MEMORY-EFFICIENT VIEW-DEPENDENT LEVEL OF DETAIL OF HIGH-DETAILED MESHES

Mathias Holst and Heidrun Schumann
Institute of Computer Science, University of Rostock, Albert Einstein Str. 23, Rostock, Germany

Keywords: Multi-Resolution, Out-of-Core, Real-Time Rendering.

Abstract: In this paper we propose an effective and efficient out-of-core LOD framework for arbitrary irregular high-detailed triangular meshes and scenes. We modify and extend the standard Multi-Triangulation framework to reduce CPU load and to increase RAM efficiency by paging. To reduce the number of IO-operations we propose a novel partitioning scheme well suitable for view-dependant LOD. In addition a fast but very effective caching strategy is proposed, which bases on a LOD prediction for future frames. The efficiency of our framework is shown by results.

1 INTRODUCTION

In this paper we focus on continuous LOD (short *CLOD*). Generally speaking for this purpose a hierarchy or sequential list is generated for a given object in a preprocess that contains the whole LOD spectrum. This structure is processed in every frame to get a proper LOD relating some restrictions (e.g. triangle number or image error). The benefits of CLOD are a nearly seamless LOD-transition and a nearly optimal adaption of the polygon number to the desired image quality. On the other hand, LOD estimation can be slow (sometimes even slower than the following polygon rendering) and the requirement of main memory is high (generally a multiple of the original model size).

To attenuate the memory costs we propose an out-of-core framework, that contains a new algorithm to increase memory efficiency by using secondary memory. This algorithm contains several strategies for partitioning and caching to reduce the number of necessary file operations. For this purpose we use the elegant Multi-Triangulation (or Multi-Tesselation) framework (short *MT*) developed by DeFloriani et al. (Floriani et al., 1997), that we adapt to our needs.

This paper is structured as follows: In section 2 previous works are introduced. In section 3 we describe the basics of the MT hierarchy and its reduction to accelerate rendering. In section 4 our new algorithm to handle the MT efficiently out-of-core is described. At the end we show the efficiency of our method by results in section 5 and give some concluding remarks in section 6.

2 PREVIOUS WORK

In this section we briefly summarize related publications that focus on out-of-core continuous level of detail of polygonal meshes.

In (El-Sana and Chiang, 2000) a merge-tree is used out-of-core. To reduce IO-operations a segmentation of the original mesh into patches is proposed, using the edge collapse order given by the simplification error. This inherently preserves boundaries between patches. The XFastMesh framework (DeCoro and Pajarola, 2002) also uses a merge-tree, but a layer-based partitioning and a very compact file format. A space partitioning approach was proposed in (Lindstrom, 2003). Here, the original model is quantized to a regular 3D-grid which allows a memory-efficient storage. In contrast to these frameworks, we use a MT-hierarchy of static patches that can be described and rendered more efficiently by using triangle strips. The QuickVDR system (Yoon et al., 2005) also uses

Holst M. and Schumann H. (2007).
MEMORY-EFFICIENT VIEW-DEPENDENT LEVEL OF DETAIL OF HIGH-DETAILED MESHES.
In *Proceedings of the Second International Conference on Computer Graphics Theory and Applications - GM/R*, pages 323-326
DOI: 10.5220/0002077003230326

a hierarchical set of patches, which are independently converted into progressive meshes and then merged bottom-up. Another system is the Batched-MT framework (Cignoni et al., 2005). Here, a leveled patch partitioning of the original mesh is used from smaller to larger patches. These patches are simplified without changing the boundary. In doing so, a MT hierarchy can be generated that contains many triangles in its arcs[1] which is useful for a fast LOD rendering. We use a similar MT-hierarchy, but choose an contrary approach. Firstly, a classical MT-hierarchy is created that is reduced afterwards to increase arc patches. Thus, we do not have any constrictions to boundary simplification which seems to be a strong constraint. Like QuickVDR the Batched-MT framework does not manage the hierarchy out-of-core, but only the geometry.

