetrahedra are often needed (for instance for MPVO-based sorting [Wil92]). A fast method to adapt the adjacencies between segments is needed whenever a segment is replaced by another segment. We introduce a special segment that handles the adjacency between any two neighboring segments. Because this special segment has the unique segment index 0, we call it 0-segment.

The 0-segment is defined as the cut of all segments of the original mesh. Thus, it contains

- all vertices that are on boundaries between two neighboring segments in the original mesh (white vertices in figure 1b), and
- all triangles that form the border between two neighboring segments, and
- (theoretically) all edges (not stored).

The triangles of the 0-segment are now used as a buffer (or a docking station) between two segments. Instead of storing the index of the adjacent tetrahedron, the index of the shared triangle in the 0-segment is stored in the adjacency information. Because the triangles never change and have always the same index, the adjacency to this triangle can be stored in the mesh (and in the file).

Each triangle stores two adjacency index-triples that point to tetrahedra in the adjacent segments, see also figure 7b. The tetrahedra of a segment that have a border to the 0-segment store an index-triple (s_i, t_i, c_i) as usual that points into the 0-segment with $s_0 = 0$ and t_i as the triangle index. c_i can be 0 or 1 and points to one of the two adjacency index-triples of the triangle.

When a segment s replaces another segment r, it must update the 0-segment as follows. All border tetrahedra of s are traversed. At least one of the four adjacency index-triples of these tetrahedra point to a triangle t in the 0-segment (the other index-triples point to tetrahedra inside s itself). The index-triple of t points to the segment r and must be replaced by the index-triple of s, i.e. (s_i, t_i, c_i) where $s_i = s$, t_i is the index of the border tetrahedron and c_i is the local index of the opposite vertex within t_i, see also figure 7.

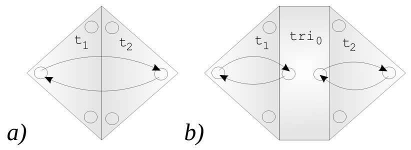

Figure 7: *Instead of storing the index-triple of the neighboring tetrahedron at the border of two segments (a), the index of the triangle in the* 0*-segment is stored (b).*

In order to find all border tetrahedra fast, they are stored before all other (inner) tetrahedra in the file. So iterating all border tetrahedra can be accomplished by iterating over all tetrahedra until a non-border tetrahedron is found.

The 0-segment does not need to exist at the construction phase. It must exist only at run-time when segments are to be exchanged very fast and can be computed once after (or before) the construction phase.

6 VIEW-DEPENDENT RENDERING

In order to adapt the mesh to current viewing and classification parameters, we need to decide which segments of the segment front must be coarsened (replaced by the parent) or refined (replaced by the children). Therefore, the following values are stored with each segment

- The histogram H of the attribute values (which is a lookup-table of resolution N with normalized entries $H_i \in [0,1]$), and
- A look-up table E of the same resolution N which specifies the maximal error that the segment contains for the according attribute value, and
- The (axis aligned) bounding box.

The user can specify a maximal field error ε_{max}.

For each frame, the segment front is traversed. Every segment of the front is marked by the tags COARSEN, REFINE, and NOTHING that help later to adapt the mesh:

1. If the bounding box is outside the view-frustum and if the sibbling exists, mark as COARSEN, else
2. Compute the average $S = \sum_i^N H_i E_i \alpha_i$ where the sum runs over all histogram values, H_i is the (normalized) i-th histogram value, E_i is the according error and α_i is the according classified opacity.

   If $S > \varepsilon_{max}$, mark as REFINE, else
3. Mark as NOTHING.

After all segments of the front are marked, the front is traversed again and every segment is adapted:

1. If a segment is marked as REFINE, it is replaced by its children.
2. If a segment and its sibbling are marked as COARSEN, both are replaced by their parent.

The histogram H as well as the look-up table E are computed during the construction of the multi-resolution model. Every edge collapse introduces a particular error for a scalar field attribute which is stored in the look-up table E if it is greater than the already stored error.

A simple LRU queue keeps track of the mapped segments. We refer the reader to the more elaborated caching methods of for instance [YLPM05].

7 RESULTS

We implemented the technique with memory mapped files which are a operating system opportunity for memory allocation such that parts of a file can be directly mapped into memory. The operating system performs all necessary swapping.

Name	# Vertices	# Tetra	# Segments	Min Vertices per segment	Max Vertices per segment	Min Tets per segment	Max Tets per segment	Time hh:mm:ss
Seaway	102,165	524,640	18	3,050	5,989	17,382	38,929	0:00:10
Fighter	256,614	1,403,504	48	3,175	5,078	21,648	31,660	0:00:15
Rbl	730,273	3,886,728	131	2,714	6,599	17,378	36,284	0:00:26
F16	1,124,648	6,345,709	212	2,874	6,169	26,104	37,087	0:00:58

Table 1: *The properties of the datasets and the timings for the construction of the segmented mesh from the original mesh. The octree was steered to contain at most 5,000 vertices per leaf node.*

So if a segment *s* replaces another segment, our implementation calls the operating system to map the parts of the file that correspond to *s* into the memory. We found that the operating system (we use Windows XP) needs nearly constant time to map a segment if the mesh part has been accessed some time before due to caching. However, sometimes the caching misses and delays of at most one second may occur.

