Parallel iso-surface extraction and simplification

Christine Ulrich
University Marburg
FB 04 – Psychology
35032 Marburg, Germany
ulrichch@staff.uni-marburg.de

Nico Grund, Evgenij Derzapf
Sirona Dental Systems GmbH
64625 Bensheim, Germany
nico.grund@sirona.com
evgenij.derzapf@sirona.com

Oleg Lobachev, Michael Guthe
University Bayreuth
CS 5 – Visual Computing
95447 Bayreuth, Germany
oleg.lobachev@uni-bayreuth.de
michael.guthe@uni-bayreuth.de

ABSTRACT

When extracting iso-surfaces from large volume data sets, long processing times are required and a high number of polygons is generated. We propose a massively parallel iso-surface extraction and simplification algorithm. The extraction is based on the marching cubes algorithm. In order to process large volume data sets, we perform the extraction with an interleaved simplification step using parallel edge collapses and the quadric error metrics. Interleaving extraction and simplification is based on locally postponing collapse operations close to the processing front. In contrast to previous methods, we do not need an explicit simplification error fall-off close to the front. Thus we can produce meshes with the same quality as if we would simplify the complete mesh after extraction. By implementing both extraction and simplification on the GPU, we can reconstruct high quality iso-surfaces from large data sets within a few seconds.

Keywords
marching cubes, simplification, reconstruction.

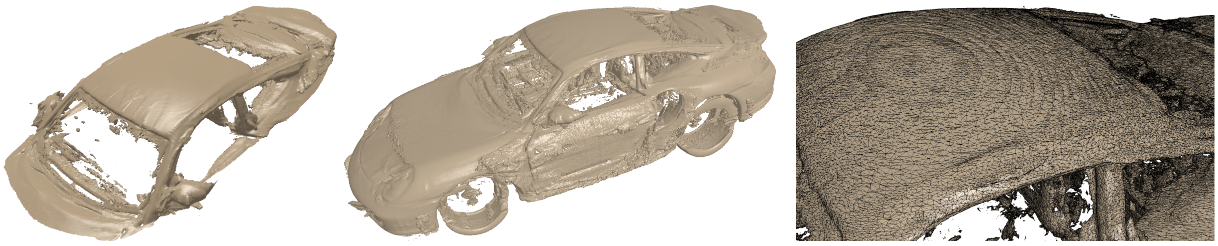
Figure 1: Iso-surface extraction and simplification results for the Porsche data set with an iso-value of 14. The left image shows an intermediate result during the interleaved extraction and simplification and the right image shows the triangles of the generated mesh.

1 INTRODUCTION

In the last decades, the resolution of volume data sets has constantly grown and memory requirements increased. When extracting iso-surfaces from such large volume data sets, a high number of polygons is generated, which also results in long processing times. To efficiently generate meshes from such volumes (especially medical data sets like high resolution histological scans), a fast and efficient algorithm is needed. We combine a massively parallel iso-surface reconstruction algorithm with efficient, high quality parallel simplification.

The marching cubes algorithm is commonly used for iso-surface extraction. It allows producing meshes from large data sets in reasonable time, but – like almost all of its variants – produces a high number of triangles. Hierarchical approaches (e.g. [MS93]) were proposed to reduce the triangle count during iso-surface extraction. These are however less efficient for tubular structures because they can only uniformly adapt the resolution. To further reduce the mesh, simplification algorithms can be used *after* the extraction or directly *during* this process. The latter can be implemented using a plane sweep algorithm or by partitioning the volume into a regular grid or a space partitioning hierarchy. The processing front – i. e. the outer surface of the already processed volume – needs to be constrained during simplification. Vertices on the front must not collapse, which leads to artifacts, like sliver triangles. The so-called tandem algorithm [ACE05] alleviates this problem using a time-lag. The simplification error gradually falls off closer to the front and thus the vertex density increases. While the mesh quality is better, the error falloff constitutes a trade-off between memory consumption and mesh quality.

The main contribution of this paper is a novel formulation of the tandem algorithm that does not require a time-lag. The key advantage is that the resulting mesh is equivalent to that produced by a complete extraction followed by a sequential simplification. Our implementation is based on a parallel edge collapse algorithm using the quadric error metrics to optimize the vertex positions and normals. Using the GPU for both extraction and simplification allows us to quickly produce high quality meshes from large volume data sets.

2 RELATED WORK

The main idea of our approach is to interleave iso-surface extraction and simplification on the GPU. We therefore review related approaches on iso-surface extraction and simplification, as well as combinations of both.