3 BASIC MT-FRAMEWORK

The MT is a general framework for fast adaptive estimation of many specific LOD for three-dimensional triangular meshes. It is based on a local simplification operator, which replaces an adjacent triangle set with a less complex (and a generally smaller) one of the same border. Without loss of generality we use edge-collapse operators for this purpose. These operators are represented as nodes in the corresponding MT hierarchy, which is a DAG in general. The triangle set that is simplified is represented by outgoing edges of the node and the simplified set is represented by ingoing edges (fig. 1(b)).

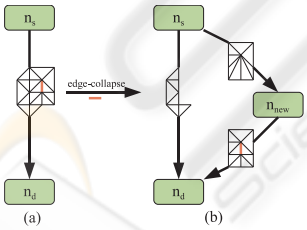

Figure 1: A given mesh and its initial MT (a). MT after one edge-collapse operation (the thicker red edge is collapsed) (b).

3.1 Hierarchy Reduction

A MT which was build using an edge-collapse operator contains about 5 times more triangles than the original object. Even worse, it can be shown that every hierarchy arc contains less than 2.5 triangles in average. Since every arc has to be rendered separately, the CPU load is very high and optimizations, like triangle strips, are less efficient.

Therefore it is useful to modify the MT to get more compact in order to contain larger triangle sets in its arcs. For this purpose we use the simplification scheme, proposed in (Holst and Schumann, 2006). By an iterative application of so called arc-collapse operations in a well chosen order, the hierarchy can be reduced very easily. How to determine this order is described in (Holst and Schumann, 2006). This reduction is stopped, if an average patch size of 10 is reached.

4 OUT-OF-CORE MT

Besides a fast LOD estimation and rendering our goal is to decrease the amount of RAM required to store the MT by an increased use of secondary memory. Therefore, we have to store the MT in a file or, more generally, in a data array. In the following we will explain the layout of such file. After this we show how the efficiency of file processing can be increased.

4.1 File Layout

We decided to store not only the geometry data in the MT-file, but also the hierarchy itself, to further decimate RAM requirement. This is reasonable especially for large hierarchies and complex scenes.

In most situations the LOD cut is only adapted somewhat for the current frame. This means that if a new arc is traversed its geometry is also likely used for the new LOD. Therefore, it is appropriate to store the geometry of an arc together with the arc, and not in a separate file part.

When traversing the hierarchy, for every node its outgoing and ingoing arcs are required. To support a sequential data reading with few file accesses only we decided to store after each node immediately its outgoing arcs. To address an ingoing arc $a = (n', n)$ of a node n we store a tuple consisting of the father node index n' and the position index of the outgoing arc a in n'.

The resulting file layout is shown in fig. 2.

4.2 Partitioning

Partitioning is a common technique to reduce file operations. However, partitioning can lead to a reading of much unused data. Therefore, it is useful to adapt

[1]Edges of MT-hierarchies are called arcs to distinguish them from mesh edges.

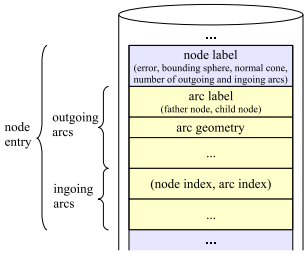

Figure 2: Layout of our MT file.

the partitioning scheme to the LOD selection mechanism. But the iteration order of hierarchy nodes and arcs is not predefined, in general. Thus, we propose to use a local partitioning, which is based on the following two observations:

- Nodes are often used together in one LOD, if they are close in the hierarchy.
- If there are many arcs between two given partitions, then the propability is high that both partitions are used together for many LOD.