The frame rates depend mainly on the power of the volume renderer that uses preintegrated projected tetrahedra. Because the size of the segment front is small, the costs for checking if a segment can be refined or coarsened, are neglectable. We experienced frame rates of about 3-4 frames per second with a workload of about 250,000 tetrahedra. We can render the F16 model interactively which is impossible for the full resolution mesh (it would take at least 6 seconds to render a single frame).

In average, 80% of the time of a frame is used by the volume renderer whereas 16% are used for reloading segments (file IO) and 4% are used for segment adjacency adaption (measured average values for the NASA Fighter dataset, the other datasets perform similiar).

The construction timings of the segmented meshes (section 3) are shown in table 1. Most of the time is needed for file IO. For the Rbl dataset, for instance, the IO to write the mesh to disc needed 20 seconds (out of the 26 seconds total construction time).

The main time was spent to simplify the meshes and to construct the hierarchy, see table 2. Although the timings do not compare to the (much faster) timings of Lindstrom [VCL+05], we differ from Lindstrom because we need to store all segments and do not use the randomized edge collapses.

Furthermore, the file sizes are huge because we store each segment in a raw format such that it is ready to be mapped to memory.

Name	# Tetra in base mesh	Time hh:mm:ss	file size [MB]
Seaway	18,763	0:13:47	67
Fighter	30,042	0:19:23	101
Rbl	259,363	0:31:05	349
F16	84,246	1:08:23	514

Table 2: *The simplification timings and the sizes of the stored multi-resolution files.*

8 CONCLUSION AND FUTURE WORK

We presented an out-of-core data structure that enables to simplify a tetrahedral mesh with a small memory footprint. The segments are constructed by combining leaves of an octree such that the segments contain similiar parts of the mesh.

A multi-resolution hierarchy is built based on the segments where pairs of segments are merged and simplified. The segments connect to each other using the 0-segment. It allows for the multi-resolution mesh to adapt its adjacency information efficiently which is mandatory for tetrahedral sorting algorithms like MPVO.

The multi-resolution model is included into a direct volume rendering frame work that adpats the mesh to viewing and classification parameters using a histogram of attribute values and errors.

Future work should make use of a compression scheme for the segments in order to shrink the file sizes of the models. Furthermore, more elaborated segmentation techniques that cluster vertex data based on spatial density and attribute values can be used.

ACKNOWLEDGEMENTS

The authors would like to thank Udo Tremel from EADS Military Aircraft for the beautiful F16-like simulation data, and Uli Bieg from the University of Tübingen for the mesh of the Burdigalian Seaway. This work is supported by EC within FP6 under Grant 511568 with the acronym 3DTV. This work has been funded by the SFB grant 382 of the German Research Council (DFG).

REFERENCES

[CCM+00] P. Cignoni, D. Constanza, C. Montani, C. Rocchini, and R. Scopigno. Simplification of tetrahedral meshes with accurate error evaluation. In *Proceedings of the conference on IEEE Visualization '00*, pages 85–92, 2000.

[CDFL+04] P. Cignoni, L. De Floriani, P. Lindstrom, V. Pascucci, J. Rossignac, and C. Silva. Multi–resolution modeling, visualization and streaming of volume meshes. *Eurographics '04 Tutorial Notes*, 2004.

[CGG+04] 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 Transactions on Graphics*, 23(3):796–803, 2004.

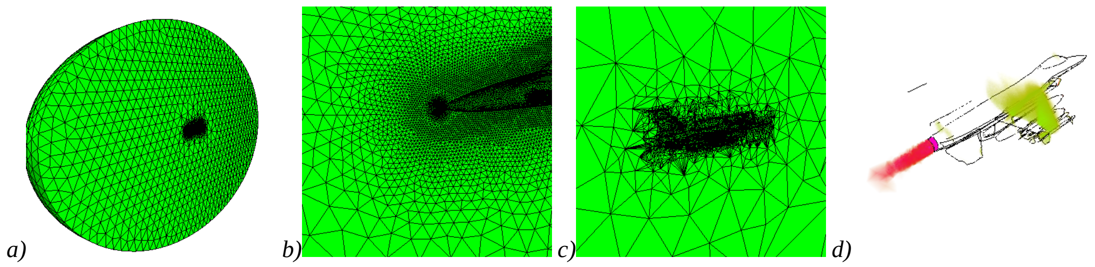

Figure 8: *This model is used for a CFD simulation of a F16-like aircraft and contains about 6 million tetrahedra. Figure (a) shows the full resolution mesh and (b) shows a zoom into the full resolution mesh. Note how the size of the tetrahedra varies which needs to be captured by the segmentation. Figure (c) shows how the mesh is adapted to the viewpoint in the volume rendering (d).*

