Out of Core continuous LoD-Hierarchies for Large Triangle Meshes

Hermann Birkholz
Research Assistant
Albert-Einstein-Str. 21
Germany, 18059, Rostock
hb01@informatik.uni-rostock.de

ABSTRACT

In this paper, algorithms for the simplification and reconstruction of large triangle meshes are described. The simplification process creates an edge-collapse hierarchy in external memory, which is used for online reconstruction. The hierarchy indices are renamed after simplification, in order to allow fast reconstructions and the hierarchy is extended with information for view-dependent rendering.

The simplification makes no restrictions with the production of the hierarchy, but produces the same hierarchy as In-Core algorithms. The amount of memory, which is used for the simplification is adjustable.

Keywords
Out of Core, Level of Detail, triangle meshes, view dependent rendering

1. INTRODUCTION

Large polygonal meshes can easily be acquired with current 3d scanning hardware [Lev00]. Those meshes exceed the internal memory and the rendering capabilities of modern personal computers. For interactive visualization only partitions of the mesh can be used. In order to offer this, view-dependent approximations of the mesh must be fast computable. Such as for In-Core-meshes, continuous Level of Detail (cLoD) methods can be used to create fast view-dependent approximations. Because the mesh data does not fit into main memory these techniques must be adopted for such large meshes. Once created, a cLoD-hierarchy in external memory can be used for view-dependent online approximation of the original mesh. Therefor only visible parts of the hierarchy are read from external into internal memory and refined, until a time- or memory-limit is reached. These parts can then be visualized with the graphic hardware. In this paper a new simplification algorithm is presented, which creates cLoD-hierarchies for large meshes. This algorithm produces the same simplification hierarchy such as In-Core methods but it stores only partitions of the mesh in internal memory. The maximum memory footprint of the mesh is adjustable by the user. Furthermore it is demonstrated how to use the resulting hierarchy file to generate view-dependent approximations of the mesh.

2. PREVIOUS WORK

In-Core cLoD Systems often build their hierarchies by collapsing edges [Hop96] or contracting vertices [Gar97] of the mesh surface. For each collapse/contraction operation an error value is computed, which determines the sequence of the collapses or of the contractions. For each edge collapse operation in manifold meshes two triangles are removed from the mesh surface. The two merged vertices and the removed triangles are stored in the LoD-hierarchy together with the error value.

For the simplification of large meshes various techniques have been presented. The spanned mesh simplification algorithm [San00] uses an external indexed mesh and an external heap with all collapsible edges, in order to determine the simplification sequence. The algorithm reads the first k (depending on internal memory size) edges from the queue and the mesh parts, which belong to the edges. Afterwards all possible simplification operations for the edges, which are in memory, can be accomplished. Due to the nearly uniform distribution of the edges with small collapse errors, there will be hardly advantages from the locality of the read mesh parts. This will result in frequent mesh updates and load/store operations and thus long computation times. An advantage however is the fact that the simplification sequence is identical to In-Core algorithms.

In order to overcome the problem of the uniform distribution, hierarchical clustering is used to partition the mesh into locally connected blocks. Hoppe [Hop98] creates a block hierarchy and simplifies the mesh portions in the leaf blocks (edge collapse) except the edges, which cross the block borders. After the simplification, the leaf blocks are hierarchically merged and then simplified again. This is repeated, until the whole mesh is stored in the root cluster. Due to the forbidden collapses of edges, which cross the cluster-borders, this algorithm cannot limit the internal memory, used for the clusters. Furthermore the simplification sequence is limited by the cluster borders and might deliver bad results.

Cignoni [Cig02] suggests an external memory management based on octree subdivision. The management is able to overcome the problems with block borders and has a wide field of applications. For their simplification example however, they avoid the use of a heap structure for the correct simplification sequence, in order to take advantage of locality.

Another approach based on cluster hierarchies was presented by Lindstrom [Lin03]. His method consists of three steps. First, the mesh is clustered with an memory insensitive clustering technique, which is based on the clustering schema of Rossignac and Borrel [Ros93]. This step produces an uniform grid with cells, which contain either one or no vertex. In the second step, an octree is constructed over the grid, where the position of the merged vertices is determined with the QEM [Gar97]. In the third phase the hierarchy is used for view-dependent rendering of the mesh. The drawbacks of this method are the initial clustering, which might remove mesh details, if the grid-size is too high, and the octree schema, which produces hierarchies of a lower quality as edge collapse hierarchies.