2.1 Iso-surface extraction

The marching cubes (MC) algorithm [LC87] divides a voxel grid into cubes and processes each of them independently. Several improvements, including adaptive and dual algorithms, were proposed [WMW86, HGB93, MSS94, SW04], mainly to prevent holes in the mesh or to improve the mesh quality. Using tetrahedrons [TPG99] instead of cubes is also possible. It prevents holes but drastically increases the amount of triangles. Chernyaev proposed a topologically correct iso-surface generation based on tri-linear interpolation [Che95]. Other approaches directly reconstruct the surface from sets of orthogonal slices (e.g. [SS02, SS04]) using contour matching instead of interpolation. Newman and Yi [NY06] provide an extensive survey on different MC algorithms and variants.

Performance optimizations include using a modified branch-on-need-octree with min-max decisions [WG90] and span spaces [SHLJ96, Liv99] which represent the cubes as two-dimensional points of their min-max values. For a fast extraction of the iso-surface, partitions (lattice elements) or *k*-d-trees are used.

The splitting boxes algorithm [MS93] was one of the first adaptive methods. A cube intersected by the iso-surface is recursively split and simply checked for sign-changes in every edge. The dual marching cubes algorithm [SW04] improves the mesh quality and is able to also generate quad meshes. It is based on using the dual of the volume grid, i.e. it places a vertex in each cell that is crossed by the iso-surface. The quadric error function combined with the method of Lindstrom [Lin00] for positioning dual vertices generate a mesh with better quality and less triangles. This algorithm is also very good in reconstructing sharp features (e.g. edges or corners) but the surface does not accurately approximate the tri-linearly interpolated iso-value.

Recent approaches exploit the processing power of massively parallel graphics processors (GPUs). Reck et al. [RDG$^+$04] proposed an algorithm to extract iso-surfaces from tetrahedral volumes. They pre-select the intersections of surface and voxel grid on the CPU and generate the mesh on the GPU using an interval tree. For rectilinear grids, a tetrahedrization is required, which leads to a higher number of triangles and also introduces artifacts. Johansson et al. [JC06] also use span spaces for pre-selection and pre-classification. They utilize GPU for interpolation and their approach is not restricted to tetrahedral grids any more. Tatarchuk et al. [TSD07] propose a hybrid implementation of marching cubes and marching tetrahedra running on the GPU. The iso-surface is generated on the GPU but again contains a higher number of triangles than the original marching cubes algorithm due to the tetrahedrization. A recent improvement to marching cubes on the GPU uses the so-called HistoPyramids [DZTS08].

2.2 Simplification

Mesh simplification is one of the most common techniques for real-time rendering of complex polygonal models and has been an active field of research over the last two decades. A detailed review of simplification algorithms is given by Luebke [Lue01]. As we aim at efficient iso-surface reconstruction and simplification from large volumes, we focus on real-time capable simplification algorithms.

Uniform vertex clustering [RB93] subdivides the bounding box of the model into cells using a regular grid. All vertices inside the same grid cell are collapsed to their mean. An improved variant is weighted vertex clustering [LT97]. It better preserves features that are not aligned with the grid. Uniform clustering is relatively fast and gives a precise upper bound for the simplification error. However, a further reduction in flat regions is still possible without increasing the simplification error.

The vertex pair contraction [GH97, PH97] has become the most common technique for the simplification of mid-sized triangle meshes. In combination with the introduced quadric error metric, it allows a flexible control over the geometric error and can be used to calculate optimal vertex positions. This approach was also extended to handle an arbitrary number of vertex attributes [GH98]. A combination of vertex clustering with error quadrics [Lin00] improves the placement of the clustered vertices, but still uses a high number of triangles in flat regions. Shaffer and Garland [SG01] proposed to overcome this problem by using a BSP tree instead of a uniform grid. This increased the run time significantly compared to uniform clustering, but the method is still faster than edge collapse simplification. An adaptive vertex clustering using octrees was also proposed by Schaefer and Warren [SW03]. Here the run time is even higher than using a BSP tree, but the quality of the simplified mesh can almost compete with edge collapse simplification.

DeCoro and Tatarchuk [DT07] proposed a parallel GPU implementation of vertex clustering [SW03] by implementing an efficient GPU based data structure. While the performance is very high, it still has the same quality problems as uniform vertex clustering. Recently, a parallel GPU implementation of edge collapse simplification using quadric error metrics [GDG11] has been proposed, which we extend in our work.

2.3 Hybrid algorithms

The hybrid algorithms of Attali et al. [ACE05] and Dupuy et al. [DJG$^+$10] directly simplify the iso-surface mesh during extraction. The first one uses a sequential marching cubes in a tandem with a simplification algorithm. The extraction and simplification steps alternate layer-by-layer. A *time-lag* is introduced to delay collapses until the extraction front is further away, resulting in a better quality of the simplified mesh. The algorithm of Dupy et al. [DJG$^+$10] improves this approach by using a load-balanced cluster to parallelize extraction and simplification. In addition, they do not use a plane sweep any more but partition the volume using an octree. Finally, those parts of the mesh that cannot be further simplified are stored on disk to reduce the memory consumption. A similar approach has been proposed for the reconstruction of surfaces from point clouds has been proposed by Cuccuru et al. [CGM$^+$09]. It combines streaming, iso-surface extraction, and simplification by applying local vertex clustering with topology preservation to produce good quality meshes.