Starting from this, a partition scheme can be formulated. Firstly, for every node $n_i \in \mathcal{N}$ a partition $p_i = \{n_i\}$ is created. Every connected partition pair is inserted into a priority queue together with the number of connecting arcs in increasing order. While processing this queue, partition pairs are merged to form a larger partition, but only if the size of the new partition does not exceed a given threshold θ. This avoids an enlargement of already large partitions that have many connecting arcs anyway. If two partition pairs have the same number of connecting arcs, we prefer that pair with the smallest layer span. Therefore, we use a top-down layering of the hierarchy that is generated in a preprocess. In doing so, we prevent that partitions contain too many different LOD. After creating a new partition out of two other ones, the queue is updated with entries for this new partition. This is done until the queue is empty.

It is not trivial to answer the question, which partition size is appropriate, because it depends on the application scenario. If there is only one object in the scene, then a smaller partition size of $\theta \leq 20$ is appropriate, because only one file is accessed. Tests have shown that if there are many objects in the scene the partition size should be enlarged to $50-100$, to reduce file operations.

4.3 Node Caching

A consequence of reading partitions instead of nodes is that more nodes and arcs are kept in main memory than really used. We propose to cache them for future frames, but only if the probability of using them is high. To estimate such nodes we use a simple but efficient selection criterion, which is based on the observation, that if a node is the father node (resp. child node) of an arc in the cut that is estimated by a top-down (resp. bottom-up) MT iteration, then this node or nodes below (resp. above) this node probably will be used in one of the following frames as well. To distinguish these nodes from other obsolete partition nodes, a fast criterion has to be found, that works without a complete scan of all partition nodes in every frame, because this would slow down LOD estimation heavily.

For this purpose we use a layer based approach. For every partition that is partially in main memory we maintain two tuples $T_{min} = (f_{min}, l_{min})$ and $T_{max} = (f_{max}, l_{max})$ consisting of a frame index and a layer index. After a partition is loaded both tuples are initialized with $(-1,-1)$ to distinguish them from meaningful tuples. If an arc a in the current cut was estimated by a top-down iteration of the hierarchy, then we will update T_{min} of the father node's and child node's partition as follows:

$$T_{min} \leftarrow \begin{cases} (f, \min(l_{min}, l_n)), & \text{if } f = f_{min} \\ (f, l_n), & \text{otherwise} \end{cases}, \quad (1)$$

where l_n denotes the layer of the particular node and f is the current frame number. Similarly, we update T_{max} if a was estimated by a bottom-up iteration:

$$T_{max} \leftarrow \begin{cases} (f, \min(l_{max}, l_n)), & \text{if } f = f_{max} \\ (f, l_n), & \text{otherwise} \end{cases}, \quad (2)$$

If an arc of the current cut was also in the cut of the last frame then f_{min} and f_{max} has only be set to f.

We maintain a priority queue of all nodes in main memory, sorted by the last frame they were part of a cut (resp. LOD) in increasing order. If a new partition is loaded, then all nodes will be inserted into this queue with frame index -1 because they do not belong to a LOD yet. After every frame, the queue is iterated from the beginning until a node of the current frame is reached. If an iterated node n is the father or child node of an arc in the cut it is re-inserted into this queue with the actual frame. Otherwise we use T_{min} and T_{max} to decide whether to cache it for future frames. If

$$(f_{min} = f \wedge l_n > l_{min}) \vee (f_{max} = f \wedge l_n > l_{max}), \quad (3)$$

then n is kept. In fig. 3 this procedure is illustrated for the top-down case.

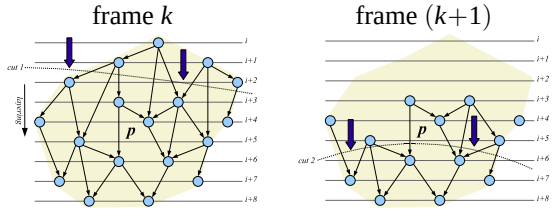

Figure 3: Node caching algorithm: In frame k a certain partition is loaded during a top down iteration (a). In frame $(k+1)$ the cut through this partition is moving down and nodes on higher layer are deleted (b).