So called "Processing Sequences" for computations on large meshes were introduced by Isenburg [Ise03]. Their mesh representation allows them to stream the mesh through internal memory and to apply changes to this local region. They show the simplification of large meshes as an example, but they neither produce a hierarchy, which can be used for reconstruction, nor they use a simplification order similar to In-Core methods.

The algorithm, which is presented in this publication produces simplification sequences, which are identical to In-Core algorithms ,and it takes advantage of local surface regions.

3. OUT OF CORE LOD

Simplification

For mesh simplification, the half-edge collapse [Kob98] method is used. This means that an edge is collapsed to one of its vertices.

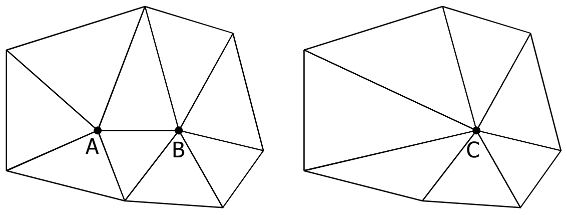

Figure 1. Half.-edge collapse

Figure 1 shows a half-edge collapse operation. The edge between vertex A and B is collapsed to vertex C. Vertex C is placed on the same position as vertex B. The simplification error of each vertex is computed with the Quadric Error Metrics [Gar97]. For each vertex all adjacent vertices are tested as potential collapse targets. The target which causes the smallest error value is stored for each vertex. Other error metrics, are applicable too. A simplification sequence equal to In-Core methods can be reached by executing only collapses which are locally minimal. That means, all adjacent vertices will cause higher or equal collapse errors. By iteratively applying locally minimal collapses, the hierarchy will become the same as if the collapses are applied in a global sequence (lowest first).

This fact can be used while simplifying large meshes. Local connected portions of the mesh can be read and all collapses for locally minimal errors can be applied.

Figure 2 shows the distribution of collapse errors for the "Stanford Bunny" mesh. The red/orange colors indicate high/medium collapse errors, while green colors indicate low collapse errors. As noticeable, the collapse errors are evenly distributed over the mesh. This also leads to an even distribution the locally minimal collapse errors (bright green dots). Furthermore the figure shows regions of low collapse errors that are surrounded by regions of low collapse errors. The surrounding vertices with high error values will not be collapsed until their neighbor vertices reach similar values. Higher collapse errors can only be reached by applying local simplifications in the low error regions.

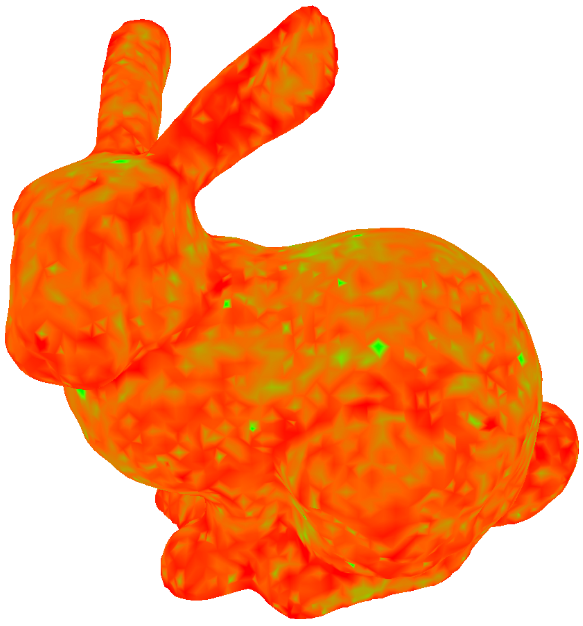

Figure 2. Collapse error distribution for the "Stanford Bunny" mesh

Starting with a small partition of the mesh in internal memory, all locally minimal collapses in the partition are executed or the partition is expanded when no locally minimal collapse error can be found. If the internal memory limit is reached, the memory data is written back to external memory and the process is restarted around the vertex with the smallest collapse error in the last partition.