3 OVERVIEW

The core idea of our approach is to interleave a parallel marching cubes with a parallel stream simplifier. As both algorithms run on the GPU, we also minimize communication between host and device as we only transfer the reduced mesh. We chose a plane sweep partitioning due to simplicity, i.e. we process the data in layers, although other partitionings would easily be possible.

Our method consists of two basic modules. The first one is the actual surface extraction based on the marching cubes algorithm. The second module is the mesh simplification that receives the output of the first one as input. The simplification is based on edge collapse operations that contract edges by collapsing two connected vertices. The position and normal of the collapse vertex are computed by minimizing the quadric error metric [GH97] which also defines the collapse cost.

4 PARALLEL MARCHING CUBES

Similar to Attali et al. [ACE05], we partition the volume into layers for processing. We do however split the volume into partitions containing multiple layers if they fit into memory and process them in parallel. The first kernel calculates the *cube codes* – i.e. eight-bit values encoding if the vertices are inside or outside. This kernel also calculates the intersections between the iso-surface and the cube edges if there are any. Assuming that the four corners and four edges of the cube's bottom are already processed, every thread processes vertex v_4 and the edges e_4, e_7 and e_8 only (drawn blue in Figure 2). In addition, the kernel computes the gradient at vertex v_4. Note, that we use a specific kernel for the first slice of each partition to improve performance.

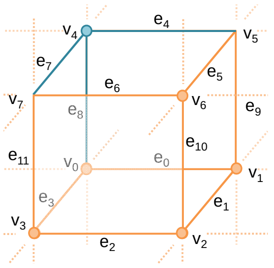

Figure 2: Local edge and vertex indices used during the extraction. Each thread processes the blue elements only.

The second kernel produces the triangles and thus generates the mesh contained in the layers. It uses the classical marching cubes look up table defining the surface topology. This kernel also removes degenerate triangles and feeds the mesh to the second module, the simplification.

We divide a volume of dimension $dim_X \times dim_Y \times dim_Z$ into *layers* and *slices* like Attali et al. [ACE05]. The k^{th} *slice* contains all vertices with the same y-coordinate. The k^{th} *layer* is the set of all vertices, edges and patches between or on the k^{th} and the $(k+1)^{\text{st}}$ slice. Figure 3 shows the relationship between slices and layers. So the volume comprises the slices from 0 to $dim_Y - 1$ and layers from 0 to $dim_Y - 2$. We then process the layers in ascending order.

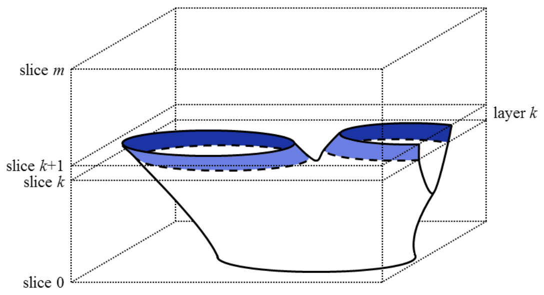

Figure 3: Relationship between slices and layers. The k^{th} layer is composed of slices k and $k+1$.

We group the slices and layers into partitions during surface extraction. Every partition is composed of N

slices and $N-1$ layers respectively. These are processed in two loops and mainly two kernels (see Algorithm 1). Note that two additional slices are required to compute the surface normals from the gradients. Consequently, the partitions overlap by one slice in each direction. As the vertices of the first slice were already calculated in the previous partition, we do however only need one additional slice in each partition. This means that a partition with N slices only contains $N-2$ layers that can be used for the extraction (see Algorithm 1). In addition, we use separate kernels for the first and last slices for this reason.

```
$Partitions$, $Remainder$ = gen_partitions($dimY$)
kernel_cubecode_init()
**for each** $partition\_index$ in $Partitions$ **do**
    kernel_cubecode($partition\_index$, $N-2$)
    kernel_generate($partition\_index$, $N-2$)
    call_simplification_module()
kernel_cubecode_other($Remainder$)
kernel_generate_other($Remainder$)
call_simplification_module()
```

Algorithm 1: Parallel extraction and simplification.

The cube code kernel is executed for each cube (see Algorithm 2) using a thread block dimension of 16×16 or 32×32 depending on the GPU. Assuming that the four corners and four edges of the cube's bottom are already processed, every single thread just processes vertex v_4 and the edges e_4, e_7 and e_8 (see Figure 2). An exception are threads on the 'right' and/or 'front' border of a grid that need to process additional vertices.