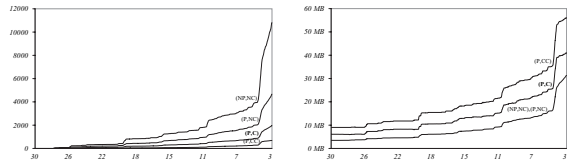

Figure 4: Results for moving object from far to near. Total sum of io operations since begin (a) and memory requirement (b).

5 RESULTS

We compared our out-of-core approach, using partitioning and caching (P,C), with three more simpler approaches:

1. No partitioning, no caching (NP,NC): All nodes are loaded if they are required for MT-iteration, and they are deleted afterwards, if they do not belong to the cut.
2. Partitioning, no caching (P,NC): The same as before, but nodes are loaded in a bundle using our partitioning scheme.
3. Partitioning, conservative caching (P,CC): The same as before, but nodes are deleted only if no node of their partition belongs to the cut.

Exemplarily, the Armadillo object is used in two different situations: First, moving from far to near (fig. 4) and second, rotating around the y-axis in view space (fig. 5). The sum of file accesses since animation begin (4(a),5(a)) and the memory requirement (4(b),5(b) were measured during both simulations. As it can be seen our approach provides a very good trade-off. It needs slightly more file accesses than (P,CC) but the memory requirement is halved on average with respect to (NP,NC) and (P,NC).

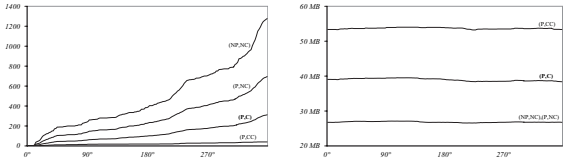

Figure 5: Results for rotating object. Total sum of io operations since begin (a) and memory requirement (b).

6 CONCLUSION

In this paper we described a LOD-framework for arbitrary irregular 3D-meshes that provides a good trade-off between rendering efficiency and memory efficiency. It uses a reduced MT hierarchy. To increase memory efficiency we handle the reduced MT out-of-core. The number of file operations is reduced by using an effective and easy to implement partitioning scheme. Moreover, a caching algorithm is proposed to reduce unnecessary reloadings.

REFERENCES

Cignoni, P., Ganovelli, F., Gobbetti, E., Marton, F., Ponchio, F., and Scopigno, R. (2005). Batched multi triangulation. In *Vis'05 conf.proc.*, Minneapolis, MI, USA. IEEE Computer Society Press.

DeCoro, C. and Pajarola, R. (2002). Xfastmesh: fast view-dependent meshing from external memory. In *VIS '02 conf.proc.*, pages 363–370, Washington, DC, USA. IEEE Computer Society.

El-Sana, J. and Chiang, Y.-J. (2000). External memory view-dependent simplification. *Computer Graphics Forum*, 19(3):139–150.

Floriani, L. D., Magillo, P., and Puppo, E. (1997). Building and traversing a surface at variable resolution. In *Vis'97 conf.proc.*, pages 103–ff. IEEE Computer Society Press.

Holst, M. and Schumann, H. (2006). Efficient rendering of high-detailed objects using a reduced multiresolution hierarchy. In *GRAPP'06 conf.proc.*, pages 3–10, Setúbal, Portugal. INSTICC Press.

Lindstrom, P. (2003). Out-of-core construction and visualization of multiresolution surfaces. In *SI3D'03 conf.proc.*, pages 93–102, New York, NY, USA. ACM Press.

Yoon, S.-E., Salomon, B., Gayle, R., and Manocha, D. (2005). Quick-vdr: Out-of-core view-dependent rendering of gigantic models. *IEEE Transactions on Visualization and Computer Graphics*, 11(4):369–382.