In this implementation, **inner-vertices**, **border-vertices** and **near-border-vertices** are distinguished. For all vertices in internal memory all surrounding triangles are also read into the internal memory. For **border-vertices** not all vertices in the neighborhood have already been read from external memory. For this kind of vertex no collapse weight is computed, because not all possible collapse targets remain in memory. The **near-border-vertices** are completely surrounded by already loaded vertices but they have at least one **border-vertex** in their neighborhood. Because of the complete neighborhood, a collapse-target and –weight can be computed, but due to the at least one **border-vertex** in the neighborhood, it is impossible to determine if the weight is locally minimal. The **inner-vertices** have only other **inner-** or **near-border-vertices** around itself. Therefore locally minimum collapse errors can only be found among **inner-vertices**. Figure 3 shows a configuration with one **inner-vertex** (black), the surrounding **near-border-vertices** (black outline) and the **border-vertices** (stippled outline).

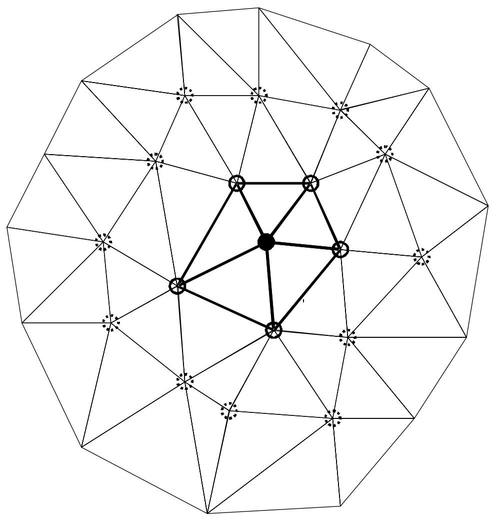

Figure 3. Local mesh partition with one inner vertex

Whenever no locally minimal collapse error can be found, the border of the local partition has to be expanded. Therefore the **near-border-vertex** with the smallest collapse error is chosen and all of its linked **border-vertices** are updated to **near-border-vertices**, by reading their neighborhood from external memory. This changes the state of the prior **near-border-vertex** to an **inner-vertex**, which can be checked for a locally minimal collapse error. Figure 4 shows the expansion of the upper right **near-border vertex** from figure 3.

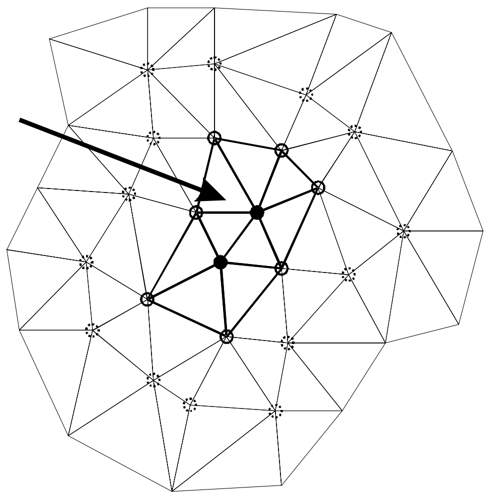

Figure 4. Local mesh partition with two inner vertices

Because always the **near-border-vertex** with the smallest collapse error is chosen, the In-Core part will always grow towards vertices with locally minimal collapse errors. If a maximum number of vertices or triangles is exceeded in main memory, the In-Core vertices and triangles are written back to external memory. The simplification process then continues with the prior **near-border-vertex**, which would cause the smallest collapse error.

The original data is presented as an array of triangles and an array of vertices in external memory. For each vertex we also store the list of adjacent triangles. Changed and new vertices are stored separately and provided with hierarchy informations.

For each collapse operation the pair of merged vertices is stored in the resulting vertex. Furthermore the collapsed triangles are referenced in the resulting vertex. The collapsed vertices and triangles are removed from internal memory and all vertices, whose neighborhood changed (including the new vertex), are updated and checked for locally minimal collapse errors.