Every thread also calculates the gradient of vertex v_4. The gradients and the cube codes always have to be calculated for the previous layer as well. A thread sets the first four bits of the cube code and then shifts the code four bits to the right at the end. The parallelization enforces to enumerate cube edges not just locally, as depicted in Figure 2, but also globally. This prevents a repeated calculation of intersection points between the iso-surface and cube edges. Algorithm 2 gives an overview of the cube code kernel. Note that Algorithms 1 and 2 omit special cases that have to be handled explicitly, e. g. when the volume contains only a single partition.

```
$k = partition\_index(N-2)$
// compute the global index k
**for each** layer in a $partition$ **do**
// a partition consists of $N-2$ layers
    **for each** $cube \in k^{th}$-layer **do in parallel**
        calculate_gradients()
        generate_cubecode()
        **if** $0 < cubecode < 255$
            calculate_intersections()
        shift_cubecode()
    $k = k+1$
```

Algorithm 2: Cube code kernel.

After the first kernel has calculated the intersections, gradients and cube codes, *kernel_generate* produces the mesh using the classical marching cubes look up table. The look up table stores the topology of the surface. This means that and only intersections of cube edges and the iso-surface have to be calculated and the corresponding mesh is fetched from the look up table. The result is a level set $I_\rho := \{v \in \mathbb{R}^3 : f(v) = \rho\}$, whose equation can be simplified by subtracting the iso value ρ from the whole data set. Table 1 lists all data structures required during iso-surface extraction. Finally, this kernel also removes degenerate triangles and feeds the mesh to the second module, the simplification.

buffers	memory (bytes)
slices	$Ndim_X dim_Z$
gradients	$12Ndim_X dim_Z$
voxel edges	$24((3N-1)dim_X dim_Z - N(dim_X + dim_Z))$
cube code	$(N-1)(dim_X - 1)(dim_Z - 1)$
triangles	$60(N-1)(dim_X - 1)(dim_Z - 1)$

Table 1: Memory consumption and data structures required for the iso-surface extraction. The size of the input grid is $dim_X \times dim_Y \times dim_Z$, while N is the number of slices in each partition.

5 PARALLEL SIMPLIFICATION

The simplification module is based on a parallel GPU simplification [GDG11] originally developed for in-core processing of a single mesh. We implemented the following changes to combine it with the stream extraction: First, the mesh generated from N slices is transferred from extraction module. At this point, it does not matter

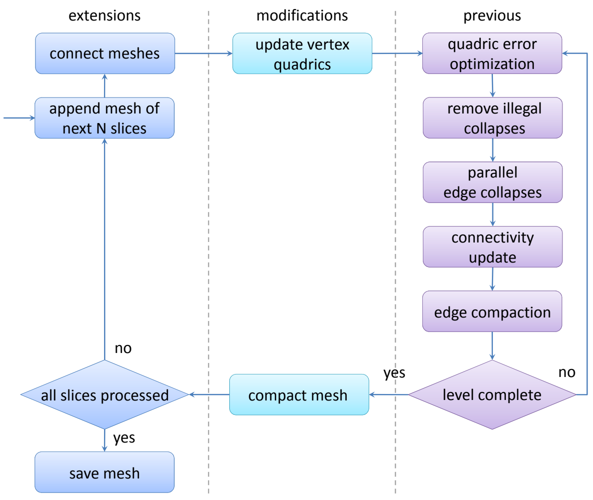

Figure 4: Extensions (left) and modifications (middle) of the previous simplification algorithm (right).

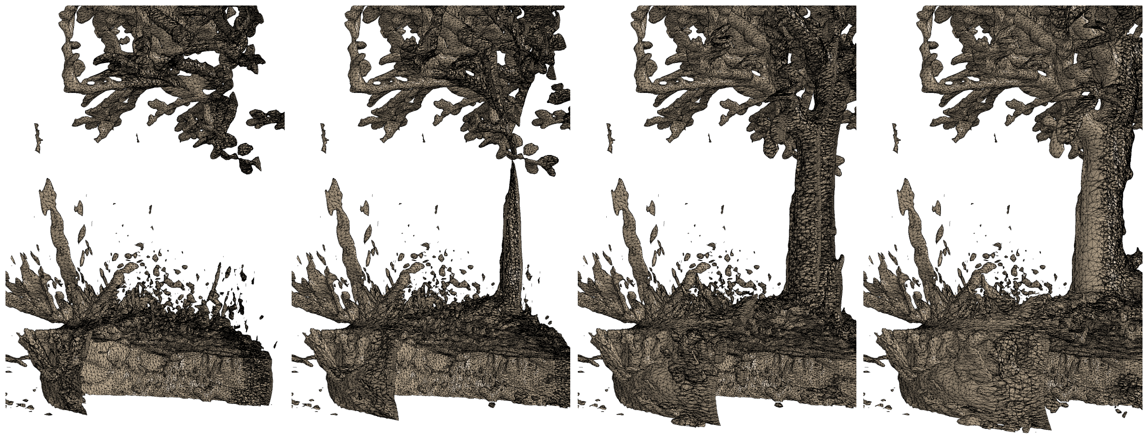

Figure 5: Interleaved extraction and simplification (bonsai #2, iso 50). The mesh is gradually simplified as the processing front moves without using an explicit time-lag.