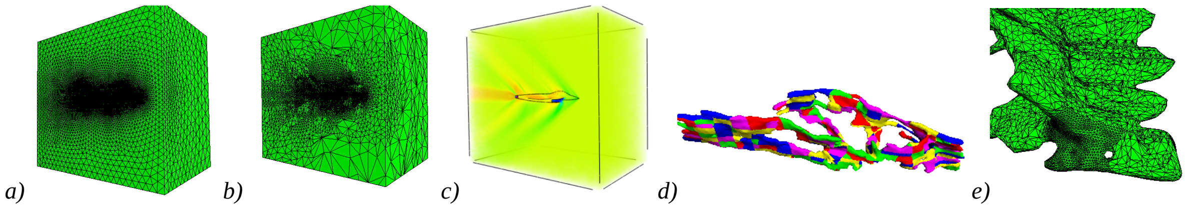

Figure 9: *The NASA fighter dataset (a-c) shows a fighter in a wind tunnel and contains 1.5 million tetrahedra. The full resolution mesh (a) is adpated to the viewpoint and the classification (b) in order to render picture (c) with 250,000 tetrahedra interactively. The Rbl dataset (d) is a portion of an endoplastic reticulum in a cell. Its 3.8 million tetrahedra are simplified to 260,000 tetrahedra using the segmentation in (d). Picture (e) shows a zoom to an adaption of the mesh.*

[CL03] Y. Chiang and X. Lu. Progressive simplification of tetrahedral meshes preserving all isosurface topologies. In *Computer Graphics Forum (Special Issue for Eurographics '03)*, volume 22, pages 493–504, 2003.

[CM02] Prashant Chopra and Jörg Meyer. Tetfusion: an algorithm for rapid tetrahedral mesh simplification. In *Proceedings of the conference on Visualization '02*, pages 133–140, 2002.

[CMRS03] P. Cignoni, C. Montani, C. Rocchini, and R. Scopigno. External memory management and simplification of huge meshes. *IEEE Transactions on Visualization and Computer Graphics*, 9:525–537, Nov 2003.

[DDFM+04] E. Danovaro, L. De Floriani, P. Magillo, E. Puppo, D. Sobrero, and N. Sokolovsky. A compact data structure for level–of–detail tetrahedral meshes. Technical report, University of Genova, 2004.

[GI03] Stefan Gumhold and Martin Isenburg. Out-of-core compression for gigantic polygon meshes. In *ACM Transactions on Graphics*, volume 22, pages 935–942, 2003.

[GZ05] Michael Garland and Yuan Zhou. Quadric-based simplification in any dimension. *ACM Transactions on Graphics*, 24(2), 2005.

[Hop96] Hugues Hoppe. Progressive meshes. *Computer Graphics*, 30(Annual Conference Series):99–108, 1996.

[IL05] Martin Isenburg and Peter Lindstrom. Streaming meshes. In *IEEE Visualization 05*, pages 231–238, 2005.

[KE00] M. Kraus and T. Ertl. Simplification of nonconvex tetrahedral meshes. *Electronic Proceedings of NSF/DoE Lake Tahoe Workshop for Scientific Visualization*, 2000.

[KQE04] M. Kraus, W. Qiao, and D. Ebert. Projecting tetrahedra without rendering artifacts. In *Proceedings of IEEE Visualization '04*, pages 27–34, 2004.

[KSE04] T. Klein, S. Stegmaier, and T. Ertl. Hardware–accelerated reconstruction of polygonal isosurface representations on unstructured grids. In *Proceedings of Pacific Graphics '04*, pages 186–195, 2004.

[MHC90] N. L. Max, P. Hanrahan, and R. Crawfis. Area and volume coherence for efficient visualization of 3d scalar functions. *Computer Graphics (San Diego Workshop on Volume Visualization)*, 24(5):27–33, 1990.

[PH97] Jovan Popovic and Hugues Hoppe. Progressive simplicial complexes. In *SIGGRAPH*, pages 217–224, 1997.

[RKE00] S. Roettger, M. Kraus, and T. Ertl. Hardware–accelerated volume and isosurface rendering based on cell projection. In *IEEE Proceedings Visualization '00*, pages 109–116, 2000.

[RO96] Kevin J. Renze and James H. Oliver. Generalized unstructured decimation. *IEEE Computer Graphics and Applications*, Nov 1996.

[SG98] O. G. Staadt and M. H. Gross. Progressive tetrahedralizations. In *Proceedings of IEEE Visualization '98*, pages 397–402, Oct 1998.

[ST90] P. Shirley and A. Tuchman. A polygonal approximation to direct scalar volume rendering. *ACM Computer Graphics (San Diego Workshop on Volume Visualization)*, 5(4):63–70, 1990.

[VCL+05] Hyu Vo, Steven Callahan, Peter Lindstrom, Valerio Pascucci, and Claudio Silva. Streaming simplification of tetrahedral meshes. Technical report, LLNL technical report UCRL-CONF-208710, 2005.

[Wil92] Peter L. Williams. Visibility-ordering meshed polyhedra. *ACM Transactions on Graphics*, 11(2):103–126, 1992.

[YLPM05] S.-E. Yoon, P. Lindstrom, V. Pascucci, and D. Manocha. Cache-oblivious mesh layouts. In *ACM Transactions on Graphics (SIGGRAPH)*, pages 886–893, 2005.