The simplification algorithm can be summarized as follows:

1. Choose and read a start vertex.
2. Read the adjacent vertices of the start vertex. (Make the start vertex to **near-border-vertex**.)
3. Expand the local partition by choosing the **near-border-vertex** with the smallest collapse error and update all **border-vertices** around it to **near-border-vertices**.
4. Execute all possible collapses among **inner-vertices**.
5. If the internal memory limit is reached, write internal data to external memory. Choose the **near-border-vertex** with the smallest error as new start vertex, read it and continue with step 2.
6. If further simplification is required, continue with step 3.

In the first step the vertex with the index 0 is normally chosen as start vertex. The second step reads the local neighborhood of the start vertex (triangles and vertices). After the second step, the start vertex is in the **near-neighbor-vertex** group and the other vertices are in the **border-vertices** group. The third step always expands the partition in internal memory, to find **inner-vertices**, whose collapse error is locally minimal. The fourth step executes all possible collapses within the **inner-vertices**. After that, a memory check is performed, to limit the use of internal memory. If no further local minimal collapse errors could be found, the algorithm restarts at the best position to find a local minimal collapse error. The last steps are repeated until a stop condition is reached. This can be for instance a maximum simplification error or a minimum number of triangles.

The indexed vertices and triangles of the internal partition are stored in AVL-Trees for fast access. With these trees, fast search operations can be performed in the local partition of the indexed mesh. The **near-border-vertices** are referenced in additional priority queues, for fast access to the vertex that will cause the smallest collapse error.

After the simplification, the remaining vertices and triangles are written to the hierarchy file for the use as approximation root.

Vertex Index Translation

In order to use the hierarchy file for the approximation of the original mesh, it has to be reconstructed top-down. This demands fast decisions how to distribute the triangles after each vertex split. This step can be accelerated by translating the hierarchy indices to an inorder structure. That means, the left subtree always contains only vertex indices which are smaller than, and the right subtree contains only vertices, which are greater than the according root node. Together with the information of the triangle vertices in the finest level, one can decide the correct child vertex in each split by only comparing vertex indices. All vertex indices in the left subtree are applied to the left child and the same is true for the right subtree.

View Dependent Rendering

After the translation of the hierarchy indices, some information for view-dependent rendering is added to the hierarchy. Fast view-frustum culling for each vertex in the hierarchy is supported with bounding-spheres. As soon as the bounding sphere of a vertex is outside the frustum, its complete subtree can be culled. For backface-culling and contour-based approximations, normal-cones [Xia96] for each vertex are computed. Whenever the cone points away from the viewer in the whole bounding-volume, the associated subtree can be culled.

The rendering process starts with the root of the hierarchy. All vertices, which were not collapsible and the triangles, which were not collapsed in the simplification process, construct the base mesh. These base vertices are put into a priority queue sorted according to their collapse error, which is divided by the distance to the viewer plane (greatest weight first). Vertices, which does not pass the view frustum test or the normal cone test are not put into the queue. Now the first vertex from the queue can be split into its child vertices iteratively and be replaced in the queue by its child vertices. The two associated triangles are appended to the triangle list during each split. Before the split, it is examined whether the child vertices are already in internal memory. If not, they are loaded from external memory. For internal memory management, an individual frame number is assigned to all internal vertices. This variable is always set to the number of the frame, in which its associated vertex was last split. All parent vertices of a pair of leaf-nodes in the internal vertex tree remain in a priority queue, sorted by their frame number (smallest first). Whenever memory must be reallocated, the first vertex from the queue is used to remove its child vertices from memory. After that, its parent vertex is put into the queue, if both of its child-vertices are leaf-nodes in internal memory. This memory management allows to restrict the memory, which is used by the internal vertex tree.

The whole approximation process is terminated when a given time period elapses or a desired number of triangles is reached. So a minimum frame rate can be guaranteed. In order to reach higher frame rates it would also be possible to make use of frame-to-frame coherency. Therefor one must use the approximated vertex tree from the last frame and apply both collapse- and split-operations to it. Furthermore two priority queues are required. One for the split-candidates (greatest error first) and one for the collapse-candidates (smallest error first). Both queues must be balanced in each frame depending on the view parameters.