anymore how the extraction module operates, i.e. we could simply replace the marching cubes algorithm with marching tetrahedrons or dual marching cubes. Then the vertex quadrics are computed for all new vertices and vertices on the previous processing front. The vertices along the processing front are marked in the extracted mesh such that they can be identified for the constrained simplification and to later combine the mesh with the next partition. The edge data structure is filled as in the original simplification algorithm [GDG11] and the parallel simplification loop starts. The quadric error is optimized and illegal collapses are removed. An edge can only collapse if it is not connected to the processing front, i.e. none of its vertices are marked. In addition, collapses of direct neighbors are also not possible since we do not know their local ordering yet. This can be easily achieved by setting the error of edges connected to the processing front to -1. After removing all illegal collapses, the operations are applied. Finally, the collapsed edges are removed after updating the face and edge connectivity. If no further collapses are possible, the next partition is added to the partially simplified mesh. During compaction, the new positions of the processing front vertices inside the vertex buffer are stored in a look up table. Figure 4 shows an overview of the simplification process.

Instead of a time-lag [ACE05], our simplification algorithm locally postpones only those collapse operations that cannot be performed yet. As the collapse operations are local, a mesh with same quality can be produced by any global operation ordering, as long as the local order remains fixed. We exploit this by simply blocking all operations in the direct neighborhood of the processing front. When additionally enforcing the correct local ordering, the simplification error automatically decreases for vertices close to the current boundary (see Figure 5). This way, the result will be identical to a simplification of the complete mesh and the operations are performed as soon as possible minimizing memory consumption.

The claim above can be proven by analyzing which operations are performed in a sequential simplification and if these are the same in our algorithm. A sequential simplification collapses the vertex pair with the smallest error until a given threshold is reached. As the error of the neighboring edges increases during the collapse, their error will always be higher after collapsing them, if it was higher before. This implies that the edge is also collapsed by our method at some time during the simplification. If an edge is collapsed by our method, then all neighboring edges have a higher error. This means that a sequential algorithm also has to collapse exactly this edge before neighboring edges could be collapsed. Finally, setting the error of edges connected to boundary vertices temporally to -1 and preventing their collapse, only causes a local delay of the simplification, because the neighboring edges also cannot collapse yet.

When the next partition is added, we simply need to check the marked vertices – i.e. those on the processing front – and find the in the next partition. Using a plane sweep algorithm, these will simply be the first extracted vertices. For other partitioning schemes we simply need to assure that the vertex order is the same in the partitions to combine both meshes in linear time.

6 RESULTS

We performed our evaluations on a PC with Intel Core i7 CPU (3.33 GHz), 6 GB of main memory, and an NVIDIA GTX 580 graphics card. We used CUDA 5.0 to implement the parallel reconstruction and simplification. The models are taken from "The Volume Library" [Roe13]. Table 2 gives an overview of the volume data sets we used for evaluation. The largest volume is the "Porsche" car model. It also produces the highest number of faces and thus has the highest resource consumption. Bith, "bonsai #2" and "CTA head" are medium sized. The bonsai contains many ramifications, leading to a high number of faces in the simplified model. Table 3 shows the relative and absolute number

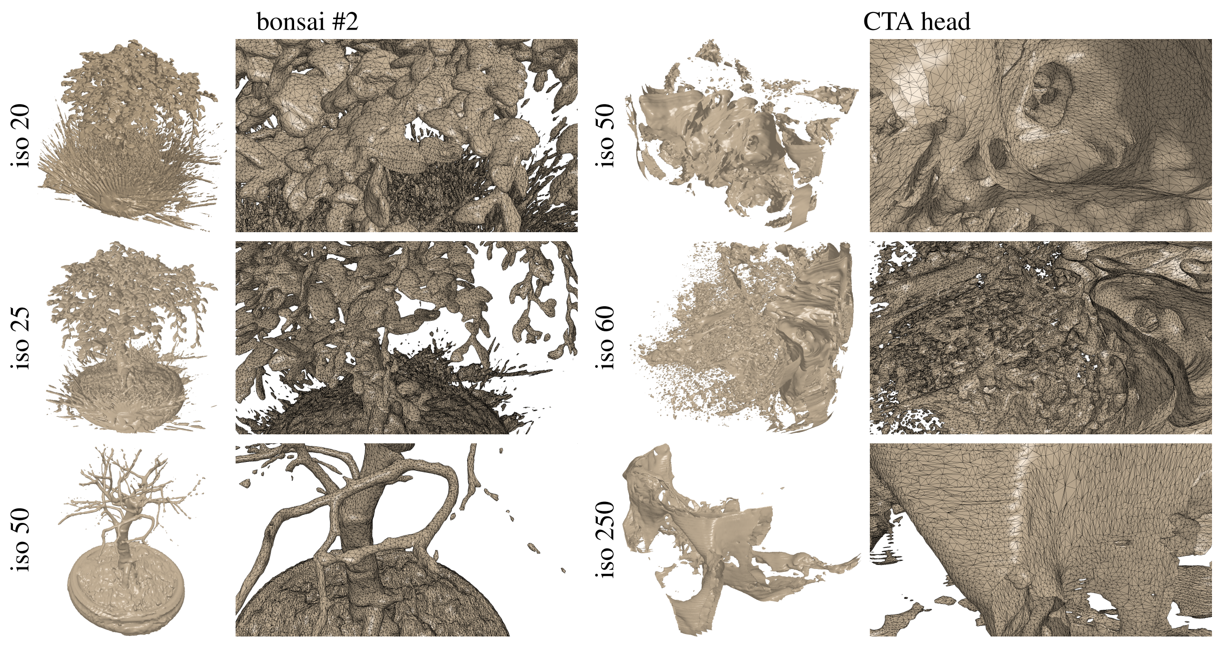

Figure 6: Renderings of the generated meshes of bonsai #2 (iso 20, 25, and 50) on the left and the CTA head (iso 50, 60, and 250) on the right. For all models, closeups with wire frame overlay are shown besides of them.

model	dimensions	size
bonsai #2	512 × 189 × 512	48.384
CTA head	512 × 120 × 512	30.720
Porsche	559 × 347 × 1023	193.784

Table 2: Dimension and file size of the models [Roe13] used for evaluation. The size denotes MBytes in RAW format.

of cubes crossed by the iso-surface, where the number of generated triangles is roughly twice of that. As the generated mesh also depends on the iso-value, we denote it together with the model in the following.

model	crossed cubes		# faces
	%	#	
bonsai #2 (20)	4.49	2,203,645	4,405,952
bonsai #2 (25)	2.29	1,125,808	2,252,046
bonsai #2 (50)	0.67	329,858	658,158
CTA head (50)	3.08	956,441	1,913,256
CTA head (60)	9.53	2,961,041	5,878,764
CTA head (250)	2.24	694,745	1,392,432
Porsche (14)	2.40	4,744,499	9,580,084

Table 3: Relative and absolute number of crossed cubes depending on the iso-value, given in parenthesis. In addition, the number of generated faces before simplification is shown.

6.1 Performance

We used 12 layers per iteration of the algorithm as partition size. The extraction time is almost linear in the number of cubes and slightly increases with the number of generated faces (see Table 4) due to the embarrasignly parallel nature of the marching cubes algorithm. The extraction performance ranges from 12.0M (CTA head, 60) to 12.9M (Porsche, 14) cubes per second and the number of triangles from 174k (bonsai #2, 25) to 2.24M (CTA head, 60) per second. The memory consumption is dominated by the volume and gradient data in the current partition with a small overhead for the simplified mesh. Note that there is no significant difference in processing times or memory consumption between medical and other data sets.

model	extr.	simp.	# faces	mem.
bonsai #2 (20)	4.09s	5.74s	4,396,060	480.5
bonsai #2 (25)	3.99s	2.86s	2,245,412	430.6
bonsai #2 (50)	3.78s	0.95s	658,158	394.7
CTA head (50)	2.58s	1.19s	1,845,976	423.4
CTA head (60)	2.62s	5.31s	1,216,734	514.2
CTA head (250)	2.58s	1.37s	1,392,432	411.5
Porsche (14)	15.34s	21.12s	4,140,690	923.2

Table 4: Computation time in seconds for surface extraction and simplification, number of faces after simplification and maximum memory consumption (in MBytes). Cf. Table 2 for number of faces before simplification.