4. RESULTS

Simplification

A prototype implementation of the simplification algorithm was tested with several meshes. The results for small meshes were determined from the “Armadillo” mesh from "Stanford University Computer Graphics Laboratory". For medium sized meshes we used the “Asian Dragon” from “XYZ RGB Inc”. For tests with large meshes the “David” mesh from the “Digital Michelangelo Project” and a randomly created rough planet surface were used. Table 1 shows the data of the meshes.

Name	Vertices	Triangle
Armadillo	172,974	345,944
Asian Dragon	3,609,455	7,218,906
Rough Planet	67,108,866	134,217,724
David[1]	69,881,083	139,749,343

Table 1. Test meshes for simplification

The amount of triangles in internal memory was limited to 1,500,000 triangles for the local surface portions. This is equal to a memory consumption of around 230 MB. Due to the repeated tests for local minimal collapse errors, this algorithm executes of course much slower than In-Core algorithms. Table 2 shows the minimum, maximum and average "collapses per second", the overall simplification time and the hierarchy size in external memory for the test models. All test have been done on a Athlon 3800+ PC with 4GB of internal memory.

Name	Armadillo	Asian Dragon	Rough Planet	David
Avrg col/s	1663	1137	672	667
Min col/s	1397	797	97	103
Max col/s	1904	1825	1783	1764
Time h:m	0:02	0:53	27:44	29:14
Disc size in MB	19.7	413	7680	7855

Table 2. Simplification test results

The watertight meshes “Armadillo”, “Asian Dragon” and “Rough Planet” were all simplified to a tetrahedron as the base mesh. The base mesh of the “David” model consists of 1463 vertices after simplification due to small errors in the mesh surface. Small meshes, which can be cached completely by the operating system, show significantly higher collapse rates as large meshes. But the average rate of large meshes does not fall below 40% of the average rate of small meshes. Compared with other methods the achieved collapse rates are relatively low. But an implementation similar to the Spanned Mesh [San00] algorithm, which uses the correct collapse sequence, delivered much lower ratios, especially for large meshes (e.g 40 hours for the “Asian Dragon”). The main advantages of the new algorithm are, the hierarchy structure, which is equal to In-Core simplification algorithms, and the use of locality on the mesh surface for simplification.

View Dependent Rendering

The extraction performance for the external collapse hierarchy is comparable to In-Core variants. The only difference is, that parts of the hierarchy which does not remain in internal memory, have to be read during some frames from external memory. As soon as the desired hierarchy data remains in internal memory, there is no difference to InCore algorithms. Due to the fine granularity of the hierarchy, the extraction wont influence the desired frame rate very much.

All hierarchies were approximated with a maximum of 15,000 triangles. The relatively low number of triangles was chosen due to the approximation algorithm that does not make use of frame to frame coherency. An improved approximation algorithm should easily reach higher numbers of triangles at high frame-rates. Figures 5 shows approximated views of the four test models. The approximation detail can change while the view is moving, because

[1] Version of the mesh repaired with PolyMender [Ju04] written by Tao Ju

access to external memory may decrease the number of triangles, which are visible, in the desired frame time temporarily.