The generated meshes are shown in Figure 1 and 6. The lowest simplification performance is achieved for the "bonsai #2" due to the highly curved surfaces and the highest performace is achieved for the "Porsche" data set which contains many almost flat regions. The ratio of triangles before and after simplification ranges from

20.7% (CTA head, 60) to 100% (bonsai #2, 50). The meshes are simplified at up to 258k collapses per second. The bonsai #2 data set (iso 25) has similar characteristics as the "old bone" [ACE05] but almost three the size. With an iso-value of 20 it resembles the "young bone" data set, again with about three times the size. Unfortunately, the original data sets used by Attali et al. [ACE05] were not available for a direct comparison. On our system, the algorithm of Attali et al. needs about 60 seconds for the bonsai #2 (iso 25) and approximately 125 seconds for bonsai #2 (iso 20), so our algorithm is 8.8 times and 12.7 times faster respectively. Our approach also compares favorably with[DJG+10] although that scales almost linear with the number of CPU cores (4 in our system) and is thus almost four times faster than [ACE05].

6.2 Memory Consumption

Figure 7 shows the total memory consumption when processing the "Porsche" data set in detail. Note that the GTX 580 does not have enough memory to store the complete model, so we have to use the partitioning. For each partition, we plot the maximum memory consumption during extraction and simplification along with the number of faces after simplification. While the memory gradually increases with the size of the generated mesh, it is dominated by the data required to process the current partition.

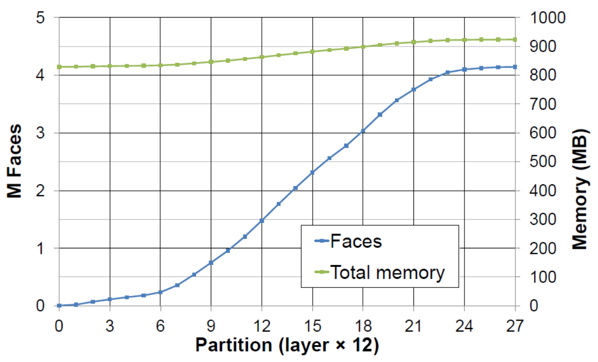

Figure 7: Total memory consumption and number of faces in the extracted and simplified mesh after complete processing of the "Porsche" data set.

6.3 Mesh Quality

As shown above, our method produces meshes of the same quality as the underlying simplification algorithm. We compared our simplification with simplifying the complete extracted mesh using the algorithm of Grund et al. [GDG11]. For the smaller models, the runtime and the simplified mesh were the same (up to floating point round-off errors). The larger models like "CTA-head" at iso-value 60 and "Porsche" do not fit into the graphics memory of the GTX 580 and cannot be simplified with that algorithm.

We experimented with the error-threshold and analyzed the directional bias. In contrast to Attali et al. [ACE05], the directional bias is – as expected – almost zero because we do not require any time lag.

7 CONCLUSION AND FUTURE WORK

Removing the explicit time-lag by locally blocking/postponing simplification operations enables us to reduce memory consumption and processing time. Due to the massively parallel implementation on the GPU, our approach can process large volumes within a few seconds. In addition, it directly benefits from the future improvements of graphics hardware or other parallel systems.

By guaranteeing the same local ordering of collapse operations as a sequential simplification of the complete mesh, we achieve the same quality as the underlying simplification algorithm. This also implies that no artifacts at partition boundaries are introduced. Our improved simplification algorithm works with *any* mesh generation algorithm. So we could easily replace the marching cubes algorithm with dual marching cubes, marching tetrahedrons, or others.

Our current implementation is limited to models for which at least three slices – i.e. one layer – fit into memory because of the plane sweep partitioning. For very large volume data sets, we could use a regular grid instead without changing the core algorithm. The only part that needs to be modified is the fusion of the next partition's mesh with the already simplified one. However, the run time complexity of this step will still be linear as the vertex ordering of the iso-surface extraction is fixed within each partition.

8 REFERENCES

[ACE05] D. Attali, D. Cohen-Steiner, and H. Edelsbrunner. Extraction and simplification of iso-surfaces in tandem. In Proc. 3rd Eurographics Symp. Geometry Processing, SGP '05. Eurographics, 2005.

[CGM+09] G. Cuccuru, E. Gobbetti, F. Marton, R. Pajarola, and R. Pintus. Fast low-memory streaming mls reconstruction of point-sampled surfaces. In Graphics Interface, pages 15–22, May 2009.

[Che95] E. V. Chernyaev. Marching cubes 33: Construction of topologically correct isosurfaces, 1995. (Retrieved 2013-05-22).

[DJG+10] G. Dupuy, B. Jobard, S. Guillon, N. Keskes, and D. Komatitsch. Parallel extraction and simplification of large isosurfaces using an extended tandem algorithm. Comput. Aided Design, 42(2):129–138, 2010.

[DT07] C. DeCoro and N. Tatarchuk. Real-time mesh simplification using the GPU. In Proc. Symp. Interactive 3D Graphics and Games, pages 161–166, 2007.

[DZTS08] C. Dyken, G. Ziegler, C. Theobalt, and H.-P. Seidel. High-speed marching cubes using histopyramids. Comput. Graph. Forum, 27(8):2028–2039, 2008.