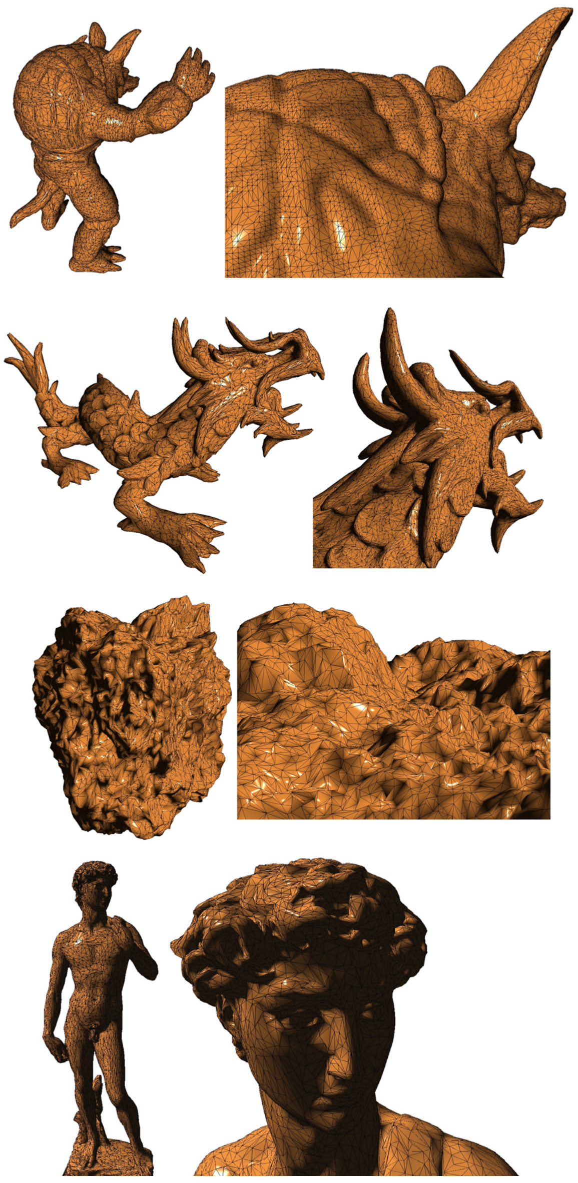

Figure 5. Approximated views of the test meshes (full view and cutout)

5. CONCLUSION

In this paper a new algorithm for the simplification of large meshes was described. It was shown how to use locally minimal collapse errors on the mesh surface in order to create collapse hierarchies, which are equal to ones produced by In-Core methods, while making use of locality on the mesh surface. The algorithm shows good simplification ratios compared with the algorithm of El-Sana [San00], which uses collapse sequences equal to In-Core algorithms. Furthermore it was described how to manipulate the hierarchy for fast triangle updates in the reconstruction process and its usage the hierarchy for view-depended rendering.

6. ACKNOWLEDGMENTS

I would like to thank Mr. Marc Levoy and the people working on the Digital Michelangelo Project for providing their models.

7. REFERENCES

[Cig02] P. Cignoni , C. Montani, C. Rocchini, R. Scopino, “External memory management and simplification of huge meshes”. IEEE Transactions on Visualization and Computer Graphics. 2002

[Gar97] M. Garland, P.S. Heckbert, “Surface Simplification Using Quadric Error Metrics”, SIGGRAPH ’97 Conf. Proc., pp. 209-216, 1997

[Hop96] H. Hoppe, “Progressive meshes“, Computer Graphics, 30(Annual Conference Series),pp. 99-108, 1996

[Hop98] H. Hoppe, “Smooth View-Dependent Level-of-Detail Control and its Aplications to Terrain Rendering,” Proc. IEEE Visualization ’98 Conf., pp. 35-42, 1998

[Ise03] M. Isenburg, P. Lindstrom, S. Gumhold, and J. Snoeyink, “Large Mesh Simplification using Processing Sequences”, IEEE Visualization 2003, pp. 465-472, 2003

[Ju04] T. Ju, “Robust Repair of Polygonal Models”, Proceedings of ACM SIGGRAPH, pp. 888-895, 2004

[Kob98] L. Kobbelt, S. Campagna, J. Vorsatz, and H.-P. Seidel. „Interactive multi-resolution modeling on arbitrary meshes”, In Proceedings of the 25th annual conference on Computer graphics and interactive techniques, pp. 105-114, 1998

[Lev00] 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,” SIGGRAPH 2000, Computer Graphics Proc., pp. 131-144, 2000

[Ros93] J. Rossignac and P. Borrel, “Multi-Resolution 3D Approximation for Rendering Complex Scenes,” Geometric Modeling in Computer Graphics, pp. 455-465, 1993

[Lind03] P. Lindstrom, “Out-of-core construction and visualization of multiresolution surfaces”, Proceedings of the 2003 symposium on Interactive 3D graphics, pp. 93-102, 2003

[San00] J. El-Sana and Y.-J. Chiang, “External Memory View-Dependent Simplification,” Computer Graphics Forum, vol. 19, no. 3, pp. 139-150, 2000

[Xia96] J. Xia and A.Varshney, “Dynamic View-dependent Simplification for Polygonal Models”, Proceedings of IEEE Visualization, pp. 327-334, 1996