[GDG11] N. Grund, E. Derzapf, and M. Guthe. Instant level-of-detail. In Vision, Modeling, and Visualization, VMV '11, pages 293–299, 2011.

[GH97] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In Proc. 24th Conf. Computer Graphics and Interactive Techniques, SIGGRAPH '97, pages 209–216, 1997.

[GH98] M. Garland and P. S. Heckbert. Simplifying surfaces with color and texture using quadric error metrics. In Proc. Conf. Visualization, pages 263–269, 1998.

[HGB93] W. Heiden, T. Goetze, and J. Brickmann. Fast generation of molecular surfaces from 3D data fields with an enhanced "marching cubes" algorithm. J. Comput. Chem., 14(2):246–250, 1993.

[JC06] G. Johansson and H. Carr. Accelerating marching cubes with graphics hardware. In Proc. Conf. Center for Advanced Studies on Collaborative research, CASCON '06. ACM, 2006.

[LC87] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. Comput. Graph. (ACM), 21(4):163–169, 1987.

[Lin00] P. Lindstrom. Out-of-core simplification of large polygonal models. In Proc. Conf. Computer Graphics and Interactive Techniques, SIGGRAPH '00, pages 259–262, 2000.

[Liv99] Y. Livnat. Noise, Wise and Sage: Algorithms for Rapid Isosurface Generation. PhD thesis, University of Utah, 1999.

[LT97] K.-L. Low and T.-S. Tan. Model simplification using vertex-clustering. In Proc. Symp. Interactive 3D Graphics, pages 75–81, 1997.

[Lue01] D. P. Luebke. A developer's survey of polygonal simplification algorithms. IEEE Comput. Graph., 21, 2001.

[MS93] H. Müller and M. Stark. Adaptive generation of surfaces in volume data. Visual Comput., 9(4):182–199, 1993.

[MSS94] C. Montani, R. Scateni, and R. Scopigno. A modified look-up table for implicit disambiguation of marching cubes. Visual Comput., 10(6):353–355, 1994.

[NY06] T. S. Newman and H. Yi. A survey of the marching cubes algorithm. Comput. Graph., 30(5):854–879, 2006.

[PH97] J. Popović and H. Hoppe. Progressive simplicial complexes. In Proc. Conf. Computer Graphics and Interactive Techniques, SIGGRAPH '97, pages 217–224. ACM/Addison-Wesley, 1997.

[RB93] J. Rossignac and P. Borrel. Multi-resolution 3D approximations for rendering complex scenes. In Modeling in Computer Graphics, IFIP, pages 455–465. Springer, 1993.

[RDG$^+$04] F. Reck, C. Dachsbacher, R. Grosso, G. Greiner, and M. Stamminger. Realtime isosurface extraction with graphics hardware. In Eurographics 2004, Short Presentations and Interactive Demos, pages 33–36, 2004.

[Roe13] S. Roettger. The volume library, 2013. Retrieved 2013-05-22.

[SG01] E. Shaffer and M. Garland. Efficient adaptive simplification of massive meshes. In Proc. Conf. Visualization '01, pages 127–134, 2001.

[SHLJ96] H.-W. Shen, C. D. Hansen, Y. Livnat, and C. R. Johnson. Isosurfacing in span space with utmost efficiency (ISSUE). In Proc. Conf. Visualization '96, VIS '96, pages 287–295. IEEE CS, 28-29 October 1996.

[SS02] R. Sviták and V. Skala. Surface reconstruction from orthogonal slices. In ICCVG 2002, 2002.

[SS04] R. Sviták and V. Skala. A robust technique for surface reconstruction from orthogonal slices. MG&V, 13(3):221–233, January 2004.

[SW03] S. Schaefer and J. Warren. Adaptive vertex clustering using octrees. In Proc. SIAM Conf. Geometric Design and Computing, 2003.

[SW04] S. Schaefer and J. Warren. Dual marching cubes: Primal contouring of dual grids. In Proc. Pacific Conf. Computer Graphics, pages 70–76, 2004.

[TPG99] G. M. Treece, R. W. Prager, and A. H. Gee. Regularised marching tetrahedra: improved isosurface extraction. Comput. Graph., 23(4):583–598, 1999.

[TSD07] N. Tatarchuk, J. Shopf, and C. DeCoro. Real-time isosurface extraction using the GPU programmable geometry pipeline. In ACM SIGGRAPH 2007 courses, SIGGRAPH '07, pages 122–137. ACM, 2007.

[WG90] J. Wilhelms and A. V. Gelder. Octrees for faster isosurface generation. Comput. Graph. (ACM), 24(5):57–62, November 1990.

[WMW86] G. Wyvill, C. McPheeters, and B. Wyvill. Data structure for soft objects. Visual Comput., 2(4):227–234, 1986.